BlindBox: Deep Packet Inspection over Encrypted Traffic

Justine Sherry
UC Berkeley

Chang Lan
UC Berkeley

Raluca Ada Popa
ETH Zürich and UC Berkeley

Sylvia Ratnasamy
UC Berkeley

ABSTRACT

Many network middleboxes perform *deep packet inspection* (DPI), a set of useful tasks which examine packet payloads. These tasks include intrusion detection (IDS), exfiltration detection, and parental filtering. However, a long-standing issue is that once packets are sent over HTTPS, middleboxes can no longer accomplish their tasks because the payloads are encrypted. Hence, one is faced with the choice of *only one* of two desirable properties: the functionality of middleboxes and the privacy of encryption.

We propose *BlindBox*, the first system that simultaneously provides *both* of these properties. The approach of BlindBox is to perform the deep-packet inspection *directly on the encrypted traffic*. BlindBox realizes this approach through a new protocol and new encryption schemes. We demonstrate that BlindBox enables applications such as IDS, exfiltration detection and parental filtering, and supports real rulesets from both open-source and industrial DPI systems. We implemented BlindBox and showed that it is practical for settings with long-lived HTTPS connections. Moreover, its core encryption scheme is 3-6 orders of magnitude faster than existing relevant cryptographic schemes.

CCS Concepts

• Security and privacy → Cryptography; Security protocols; • Networks → Middleboxes / network appliances;

Keywords

middlebox privacy; network privacy; searchable encryption

1. INTRODUCTION

Many network middleboxes perform *deep packet inspection* (DPI) to provide a wide range of services which can benefit both end users and network operators. For example, Network Intrusion Detection/Prevention (IDS/IPS) systems (e.g., Snort [9] or Bro [36]) detect if packets from a compromised sender contain an attack. Exfiltration prevention devices block accidental leakage of private data in enterprises by searching for document confidentiality watermarks in the data transferred out of an enterprise network [45]. Parental filtering devices prevent children from accessing adult material in schools, libraries, and homes [13]. These devices and many others [6, 8, 7] all share the common feature that they inspect packet payloads; the market for such DPI devices is expected to grow to over $2B by 2018 [44].

At the same time, HTTPS and other encryption protocols have seen dramatic growth in usage in recent years [35]; encryption protects users' private data from eavesdroppers anywhere in the network, including at a middlebox. Unfortunately, HTTPS poses a long-standing challenge for DPI devices. Since packet payloads are encrypted, network middleboxes can no longer inspect payloads [35] and accomplish their tasks. To enable middlebox processing, some currently deployed middlebox systems support HTTPS in an *insecure* way: they mount a man-in-the-middle attack on SSL and decrypt the traffic at the middlebox [28, 27]. This approach violates the end-to-end security guarantees of SSL and thus causes an unfortunate set of issues as surveyed in [28]. Moreover, users and clients have criticized this approach [51, 31, 41, 48], some expressing worry that the private data logged at the middlebox is given to marketers.

Therefore, one is faced with an unfortunate choice of *only one* of two desirable properties: the functionality of the middleboxes and the privacy given by encryption. At a first glance, it may appear that these two properties are fundamentally at odds with each other: how can a network appliance make decisions based on connection contents when it cannot see these contents?

In this paper, we demonstrate that it is possible to build a system that satisfies both of these seemingly conflicting properties. We present *BlindBox*, the first system that provides both the benefits of encryption and functionality at a DPI middlebox. The name "BlindBox" denotes that the middlebox cannot see the private content of traffic.

Our approach is to perform *the inspection directly on the encrypted payload*, without decrypting the payload at the middlebox. Building a practical such system is challenging: networks operate at very high rates requiring cryptographic operations on the critical path to run in micro or even nano seconds; further, some middleboxes require support for rich operations, such as matching regular expressions. A potential candidate is expressive cryptographic schemes such as fully homomorphic or functional encryption [24, 23, 26],

SIGCOMM '15, August 17 - 21, 2015, London, United Kingdom

ISBN 978-1-4503-3542-3/15/08…$15.00

DOI: http://dx.doi.org/10.1145/2785956.2787502

是为http而键的

but these are prohibitively slow, decreasing network rates by many orders of magnitude.

To overcome these challenges, BlindBox explores and specializes on the network setting. BlindBox enables two classes of DPI computation each having its own privacy guarantees: *exact match privacy* and *probable cause privacy*. Both of BlindBox's privacy models are much stronger than the state-of-the-art "man in the middle" approach deployed today. In both of these models, BlindBox protects the data with strong randomized encryption schemes providing similar security guarantees to the well-studied notion of searchable encryption [46, 29]. Depending on the class of computation, BlindBox allows the middlebox to learn a small amount of information about the traffic to detect rules efficiently.

The first class of computation consists of DPI applications that rely only on exact string matching, such as watermarking, parental filtering, and a limited IDS. Under the associated privacy model, exact match privacy, the middlebox learns at which positions in a flow attack keywords occur; for substrings of the flow that do not match an attack keyword, the middlebox learns virtually nothing.

The second class of computation can support all DPI applications, including those which perform regular expressions or scripting. The privacy model here, probable cause privacy, is a new network privacy model: the middlebox gains the ability to see a (decrypted) individual packet or flow *only if the flow is suspicious*; namely, the flow contains a string that matches a known attack keyword. If the stream is not suspicious, the middlebox cannot see the (decrypted) stream. Hence, privacy is affected *only with a cause*. In practice, most traffic remains encrypted. BlindBox allows users to select which privacy model they are most comfortable with.

To implement these two models, we developed the following techniques:

- **DPIEnc** and **BlindBox Detect** are a new searchable encryption scheme [46] and an associated fast detection protocol, which can be used to inspect encrypted traffic for certain keywords efficiently. As we explain in §3, existing searchable encryption schemes [46, 29, 15] are either deterministic (which can enable fast protocols, but provide weak security) or randomized (which have stronger security, but are slow in our setting). DPIEnc with BlindBox Detect achieve both the speed of deterministic encryption and the security of randomized encryption; detection on encrypted traffic runs as fast as on unencrypted traffic.
- **Obfuscated Rule Encryption** is a technique to allow the middlebox to obtain encrypted rules based on the rules from the middlebox and the private key of the endpoints, without the endpoints learning the rules or the middlebox learning the private key. This technique builds on Yao's garbled circuits [50] and oblivious transfer [34, 39, 22].
- **Probable Cause Decryption** is a mechanism to allow flow decryption when a suspicious keyword is observed in the flow; this is the mechanism that allows us to implement our probable cause privacy model.

有可疑的关键字就对流进行解密

We implemented BlindBox as well as a new secure transport protocol for HTTP, which we call BlindBox HTTPS. We evaluated BlindBox HTTPS using several real DPI applications: data watermarking [45], parental filtering, and intrusion detection. We used attack signatures from the open-source community (Snort [9]) and industrial IDSes (McAfee Stonesoft and Lastline).

We show that BlindBox's performance is practical for many settings. For example, the rate at which the middlebox can inspect packets is as high as 186Mbps per core in our experiments. Given that standard IDS implementations, such as Snort [9], peak at under 100Mbps, this performance is competitive with existing deployments. We achieve this performance due to DPIEnc and BlindBox Detect. When compared to two strawmen consisting of a popular searchable encryption scheme [46] and a functional encryption scheme [30], DPIEnc with BlindBox Detect are 3-6 orders of magnitude faster. Nevertheless, a component of BlindBox is not yet as fast as desirable: the setup of an HTTPS connection. This setup performs obfuscated rule encryption and it takes time proportional to the number of attack rules. For rulesets with tens of keywords, this setup completes in under a second; however, for large IDS installations with thousands of rules, the setup can take up to 1.5 minutes to complete. Hence, BlindBox is most fit for settings using long or persistent connections through SPDY-like protocols or tunneling, and not yet practical for short, independent flows with many rules.

We see BlindBox as a first step towards a general protocol that will allow middleboxes and end-to-end encryption to coexist, without sacrificing the benefits of either. BlindBox shows that this coexistence is possible for the class of middleboxes performing deep packet inspection. A future, general protocol will devise mechanisms to address the full range of middleboxes including those that process packet headers, modify packet payloads (such as transcoders and WAN optimizers), or terminate sessions (such as Web or SIP proxies). Consequently, the subject of our current and future work focuses on achieving this end goal: a general protocol to allow all middleboxes to coexist with encryption.

2. OVERVIEW

Fig. 1 presents the system architecture. There are four parties: sender (S), receiver (R), middlebox (MB), and rule generator (RG) – these reflect standard middlebox deployments today. RG generates attack rules (also called signatures) to be used by MB in detecting attacks. Each rule attempts to describe an attack and it contains fields such as: one or more keywords to be matched in the traffic, offset information for each keyword, and sometimes regular expressions. The RG role is performed today by organizations like Emerging Threats [3], McAfee [4], or Symantec [11]. S and R send traffic through MB. MB allows S and R to communicate unless MB observes an attack rule in their traffic.

