IEEE/ACM *Transactions on*
Networking

SPABox: Safeguarding Privacy during Deep Packet Inspection at a MiddleBox

Jingyuan Fan, Chaowen Guan, Kui Ren, *Fellow, IEEE*, Yong Cui, and Chunming Qiao, *Fellow, IEEE*

***Abstract*—Widely used over the Internet to encrypt traffic, HTTPS provides secure and private data communication between clients and servers.**

However, to cope with rapidly changing and sophisticated security attacks, network operators often deploy middleboxes to perform Deep Packet Inspection (DPI) to detect attacks and potential security breaches, using techniques ranging from simple keyword matching to more advanced machine learning and data mining analysis. But this creates a problem: how can middleboxes, which employ DPI, work over HTTPS connections with encrypted traffic while preserving privacy?

In this paper, we present SPABox, a middlebox based system that supports both keyword based and data analysis based DPI functions over encrypted traffic. SPABox preserves privacy by using a novel protocol with a limited connection setup overhead. We implement SPABox on a standard server and show that SPABox is practical for both long-lived and short-lived connection. Compared with the state-of-the-art Blindbox system, SPABox is more than 5 orders of magnitude faster and requires 7 orders of magnitude less bandwidth for connection setup while SPABox can achieve a higher security level.

***Index Terms*—DPI, privacy preserving, middlebox.**

I. Introduction

HTTPS is a popular Internet protocol which uses Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL) [27] to encrypt communication between clients and servers to ensure data integrity and privacy.

Traditionally, many middleboxes that provide deep packet inspection (DPI) functionalities are deployed by network operators to detect attacks by searching for specific keywords or signatures in non-encrypted traffic [23]. As malwares use various concealment techniques such as obfuscation, and polymorphic or metamorphic strategies to try to evade detection [15], [16], both industry and academia have considered adding more advanced machine learning and data mining analysis in DPI [50], [8], [12]. For example, both Symantec and Proofpoint claimed that with machine learning, they are able to detect more attacks and threats than systems without machine learning technology [10], [9]. Nevertheless, with the growing adoption of HTTPS, existing approaches are unable to perform keyword or signature matching, let alone advanced machine learning analysis for malware detection of the encrypted traffic.

Naylor et al. [39] tried to tackle the problem by proposing mcTLS which allows sharing of the encryption keys with middlebox services. However, this approach cannot protect users' private information from a service provider who deploys middleboxes. As a matter of fact, service providers have been using their deployed middleboxes to capture Internet traffic for surveillance and marketing purposes [6], a practice that has been widely criticized. Sherry et al. [43] took the first step to bridge the gap by introducing Blindbox, a system which supports keyword matching and regular expression evaluation over encrypted traffic. They realized it by developing a novel searchable encryption scheme and experimentally proved its feasibility. However, its scope of functionality is limited by the underlying cryptographic method. Specifically, Blindbox still needs to decrypt the traffic at a middlebox in order to perform regular expression evaluation which may reveal information that end users consider private. In addition, it cannot support complex machine learning analysis for malware detection. Furthermore, the overhead of connection setup makes Blindbox impractical. A follow-up work in [35] tried to address the high setup overhead by employing a modified architecture.

In this paper, we present SPABox, the first middlebox based system that supports both signature and data analysis based DPI functionalities over encrypted traffic with a limited overhead. Compared to [35], SPABox achieves an even greater reduction in the setup overhead under more general conditions while [35] only applies to an enterprise environment. With SPABox, when two end points create a communication link, there is no involvement of the middlebox, which preserves its transparency as in other existing middlebox deployments. SPABox accomplishes these by introducing a novel, and yet simple, efficient encryption strategy built on top of the *Discrete Logarithm Problem* [36] and a novel protocol for secure communication between two parties. Although our protocol has a mapping of encrypted keywords (and partial keywords) to their plain text representations, SPABox can neither decrypt the traffic, nor infer any private information, which is stronger than the privacy models in the previous work [43].

Specifically, SPABox supports three types of operations: keyword matching (Section IV-B), regular expression evaluation (Section IV-C) and malware detection via machine learning (Section IV-D). The main ideas of our design behind these three operations are 1) instead of matching keywords directly, we tokenize the keywords so that we can build a

Manuscript received January 20, 2002; revised January 30, 2002. This work was supported in part by the National Science Foundation of China under Grant 61422206 and in part by the NSF under Grant CSR-1409809. Kui's research is supported in part by National Science Foundation under Grant CNS-1262277.

J. Fan, C. Guan, K. Ren and C. Qiao are with the Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY 14260 USA (e-mail: jfan5@buffalo.edu; chaoweng@buffalo.edu; kuiren@buffalo.edu; qiao@computer.org).

Y. Cui is with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, P.R.China (e-mail: cuiyong@tsinghua.edu.cn).

Digital Object Identifier: 10.1109/TNET.2017.2753044

Trie-like structure to accelerate searching and matching at a DPI middlebox, 2) regular expressions are processed at the receiver side, and garbled circuits and oblivious transfer are used to protect the privacy of both traffic and rule sets, and 3) we utilize the homomorphic properties on addition and scalar multiplication, respectively.

To validate our design, we analyze the security guarantees of our design, implement SPABox on a standard server and evaluate its performance with several real data sets and traffic. We consider performance metrics such as throughput, bandwidth and memory overhead on the sender, middlebox and receiver sides, and show that SPABox is practical for both long-lived and short-lived connections.

The rest of the paper is organized as follows. Section II describes the security models and protocol requirements. Section III presents the proposed system architecture, followed by detailed protocol designs in Section IV. Section V discusses the implementation of our system. We evaluate the performance in Section VI to show the effectiveness and efficiency of SPABox. Section VII surveys the related work and Section VIII discusses possible extensions, alternative approaches, and differences between our scheme and Blindbox, followed by conclusions and future work in Section IX.

II. Security Models and Protocol Requirements

The goal of this work is to protect the privacy of user traffic from a middlebox which performs DPI while leveraging advanced DPI functionalities to detect malicious traffic. In other words, the middlebox can detect attacks over encrypted traffic with rule sets and well-trained machine learning (ML) models, which are provided by a (third party) rule generator. In this section, we first describe the security models and then outline the protocol requirements.

A. Security Models

In the scenario being considered in this work, we assume that 1) the rule generator is honest, and 2) the middlebox is *honest-but-curious*, i.e., it will implement the rules and follow the protocol honestly but may attempt to infer private information from the encrypted traffic during the execution of DPI.

Thus the goal of our proposed system is to realize the DPI procedures while protecting the data privacy of endpoints. More formally, the security goals that SPABox is designed to achieve are summarized as follows:

1) To guarantee the confidentiality of the unmatched traffic, that is, the traffic that doesn't match the known attack keywords in the rule set will remain secret from the middlebox.
2) To make sure that no private information can be inferred by the middlebox and all analysis results can only be seen by the clients.

This first goal indicates that the middlebox is allowed to learn only the data that are exactly identical to the known suspicious keywords in the middlebox rule set, and the second goal aims to make the middlebox unable to apply data analysis technique to infer other private information about the traffic, such as leakage-abuse attacks [24].

B. Protocol Requirements

Based on the above security models, we identify a few requirements that our protocol should meet. As our goal is to provide privacy of user data at a middlebox over HTTPS connection, first we require SPABox to maintain and extend the properties provided by existing TLS/SSL as follows.

Private Connection. All traffic should be encrypted with the secret key negotiated at the beginning of the session, and the negotiation should be both secure and reliable. Note that only the endpoints can read the unencrypted traffic, while all the traffic exposed to the middlebox should remain encrypted at all times.

Identity Authentication. The identity of endpoints can be authenticated by each other and even the DPI appliance. This can be achieved using public key cryptography and made optional to reduce overhead.

Reliable Connection. The integrity of each message transmitted by one party of the session should be able to be verified by other parties in this session, including trusted middleboxes, to detect unauthorized modification and prevent undetected loss during the transmission.

Besides, we also require SPABox to meet the following new requirements:

Middlebox Transparency. During communication, clients typically do not communicate directly with the DPI middleboxes in existing deployments. In order to comply with current deployments, we try to preserve the transparency of the DPI appliance in our protocol.

Privacy Preservation. In our case, attackers at middleboxes should never be able to read the plaintext exchanged between two endpoints nor extract private information from the data using analysis techniques. The TLS/SSL session key should never be exposed to the attackers at a middlebox under any circumstances.

Endpoint Verification. For the sake of security, besides being able to authenticate the identity of the other communicating party in the session, one should be able to verify whether the other party follows the protocol to prevent her from behaving maliciously.

Function Variety. The protocol should be general enough to support both signature or keyword based and data analysis based DPI functionalities.

Minimum Overhead. Finally, our protocol should operate without a substantial overhead, including bandwidth, latency, computation, etc. Specifically, the latency and bandwidth overhead to establish connections should be small in order to support short, independent flows and a large rule set. The computational overhead at the endpoints should be limited. We also intend to make the computation independent of any specific hardware platforms such that all users can benefit from our protocol.