Network administrators today deploy in-network middleboxes in this way because it gives them a central point of control to enforce security policies in their network (since they often do not control the endpoints). Moreover, having a single in-network device is also easy to manage and

RG是攻击规则，在正常的发送和接收之间插入一个发送方，希望BB能识别这种流量

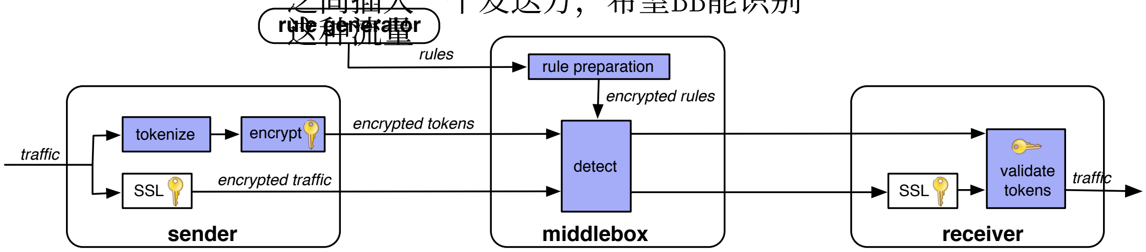


Figure 1: System architecture. Shaded boxes indicate algorithms added by BlindBox.

upgrade. Some research [21] considered alternative (edge) deployments in which endpoints perform all middlebox processing on their own traffic and rules are distributed to the endpoints. BlindBox instead aims to make changes compatible with the network approach in widespread use today. Further, this edge-based model is not compatible with the requirement of keeping rules hidden from endpoints, which we discuss in §2.2.1.

In today's deployments, MB can read any traffic sent between S and R. With BlindBox, MB should be able to detect if attack rules generated by RG match the traffic between R and S, but should not learn the contents of the traffic that does not match RG's attack rules.

2.1 Usage Scenarios

Before formalizing our threat model, we illustrate our usage scenario with three examples. For each individual in these examples, we indicate the party in our model (R, S, MB, or RG) that they correspond to.

Example #1: University Network. Alice (R or S) is a student at the University of SIGCOMM and brings her own laptop to her dorm room. However, university policy requires that all student traffic be monitored for botnet signatures and illegal activity by a middlebox (MB) running an IDS. Alice is worried about her computer being infected with botnet software, so she also desires this policy applied to her traffic. McAfee (RG) is the service that provides attack rules to the middlebox and Alice trusts it. However, she is uncomfortable with the idea of someone she doesn't know (who has access to the middlebox) potentially being able to read her private Facebook messages and emails. Alice installs BlindBox HTTPS with McAfee's public key, allowing the IDS to scan her traffic for McAfee's signatures, but not read her private messages.

Example #2: ISP Service. Bob has two young children (S or R) at home, and registers for parental filtering with his ISP so that all traffic is filtered for adult content. However, Bob has read stories in the news of ISPs selling user browsing data to marketers [48] and wants to prevent his ISP (MB) from using his data in this way. Bob trusts the Electronic Filtering Foundation (RG), a non-profit which generates rulesets for filtering and pledges not to sell user data. Bob installs BlindBox HTTPS on his home computer with the Electronic Filtering Foundation's public key, allowing his traffic to be scanned for EFF rules, but no other data.

In the above examples, Alice and Bob want to have a middlebox in their network check *for the attack rules the corresponding trusted parties permit*, but the middlebox should not learn anything else about the content of the traffic. A key requirement is that there exists an RG which Alice, Bob and the MB trust with rule generation; if this is not the case, the parties cannot use BlindBox.

Anti-Example #1: Political Dissident. Charlie (R or S) is a political dissident who frequently browses sensitive websites, and is concerned about government monitoring. If the government coerces one of MB or RG, Charlie remains protected. However, BlindBox should not be used in a setting in which *both* MB and RG can be controlled by an attacker: in this case, RG can produce signatures for sensitive terms and MB will use these to match the traffic. Hence, if the government can coerce both MB and RG together, Charlie should not use BlindBox. Similarly, if the government can coerce root certificate generators, Charlie should not use vanilla HTTPS either because it may allow man-in-the-middle attacks on his traffic.

2.2 Security and Threat Model

The goal of our work is to protect the privacy of user traffic from MB. Any solution must satisfy a set of systems requirements we discuss in §2.2.1. We then discuss the threat model in §2.2.2 and the privacy guarantees BlindBox provides in §2.2.3.

2.2.1 *System Requirements*

BlindBox retains key system goals of traditional IDS deployments today: (1) BlindBox must maintain MB's ability to enforce its policies (i.e., detect rules and drop/alert accordingly), and (2) endpoints must not gain access to the IDS rules. The rationale behind the second requirement is twofold. First, in order to make IDS evasion more difficult for an attacker at the user, the rules should be hidden from the endpoints [36]. Second, most vendors (e.g., Lastline and McAfee Stonesoft) rely on the secrecy of their rulesets in their business model, as their value added against competitors often includes more comprehensive, more efficient, or harder to evade rules.

BlindBox maintains these two requirements, and adds an additional one: (3) that the middlebox cannot read the user's traffic, except the portions of the traffic which are considered *suspicious* based on the attack rules.

2.2.2 *Threat Model*

There are two types of attackers in our setup.

The original attacker considered by IDS. This is the same attacker that traditional (unencrypted) IDS consider and we do not change the threat model here. Our goal is to detect such an attacker over *encrypted* traffic. As in traditional IDS, one endpoint can behave maliciously, but at least one endpoint must be honest. This is a fundamental requirement of any IDS [36] because otherwise two malicious endpoints can agree on a secret key and encrypt their traffic under that key with a strong encryption scheme, making prevention impossible by the security properties of the encryption scheme. Similarly, the assumption that one endpoint is honest is also the default for exfiltration detection and parental filtering today. Parental filters can assume one endpoint is innocent under the expectation that 8-year-olds are unlikely replace their network protocol stack or install tunneling software. Commercial exfiltration detection devices primarily target *accidental* exfiltration (e.g., where an otherwise innocent employee attaches the wrong file to an email), recognizing that deliberate exfiltration requires control of the end host.

The attacker at the middlebox. This is the new attacker in our setting. This attacker tries to subvert our scheme by attempting to extract private data from the encrypted traffic passing through the middlebox. We assume that the middlebox MB performs the detection honestly, but that it tries to learn private data from the traffic and violate the privacy of the endpoints. In particular, we assume that an attacker at MB reads *all* the data accessible to the middlebox, including traffic logs and other state. Given this threat model, BlindBox's goal is to hide the content of the traffic from MB, while allowing MB to do DPI. We do not seek to hide the attack rules from the MB itself; many times these rules are hardcoded in the MB.

2.2.3 Privacy Models

We now describe our privacy models.

Exact Match Privacy gives the following guarantee: the middlebox will be able to discover only those substrings of the traffic that are exact matches for known attack keywords. For example, if there exists a rule for the word "ATTACK", the middlebox will learn at which offset in the flow the word "ATTACK" appears (if it appears), but does not learn what the other parts of the traffic are. Traffic which does not match a suspicious keyword remains unreadable to the middlebox.

Probable Cause Privacy gives a different guarantee: that the middlebox will be able to *decrypt a flow* only if a substring of the flow is an exact match for a known attack keyword. Probable cause privacy is useful for IDS tasks which require regular expressions or scripting to complete their analysis. This model is inspired from two ideas. First, it is inspired from the notion of probable cause from United States' criminal law: one should give up privacy *only* if there is a reason for suspicion. Second, most rules in Snort that contain regular expressions first attempt to find a suspicious keyword in the packet – this keyword is selective so only a small fraction of the traffic matches this string and is passed through the regexp. Indeed, the Snort user manual [47] urges the presence of such selective keywords because otherwise, detection would be too slow. Since rules are structured this way, it becomes easier to implement our probable cause privacy model by decrypting the stream if there is a match to the suspicious keyword.

Exact match privacy provides security guarantees as in searchable encryption [46], which are well-studied. Probable cause privacy is a new privacy model, and we believe it may be useful in other network domains beyond middleboxes (e.g. network forensics or search warrants), although we leave such investigation to future work. We formalize and prove the security guarantees of BlindBox using standard indistinguishability-based definitions in our extended paper [43]. Both models are stronger than the "man in the middle" approach in deployment today, where all traffic is decrypted regardless of suspicion. A user who prefers exact match privacy over probable cause privacy can indicate so within BlindBox HTTPS.

2.3 System Architecture

We now return to Fig. 1 to explain each module and how BlindBox functions from a high level; we delve into the protocol and implementation details in the following sections.

Prior to any connection, RG generates a set of *rules* which contain a list of suspicious keywords known to formulate parts of attacks; RG signs these rules with its private key and shares them with MB, its customer. S and R, who trust RG, install a BlindBox HTTPS configuration which includes RG's public key. Beyond this initial setup, RG is never directly involved in the protocol. We now discuss the interactions between R, S, and MB when R and S open a connection in a network monitored by MB.

Connection setup. First, the sender and receiver run the regular SSL handshake which permits them to agree on a key k_0. The sender and receiver use k_0 to derive three keys (e.g., using a pseudorandom generator):

- k_{SSL}: the regular SSL key, used to encrypt the traffic as in the SSL protocol,
- k: used in our detection protocol, and
- k_{rand}: used as a seed for randomness. Since both endpoints have the same seed, they will generate the same randomness.

At the same time, MB performs its own connection setup to be able to perform detection over S and R's traffic. In an exchange with S and R, MB obtains each rule from RG deterministically encrypted with key k – this will later enable MB to perform the detection. However, this exchange occurs in such a way that MB *does not learn the value of* k and in such a way that R and S *do not learn what the rules are*. We call this exchange *obfuscated rule encryption* and we describe how it is implemented in the following section.

Unlike the above handshake between S and R, which bootstraps off the existing SSL handshake, obfuscated rule encryption is a new exchange. In existing deployments, clients typically do not communicate directly with DPI middleboxes (although for other kinds of middleboxes, such as explicit proxies [17] or NAT hole-punching [20], they may do so). Even though this step removes the complete "transparency" of the DPI appliance, it is an incremental change that we consider an acceptable tradeoff for the benefits of BlindBox.

Sending traffic. To transmit, the sender: (1) encrypts the traffic with SSL as in a non-BlindBox system; (2) tokenizes the traffic by splitting it in substrings taken from various offsets (as discussed in §3); and (3) encrypts the resulting tokens using our DPIEnc encryption scheme.

Detection. The middlebox receives the SSL-encrypted traffic and the encrypted tokens. The detect module will search for matchings between the encrypted rules and the encrypted tokens using BlindBox Detect (Sec. 3.2). If there is a match, one can choose the same actions as in a regular (unencrypted IDS) such as drop the packet, stop the connection, or notify an administrator. After completing detection, MB forwards the SSL traffic and the encrypted tokens to the sender.

Receiving traffic. Two actions happen at the receiver. First, the receiver decrypts and authenticates the traffic using regular SSL. Second, the receiver checks that the encrypted tokens were encrypted properly by the sender. Recall that, in our threat model, one endpoint may be malicious – this endpoint could try to cheat by not encrypting the tokens correctly or by encrypting only a subset of the tokens to eschew detection at the middlebox. Since we assume that at least one endpoint is honest, such verification will prevent this attack.

Because BlindBox only supports attack rules at the HTTP application layer, this check is sufficient to prevent evasion. Almost all the rules in our datasets were in this category. Nonetheless, it is worth noting that, if an IDS were to support rules that detected attacks on the client driver or NIC – before verification –, an attacker *could* evade detection by not tokenizing.

2.4 Protocols

BlindBox provides three protocols. In Protocol I, a rule consists of one keyword. MB must be able to detect if the keyword appears at any offset in the traffic based on equality match. This protocol suffices for document watermarking [45] and parental filtering [13] applications, but can support only a few IDS rules. In Protocol II, a rule consists of multiple keywords as well as position information of these keywords. This protocol supports a wider class of IDS rules than Protocol I, as we elaborate in §7. Protocol I and II provide Exact Match Privacy, as discussed in §2.2.3. Protocol III additionally supports regular expressions and scripts, thus enabling a full IDS. Protocol III provides Probable Cause Privacy, as discussed in §2.2.3.

只使用一个关键字进行匹配

3. PROTOCOL I: BASIC DETECTION

Protocol I enables matching a suspicious keyword against the encrypted traffic. An attack rule in this protocol consists of one keyword. Even though this protocol is the simplest of our protocols, it introduces the majority of our techniques. The other protocols extend Protocol I.

To detect a keyword match on encrypted text, one naturally considers searchable encryption [46, 29]. However, existing searchable encryption schemes do not fit our setting for two reasons. First, the setup of searchable encryption requires the entity who has the secret key to encrypt the rules; this implies, in our setting, that the endpoints see the rules (which is not allowed as discussed in §2.2.2). Our obfuscated rule encryption addresses this problem.

Second, none of the existing schemes meet both of our security and network performance requirements. There are at least two kinds of searchable encryption schemes: deterministic and randomized. Deterministic schemes [15] leak whether two words in the traffic are equal to each other (even if they do not match a rule). This provides weak privacy because it allows an attacker to perform frequency analysis. At the same time, these schemes are fast because they enable MB to build fast indexes that can process each token (e.g., word) in a packet in time logarithmic in the number of rules. On the other hand, randomized schemes [46, 29] provide stronger security guarantees because they prevent frequency analysis by salting ciphertexts. However, the usage of the salt in these schemes requires combining each token with each rule, resulting in a processing time linear in the number of rules for each token; as we show in §7, this is too slow for packet processing. In comparison, our encryption scheme DPIEnc and detection protocol BlindBox Detect achieve the best of both worlds: the detection speed of deterministic encryption and the security of randomized encryption.

确定性方案会透露关键字相同的信息，隐私不够安全。如果通过增减salt对原密文进行保护，处理的时间会线性增加，太慢。本文的加密方案DPIEnc和检测协议BlindBox可以增加检测速度并对安全性也有保证。

Let us now describe how each BlindBox module in Fig. 1 works in turn. Recall that S and R are the sender and receiver, MB the middlebox and RG the rule generator.

Tokenization. The first step in the protocol is to tokenize the input traffic. We start with a basic tokenization scheme, which we refer to as "window-based" tokenization because it follows a simple sliding window algorithm. For every offset in the bytestream, the sender creates a token of a fixed length: we used 8 bytes per token in our implementation. For example, if the packet stream is "alice apple", the sender generates the tokens "alice ap", "lice app", "ice appl", and so on. Using this tokenization scheme, MB will be able to detect rule keywords of length 8 bytes or greater. For a keyword longer than 8 bytes, MB splits it in substrings of 8 bytes, some of which may overlap. For example, if a keyword is "maliciously", MB can search for "maliciou" and "iciously". Since each encrypted token is 5 bytes long and the endpoint generates one encrypted token per byte of traffic, the bandwidth overhead of this approach is of 5×.

令牌长度为8字节，使用滑动窗口生成，绝大部分字节都会生成8遍。加密令牌5字节，此方法的带宽开销为正常的5倍

We can reduce this bandwidth overhead by introducing some optimizations. First, for an HTTP-only IDS (which does not analyze arbitrary binaries), we can have senders ignore tokenization for images and videos which the IDS does not need to analyze. Second, we can tailor our tokenization further to the HTTP realm by observing how the keywords from attack rules for these protocols are structured. The keywords matched in rules start and end before or after a delimiter. Delimiters are punctuation, spacing, and special symbols. For example, for the payload "login.php?user=alice", possible keywords in rules are typically "login", "login.php", "?user=", "user=alice", but not "logi" or "logi.ph". Hence, the sender needs to generate only those tokens that could match keywords that start and end on delimiter-based offsets; this allows us to ignore redundant tokens in the window. We refer to this tokenization as "delimiter-based" tokeniza-

我们可以通过引入一些优化来减少这种带宽开销。首先，对于一个只支持超文本传输协议的入侵检测系统(它不分析任意二进制文件)，忽略入侵检测系统不需要分析的图像和视频的标记。其次，观察这些协议的攻击规则中的关键字是如何构造的，来进一步针对HTTP领域定制令牌化。规则中匹配的关键字在分隔符之前或之后开始和结束。分隔符是标点、空格和特殊符号。比如对于有效载荷“ login.php？user=alice ”，规则中可能的关键字通常是" login "，" login.php "，"？user= "，" user=alice "，但不是" logi "或" logi.ph "。因此，发送者只需要生成那些能够匹配关键字的标记，这些关键字以基于分隔符的偏移量开始和结束；忽略窗口中的冗余标记。我们将这种标记化称为“ 基于分隔符 ”的标记化。在第7章中，我们比较了这两种令牌化协议的开销和覆盖范围。

因此，天真地，MB对每个令牌t和规则r执行匹配测试，这导致每个令牌的性能与规则数量成线性关系；这太慢了。为了解决这一问题，我们下面的检测算法使这一成本在规则数量上成对数关系，与对未加密流量的普通检查相同。这将显著提高性能:例如，对于一个有10000个匹配关键字的规则集，对数查找比线性扫描快四个数量级。

tion. In §7, we compare the overheads and coverage of these two tokenization protocols.

我们介绍了我们新的DPIEnc加密方案，它由图1中的加密模块使用。发送方使用我们的加密方案对从令牌化中获得的每个令牌t进行加密。DPIEnc中令牌t的加密是

3.1 The DPIEnc Encryption Scheme

In this subsection, we present our new DPIEnc encryption scheme, which is used by the Encrypt module in Fig. 1. The sender encrypts each token t obtained from the tokenization with our encryption scheme. The encryption of a token t in DPIEnc is:

$$\mathsf{salt},\ \mathsf{AES}_{\mathsf{AES}_k(t)}(\mathsf{salt}) \bmod \mathsf{RS}, \qquad (1)$$

让我们解释一下DPIEnc背后的原理。为此，假设MB正在被移交，对于每个规则r，对(r，AESk(r))，但不是密钥k。我们在3.3中解释MB实际上如何获得AESk(r)。让我们从考虑一个简单的确定性加密方案而不是从DPIEnc开始:加密t使用AESk(t)。那么，要检查t是否等于一个关键字r，MB可以简单的检查一下AESk(t)是否等于AESk(r)。不幸的是，由此产生的安全性很弱，因为t的每次出现都会有相同的密文。为了解决这个问题，我们需要随机化加密。

where salt is a randomly-chosen value and RS is explained below.

Let us explain the rationale behind DPIEnc. For this purpose, assume that MB is being handed, for each rule r, the pair $(r, \mathsf{AES}_k(r))$, but not the key k. We explain in §3.3 how MB actually obtains $\mathsf{AES}_k(r)$.