III. System Architecture

Fig. 1 shows the system architecture, and the highlighted boxes indicate components added by SPABox. As in the previous work [43], there are four parties: sender (S), receiver (R), middlebox (MB) and rule generator (RG). In order to define an

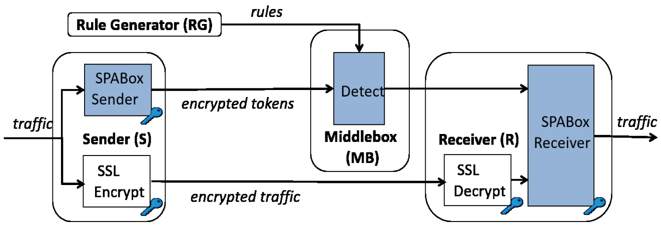

Fig. 1: SPABox system architecture

Sender	$g, n, salt_0, d, r, s, N^2, H(\cdot)$
Middlebox	$g^{salt_0}, g^d, n, g^s, N^2, H(\cdot)$
Receiver	$g, n, salt_0, d, s, N^2, H(\cdot), \lambda, \mu$

TABLE I: Protocol parameter list

attack, the RG such as Symantec [13] and McAfee [4] usually provides a list of attack rules. Each of the attack rules contains a set of keywords and possibly other information such as offset of each keyword, distance between two pattern matches and a regular expression. Besides, the RG also provides a trained model for classifying the encrypted traffic to decide whether the traffic contains malicious program. The MB are usually deployed by network operators like AT&T [46]. The security threats in the traffic can be identified in two ways: *a*) The MB compares the encrypted traffic with the rule set provided by the RG, and observes a matching between the traffic and one of the attack rules in the rule set; *b*) The MB classifies the encrypted traffic with the trained model provided by the RG and the results from the classifier are used by R to decide whether the traffic contains malware.

When S and R want to establish a HTTPS connection in a network monitored by the MB, the following steps take place:

Connection Setup. When S and R try to communicate, they first run SPABox handshake to exchange SSL session key k_{ssl} just as an SSL handshake. Besides, during the SPABox handshake process, S and R also need to negotiate 7 more parameters (SPAPara), g, n, r, s, N^2, $salt$ and a hash function $H(\cdot)$, all of which are used for encryption, decryption and detection in the SPABox protocol. Note that only S and R are involved in an SPABox handshake which bootstraps off the existing SSL handshake, preserving the MB transparency property.

Sending Traffic. At S, two logical connections are set up, to be referred to as an SSL connection and an SPA connection, respectively. On the SSL connection, S encrypts the traffic with unmodified SSL. On the SPA connection, S makes a copy of the traffic, tokenizes it and then encrypts the tokens using the proposed approach based on Discrete Logarithm Problem (Section IV-A). To support malware detection, tokens need to be pre-processed before getting encrypted (Section IV-D).

Detection at MB. There are two tasks for MB to perform once it receives the encrypted tokens from S over the SPA connection. *a*) The MB compares the encrypted traffic with the rule set provided by the RG (including keyword and regular expression); *b*) The MB classifies the encrypted traffic with the trained model provided by the RG; If there is a match between the traffic and the rule set and there is no regular expression in the corresponding attack rule, the MB can choose to drop the packet and inform administrator or issue a warning just as what a regular MB would do over unencrypted traffic. If a regular expression needs to be further evaluated, it will be forwarded to R for further processing (Section IV-C). For malware detection using ML, the classification results should not be seen by the MB. Instead, they should be sent to and can only be seen at R (Section IV-D). The reason to do so is to prevent attackers at the MB from analyzing data using scripts (Protocol Requirement Privacy Preservation in Section II-B).

Receiving Traffic. Upon receiving SSL traffic at R, R would decrypt and authenticate the traffic using regular SSL. Then R will tokenize and encrypt the recovered plaintext which will be compared with the traffic from the SPA connection. The reason we proceed in this way rather than decrypting and detokenizing the traffic from the SPA connection for comparison is that encryption is much faster than decryption in our protocol. By comparing the ciphertexts generated from the traffic over the SSL connection against the encrypted tokens received from the SPA connection, R can determine whether S in this session follows the SPABox protocol correctly, including both keyword matching and malware detection using ML (Protocol Requirement Endpoint Verification in Section II-B). If there is any discrepancy, R may think S is an attacker and immediately drop the connection. Otherwise, R may process information forwarded by the MB containing the classification results and regular expressions to decide whether the traffic from S over the SSL connection is malicious or not.

IV. Protocol Designs

In this section, we give a detailed description of our proposed protocol. First, we describe a hard problem based on which we construct our protocol. Then we present the encryption procedures on how our protocol handles **Keyword Matching**, **Regular Expression** and **Malware Detection via Machine Learning**, respectively. Table I summarizes the parameters that each entity uses, where d is used to compute pseudorandom salt, (λ, μ) is the private key pair for decrypting malware detection classification results (see more detail in Section IV-B, IV-C and IV-D) and the rest has been described earlier (Section III).

A. Intuition

Blindbox requires interactive initialization between a sender and a middlebox (Protocol Requirement Middlebox Transparency in Section II-B) which results in a slow connection setup. In order to avoid such an interaction, public-key cryptographic primitive is preferred in our protocol. Therefore, the key to design a protocol that meets our requirements on both efficiency and security is to select a suitable computational hard problem on which the protocol to be built. As mentioned in the protocol requirement, we aim to design a protocol with the following three properties: *a*) it doesn't rely on any specific hardware support but can work efficiently for all parties to process the data; *b*) the MB is able to perform necessary operations for different kinds of DPI functionalities over encrypted traffic; c) no information exchange is required between clients and the MB during connection setup. Note that the third property can't be satisfied if we use any searchable encryption schemes [18], [44] since these schemes require the MB to obtain search keys from clients, which may become the major performance bottleneck. Taking all these three properties into consideration, we construct our encryption strategy based on the intractability of **Discrete Logarithm Problem** defined as follows.

Definition 1. ***Discrete Logarithm Problem.*** *Given a finite group $\mathbb{G}$, an element $g \in \mathbb{G}$ and an element b in the subgroup generated by g, i.e., $\langle g \rangle$, find an integer x such that $g^x = b$.*

Such an integer x is the discrete logarithm of b to the base g. The Discrete Logarithm Problem is one of the most important one-way functions used in modern asymmetric cryptography. Many public-key algorithms are built based on it, such as the widely used Diffie-Hellman key exchange protocol [28] and the ElGamal encryption scheme [29].

Another two important and useful properties when encrypting message m as an element g^m in $\mathbb{G}$ are the homomorphic features on **addition** and **scalar multiplication**. In a nutshell, these two properties enable our protocol to work directly over the encrypted traffic without decrypting the payload at the MB. Assume $c_1 = g^{m_1}$ and $c_2 = g^{m_2}$, where m_1 and m_2 are plaintexts. The two properties can be summarized as follows:

- **Homomorphic Addition**: the product of c_1 and c_2 gives $g^{m_1+m_2}$ which encrypts message $(m_1 + m_2)$.
- **Homomorphic Scalar Multiplication**: raising c_1 to the power of m_3 produces $g^{m_1 \cdot m_3}$ which encrypts message $m_1 \cdot m_3$, where m_3 is another plaintext in the same message space as m_1, m_2.

B. Keyword Matching

In this subsection, we present our protocol design for performing keyword matching at S, R and the MB. Note that only few minor changes need to be made at the MB, S and R to support the other two functions to be described in Section IV-C and IV-D. An attack rule may contain multiple keywords as well as position information of these keywords. Only if all the keywords in one attack rule are matched with the correct offset, we consider this attack rule is matched.

1) On The Sender Side: Fig. 2 shows the architecture on the SPABox sender side. The first step is to tokenize the traffic to be sent over the SPA connection using a fixed-length sliding window. For example, assume the traffic to send is ”NETWORKING” and we use 5 bytes per token, the tokens generated are ”NETWO”, ”ETWOR”, ”TWORK”, ”WORKI”, ”ORKIN” and ”RKING”. If we tokenize the keywords in the rule set at the MB in a similar way, we can search for keywords whose length are equal to or greater than 5. For instance, say one of the keywords is ”NETWORK”. If the MB tokenizes this keyword using a window of 5 bytes long, two keyword tokens ”NETWO” and ”TWORK” are generated. Then the MB can compare all the tokens received over the SPA connection with these two keyword tokens and see if there are two received tokens that are separated by 2 tokens and equal to these two keyword tokens respectively. The reason we choose to tokenize the traffic in this way will be discussed in Section IV-D. In our implementation, we use 5 bytes as the sliding window length. The reasons that we choose to use 5 bytes are (i) the message space is large enough for our encryption to be secure; (ii) longer sliding window will result in slower encryption speed. Note that a token needs to be converted into an integer of proper format (i.e., mpz_t) before being encrypted.

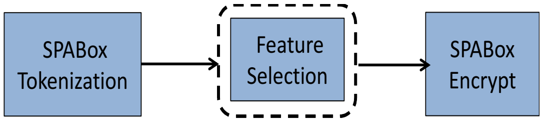

Fig. 2: SPABox sender architecture

After tokenization, S can encrypt each token with our encryption scheme based on *Discrete Logarithm Problem* (feature selection module is used in Section IV-D). Assume the plaintext for a token obtained from the tokenization is t, the ciphertext c is given as:

$$c = g^{salt \cdot t} \mod n \tag{1}$$

where n is a prime number, g is an element in multiplicative group $\mathbb{Z}_n^*$ and $salt$ is an element in the additive group $\mathbb{Z}_n$, all of which are defined during the connection setup phase. One may find that our encryption scheme is similar to the well-known Paillier encryption method [40], but there are major differences, and we will discuss them in Section VIII. Specifically, $salt$ is for randomizing the ciphertext, that is, ensuring that no two ciphertexts consecutively encrypting the same message are identical (similar as randomized encryption methods), and therefore makes the encrypted tokens resistant to frequency analysis. However, if a new random $salt$ is selected for each token, it makes the resulting protocol difficult to support efficient keyword matching operation at the MB, and also pre-computing $g^{salt} \mod n$ is no longer possible. To address this, in our protocol, an initial $salt_0, d \in \mathbb{Z}_n$ are defined. A counter table is also defined for storing k_m, the number of times that each plaintext m has been encrypted so far in the SPABox tokenization module. Therefore, to encrypt a token t, S first looks up the number k_t (i.e., how many times token t has been encrypted so far) in the counter table and then encrypts t as $c = g^{salt_k \cdot t} \mod n$ where $salt_k = salt_0 + k_t d \mod \mathbb{Z}_n$. Note that k is initially set to one.