Let's start by considering a simple deterministic encryption scheme instead of DPIEnc: the encryption of t is $\mathsf{AES}_k(t)$. Then, to check if t equals a keyword r, MB can simply check if $\mathsf{AES}_k(t) \stackrel{?}{=} \mathsf{AES}_k(r)$. Unfortunately, the resulting security is weak because every occurrence of t will have the same ciphertext. To address this problem, we need to randomize the encryption.

因此，我们使用一个“随机函数”H和一个随机salt，密文变成:salt，H(salt，AESk(t))。直觉上，H一定是伪随机的，不可逆的。为了执行匹配，MB然后可以基于AESk(r)和salt计算H(salt，AESk(r))，并再次执行相等检查。H的典型实例是SHA-1，但SHA-1没有AES快(因为AES是在现代处理器的硬件上实现的)，可以降低BlindBox的网络吞吐量。相反，我们用AES实现H，但这必须小心进行，因为这些原语具有不同的安全属性。为了获得H的属性，当攻击规则不匹配时，AES必须用MB不知道的值来加密——因此，该值是AESk(t)。我们的算法现在完全在AES中实现，这使得它很快。

Hence, we use a "random function" H together with a random salt, and the ciphertext becomes: $\mathsf{salt}, H(\mathsf{salt}, \mathsf{AES}_k(t))$. Intuitively, H must be pseudorandom and not invertible. To perform a match, MB can then compute $H(\mathsf{salt}, \mathsf{AES}_k(r))$ based on $\mathsf{AES}_k(r)$ and salt, and again perform an equality check. The typical instantiation of H is SHA-1, but SHA-1 is not as fast as AES (because AES is implemented in hardware on modern processors) and can reduce BlindBox's network throughput. Instead, we implement H with AES, but this must be done carefully because these primitives have different security properties. To achieve the properties of H, AES must be keyed with a value that MB does not know when there is no match to an attack rule – hence, this value is $\mathsf{AES}_k(t)$. Our algorithm is now entirely implemented in AES, which makes it fast.

最后，RS只是简单地减小密文的大小以减少带宽开销，但并不影响安全性。在我们的实现中，RS是2^40，产生5字节的密文长度。结果，密文不再可解密；这不是问题，因为盲箱总是从主要的SSL流解密流量。

Finally, RS simply reduces the size of the ciphertext to reduce the bandwidth overhead, but it does not affect security. In our implementation, RS is 2^{40}, yielding a ciphertext length of 5 bytes. As a result, the ciphertext is no longer decryptable; this is not a problem because BlindBox always decrypts the traffic from the primary SSL stream.

现在，为了检测关键字r和令牌t的加密之间的匹配，MB使用salt和它对AESk(r)的知识计算AESk(r)(salt) mod RS，然后测试AESk(t)(salt) mod RS的相等性。

Now, to detect a match between a keyword r and an encryption of a token t, MB computes $\mathsf{AES}_{\mathsf{AES}_k(r)}(\mathsf{salt}) \bmod \mathsf{RS}$ using salt and its knowledge of $\mathsf{AES}_k(r)$, and then tests for equality with $\mathsf{AES}_{\mathsf{AES}_k(t)}(\mathsf{salt}) \bmod \mathsf{RS}$.

Hence, naïvely, MB performs a match test for every token t and rule r, which results in a performance per token linear in the number of rules; this is too slow. To address this slowdown, our detection algorithm below makes this cost logarithmic in the number of rules, the same as for vanilla inspection of unencrypted traffic. This results in a significant performance improvement: for example, for a ruleset with 10000 keywords to match, a logarithmic lookup is four orders of magnitude faster than a linear scan.

3.2 BlindBox Detect Protocol

现在讨论我们的检测算法如何实现对数查找时间，解决安全性和性能之间的紧张关系

We now discuss how our detection algorithm achieves logarithmic lookup times, resolving the tension between security and performance. For simplicity of notation, denote $\mathsf{Enc}_k(\mathsf{salt}, t) = \mathsf{AES}_{\mathsf{AES}_k(t)}(\mathsf{salt})$.

不能通过遍历随机salt或者使用少量salt的方法减少计算

The first idea is to precompute the values $\mathsf{Enc}_k(\mathsf{salt}, r)$ for every rule r and for every possible salt. Recall that MB can compute $\mathsf{Enc}_k(\mathsf{salt}, r)$ based only on salt and its knowledge of $\mathsf{AES}_k(r)$, and MB does not need to know k. Then, MB can arrange these values in a search tree. Next, for each encrypted token t in the traffic stream, MB simply looks up $\mathsf{Enc}_k(\mathsf{salt}, t)$ in the tree and checks if an equal value exists. However, the problem is that enumerating all possible salts for each keyword r is infeasible. Hence, it would be desirable to use only a few salts, but this strategy affects security: an attacker at MB can see which token in the traffic equals which other token in the traffic whenever the salt is reused for the same token. To maintain the desired security, every encryption of a token t must contain a different salt (although the salts can repeat across different tokens).

通过一个随机数的累加。实现一系列随机数。司机数不在发送，而是和消息一起加密。MB不知道随机数的累加数目

To use only a few salts and maintain security at the same time, the idea is for the sender to *generate salts based on the token value* and *no longer send the salt in the clear along with every encrypted token*. Concretely, the sender keeps a *counter table* mapping each token encrypted so far to how many times it appeared in the stream so far. Before sending encrypted tokens, the sender sends one initial salt, salt_0, and MB records it. Then, the sender no longer sends salts; concretely, for each token t, the sender sends $\mathsf{Enc}_k(\mathsf{salt}, t)$ but not salt. When encrypting a token t, the sender checks the number of times it was encrypted so far in the counter table, say ct_t, which could be zero. It then encrypts this token with the salt $(\mathsf{salt}_0 + \mathsf{ct}_t)$ by computing $\mathsf{Enc}_k(\mathsf{salt}_0 + \mathsf{ct}_t, t)$. Note that this provides the desired security because no two equal tokens will have the same salt.

考虑发送者需要加密令牌A、B、A。发送者计算并传输:salt0、Enck(salt0，A)、Enck(salt0，B)和Enck(salt0+1，A)。不为每个密文发送salt既减少了带宽，又是安全所必需的:如果发送者发送了salt，MB就可以知道第一个和第二个令牌具有相同的salt，因此它们是不相等的。不同令牌的salt不同，计数也是从头开始的

For example, consider the sender needs to encrypt the tokens A, B, A. The sender computes and transmits: salt_0, $\mathsf{Enc}_k(\mathsf{salt}_0, A)$, $\mathsf{Enc}_k(\mathsf{salt}_0, B)$, and $\mathsf{Enc}_k(\mathsf{salt}_0 + 1, A)$. Not sending a salt for each ciphertext both reduces bandwidth and is required for security: if the sender had sent salts, MB could tell that the first and second tokens have the same salt, hence they are not equal.

To prevent the counter table from growing too large, the sender resets it every P bytes sent. When the sender resets this table, the sender sets $\mathsf{salt}_0 \leftarrow \mathsf{salt}_0 + \max_t \mathsf{ct}_t + 1$ and announces the new salt_0 to MB.

For detection, MB creates a table mapping each keyword r to a counter ct^*_r indicating the number of times this keyword r appeared so far in the traffic stream. MB also creates a search tree containing the encryption of each rule r with a salt computed from ct^*_r: $\mathsf{Enc}_k(\mathsf{salt}_0 + \mathsf{ct}^*_r, r)$. Whenever there is a match to r, MB increments ct^*_r, computes and inserts the new encryption $\mathsf{Enc}_k(\mathsf{salt}_0 + \mathsf{ct}^*_r, r)$ into the tree, and deletes the old value. We now summarize the detection algorithm.

为了防止计数器表变得太大，发送方每发送一个P字节就重置它。当发送方重置此表时，发送方设置salt0← salt0+ maxt ctt+ 1，并告诉MB新的salt0。为了进行检测，MB创建一个表，将每个关键字r映射到一个计数器ct，该计数器指示到目前为止该关键字r在流量流中出现的次数。MB还创建了一个搜索树，该树包含每个规则r的加密，其中salt是根据CT*r:Enck(salt0+CTr，r)计算的。每当与r匹配时，MB增加CTr，计算新的加密Enck(salt 0+CTr，r)并将其插入到树中，并删除旧的值。我们现在总结一下检测算法。

MB处的状态由每个规则r的计数器CT∫r和每个规则r的Enck(salt 0+CT∫r，r)组成的快速搜索树组成。

BlindBox Detect: The state at MB consists of the counters ct_r^* for each rule r and a fast search tree made of $\mathsf{Enc}_k(\mathsf{salt}_0 + \mathsf{ct}_r^*, r)$ for each rule r.

1: For each encrypted token $\mathsf{Enc}_k(\mathsf{salt}, t)$ in a packet:

1.1: If $\mathsf{Enc}_k(\mathsf{salt}, t)$ is in the search tree:

1.1.1: There is a match, so take the corresponding action for this match.

1.1.2: Delete the node in tree corresponding to r and insert $\mathsf{Enc}_k(\mathsf{salt}_0 + \mathsf{ct}_r^* + 1, t)$

1.1.3: Set $\mathsf{ct}_r^* \leftarrow \mathsf{ct}_r^* + 1$

使用这种策略，对于每个标记t，MB执行一个简单的树查找，这是规则数量的对数。

With this strategy, for every token t, MB performs a simple tree lookup, which is logarithmic in the number of rules. Other tree operations, such as deletion and insertion, happen rarely: when a malicious keyword matches in the traffic. These operations are also logarithmic in the number of rules.

3.3 Rule Preparation

上面的检测协议假设每次建立新的连接(具有新的密钥k)时，MB为每个关键字r获得AESk(r)。但是MB如何获得这些值呢？这里的挑战是，没有一方，MB或者S/R，看起来适合计算AESk(r): MB知道R，但是不允许学习k；s和R知道k，但不允许学习规则R(如2.2.2所述)。

The detection protocol above assumes that MB obtains $\mathsf{AES}_k(r)$ for every keyword r, every time a new connection (having a new key k) is setup. But how can MB obtain these values? The challenge here is that no party, MB or S/R, seems fit to compute $\mathsf{AES}_k(r)$: MB knows r, but it is not allowed to learn k; S and R know k, but are not allowed to learn the rule r (as discussed in §2.2.2).

我们提供了一种称为模糊规则加密的技术来解决这个问题。其思想是发送者向中间盒提供一个函数AES的"混淆"，密钥k硬编码在其中。这种混淆隐藏了密钥k。中间框在规则r上运行这种混淆，并获得AESk(r)，而不学习k。我们用ObfAESk表示这种混淆的函数。由于实际的混淆不存在，我们用Yao乱码电路来实现它，我们将在下面详细说明。乱码电路，MB不能直接插r作为ObfAESk()的输入；相反，它必须从端点获得一个与ObfAESk一起工作的r编码。对于这个任务，发送者使用一个称为不经意传输的协议，它不向端点透露r。此外，MB需要为每个关键字r获得一个新鲜的、重新加密的乱码电路ObFaesk()；原因是，如果宏块为同一个乱码电路接收一个以上的编码，那么乱码电路的安全性就不成立。一个问题是，MB可能试图在它选择的规则上运行模糊加密功能，而不是来自RG的规则。为了防止这种攻击，来自RG的规则必须由RG签名，混淆(乱码)函数必须在加密之前检查输入规则上是否有有效的签名。如果签名无效，它将输出null。现在让我们更详细地介绍构建模块和我们的协议。

Intuition. We provide a technique, called *obfuscated rule encryption*, to address this problem. The idea is that the sender provides to the middlebox an "obfuscation" of the function AES with the key k hardcoded in it. This obfuscation hides the key k. The middlebox runs this obfuscation on the rule r and obtains $\mathsf{AES}_k(r)$, without learning k. We denote this obfuscated function by ObfAES_k.

Since practical obfuscation does not exist, we implement it with Yao garbled circuits [50, 33], on which we elaborate below. With garbled circuits, MB cannot directly plug in r as input to $\mathsf{ObfAES}_k()$; instead, it must obtain from the endpoints an encoding of r that works with ObfAES_k. For this task, the sender uses a protocol called oblivious transfer [34, 14], which does not reveal r to the endpoints. Moreover, MB needs to obtain a fresh, re-encrypted garbled circuit $\mathsf{ObfAES}_k()$ for every keyword r; the reason is that the security of garbled circuits does not hold if MB receives more than one encoding for the same garbled circuit.

A problem is that MB might attempt to run the obfuscated encryption function on rules of its choice, as opposed to rules from RG. To prevent this attack, rules from RG must be signed by RG and the obfuscated (garbled) function must check that there is a valid signature on the input rule before encrypting it. If the signature is not valid, it outputs null.

Let us now present the building blocks and our protocol in more detail.

Yao garbling scheme [50, 33]. At a high level, a garbled circuit scheme, first introduced by Yao, consists of two algorithms Garble and Eval. Garble takes as input a function F with n bits of input and outputs a garbled function ObfF and

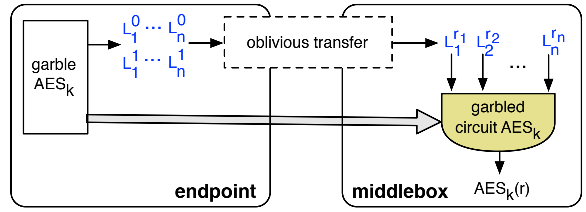


Figure 2: Rule preparation. The endpoint has a key k and the middlebox has a keyword r.

在更高的层次上，由姚首先提出的乱码电路方案由两个算法组成:加布尔算法和Eval算法。加布尔将具有n位输入的函数F作为输入，并输出加有n对标签的加错函数ObfF(L01，L11)。。。(L0 n，L1 n)，f的每一个输入位对应一对标签。考虑n个输入位中的任意x个输入位的第一位。ObfF具有ObfF(Lx1 1,。。，Lxn n) = F(x)。基本上，如果给定对应于x的每个位的标签，ObfF产生与F相同的输出。只要对手只收到一个输入x的标签Lx1。。，Lxn不会泄漏任何关于F和x的信息。

n pairs of labels $(L_1^0, L_1^1), \dots, (L_n^0, L_n^1)$, one pair for every input bit of F. Consider any input x of n bits with x_i being its i-th bit. ObfF has the property that $\mathsf{ObfF}(L_1^{x_1}, \dots, L_n^{x_n}) = F(x)$. Basically, ObfF produces the same output as F if given the labels corresponding to each bit of x. Regarding security, ObfF and $L_1^{x_1}, \dots, L_n^{x_n}$ do not leak anything about F and x beyond $F(x)$, as long as an adversary receives labels for only one input x.

2选1不经意传输[34，14]。假设甲方有两个值，L0和L1，乙方有一个bit位b。假设乙方想从甲方获得第B个标签，Lb，但乙方不想告诉乙方给甲方。此外，甲方不想让乙方学习另一个标签L1-B。因此，乙方不能将B发送给甲方，甲方也不能将两个标签都发送给乙方。不经意传输(ot)正好实现了这一点:乙方可以获得Lb而不学习L1-B，甲方不学习B。

1-out-of-2 oblivious transfer (OT) [34, 14]. Consider that a party A has two values, L^0 and L^1, and party B has a bit b. Consider that B wants to obtain the b-th label from A, L^b, but B does not want to tell b to A. Also, A does not want B to learn the other label L^{1-b}. Hence, B cannot send b to A and A cannot send both labels to B. Oblivious transfer (OT) enables exactly this: B can obtain L^b without learning L^{1-b} and A does not learn b.

规则准备。图2示出了一个关键字r的规则准备过程。一个端点可能是恶意的，并试图不正确地执行篡改以避免检测。为了防止这种攻击，两个端点都必须准备乱码电路，并将其发送到MB，以检查它们是否产生了相同的结果。如果乱码电路和标签匹配，宏块保证它们是正确的，因为至少有一个端点是诚实的(如第节所述2.2.2).要启用此检查，端点必须使用从植入krand的伪随机发生器获得的相同随机性(在第节中讨论2.3).

Rule preparation. Fig. 2 illustrates the rule preparation process for one keyword r. One endpoint could be malicious and attempt to perform garbling incorrectly to eschew detection. To prevent such an attack, both endpoints have to prepare the garbled circuit and send it to MB to check that they produced the same result. If the garbled circuits and labels match, MB is assured that they are correct because at least one endpoint is honest (as discussed in Sec. 2.2.2). To enable this check, the endpoints must use the same randomness obtained from a pseudorandom generator seeded with k_{rand} (discussed in Sec. 2.3).

Rule preparation:

1: MB tells S and R the number of rules N it has.

2: For each rule $1, \dots, N$, do:

2.1: S and R: Garble the following function F.

混淆

F on input $[x, \mathsf{sig}(x)]$ checks if $\mathsf{sig}(x)$ is a valid signature on x using RG's public key. If yes, it encrypts x with AES_k and outputs $\mathsf{AES}_k(x)$; else, it outputs $\perp$.

f在输入[x，sig(x)]时，使用RG的公钥检查sig(x)是否是x上的有效签名。如果是，用AESkand加密x，输出AES kx；否则，它输出⊥

In the garbling process, use randomness based on k_{rand}. Send the resulting garbled circuit and labels to MB.

在乱码过程中，使用基于随机数的随机性。将产生的乱码电路和标签发送到MB。

2.2: MB: Verify that the garbled circuits from S and R are the same, and let ObfAES_k be this garbled circuit. Let r and $\mathsf{sig}(r)$ be the current rule and its signature. Run oblivious transfer with each of S and R to obtain the labels for r and $\mathsf{sig}(r)$. Verify that the labels

验证S和R的混淆电路是相同的，让ObfAESkbe成为这个混淆电路。设r和sig(r)为当前规则及其签名。对S和R中的每一个运行不经意传输，以获得R和sig(r)的标签。验证S和R的标签是否相同，并用Lr1-1。。。Lrn-n 表示.

> from S and R are the same, and denote them $L_1^{r_1}, \ldots, L_n^{r_n}$.
>
> 2.3: MB: Evaluate ObfAES_k on the labels $L_1^{r_1}, \ldots, L_n^{r_n}$ to obtain $\mathsf{AES}_k(r)$.
>
> 评估标签Lr1-1。。。，Lrn上的ObfAESk,以获得AESk(r)
>
> 。

Rule preparation is the main performance overhead of BlindBox HTTPS. This overhead comes from the oblivious transfer and from the generation, transmission and evaluation of the garbled circuit, all of which are executed once for every rule. We evaluate this overhead in §7.