For security purpose, the bit length of n should be at least 1024 (2048 recommended by NIST). This unfortunately, will lead to huge communication overhead. To maintain the security level while reducing the overhead, we hash each token t using a cryptographic hash function $H(\cdot)$ after tokenization and set n to 160 bits so that each token can be represented with 20 bytes after encryption. In our implementation, we use a hash function SHA-1 with output size of 160 bits. This is a valid step because the message space (40 bits) of a token is greatly smaller that 160 bits. With this modification, S encrypts a token by computing

$$c = g^{salt_k \cdot H(t)} \mod n \tag{2}$$

where $salt_k = salt_0 + kd \mod \mathbb{Z}_n$. Note that this ensures at least two consecutive encryptions of the same message won't produce the same ciphertext (similar as randomized encryption methods), and therefore makes the encrypted tokens resistant to frequency analysis. Unfortunately, the above design is not secure enough (Section IV-E) when the first encrypted token is identified as malicious keyword (i.e., one of the keyword tokens in the rule set). To make our protocol secure against this case, S is required to pick out a random number r of benign tokens and set this sequence of tokens as the prefix preceding the tokenized traffic over the SPA connection. The source from which S selects the benign tokens can be a trusted third party like Certificate Authority (CA) or any system file. We call this process *Benign Prefix Padding* (BPP).

After the tokens are encrypted, they are sent out together over the SPA connection with auxiliary information (AUXInfo) g^{salt_0}, g^d, r, n and the hash function $H(\cdot)$, which are used by the MB for keyword matching. One can think of the AUXInfo as public keys for one particular session. For simplicity of notation, we denote the Eq. 2 as $enc(t,k)$, where k is the times that token t has appeared in the traffic so far. To prevent the counter table from growing too large, S resets $salt_0$ every M distinct token sent, where M can be set to a large number, say 10M, so as to keep the overhead for resetting at a minimal level.

2) On The Middlebox Side: In order to support keyword matching, the MB needs to perform the following operations.

Rule Preparation. The keywords in the rule set need to be processed. Upon receiving the rule set from the RG, the MB first tokenizes all the keywords in the rule set using a window of the same length which R uses to tokenize the traffic sent over the SPA connection. For a keyword of length l, $\lceil \frac{l}{5} \rceil$ keyword tokens are generated. Then the MB would transform all tokenized keywords into large numbers as mentioned earlier (i.e., in mpz_t format).

When S tries to send traffic to a R, the MB would receive from S a set of AUXInfo.With such information, the MB can hash and then encrypt all the tokenized keywords w as $enc(w,1)$. The reason $k = 1$ here is that the MB has not yet seen keyword w in the traffic. It is worth noting that in our protocol, no information exchange between S and the MB is needed during the rule preparation step, while this is the major performance bottleneck in [43]. We will compare the overhead in Section VI.

Keyword Matching. With encrypted keyword tokens, the MB can perform keyword matching as follows. First consider a simple example with one received encrypted token $enc(t,1)$ and one encrypted keyword token $enc(w,1)$. To check if t is equal to w, the MB only needs to check if $enc(t,1)$ is equal to $enc(w,1)$. Upon matching, the MB simply calculates $enc(w,2)$ as $enc(w,1) \cdot g^{d_0 \cdot w} \mod n$, and replace $enc(w,1)$ with this new value since the next time S sends token t over the SPA connection, the corresponding ciphertext would be $enc(t,2)$. Next, we extend our keyword matching example with multiple encrypted keyword tokens and one received encrypted token. In order to support searching over multiple keyword tokens, the MB needs to maintain a token table which records the number of times that each keyword token has appeared in the traffic. Once there is a match, the MB can re-encrypt the matched keyword token as in the one keyword token case, and replace the matched entry in the token table with the new encrypted keyword token. But a practical rule set may have hundreds of encrypted keyword tokens [11], *how can the MB perform an efficient search*?

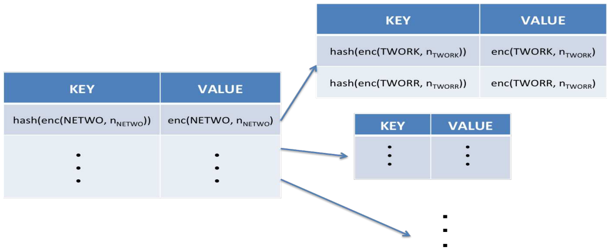

Fig. 3: Hierarchical hash table

The first idea is to build up a searching tree with each element being an encrypted keyword token. When receiving an encrypted token from S over the SPA connection, the MB can look up the tree to find if there is a match, which makes the searching time logarithmic in the number of keyword tokens. However, even with logarithmic searching time, this would still result in too much overhead: *a*) With more than 10,000 keywords in a rule set which is typical (the number of distinct keyword tokens could be even more), it may take more than 15 comparisons to find a matching. *b*) Since encountering a malicious keyword is relatively rare, computation power may be wasted on searching for an encrypted token not present in the searching tree. *c*) Since keywords are tokenized, matching a rule may needs multiple searches over the tree. In our rule set from Snort Emerging Threats [11], there are up to 81 keyword tokens for one rule. On the contrary, our solution is simple yet efficient. Instead of building a searching tree, we use a **Token Hash Table** (THT). The key for each entry is the hash value of each encrypted keyword token w, and the value of each entry is a tuple $(enc(w, n_w), w)$. The reason we need to store the value of w itself is for breaking ties in case hash table collisions happen. Moreover, THT can be built based on the token table which the MB maintains to count the number of appearances of each keyword token. Once a match is established, the MB can remove the corresponding key and value pair from THT, re-encrypt the ciphertext and insert it as a new entry.

Up till now, we have discussed how to efficiently search for an encrypted token in a rule set with hundreds of keyword tokens at the MB. The questions left to answer are

How to efficiently combine multiple matched keyword tokens to one keyword since keywords are also tokenized by the MB? How to map multiple matched keywords to a specific rule as one rule may contains multiple keywords?

We attempt to kill two birds with one stone. The insight here is that *once the MB finds a matched keyword token, it can reduce the search range for the next token, which is either* *the first token of the next keyword in the same rule or the next keyword token together with the matched keyword token constitute a part of one keyword*. For example, if there are only two keywords in the rule set with first five bytes as ”NETWO”, i.e.,”NETWORK” and ”NETWORR”, and there is a matching token ”NETWO” in the traffic, the MB only needs to find out if the third encrypted token after ”NETWO” is ”TWORK” or ”TWORR”. Therefore we can build a **Hierarchical Hash Table** (HHT) as shown in Fig. 3 to keep the information about how each rule is tokenized. The first level of the HHT has only one hash table, each entry of which corresponds to the first five bytes of the first keyword in each rule. Each entry of the first level hash table also points to a new hash table which contains all possible following keyword tokens, and so on. The position information of each keyword token can be easily embedded in HHT as well. Since S tokenizes the traffic using a sliding window, the offset of each received encrypted token over the SPA connection can be easily deduced.

The HHT indicates how each rule is structured with keyword tokens and can be pre-built or even hardcoded in the MB once it receives the rule set from the RG, while the THT needs to be built specifically for each connection. Observe that the HHT could be further compressed by combining hash tables with same entries, and we leave it as our future work. The memory overhead and the time to setup THT will be evaluated in Section VI.

3) On The Receiver Side: In order to prevent S from being malicious and sending illegal tokens, R upon receiving traffic from the SSL connection and traffic forwarded by the MB over the SPA connection can first decrypt the SSL-encrypted traffic, and then tokenize and encrypt the plaintext-form SSL traffic. R can then compare the ciphertexts generated from the SSL traffic against the encrypted tokens from the SPA connection; R also needs to check if the AUXInfo sent by S is valid by comparing it with its own copy. Only if both match, the traffic received over the SSL connection is considered secure. Decryption of the SSL-encrypted traffic can be done using the standard method specified in SSL. The reason we proceed in this manner rather than decrypting and then detokenizing the traffic from the SPA connection for comparison is that tokenizing and encrypting plaintext with Eq. 2 together are much faster than decrypting the traffic from the SPA connection. Moreover, R can start this comparison process as long as it gets the SSL traffic and the first encrypted token over the SPA connection. Specifically, R can first decrypt, tokenize and then encrypt the SSL traffic. Whenever R receives an encrypted token over the SPA connection, it can compare it right away. Nevertheless, this process may cause some delay at R side but we consider it as an acceptable trade-off.

C. Regular Expression Evaluation

The reason for evaluating regular expressions is twofold. First, some rule sets require evaluation of regular expressions besides keyword matching. By enabling such operations, all of the attack rules of many public and industrial rule sets can be addressed [43]. For instance, in our rule set from Snort Emerging Threats [11], some of the rules are allowed to be written using Perl compatible regular expressions. Take rule #74 in emerging-info.rules as an example:

```
alert http $HOME_NET any→$EXTERNAL_NET any
flow: established, to_server;
content: ”8866.org”;
pcre: ”/Host\x3A[^\r\n]*\x2E8866.org/Hi”
```

In this example, regular expression evaluation denoted by action ”pcre” would be triggered if a match of the keyword ”8866.org” can be found in the traffic. Secondly, regular expression can help the MB detect keyword that are less than 5 bytes long. Recall that S tokenizes the traffic using a 5-byte long sliding window so that the MB can search for keywords whose length are equal to or greater than 5. However, if one rule contains keywords that are shorter that 5 bytes, methods mentioned in Section IV-B would not work. In our rule set, more than 25% of the keywords are shorter than 5 bytes long. To enable matching on keywords that are shorter than 5 bytes long, the MB can build one regular expression out of the keywords that are shorter than 5 bytes long per rule easily to make the keyword searching problem a regular expression evaluation problem. Therefore, in our case, matching regular expression enables full keyword matching functionality.

Next, we try to answer *How to efficiently evaluate regular expressions over encrypted traffic?* One strawman solution is to embed the SSL key k_{ssl} in the encrypted tokens, and once there is a match between an encrypted token sent over the SPA connection and one keyword token at the MB, the MB can extract k_{ssl} by re-encrypting the matched token, say $enc(t, n_t)$ with $n_t{+}1$, and XORing it with $enc(t, n_t{+}1) \oplus k_{ssl}$ which is sent as a pair together with $enc(t, n_t)$ over the SPA connection by S. However, three problems may arise in this solution: *a*) The traffic can get doubled; *b*) Matching of a partial keyword token would would reveal the SSL key to the MB; *c*) The MB is granted with too much power for some users. Especially, the second one breaks the security requirement. For instance, if the only keyword at the MB is ”NETWORK” and the traffic contains ”NETWORR”, which doesn’t match the keyword, the SSL key would still be exposed to the MB since the token ”NETWO” matches. Therefore, we need a protocol which can perform regular expression evaluation and preserve user data privacy at the same time.

Normally, a regular expression can be converted to a deterministic finite automata (DFA) with an accepting state F indicating that the input string contains malicious factors if this DFA finishes in state F. A DFA will take as input the encrypted traffic over the SPA connection one byte at a time [30], [19]. One way to execute the regular expression in a privacy-preserving manner is encrypting each single character using Eq. 2. However, this will render the resulting protocol vulnerable to brute-force attack. Rather than revealing SSL key to the MB such that the MB can decrypt the traffic to run regular expressions, we propose to send the regular expression to R and let R decrypt the SSL encrypted traffic and run the regular expression matching algorithm, which just causes small overhead (Section VI). If the corresponding DFA can achieve an accepting state, R would know that S is malicious (a hit of an attack rule) and can drop the packet. If the RGs don't want to publicize their rule sets for any practical reasons, *Yao's garbled circuits* [49] and *1-out-of-2 oblivious transfer* (OT) [26] can be used to hide the regular expression from end users.

To further improve the performance of our protocol, instead of using Yao's garbled circuit technique which is designed for general circuits, we apply the efficient garbling technique that is customized for DFAs from [37]. Note that garbling of the DFA's can be done offline before the connection setup since the garbling process does not require information from other parties but the MB itself. Therefore, it would not affect the runtime performance of the MB.

It is worth pointing out that the MB must send a "newly" garbled DFA for each match, even if the same regular expression has been previously matched. This ensures that R doesn't learn that the same rule is matched repeatedly. Note that this doesn't mean that the DFA needs to be garbled on the fly for each match. Instead, the MB can pre-compute and store them for future usage.

One practical assumption in our protocol is that the RG will provide the MB with various DFAs and the MB will send the garbled DFA to R without letting R know which DFA is being executed. With this assumption, in our protocol, R would not have enough information to figure out the structure of any specific DFA, because R is not able to learn what garbled DFA it has computed and whether the garbled DFAs it has computed so far are correlated or not.

Recall that a garbled DFA will be sent to R when the traffic being transmitted has matches that can trigger a regular expression evaluation. One might be concerned that an attacker can cause the MB to send out many garbled DFAs to R just by sending traffic that matches a token that triggers a regular expression evaluation, which would turn into DoS on R's network link. Note that for this kind of attack to be achieved, the attacker needs to have the knowledge about what keywords can trigger regular expression evaluations. However, in this paper, we assume that the information of keywords is kept secret from users and it is only known to the (trustworthy) rule generator. In the meantime, since the underlying encryption on keywords can provide semantic security, the attack can learn nothing about the keywords themselves. Hence, this kind of attack is eliminated from the scenario being considered in the paper.

Another concern might be that since R runs the garbled DFA with somewhat plaintext payload (i.e, the payload is not encrypted but replaced with a random string input to the garbled DFA), R will at least learn the payload that apparently matched a regular-expression based rule. This can be prevented by adding dummy operations on the DFA. As a result, it will confuse R when running the garbled DFA with the somewhat plaintext payload.

D. Malware Detection via Machine Learning

In this section, we show how our protocol enables malware detection over encrypted traffic using ML analysis. Recall that for keyword matching, S needs to tokenize the traffic using a sliding window, which is also called *n-gram* or *shingles*. Over the past few decades, researchers have proposed to use *n-grams* to represent features for malware detection using ML methods [45], [34]. In our protocol, we will use *Support Vector Machine* (SVM) [41].

1) On The Rule Generator Side: The primary job of the RG is to train a ML model (SVM model). Assume that the RG has two sets of files, which are composed of a collection of malware software and benign programs, respectively. In particular, malware programs consists of variety of forms of hostile or intrusive software, such as Worms, Trojans Horses and Virus [16]. In order to build representation for all files in each set, the RG should extract n-grams (features) for all files in each set which will act as the set signature. To reduce the feature dimension in our protocol, we require the RG to count the frequency of each n-gram within each set, and use top K most frequent n-grams as the features representing that set. Then the RG can train a SVM model and obtain the necessary parameters $\boldsymbol{w}$ and b for the following decision function

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \cdot \boldsymbol{x} + b \tag{3}$$

where $\boldsymbol{x}$ is the feature vector and $\boldsymbol{w}$ is a m-dimensional vector $(w_1, w_2, \cdots, w_K)$. Note that the pair $(\boldsymbol{w}, b)$ can be updated periodically by the RGs as they receive new malwares reports everyday [15]. We also discuss how SPABox can support other ML model in Section VIII-B.

2) On The Sender Side: In order to perform ML at the MB, the object's features need to be extracted first. However, it is not trivial for the MB to extract one input object's features since it is highly possible that not all important features are malicious keywords while the MB can only know the patterns of those keywords, guaranteed by our keyword matching protocol. Hence, in our protocol, the job of selecting features of an input object is shifted to S who could do this at almost no cost by using the counter table which stores the frequencies of all tokens that have appeared so far. Specifically, S first uses the feature selection module shown in Fig. 2 to select as features, say the top 400 most frequent n-grams, i.e., $K = 400$. Then S encrypts these n-grams, that is, for $i = 1, \cdots, K$, S encrypts x_i as $x_i' = g^{s \cdot x_i} \mod N^2$ where $N = pq$ with p, q as two prime numbers of equal bit-length and s is a random element in $\mathbb{Z}_N$. After that, S sends the encrypted features over the SPA connection together with g^s and N^2 so that the MB can perform classification. To ensure desirable security, we pick p, q of length at least 1024-bit. Nevertheless, the overhead incurred by using 1024-bit long p and q is quite small in practice. Note that one can implement their own feature selection module as long as the RGs can provide a corresponding trained model.

3) On The Middlebox Side: The most notable feature of SVM is kernel function, which serves as a mapping of data to improve its resemblance to a linearly separable set. In our work, we use simple linear kernel [32], and we will discuss other feasible kernel functions in Section VIII-B. Recall that each encrypted n-gram is of the form as Eq. 2.

Upon receiving encrypted n-grams (features) from S, the MB performs the following steps using a well-trained SVM model: (i) for i-th input to the decision function (with $i = 1, \cdots, 400$), raise the encrypted n-gram x'_i to the power of value w_i, that is, ${x'_i}^{w_i} = g^{s\cdot x_i \cdot w_i}$; (ii) multiply together all the intermediate results generated in the former step; (iii) multiply the result generated in step (ii) by g^{sb} and then by $g^{s\frac{N^2}{2}}$ (this multiplication enables R to learn the classification sign) to give the classification result in an encrypted form, namely $g^{s(\sum_{i=1}^{K} w_i \cdot x_i + b + \frac{N^2}{2})}$. Finally, the MB can send the result returned in step (iii) to R. The correctness of the encrypted classification result is guaranteed by the homomorphic properties stated in Section IV-A.

4) On The Receiver Side: In order to get the classification result, all R needs to do is to decrypt the classification result sent by the MB using (λ, μ) (private keys in Paillier cryptosystem [40]) and then check whether it is greater or less than 0. Unfortunately, the latter step can't be done directly, since in our encryption method, the tokens (in mpz_t format) are non-negative. However we observe that the classification result output by Eq. 3 is within the interval $[-\frac{N^2}{2}, \frac{N^2}{2}]$. Therefore, in order to recover the sign of the classification result, R can decrypt the classification result and then compare it with $\frac{N^2}{2}$. If the recovered value is greater than $\frac{N^2}{2}$, then R knows that it is a positive result; otherwise, it is a negative result. The correctness of this operation is straightforward, because if $f(\boldsymbol{x}) = \boldsymbol{w}^\top \cdot \boldsymbol{x} + b > 0$, then $f(\boldsymbol{x}) + \frac{N^2}{2} > \frac{N^2}{2}$.

E. Security Guarantee

Keyword Matching. Recall that in keyword matching operation, S needs to tokenize the traffic, perform BPP and then encrypt the tokens using Eq. 2. While on the MB side, it checks if there is any encrypted keyword token in the rule set that is equal to the received encrypted tokens over the SPA connection.