规则准备是盲箱HTTPS的主要性能开销。这种开销来自不经意的传输以及乱码电路的生成、传输和评估，所有这些都是针对每个规则执行一次。我们在7中评估了这个开销。

3.4 Validate Tokens 验证令牌

As shown in Fig. 1, the validate tokens procedure runs at the receiver. This procedure takes the decrypted traffic from SSL and runs the same tokenize and encrypt modules as the sender executes on the traffic. The result is a set of encrypted tokens and it checks that these are the same as the encrypted tokens forwarded by MB. If not, there is a chance that the other endpoint is malicious and flags the misbehavior.

如图1所示，验证令牌过程在接收器上运行。此过程从SSL接收解密的流量，并运行与发送方对流量执行的令牌化和加密模块相同功能的模块。结果是一组加密的令牌，它检查这些令牌是否与MB转发的加密令牌相同。如果没有，则有可能另一个端点是恶意的，并标记了不良行为。

安全保证

3.5 Security Guarantees

We proved our protocol secure with respect to our exact match privacy model; the proofs can be found in our extended paper [43]. We formalized the property that DPIEnc hides the traffic content from MB using an indistinguishability-based security definition. Informally, MB is given encryptions of a sequence of tokens $t'_1, \ldots, t'_n$ and keywords r_1, $\ldots, r_m$. Then, MB can choose two tokens t_0 and t_1 which do not match any of the keywords. Next, MB is given a ciphertext $c = \mathsf{Enc}_k(\mathsf{salt}, t_b)$ for some bit b and salt generated according to the BlindBox Detect protocol. The security property says that no polynomial-time attacker at MB can guess the value of b with chance better than half. In other words, MB cannot tell if t_0 or t_1 is encrypted in c. We can see why this property holds intuitively: if MB does not have $\mathsf{AES}_k(t_b)$, this value is indistinguishable from a random value by the pseudorandom permutation property of AES. Hence, $\mathsf{Enc}_k(\cdot, t_b)$ maps each salt to a random value, and there are no repetitions among these random values due to the choice of salt in BlindBox Detect. Thus, the distributions of ciphertexts for each value of b are essentially the same, and thus indistinguishable.

我们根据精确匹配隐私模型证明了我们的协议是安全的；证明可以在我们的扩展论文[43]中找到。我们形式化了这样一个属性，即DPIEnc隐藏了基于流量内容的安全定义。非正式地，给MB一系列加密的令牌t1。。。，tn和关键字r1，.。。rm。然后，MB可以选择不匹配任何关键字的两个令牌t0和t11。接下来，为某个位b给MB一个密文c = Enck(salt，tb)，salt根据BlindBox Detect协议生成。安全属性表示，MB的多项式时间攻击者无法以概率大于一半猜测b的值。换句话说，MB无法分辨t0或t1在c中是否加密，我们可以直观地看到为什么这个属性成立:如果MB没有AESk(tb)，那么这个值就无法通过AES的伪随机置换属性与随机值区分开来。因此，Enck(，tb)将每个salt映射到一个随机值，由于在盲盒检测中选择了salt，这些随机值之间没有重复。因此，每个b值的密文分布本质上是相同的，因此无法区分。

As part of our privacy model, BlindBox reveals a small amount of information to make detection faster: BlindBox does not hide the number of tokens in a packet. Also, if a suspicious keyword matches at an offset in the traffic stream, MB learns this offset. Hence, BlindBox necessarily weakens the privacy guarantees of SSL to allow efficient detection. (Note that BlindBox preserves the authenticity property of SSL.)

BlindBox会暴露少量的信息，以加快检测速度:BlindBox不会隐藏数据包中的令牌数量。此外，如果一个可疑的关键字匹配流量流中的偏移量，MB就会学习该偏移量。因此，盲箱必然会削弱SSL的隐私保证，从而允许有效的检测。(注意，BlindBox保留了SSL的真实性属性。)

有限IDS

4. PROTOCOL II: LIMITED IDS

This protocol supports a limited form of an IDS. Namely, it allows a rule to contain: (1) multiple keywords to be matched in the traffic, and (2) absolute and relative offset information within the packet. In our industrial dataset, the average *rule* contained three *keywords*; a rule is "matched" if all keywords are found within a flow.

该协议支持有限形式的入侵检测系统。也就是说，它允许规则包含:(1)在流量中匹配多个关键字(2)数据包中的绝对和相对偏移信息。在我们的工业数据集中，平均规则包含三个关键词；如果在一个流中找到所有关键字，则规则是“匹配的”。

This protocol supports most of the functionality in the rule language of Snort [47]. A few functional commands are not supported, the most notable being *pcre*, which allows arbitrary regular expressions to be run over the payload. This command is supported by Protocol III.