In Section IV-B, we've mentioned that BPP is designed to avoid the case where the first few encrypted tokens are malicious keyword tokens. Without BPP step, the resulting protocol becomes insecure. To show that, assume that the generated tokens are ”NETWO”, ”ETWOR” and ”TWORK”, denoted as m_1, m_2 and m_3 respectively, and the first token m_1 is a malicious keyword token while the other two are benign. Thus, the MB can learn what the first encrypted token is. As the MB also knows $salt_0$, d and the counter k for each distinct token starts from 1, the MB can learn m_2 by picking message $m'_2 = ETWO*$, encrypting it and comparing the resulting ciphertext with the received encrypted token $g^{salt_1 \cdot m_2}$, where $*$ is one byte and can have up to 256 possibilities. m_2 cannot be a malicious keyword token; otherwise the MB would find a match against the rule set. Apparently, by doing this repeatedly, the MB can definitely recover m_2 within constant time (i.e., 256 comparisons in the worst case), which renders the protocol insecure. Similarly, the MB can learn all the following tokens. However, if S applies BPP to the tokenized traffic before transmitting them over the SPA connection, this undesirable scenario can be avoided because r is a random number which the MB cannot learn. As the MB cannot learn the encrypted form of these r benign tokens, the MB is not able to work out what those benign tokens are and thus can't discover the patterns of the encrypted tokens as it does, if S transmits without the BPP step.

To find the suitable value of r, we again consider the above situation where the first token is a malicious keyword token in the rule set.After applying BPP, the first r tokens are benign and the $(r+1)$-th token is a malicious token. Note that the goal of applying BPP is to disturb the original patterns of encrypted tokens. In other words, BPP is expected to bring in benign tokens that will also appear in the traffic. We denote this event by C and the probability that C happens is denoted by $\Pr[C]$. Assume that each token picked by the BPP process will appear in the original encrypted token sequence over the SPA connection with probability p. Then $\Pr[C] = 1-(1-p)^r$. Note that the value r is unknown to the MB. Hence, it is not possible (at least not feasible) for the MB to figure out what r is. Thus, an average case r would suffice for actual applications.

The security goal of our proposed protocol is to provide indistinguishability for unmatched traffic. At a high level, given encrypted tokens that are not equal to any keyword in the rule set so far, no polynomial-time adversary can infer any information of those tokens. However, when the given encrypted tokens are keywords, the adversary can know that they are in the keyword set, while not learning anything else about the underlying keywords.

To evaluate the security feature of our encryption scheme, first we give the definition of the *Decision Diffie-Hellman Problem (DDH)*.

Definition 2. ***Decision Diffie-Hellman Problem (DDH).*** *A group family $\mathbb{G}$ satisfies the DDH assumption if there is no DDH algorithm $\mathcal{A}$ for $\mathbb{G}$, such that for some $\alpha > 0$ and sufficiently large n:*

$$\begin{aligned}|\Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^{ab}) = true]\\ - \Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^c) = true]| > \frac{1}{n^\alpha}\end{aligned}$$

Then we can define the indistinguishability of our proposed protocol via the following security game.

Definition 3. ***Security Game.*** *Consider the protocol with algorithm (Setup, Enc) and associated message space $\mathcal{M}$. Let $\mathcal{A}$ be a p.p.t. adversary. The security game $\mathsf{Exp}_{\mathcal{A}}(1^\lambda)$ is defined as follows:*

1) $salt_0, d \leftarrow \mathsf{Setup}(1^\lambda)$
2) $T^0 = (t_1^0, \cdots, t_n^0), T^1 = (t_1^1, \cdots, t_n^1) \leftarrow \mathcal{A}(1^\lambda)$
3) $b \xleftarrow{\$} \{0, 1\}$
4) $c_1, \cdots, c_n \leftarrow \mathsf{Enc}(salt_0, d, t_1^b, \cdots, t_n^b)$
5) $b' \leftarrow \mathcal{A}(c_1, \cdots, c_n,)$
6) If $b' = b$, output 1.

We say that the protocol is secure if for all p.p.t. adversaries $\mathcal{A}'s$, and for all sufficiently large λ:

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

Based on the above definitions, the proposed protocol achieves above-mentioned security goal as stated in Lemma 1 below.

Lemma 1. *Assuming the intractability of the DDH problem, then our proposed keyword matching protocol in Section IV-B is secure for those tokens that are not in the rule set.*

Proof. If there is an attacker $\mathcal{A}$ at the MB who is able to learn the private value of any incoming encrypted token from S with non-negligible probability, then we can use this attacker $\mathcal{A}$ as a black-box to construct an algorithm that can solve DDH efficiently.

More specifically, given an instance of DDH (g, g^a, g^b, z), we want to build an algorithm $\mathcal{C}$ that uses $\mathcal{A}$'s ability to solve for such b. To do that, we can construct an algorithm $\mathcal{C}$ that will act as a challenger in the security game played by $\mathcal{A}$, which is proceeding as follows:

Setup Phase. $\mathcal{C}$ defines $salt_0, d$ as in Section IV-B, and computes $h = g^{salt_0} = g^a \mod n^2$ and $k = g^d = g^b \mod n$. Then h, k will be sent to $\mathcal{A}$ as the public parameter for the protocol.

Challenge Phase. $\mathcal{A}$ submits $T^0 = (t_1^0, \cdots, t_n^0)$ and $T^1 = (t_1^1, \cdots, t_n^1)$. $\mathcal{C}$ flips a coin b from $\{0, 1\}$ and then encrypts T^b set of tokens via computing $c_i = (g^a)^{t_i^{b)}} \cdot (g^{ab})^{t_i^b}$, and then forwards this set of $\{c_1, \cdots, c_n\}$ to $\mathcal{A}$.

Output Phase. The adversary $\mathcal{A}$ outputs b'. If $b' = b$, $\mathcal{A}$ wins.

We can see that the algorithm $\mathcal{C}$ is constructed as a challenger to $\mathcal{A}$. Now we can check from below that c_i is of the correct form as a ciphertext encrypting the plaintext t_i^b in our proposed protocol.

$$(g^a)^{t_i^{b)}} \cdot (g^{ab})^{t_i^b} = (g^{salt_0})^{t_i^{b)}} \cdot (g^{ad})^{t_i^b} = g^{salt_a \cdot t_i^b}$$

where $salt_a = salt_0 + ad$. This is a valid ciphertext since $\mathcal{A}$ doesn't have the knowledge of g^{dt} for unmatched token t and thus can't decide $gkdt$ where $k \in \mathbb{Z}$. Therefore, from the perspective of the attack $\mathcal{A}$, he is interacting with his challenger in a security game for the proposed protocol.

Finally, $\mathcal{C}$ determines that in the instance (g, g^a, g^b, z), $z = g^{ab}$ if $\mathcal{A}$ outputs $b' = b$; otherwise, decides $z \neq g^{ab}$.

Note that $\mathcal{A}$ has the ability to break the proposed protocol. Hence, if it is indeed $z = g^{ab}$, then all ciphertexts in $\{c_1, \cdots, c_n\}$ are of the correct form, and thus $\mathcal{A}$ decides whether they are ciphertexts of T^0 or T^1. If $z \neq g^{ab}$, then all ciphertexts will become random values, and thus $\mathcal{A}$ will be likely to fail with probablity $1/2$. Thus we have the advantage of $\mathcal{A}$ as

$$\mathrm{Adv}_{\mathcal{A}} = \Pr[\mathcal{A}(1^\lambda)\mathrm{wins}] - \frac{1}{2}$$

As a result, $\mathcal{C}$ also has advantage equal to $\mathrm{Adv}_{\mathcal{A}}$. However, there is no known algorithm that can efficiently distinguish DDH problem, which indicates that $\mathcal{C}$ has negligible probability of distinguishing DDH problems. This means that $\mathrm{Adv}_{\mathcal{A}}$ is also negligible, that is, $\mathrm{Adv}_{\mathcal{A}} \leq \mathrm{negl}(\lambda)$. Therefore, there is no such adversary $\mathcal{A}$ in the MB that can break the indistinguishability of unmatched tokens in our proposed protocol. □

On the other hand, the security provided by our protocol also depends on the security of the used hash function. Note that a secure hash function requires that it is infeasible to recover a message input from its hash value. Now we see that our protocol meets the security goal as long as the parameters for the hash function and the Discrete Logarithm Problem are appropriately set.

Regular Expression Evaluation. Recall that in our protocol for regular expression evaluation, each incoming encrypted token over the SPA connection is of the form $c = g^{salt_k \cdot m} \mod n$. As we have proved **Lemma 1** and each token is 5 bytes long, the plaintext space for m is about $\Theta(256^5) = \Theta(2^{40})$. Apparently, this can't be dealt with by brute-force method within polynomial time. This indeed prevents our protocol from being vulnerable to brute-force attack, but it causes the DFA for performing regular expression evaluation to be of a tremendous size, because the transition options (i.e., the size of DFA's alphabet) from state to state is linear to the number of possible cases for one token (about $\Theta(2^{40})$). Apparently, this would not be acceptable in real life applications. Therefore, our protocol employs the techniques of Yao's garbled circuits [49] (or a similar garbled technique as shown in Section IV-C) to keep the DFA size small while providing desired security. Then 1-out-2 oblivious transfer [26] comes in to help R know the random strings corresponding to the inputs to the circuit without letting the MB learn what strings R gets. With the combination of these two techniques, our protocol can guarantee that the MB can't learn the benign tokens and R knows nothing about the underlying structure of the regular expression. Formally, we can summarize our security guarantee as:

Lemma 2. *The protocol is fully-secure against a malicious receiver and is private against a honest-but-curious middlebox.*

Proof. As our protocol is built upon the techniques in [37], [26], one can refer to them for a detailed proof. □

Malware Detection via Machine Learning. Recall that this procedure heavily depends on the homomorphic properties as stated in Section IV-A. All the major computations of the machine learning process are performed within the MB and only the encrypted classification result will be forwarded to R. Thus, the desired security guarantees for this operation are

1) R doesn't know the parameters of the trained machine learning model.
2) The MB can't learn the classification result.

Unlike the regular expression operation, no major computation is done by R except decrypting the classification result. It is easy to see that our proposed protocol achieves the security goal (i), because R only has the knowledge of the classification result and the input features, which are not enough to solve for all the unknown parameter variables using one equation. As for the security goal (ii), it is a result stemming from both the two homomorphic properties in Section IV-A and the above security analysis of the keyword matching operation. One can see that the encryption is also similar to Paillier cryptosystem, and can refer to [40] for a detailed proof. Specifically, on one hand, the two homomophirc properties provided by our encryption method ensure that the resulting ciphertext from exponentiation and multiplication is of the correct encrypted form, i.e., $g^{salt_k \cdot m'}$ with m' as the resulting plaintext; on the other hand, we have proved above that no attackers in the MB can learn any private value that is not a keyword from a well-formed ciphertext. Therefore, with these two guarantees, no attackers in the MB is able to derive the result produced by the machine learning process, which is our security goal (ii). Furthermore, as s (salt) used in encrypting the n-grams is different from the salt used for encrypting tokens (Eq. 2), the MB cannot correlate an n-gram with any keyword token even if their plaintexts are the same.

V. System Implementation

In this section, we show a detailed implementation of SPABox for the RG, clients and the MB.

On The Rule Generator Side. LIBSVM 3.21 library [25] is modified and used for training a SVM model.

On The Client Side. We implement SPABox for clients in C on top of OpenSSL-1.0.2d library [7]. We also modify SSL handshake process in the OpenSSL library such that we can extract AUXInfo. GMP 6.0.0 library is used to convert each token, which we choose to be 5 bytes long, into a large integer (mpz_t format) and then hash and encrypt each token as described in Eq. 2 using the corresponding large integer. We choose g, $salt_0$, s, d, n and N^2 to be 80, 20, 20, 10, 160 and 4096 bits, respectively. The hash function we use is SHA-1. Based on our security analysis, r is set to be between 20 and 40. When S opens a connection, it creates two sockets, one for SPABox handshake, sending normal HTTPS traffic and the other one for the encrypted token transmission. Features for ML are sent after the tokenized traffic. The implementation of R is similar to that of S with additional implementation of garbled circuit [37] and OT [26]. If R successfully matches the traffic with the regular expression sent by the MB or gets a positive result for malware, it stops the connection.

On The Middlebox Side. We implement the MB in Click modular router [33] with DPDK [1]. We build both THT and HHT based on Google dense hash map [2]. We let THT start with 65,536 slots and resize when it's more than 50% full. For *keyword matching*, half of all the threads are used for matching encrypted tokens in the THT, and one thread is used to search over the HHT if there are keyword tokens matched. If a rule is matched and no further *regular expression* evaluation is needed, the MB would block the connection and notify R; if regular expression needs to be run on the receiver side, [26] is used for oblivious transfer of the input key strings corresponding to a garbled DFA. The rest of the threads are used for *malware detection* which is implemented based on GMP 6.0.0 library. If the connection is not blocked, the ML classification result is sent together with other traffic to R. The MB and the clients are connected with a 1Gbps LAN on campus.

VI. Performance Evaluation

To demonstrate that our proposed protocol is practical, in this section, we show the performance evaluation of SPABox on both clients and the MB sides. SPABox requires (i) extra tokeninzation and encryption on client side; (ii) building and searching over multiple hash tables, garbling and OT for regular expression evaluation and performing classification on the MB side. The statistics shown in this section are the average results.

	Blindbox	SPABox	Pailliar
Encrypt (5 bytes)	101*ns* w/ HW support 1022*ns* w/o HW support	1015*ns*	20.6*ms*

TABLE II: Encryption micro-benchmarks at S comparing SPABox and Blindbox

A. Data Sets

Our keyword rule set from the Snort Emerging Threats [11] has around 3K rules, which contains 11,202 keywords and 21,035 distinct keyword tokens.

For malware detection, we use two different data sets: a malware data set and a benign data set. Our malware corpus contains 17258 malicious program from VX Heaven [14] and Microsoft Malware classification Challenge [5], including different malware families that represent different types of malware. Our benign data set contains 1000 legitimate executables and Dynamic Linked Librarys (DLLs), most of which are system files gathered from the machines running on our campus. We use 70% of the files as the training set and the rest 30% as the testing set.

We run SPABox client and the MB over synthetic traffic which contains data sets used for malware detection, payload extracted from ICTF2010 network trace [3] and 36 hour-long unencrypted real world traces collected at the access link going in and out of our campus.

B. Performance

We now investigate the performance of SPABox at clients, the MB and the network. To the best of our knowledge, Blindbox [43] is the only system that enables part of the DPI functions considered in our paper over encrypted traffic, namely keyword matching and regular expression. We implement the encryption method, and keyword matching algorithm used in Blindbox and compare them with SPABox. All statistics are average results.

1) On The Client Side: Two desktops equipped with Intel Core i7 processors and 16GB memory are used to run our client prototypes. The machines are multicore, but we only use one thread per client (except for throughput test). Hyper-threading is enabled. The CPU supports AES-NI instructions so that we can compare our solution with the Blindbox.

How long does it take for S to encrypt a token? Table II shows the micro-benchmarks for encryption using SPABox and Blindbox. In SPABox, encryption of one 5-byte block takes 1015*ns* on average. This timing result includes the time for converting a token into a large integer, hashing with SHA-1 and encryption. Compared with Blindbox, our solution takes 9× more time. The main reason is that Blindbox takes advantage of hardware support for AES encryption. Without hardware support, Blindbox takes similar amount of time as SPABox. Compared to Paillier, our encryption method saves almost 20× time.

How long does it take for R to evaluate regular expression? How much overhead does OT incur? In SPABox, R needs to perform regular expression evaluation in case the MB finds the traffic suspicious (by keyword matching). This process

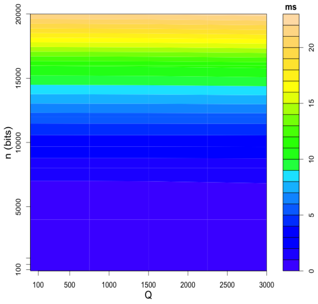

Fig. 4: Regular expression evaluation time

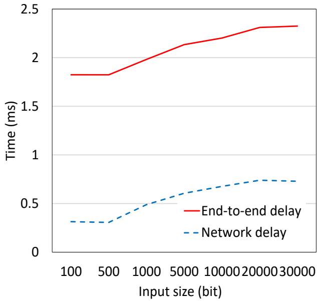

Fig. 5: End-to-end delay due to OT

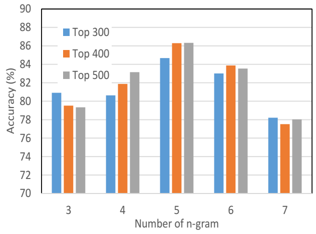

Fig. 6: Malware detection accuracy

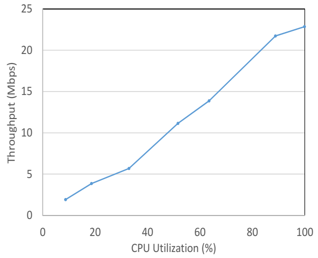

Fig. 7: Client throughput vs. CPU utilization

		Blindbox	SPABox
Keyword matching	1 Token, 1 Rule	27ns	103ns
	1 Token, 3K Rules	162ns	114ns
	1 Keyword, 3K Rules	708ns	499ns
Malware detection	Top 400 5-grams	N/A	239μs

TABLE III: Keyword matching and machine learning micro-benchmarks at the MB comparing SPABox and Blindbox

adds more computation overhead on the receiver side. Fig. 4 shows the regular expression evaluation time at R given different input string lengths (x-axis) and DFA sizes (y-axis). It shows that the evaluation time increases linearly as the input size grows, but slowly as the number of states increases. This is because for each bit, only one hash calculation is required, which dominates the evaluation time. The time for processing regular expression can be further decreased by optimizing the underlying data structure (i.e., DFA) [38], [17], but it is beyond the scope of this paper.

Another overhead when evaluating regular expression comes from OT. To meet the performance requirement, we use the OT implementation in [26]. This protocol bootstraps off Diffie-Hellman key-exchange protocol [28], and therefore is very efficient. We evaluate the total end-to-end delay when a regular expression is needed with the real world traces as shown in Fig. 5, including both the network delay and the delay caused by OT itself at the MB and R. It is clearly that the delay caused by OT grows slowly as the input size increases, and it will be within the same order of magnitude as the network delay, which means that the overhead caused by OT itself is quite limited.

How accurate is the malware detection? To answer this question, we first should find out what the best n value is for n-grams. Besides, we also need to determine how many features to select from each file while balancing performance and overhead (i.e., computation and bandwidth overhead). Fig. 6 shows the malware detection accuracy using SVM. Accuracy is defined as true positive rate, which measures proportion of positives that are correctly identified as a malware program. Although, using top 500 most frequent 6-grams gives the most accurate classification result, we choose $K = 400$ because it leads to smaller communication overhead while still achieving relatively high accuracy. Therefore, we use 5 bytes as the sliding window for tokenization (see Section IV-B for the other two reasons) and select top 400 n-grams as features for malware detection in our implementation. The detection accuracy cannot be easily improved due to the limitation of the kernel function we use. How to enable other kernel functions and classification models are discussed in Section VIII-B.