该协议支持Snort规则语言的大部分功能[47]。不支持一些函数式命令，最著名的是pcre，它允许在负载上运行任意正则表达式。该命令得到第三议定书的支持。

For example, consider rule number 2003296 from the Snort Emerging Threats ruleset:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS
                        -> $HOME_NET 1025:5000 (
  flow: established,from_server;
  content: "Server|3a| nginx/0.";
  offset: 17; depth: 19;
  content: "Content-Type|3a| text/html";
  content: "|3a|80|3b|255.255.255.255"; )
```

This rule is triggered if the flow is from the server, it contains the keyword "Server|3a| nginx/0." at an offset between 17 and 19, and it also contains the keyword "Content-Type|3a| text/html" and "|3a|80|3b|255.255.255.255". The symbol "|" denotes binary data.

如果流来自服务器，则触发此规则，它包含关键字“服务器|3a| nginx/0”偏移量在17和19之间，它还包含关键字“ContentType|3a| text/html”和“|3a|80|3b|255.255.255.255”。符号“|”表示二进制数据。

Protocol II builds on Protocol I in a straightforward way. The sender processes the stream the same as in Protocol I (including the encryption) with one exception: if the delimiter-based tokenization is used, the sender attaches to each encrypted token the offset in the stream where it appeared. In the window-based tokenization, the offset information need not be attached to each encrypted token because a token is generated at each offset and hence the offset can be deduced.

第二号议定以直接的方式建立在一号议定的基础上。发送方处理流的方式与第一协议(包括加密)相同，但有一个例外:如果使用基于令牌的加密，发送方会将流中出现的偏移量附加到每个加密的令牌上。在基于窗口的令牌化中，不需要将偏移量信息附加到每个加密的令牌，因为令牌是在每个偏移量生成的，因此可以推导出偏移量。

Detection happens similarly to before. For each encrypted token, MB checks if it appears in the rule tree. If so, it checks whether the offset of this encrypted token satisfies any range that might have been specified in the relevant rule. If all the fields of the relevant rule are satisfied, MB takes the action indicated by the rule.

检测的发生与以前类似。对于每个加密的令牌，MB检查它是否在规则树中出现过。如果是这样，它将检查这个加密令牌的偏移量是否满足相关规则中指定的任何范围。如果相关规则的所有字段都得到满足，则MB会采取该规则指示的操作。

Security Guarantee. The security guarantee is the same as in Protocol I: for each rule keyword, the middlebox learns if the keyword appears in the traffic and at what offset, but it learns nothing else about the parts of the traffic that do not match keywords. Note that the security guarantee is defined per keyword and not per rule: MB learns when a keyword matches even if the entire rule does not match.

可能导致隐私的完整身份证明

5. PROTOCOL III: FULL IDS WITH PROBABLE CAUSE PRIVACY

This section enables full IDS functionality, including regexp and scripts, based on our probable cause privacy model. If a keyword from a rule (a suspicious keyword) matches a stream of traffic, MB should be able to decrypt the traffic. This enables the middlebox to then run regexp (e.g., the "pcre" field in Snort) or scripts from Bro [36] on the decrypted data. However, if such a suspicious keyword does not match the packet stream, the middlebox cannot decrypt the traffic (due to cryptographic guarantees), and the security guarantee is the same as in Protocol II.

Protocol insight. The idea is to somehow embed the SSL key k_{SSL} into the encrypted tokens, such that, if MB has

基于我们的可能原因隐私模型，这一部分启用了完整的IDS功能，包括正则表达式和脚本。如果规则中的一个关键字(可疑关键字)与一个流量流匹配，MB应该能够解密该流量。这使得中间盒能够对解密的数据运行正则表达式(例如Snort中的“pcre”字段)或Bro中的脚本。但是，如果这样一个可疑的关键字与包流不匹配，则中间盒无法解密流量(由于加密保证)，并且安全保证与协议二中的相同。

这个想法是以某种方式将SSL密钥Kssl嵌入到加密的令牌中，这样，如果MB有一个与流量中的令牌t匹配的规则关键字r，MB应该能够计算Kssl。为了实现这个目标，我们用Enck(salt，t) ⊕ kSSL替换加密的令牌Enck(salt，t),其中⊕是按位XOR。如果r = t，MB有AESk(t)，可以构造Enck(salt，t)，然后通过一个异或运算得到kSsL。问题是，这将检测速度减慢到规则的线性扫描，因为计算异或的需要不再允许在规则树中对加密令牌进行简单的树查找(在第节中描述)。3.2).

a rule keyword r that matches a token t in the traffic, MB should be able to compute k_{SSL}. To achieve this goal, we replace the encrypted token $\mathsf{Enc}_k(\mathsf{salt}, t)$ with $\mathsf{Enc}_k(\mathsf{salt}, t) \oplus k_{\mathsf{SSL}}$, where $\oplus$ is bitwise XOR. If $r = t$, MB has $\mathsf{AES}_k(t)$ and can construct $\mathsf{Enc}_k(\mathsf{salt}, t)$, and then obtain k_{SSL} through a XOR operation. The problem is that this slows down detection to a linear scan of the rules because the need to compute the XOR no longer allows a simple tree lookup of an encrypted token into the rule tree (described in Sec. 3.2).

为了保持检测的效率，我们保留了与DPIEnc中相同的加密令牌，并将其用于检测，但另外创建了一个加密令牌，其中嵌入了如上所述的密钥。现在，令牌的加密是一对[c1= Enck(salt，t)，C2 = Enck*(salt，t)⊕ kSSL)，其中Enck*(salt，t)= aesk(t)(salt+1)，salt是在盲箱检测(3.2)中生成的。请注意，Enck*与salt不同于任何c1加密t中的salt是至关重要的，因为否则攻击者可以计算c1 ⊕C2并获得kSSL。为了在盲箱检测(Blind Box Detect)中对同一令牌的不同出现执行这一要求，发送方现在将salt增加2因为:它对c1使用偶数salt(对树中的规则使用MB也是如此)，而对c2使用奇数salt。MB像以前一样使用C1来执行检测。如果MB使用BlindBox Detect检测到与规则r的匹配，则MB使用AESk(r)计算Enck*(salt，r),并计算Enck*(salt，r) ⊕ c2，从而产生kSSL。我们在扩展的文献[43]中证明了该协议的安全性。

Protocol. To maintain the efficiency of the detection, we retain the same encrypted token as in DPIEnc and use it for detection, but additionally create an encrypted token that has the key embedded in as above. Now, the encryption of a token t becomes a pair $[c_1 = \mathsf{Enc}_k(\mathsf{salt}, t), c_2 = \mathsf{Enc}^*_k(\mathsf{salt}, t) \oplus k_{\mathsf{SSL}}]$, where $\mathsf{Enc}^*_k(\mathsf{salt}, t) = \mathsf{AES}_{\mathsf{AES}_k(t)}(\mathsf{salt} + 1)$ and the salt is generated as in BlindBox Detect (§3.2). Note that it is crucial that the salt in Enc^*_k differs from the salt in any c_1 encryption of t because otherwise an attacker can compute $c_1 \oplus c_2$ and obtain k_{SSL}. To enforce this requirement across different occurrences of the same token in BlindBox Detect, the sender now *increments the salt by two*: it uses an even salt for c_1 (and so does MB for the rules in the tree), while it uses an odd salt for c_2. MB uses c_1 to perform the detection as before. If MB detects a match with rule r using BlindBox Detect, MB computes $\mathsf{Enc}^*_k(\mathsf{salt}, r)$ using $\mathsf{AES}_k(r)$, and computes $\mathsf{Enc}^*_k(\mathsf{salt}, r) \oplus c_2$, which yields k_{SSL}. We prove the security of this protocol in our extended paper [43].

6. SYSTEM IMPLEMENTATION

We implemented two separate libraries for BlindBox: a client/server library for transmission called BlindBox HTTPS, and a Click-based [32] middlebox.

盲盒HTTPS协议是在一个C库中实现的。当一个客户端打开一个连接时，我们的协议实际上打开了三个独立的套接字:一个通过普通的SSL，一个传输“可搜索的”加密令牌，一个监听路径上的中间盒是否请求乱码电路。正常的SSL通道运行在一个修改后的GnuTLS库之上，该库允许我们提取协议三下的会话密钥。发送时，端点首先发送加密令牌，然后通过正常的HTTPS发送流量。如果路径上有一个中间盒，端点会使用JustGarble和OT扩展库生成乱码电路。

BlindBox library. The BlindBox HTTPS protocol is implemented in a C library. When a client opens a connection, our protocol actually opens three separate sockets: one over normal SSL, one to transmit the "searchable" encrypted tokens, and one to listen if a middlebox on path requests garbled circuits. The normal SSL channel runs on top of a modified GnuTLS [12] library which allows us to extract the session key under Protocol III. On send, the endpoint first sends the encrypted tokens, and then sends the traffic over normal HTTPS. If there is a middlebox on the path, the endpoints generate garbled circuits using JustGarble [16] in combination with the OT Extension library [5].

The middlebox. We implemented the middlebox in multithreaded Click [32] over DPDK [2]; in our implementation, half of the threads perform detection over the data stream ("detection" threads), and half perform obfuscated rule encryption exchanges with clients ("garble" threads). When a new connection opens, a detection thread signals to a garble thread and the garble thread opens an obfuscated rule encryption channel with the endpoints. Once the garble thread has evaluated all circuits received from the clients and obtained the encrypted rules, it constructs the search tree. The detection thread then runs the detection based on the search tree, and allows data packets in the SSL channel to proceed if no attack has been detected.

我们在DPDK 上的多线程点击中实现了中间件；在我们的实现中，一半的线程对数据流执行检测(“检测”线程)，另一半执行与客户端的模糊规则加密交换(“乱码”线程)。当一个新的连接打开时，一个检测线程向一个乱码线程发出信号，乱码线程打开一个带有端点的模糊规则加密通道。一旦垃圾线程已经评估了从客户端接收的所有电路并获得了加密的规则，它就构建了搜索树。然后，检测线程基于搜索树运行检测，如果没有检测到攻击，则允许SSL通道中的数据包继续运行。

当一个检测线程匹配一个规则时，在协议一和协议二下，中间盒阻塞连接。在协议三下，它计算解密密钥(由于匹配，这是可能的)，并将加密的流量和密钥转发给解密元素。这个元素被实现为开源ssldump 工具的包装器。解密后的流量可以转发到任何其他系统(Snort、Bro等)。)进行更复杂的处理。我们在SSL终端设备的基础上对此进行了建模，如今，在将流量传递给其他监控或DPI设备之前，SSL终端设备会对流量进行中间人处理。

When a detection thread matches a rule, under Protocols I and II, the middlebox blocks the connection. Under Protocol III, it computes the decryption key (which is possible due to a match), and it forwards the encrypted traffic and the key to a decryption element. This element is implemented as a wrapper around the open-source ssldump [10] tool. The decrypted traffic can then be forwarded to any other system (Snort, Bro, etc.) for more complex processing. We modeled this after SSL termination devices [18], which today man-in-the-middle traffic before passing it on to some other monitoring or DPI device.

7. EVALUATION

When evaluating BlindBox, we aimed to answer two questions. First, can BlindBox support the functionality of our target applications – data exfiltration (document watermarking), parental filtering, and HTTP intrusion detection? Second, what are the performance overheads of BlindBox at both the endpoints and the middlebox?

7.1 Functionality Evaluation

To evaluate the functionality supported by BlindBox, we answer a set of sub-questions.

Can BlindBox implement the functionality required for each target system? Table 1 shows what fraction of "rules" BlindBox can implement using each of Protocols I, II, and III. We evaluate this fraction using public datasets for document watermarking [45], parental filtering [13], and IDS rules (from the Snort community [9] and Emerging Threats [3]). In addition, we evaluate on two industrial datasets from Lastline and McAfee Stonesoft to which we had (partial) access.

Document watermarking and parental filtering can be completely supported using Protocol I because each system relies only on the detection of a single keyword to trigger an alarm. However, Protocol I can support only between 1.6-5% of the policies required by the more general HTTP IDS applications (the two public Snort datasets, as well as the datasets from McAfee Stonesoft and Lastline). This limitation is due to the fact that most IDS policies require detection of multiple keywords or regular expressions.