What is the throughput of a client? As our proposed encryption method takes longer than the one in Blindbox, we expect that SPABox gives a lower throughput. Fig. 7 shows the costs for encryption at the sender side with two cores and hyper-threading disabled. These numbers include the time for tokenization, token selection (for malware detection), hashing with SHA-1, encryption and transmission process. At 5Mbps, the encryption cost is very limited as CPU can continuously generate data; but when the throughput increases to 23Mbps, CPU at the sender side can no longer keep up. This overhead can be mitigated in two ways: (i) tokenization, hashing and encryption can be further parallelized with extra cores easily; (ii) we can also use a delimiter-based tokenization instead of window-based tokenization [43] to reduce the number of generated tokens. Nevertheless, the throughput achieved by our solution is enough for a typical broadband home uplink and applications dealing with small files. R incurs similar cost to S, as R must check that the encrypted tokens sent by S match the plaintext recovered from SSL.

2) On The Middlebox Side: Our MB prototype is implemented on a server with two 2.0GHz Xeon E5335 cores and 16GB RAM. The CPU doesn't have AES instruction support.

How long does it take for the MB to match keywords? How about time taken to run machine learning? To show the effectiveness of our method, we compare the keyword matching performance of SPABox and Blindbox. Due to the lack of AES hardware support, we implement the MB on a desktop with Intel Core i7 and 8GB memory for comparison with Blindbox. Table III shows the detection micro-benchmarks comparing Blindbox and SPABox with hpyer-threading enabled. For the first case where the MB needs to compare the received encrypted token over the SPA connection with one keyword token, Blindbox can finish the comparison in 27 ns while SPABox increases the value to 103 ns. Most of the overhead comes from hashing the encrypted token, since we first need to convert each token of mpz_t format into a string and then hash it. However, for 3K rules (all keywords are tokenized), SPABox only takes 114ns to match a token while Blindbox would use 162ns on average. This is because our implementation uses a hash table rather than a searching tree for keyword searching. To match one keyword which has

4 tokens on average, SPABox can save 29.5% time compared to Blindbox. To perform classification, it only takes $239\mu s$ at the MB. Apparently, the incurred overhead is quite small.

What is the memory overhead for storing THT at the middblebox? To keep up the searching speed, we use hash tables instead of searching tree in our algorithm for keyword matching. Each keyword token is hashed and stored in the hash table such that the MB can quickly find out if one encrypted token over the SPA connection is present in the rule list. One possible defect of our approach is more memory usage. Specifically, if using a searching tree, the total memory consumption is $\sim$0.822MB, while the hash table we use has a factor of 2-3 memory overhead. In our prototype, we measure a memory usage of $\sim$3MB. Even if using hash table increases the memory usage, we consider it an acceptable trade-off. Moreover, with the emergence of network function virtulization (NFV) technology [31], middleboxes could be "virtualized" and implemented on standard server which has "unlimited" memory. Therefore, the memory overhead incurred by THT is negligible.

What throughput can the MB sustain? As our client prototype cannot generate encrypted tokens fast enough (up to 23Mbps with two cores), we pre-encrypt the data and use it as traffic. In our experiment, we measure an average throughput of 69Mbps at the MB. This number includes all three DPI functionalities supported by SPABox. The major overhead at the MB comes from hashing all encrypted token received from S, while searching over the THT and performing classification take limited time. If a dedicated core is used for hashing and searching, the throughput can be increased to 81Mbps. As hashing all encrypted tokens can be paralleled easily, the throughput can be further boosted.

3) Network Overheads:

What overhead does connection setup incur? One main advantage of SPABox is that it doesn't require interaction between clients and the MB during the connection setup stage. Assume that the rule set contains 3K keywords, and that the throughput of the link between clients and the MB is 20Mbps (typical home broadband link). SPABox can finish the setup within 40μs on client side (see Table IV), while Blindbox requires lengthy interaction between clients and the MB. Specifically, in Blindbox, the garbled circuit for each keyword token is 599KB. To compute all the garbled circuits and transmit them, it would take more than 90*min* and impose huge computation burden on clients. Phones or tablets would take a even longer time. After receiving all the garbled circuits, the MB needs 7.5*s* to evaluate them all. On the contrary, in SPABox, the MB needs 228 byte information from client and the MB can set up THT within 17*ms* after receiving the information. Compared with Blindbox, SPABox has a better support for short-lived connection and mobile users.

What is the overall bandwidth overhead? Throughout the experiment, we see a bandwidth overhead a little less than 20 times. The reason is twofold: *a*) We use window-based tokenization; *b*) only very small overhead is incurred due to regular expression and malware detection. Although there is substantial bandwidth overhead, we can find that the major source of overhead is tokenization. Therefore, it can be mitigated easily: instead of using window-based tokenization, we can switch to delimiter-based tokenization and only tokenize non-binary data as in Blindbox. However, delimiter-based tokenization may fail to detect attack keywords that do not start and end before or after a delimiter. One can choose whichever method that is best for a specific application.

	Blindbox		**SPABox**	
	Client	Middlebox	Clinet	Middlebox
Time	4670*s*	7.5*s*	< 40μs	17*ms*
Bandwidth	11.4*GB*		228 *byte*	

TABLE IV: Connection setup overhead micro-benchmarks comparing SPABox and Blindbox

VII. Related Work

Searchable Encryption. To perform keyword matching over encrypted data in DPI scenario, one naturally considers searchable encryption. There have been many work on searchable encryption [18], [20], [44] since its introduction. However, applying these existing work to DPI for keyword matching requires the entity who generates search tokens to encrypt the rules, which will probably expose sensitive rule set information to end users. Moreover, none of the existing schemes meet both security and network performance requirements simultaneously. Specifically, on one hand, deterministic searchable encryption schemes [18] leak pattern information, that is, whether two words (match a rule or not) in the encrypted traffic are the same, even though these schemes enable the MB to build indexes to process each token faster. This weak privacy guarantee allows an attacker to perform frequency analysis. On the other hand, randomized schemes [44] provide stronger security guarantees because of the existence of random salts in their generated ciphertexts. However, the usage of random salt prevents the MB from building index structure for fast token matching, which results in a low throughput at the MB.

Regular Expression Evaluation. Recently, there have been some work [42], [48] on developing protocols to enable regular expression searches over encrypted data. Generally, these interactive protocols allow two party to privately evaluate DFA over encrypted files. A non-interactive case is a functional encryption scheme proposed by Waters [47]. This scheme ties a secret key to a specific DFA so that it can be used to decrypt a ciphertext only if the DFA accepts a fixed string associated with the ciphertext. However, these schemes can only support regular expression, but not other DPI functionalities discussed in our paper. Moreover, they are not able to meet the network performance requirement, especially the functional encryption scheme.

Machine Learning. Some work [21], [22] focused on performing ML over encrypted data. Bos et al.'s work [21] showed how the computation of medical prediction functions over the encrypted medical data can be performed by a third party using fully homomorphic encryption. Bost et al.

[22] constructed three major privacy-preserving classifiers, including hyperplane decision, Naïve Bayes, and decision trees. However, their schemes relied on fully homomorphic encryption (FHE), which results in significant overheads due to the fact that existing FHE constructions are still not practical. As a consequence, they are not able to meet the network performance requirement. Moreover, the protocols themselves in [22] are pretty complicated and require multiple rounds of interactions between S and the MB.

VIII. Discussion

A. Protocol using Paillier Cryptosystem

The encrypted token in our protocol is somewhat similar to that in Paillier encryption [40]. Here we point out that why it is not a good idea to use Paillier cryptosystem directly.

Recall that in Paillier cryptosystem, the ciphertext is of the form $c = g^m \cdot r^n \mod n^2$ with m as the message to encrypt. However, directly using Paillier encryption in our application scenario can't provide the MB with the ability for keyword matching. Precisely, with only c and no $r^n \mod n^2$, the MB cannot perform keyword matching operations over encrypted tokens. Unfortunately, if the MB can possess c and $r^n \mod n^2$ at the same time which makes the matching operation possible, the resulting protocol becomes insecure since the MB can derive g^m via dividing c by $r^n \mod n^2$. Hence, to have a secure protocol based on Paillier encryption (for exposition, we call this Paillier-based protocol Π_P), a straightforward way would be to introduce a new random element $salt$ in $\mathbb{Z}_n$. Now the encrypted token in protocol Π_P becomes $c' = g^{salt \cdot m} \cdot r^n \mod n^2$, and the MB will receive c', $r^n \mod n^2$ and $g^{salt} \mod n^2$ for keyword matching operations over every token. Again observe that with $r^n \mod n^2$ and n^2 which is part of R's public key, the MB can compute $r^{-n} \mod n^2$ and then derive $g^{salt \cdot m} \mod n^2$. Thus, $r^n \mod n^2$ becomes unnecessary in protocol Π_P for the MB to perform keyword matching.

Now protocol Π_P encrypts a token as $g^{salt \cdot m} \mod n^2$ which is exactly Eq. 1. To perform keyword matching, the MB would need $g^{salt \cdot m} \mod n^2$ and $g^{salt} \mod n^2$. However, this random $salt$ prevents the MB from performing efficient search for keyword matching operations. Specifically, the MB needs to encrypts each keyword, denoted by w, in rule set by computing $(g^{salt})^w$. The MB has to compare a received encrypted token with all of the encrypted keywords, whose number will probably be daunting. To improve searching performance at the MB, $salt$ can be defined as pseudorandom. This change gives our proposed protocol. Moreover, this method further reduces the communication overhead since now S only needs to transmit g^{salt} once.

B. Extensions for Malware Detection

In our protocol, we use linear kernel function with SVM as the method for malware detection. Applying other nonlinear kernel functions is also possible with our protocol. For example, consider Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\sigma^2)$. Then the decision function in Eq. 3 can be rewritten as $f(\boldsymbol{x}) = \sum_i \alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i)$ with those $\boldsymbol{x}_i$'s that constitute the support vectors. A straightforward way to perform this SVM model involves a one-round interaction between the MB and R. At a high level, the MB first securely computes $(\boldsymbol{x}-\boldsymbol{x}_i)$ for all support vectors, separately, and then forwards these generated results to R. Then R can decrypt these results to get $(\boldsymbol{x} - \boldsymbol{x}_i)$ and then calculate all $k(\boldsymbol{x} - \boldsymbol{x}_i) = \exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\sigma^2)$, which will be encrypted and sent back to the MB for computing the last step $f(\boldsymbol{x}) = \sum_i \alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i)$. Note that all of the computations done in the MB are performed in a privacy-preserving way based on the homomorphic properties stated in Section IV-A. Finally, the MB sends the encrypted classification decision to R. Note that this method may give better classification results at the expense of more overhead on both clients and the MB sides.

Another possible extension is to use other ML models instead of SVM, such as Naïve Bayes and Decision Trees[22]. However, it might incur much more overhead and lose the transparency of the MB. Normally, they would involve more than one round of interaction between the MB and R.

C. Comparison with Blindbox

Although SPABox has an architecture similar to that of Blindbox, it uses different approaches. First, Blindbox uses symmetric encryption schemes as its basis, while SPABox uses public-key encryption schemes. While this choice lowers the throughput, more operations can be supported (e.g., machine learning), and the connection setup overhead can be greatly avoided in our system. To compensate for the reduced throughput, we build a multi-layered hashtable and carefully design our encryption scheme to accelerate the keyword look-up at the MB. Second, we push the regular expression evaluation to end users while Blindbox does it at the MB by decrypting the traffic. Our design ensures that the traffic is kept encrypted from end to end and the overhead incurred at the receiver side is limited. Note that the MB is necessary in both BlindBox and SPABox designs to protect the rule set (otherwise end users will have access to it), and therefore our method serves as a tradeoff between R's performance and the purpose of offering regular expression evaluation functionality and traffic/rule set privacy simultaneously.

IX. Conclusions and Future Work

Whether or not the use of HTTPS could lead to the death of DPI has long been a hot topic of debate. In this paper, we have presented SPABox, the first middlebox based system that supports both keyword based and data analysis based DPI functionalities over encrypted traffic, while guaranteeing the privacy of the user data at the middlebox. The most notable feature of SPABox is that protocol setup does not require any interaction or data transmission between a middlebox and clients. SPABox also enables privacy-preserving regular expression evaluation and machine learning using SVM for malware detection. The performance evaluation of SPABox demonstrates that it provides privacy-preserving DPI with a limited overhead. To further improve the performance, we are working on privacy-preserving regular expression evaluation schemes, as part of our future work, which can be integrated with our solution in order to fully support outsourcing DPI.

References

[1] Data Plane Development Kit. http://dpdk.org.

[2] Google's dense hash map. http://goog-sparsehash.sourceforge.net/doc/dense_hash_map.html.

[3] ICTF Data. https://ictf.cs.ucsb.edu/.

[4] McAfee Network Security Platform. http://www.mcafee.com/us/products/network-security-platform.aspx.

[5] Microsoft Malware Classification Challenge (BIG 2015). https://www.kaggle.com/c/malware-classification.

[6] NSA Spying Relies on AT&T's 'Extreme Willingness to Help. https://www.propublica.org/article/nsa-spying-relies-on-atts-extreme-willingness-to-help.

[7] OpenSSL. https://www.openssl.org/.

[8] Overcoming Targeted Attacks: A New Approach. https://blogs.mcafee.com/mcafee-labs/overcoming-targeted-attacks-new-approach/.

[9] Proofpoint vs. Intel McAfee. https://www.proofpoint.com/sites/default/files/documents/bnt_download/intel-mcafee-cg_1.pdf.

[10] Protection from Advanced Threats with Symantec Insight and SONAR. https://www.symantec.com/content/en/us/enterprise/white_papers/protection-advanced-threats-insight-sonar-wp-21360326.pdf.

[11] Snort v2.9. https://www.snort.org/downloads.

[12] Symantec Adds Deep Learning to Anti-Malware Tools to Detect Zero-Days. http://www.eweek.com/security/symantec-adds-deep-learning-to-anti-malware-tools-to-detect-zero-days.html.

[13] Symantec Enterprise. http://www.symantec.com/index.jsp.

[14] VX Heaven. https://vxheaven.org/.

[15] McAfee Labs Threats Report. Technical report, McAfee Labs, 2015.

[16] J. Aycock. *Computer Viruses and Malware*. Springer, 2006.

[17] M. Becchi and P. Crowley. A hybrid finite automaton for practical deep packet inspection. In *ACM CoNEXT*, 2007.

[18] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*. Springer, 2007.

[19] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical computer science*, 48:117–126, 1986.

[20] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Eurocrypt*. Springer, 2004.

[21] J. W. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[22] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. *Crypto ePrint Archive*, 2014.

[23] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Deep packet inspection as a service. In *ACM CoNEXT*, 2014.

[24] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *ACM CCS*, 2015.

[25] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

[26] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *International Conference on Cryptology and Information Security in Latin America*, pages 40–58. Springer, 2015.

[27] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. Technical report, IETF, 2008.

[28] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[29] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[30] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. Di Pietro. An improved DFA for fast regular expression matching. *ACM SIGCOMM CCR*, 38(5):29–40, 2008.

[31] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM*, 2014.

[32] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification, 2003.

[33] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[34] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *ACM SIGKDD*, 2004.

[35] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu. Embark: Securely Outsourcing Middleboxes to the Cloud. In *NSDI*, 2016.

[36] K. S. McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74, 1990.

[37] P. Mohassel, S. Niksefat, S. Sadeghian, and B. Sadeghiyan. An efficient protocol for oblivious DFA evaluation and applications. In *Cryptographers Track at the RSA Conference*, pages 398–415. Springer, 2012.

[38] K. Namjoshi and G. Narlikar. Robust and fast pattern matching for intrusion detection. In *IEEE INFOCOM*, 2010.

[39] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM*, 2015.

[40] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*. Springer, 1999.

[41] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125. Springer, 2008.

[42] M. A. Salehi, T. Caldwell, A. Fernandez, E. Mickiewicz, E. W. Rozier, S. Zonouz, and D. Redberg. RESeED: Regular Expression Search over Encrypted Data in the Cloud. In *IEEE CLOUD*, 2014.

[43] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *ACM SIGCOMM*, 2015.

[44] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P*, 2000.

[45] S. M. Tabish, M. Z. Shafiq, and M. Farooq. Malware detection using statistical analysis of byte-level file content. In *ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 2009.

[46] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *ACM SIGCOMM CCR*, volume 41, pages 374–385, 2011.

[47] B. Waters. Functional encryption for regular languages. In *CRYPTO*. 2012.

[48] L. Wei and M. K. Reiter. Toward practical encrypted email that supports private, regular-expression searches. *International Journal of Information Security*, 14(5):397–416, 2015.

[49] A. Yao. How to generate and exchange secrets. In *IEEE FOCS*, 1986.

[50] Y. Ye, D. Wang, T. Li, and D. Ye. Imds: Intelligent malware detection system. In *ACM SIGKDD*, 2007.

Jingyuan Fan received the B.Eng. and M.S. degree from Fudan University, China, and the University of California, Los Angeles, USA, in 2012 and 2014, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, the State University of New York at Buffalo. His research interests lie in the field of computer networks.

Chaowen Guan received the B.Eng. Degree from Jinan University, Guangzhou, in 2011. He was a research engineer in Singapore Management University from 2012 to 2013. Now he is a Ph.D. student at SUNY, Buffalo. His primary research interest lies in the field of cryptography and information security.

Kui Ren is a professor of Computer Science and Engineering and the director of UbiSeC Lab at State University of New York at Buffalo (UB). He received his PhD degree from Worcester Polytechnic Institute. Kuibs current research interest spans Cloud & Outsourcing Security, Wireless & Wearable Systems Security, and Mobile Sensing & Crowdsourcing. His research has been supported by NSF, DoE, AFRL, MSR, and Amazon. Kui has published extensively in peer reviewed journals and conferences and received several Best Paper Awards including ICDCS 2017, IWQoS 2017, and ICNP 2011. He currently serves as an associate editor for IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Service Computing, IEEE Transactions on Mobile Computing, IEEE Wireless Communications, IEEE Internet of Things Journal, and an editor for Spinger Briefs on Cyber Security Systems and Networks. Kui is a Fellow of IEEE, a Distinguished Lecturer of IEEE, a member of ACM, and a past board member of Internet Privacy Task Force, State of Illinois.

Yong Cui received the B.E. and Ph.D. degrees both from Tsinghua University. He is currently a professor at the Computer Science Department in Tsinghua University. Co-chairing an IETF WG, he served or serves at the editorial boards on IEEE TPDS, IEEE TCC, IEEE Network and IEEE Internet Computing. He published over 100 papers with several Best Paper Awards and co-authored around 10 Internet standard documents (RFC). His research interests include mobile computing and network architecture.

Chunming Qiao is a SUNY Distinguished Professor and also the current Chair of the Computer Science and Engineering Department at University at Buffalo. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols. His current focus is on connected and autonomous vehicles. He has published extensively with an h-index of over 69 (according to Google Scholar). Two of his papers have received the best paper awards from IEEE and Joint ACM/IEEE venues. He also has 7 US patents and served as a consultant for several IT and Telecommunications companies since 2000. His research has been funded by a dozen major IT and telecommunications companies including Cisco and Google, and more than a dozen NSF grants.