Protocol II, by supporting multiple exact match keywords, extends support to 29-67% of policies for the HTTP IDS applications. Protocol III supports all applications including regular expressions and scripting, by enabling decryption when there is a probable cause to do so.

Does BlindBox fail to detect any attacks/policy violations that these standard implementations would detect? The answer depends on which tokenization technique one uses out of the two techniques we described in §3: window-based and delimiter-based tokenization. The window-based tokenization does not affect the detection accuracy of the rules because it creates a token at every offset. The delimiter-based tokenization relies on the assumption that, in IDSes, most rules occur on the boundary of non-alphanumeric characters, and thus does not transmit all possible tokens – only those required to detect rules which occur between such "delimiters". To test if this tokenization misses attacks, we ran

Dataset	I.	II.	III.
Document watermarking [45]	100%	100%	100%
Parental filtering [13]	100%	100%	100%
Snort Community (HTTP)	3%	67%	100%
Snort Emerging Threats (HTTP)	1.6%	42%	100%
McAfee Stonesoft IDS	5%	40%	100%
Lastline	0%	29.1%	100%

Table 1: Fraction of attack rules in public and industrial rule sets addressable with Protocols I, II, and III.

BlindBox over the ICTF2010 [49] network trace, and used as rules the Snort Emerging Threats ruleset from which we removed the rules with regular expressions. The ICTF trace was generated during a college "capture the flag" contest during which students attempted to hack different servers to win the competition, so it contains a large number of attacks. We detected 97.1% of the attack keywords and 99% of the attack rules that would have been detected with Snort. (Recall that an attack rule may consist of multiple keywords.)

7.2 Performance Evaluation

We now investigate BlindBox's performance overheads at both the client and the network. For all experiments, the client software uses Protocol II, which has higher overhead than Protocol I. We do not evaluate Protocol III directly; the differences we would expect from Protocol III relative to II would include a secondary middlebox to perform regular expression processing, and an increase in bandwidth due to the key being embedded in each encrypted token.

Our prototype of the client software runs on two servers with 2.60 GHz processors connected by a 10GbE link. The machines are multicore, but we used only one thread per client. The CPU supports AES-NI instructions and thus the encryption times for both SSL and BlindBox reflect this hardware support. Since typical clients are not running in the same rack over a 10GbE links, in some experiments we reduced throughput to 20Mbps (typical of a broadband home link) and increased latency to 10ms RTT. Our prototype middlebox runs with four 2.6GHz Xeon E5-2650 cores and 128 GB RAM; the network hardware is a single 10GbE Intel 82599 compatible network card. All of our experiments were performed on this testbed. For microbenchmarks (as in Table 2), we measured time to complete a loop of 10,000 iterations and took an average. For flow completion benchmarks we took an average of five runs.

To summarize our performance results, BlindBox is practical for long-lived connections: the throughput of encryption and detection are competitive with rates of current (unencrypted) deployments. Additionally, BlindBox is 3 to 6 orders of magnitude faster than relevant implementations using existing cryptography; these solutions, by themselves, are incomplete in addition to being slow. The primary overhead of BlindBox is setting up a connection, due to the obfuscated rule encryption. This cost is small for small rulesets, but can take as long as 1.5 minutes for rulesets with thousands of rules; hence, BlindBox is not yet practical for systems with thousands of rules and short-lived connections that need to run setup frequently. We now elaborate on all these points.

总结我们的性能结果，BlindBox适用于长寿命连接:加密和检测的吞吐量与当前(未加密)部署的速率相当。此外，盲箱是使用现有加密技术的3到6个有序的大规模加密工具。这些解决方案本身不仅速度慢，而且不完整。由于混淆的规则加密，盲箱的主要开销是建立连接。这个代价对于小规则集来说很小，但是对于有上千个规则的规则集来说可能需要1.5分钟之久；因此，对于具有成千上万条规则和需要频繁运行安装程序的短期连接的系统来说，盲箱还不实用。我们现在详细阐述所有这些要点。

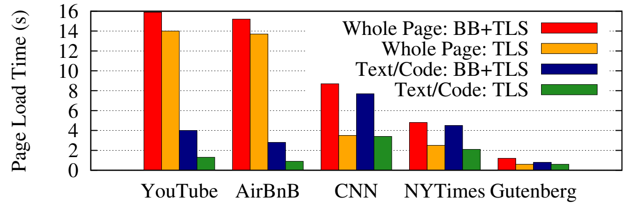


Figure 3: Download time for TLS and BlindBox (BB) + TLS at 20Mbps$\times$10ms.

7.2.1 *Strawmen*

BlindBox is the only system we know of to enable DPI over encrypted data. Nevertheless, to understand its performance, we compare it to standard SSL as well as two strawmen, which we now describe.

A searchable encryption scheme due to Song et al. [46]: This scheme does not enable obfuscated rule encryption or probable cause decryption, but can implement encryption and detection as in Protocols I and II (but not Protocol III). We used the implementation of Song et al. from [37], but replaced the use of SHA512 with the AES-NI instruction in a secure way, to speed up this scheme.

Generic functional encryption (FE) [23, 26]: Such schemes, if enhanced with our obfuscated rule encryption technique, can in theory perform Protocols I, II, and III. However, such encryption schemes are prohibitively expensive to be run and evaluated. For example, one such scheme [26] nests fully homomorphic encryption twice, resulting in an overhead of at least 10 orders of magnitude. Instead, we chose and implemented a simple and specialized functional encryption scheme due to Katz et al. [30]. The performance of this scheme is a generous lower bound on the performance of the generic protocols (the Katz et al. scheme does not support Protocol III because it can compute only inner product).

7.2.2 *Client Performance*

How long does it take to encrypt a token? Table 2 provides micro-benchmarks for encryption, detection, and setup using BlindBox, HTTPS, and our strawmen. With HTTPS (using GnuTLS), encryption of one 128-bit block took on average 13ns, and 3μs per 1400 byte packet. BlindBox increases these values to 69ns and 90μs respectively. These figures include the time to perform HTTPS transmission in the primary channel, as well as the overheads from BlindBox: the tokenization process itself (deciding which substrings to tokenize) as well as the encryption process (encrypting and then hashing each token with AES). The searchable strawman performs encryption of a single token on average 2.7μs and 257μs for an entire packet; the primary overhead relative to BlindBox here is multiple calls to `/dev/urandom` because the scheme requires random salts for every token. With fixed or pre-chosen salts, we would expect the searchable strawman to have comparable encryption times to BlindBox. As we discuss, the detection times for this strawman are slower. The FE strawman takes six orders of magnitude longer than BlindBox and is even further impractical: a client using this scheme could transmit at most one packet every 15 seconds.

		Vanilla HTTPS	FE Strawman	Searchable Strawman	BlindBox HTTPS
Client	*Encrypt* (128 bits)	13ns	70ms	2.7μs	69ns
	Encrypt (1500 bytes)	3μs	15s	257μs	90μs
	Setup (1 Keyword)	73ms	N/A	N/A	588 ms
	Setup (3K Rules)	73ms	N/A	N/A	97 s
MB	*Detection:*				
	1 Rule, 1 Token	NP	170ms	1.9μs	20ns
	1 Rule, 1 Packet	NP	36s	52μs	5μs
	3K Rules, 1 Token	NP	8.3 minutes	5.6ms	137ns
	3K Rules, 1 Packet	NP	5.7 days	157ms	33μs

Table 2: Connection and detection micro-benchmarks comparing Vanilla HTTPS, the functional encryption (FE) strawman, the searchable strawman, and BlindBox HTTPS. NP stands for not possible. The average rule includes three keywords.

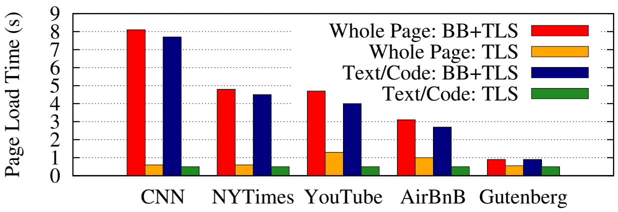


Figure 4: Download time time for TLS and BlindBox (BB) + TLS at 1Gbps$\times$10ms.

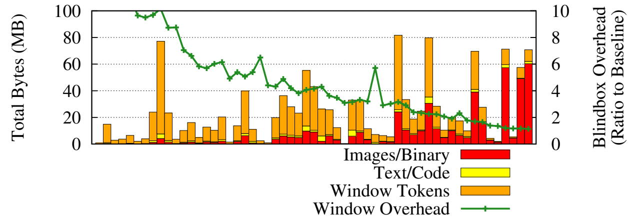
How long does the initial handshake take with the middlebox? The initial handshake to perform obfuscated rule encryption runs in time proportional to the number of rules. In the datasets we worked with, the average Protocol II rule had slightly more than 3 keywords; a typical 3000 rule IDS rule set contains between 9-10k keywords. The total client-side time required for 10k keywords was 97 seconds; for 1000 keywords, setup time was 9.5s. In a smaller ruleset of 10 or 100 keywords (which is typical in a watermark detection exfiltration device), setup ran in 650ms and 1.6 seconds, respectively. These values are dependent on the clock speed of the CPU (to generate the garbled circuits) and the network bandwidth and latency (to transmit the circuits from client to sender). Our servers have 2.6GHz cores; we assumed a middlebox on a local area network near the client with a 100μs RTT between the two and a 1Gbps connection. Garbling a circuit took 1042μs per circuit; each garbled circuit transmission is 599KB.

Neither strawman has an appropriate setup phase that meets the requirement of not making the rules visible to the endpoints. However, one can extend these strawmen with BlindBox's obfuscated rule encryption technique, and encrypt the rules using garbled circuits. In this case, for the scheme of Song et al., the setup cost would be similar to the one of BlindBox because their scheme also encrypts the rule keywords with AES. For the scheme of Katz et al., the setup would be much slower because one needs garbled circuits for modular exponentiation, which are huge. Based on the size of such circuits reported in the literature [16], we can compute a generous lower bound on the size of the garbled circuits and on the setup cost for this strawman: it is at least $1.8 \cdot 10^3$ times larger/slower than the setup in BlindBox.

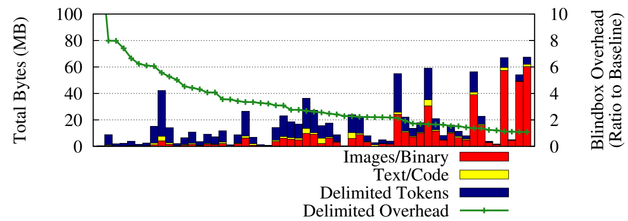
How long are page downloads with BlindBox, excluding the setup (handshake) cost? Figure 3 shows page download times using our "typical end user" testbed with 20Mbps links. In this figure, we show five popular websites: YouTube, AirBnB, CNN, The New York Times, and Project Gutenberg. The data shown represents the post-handshake (persistent connection) page download time. YouTube and AirBnB load video, and hence have a large amount of binary data which is not tokenized. CNN and The New York Times have a mixture of data, and Project Gutenberg is almost entirely text. We show results for both the amount of time to download the page including all video and image content, as well as the amount of time to load only the Text/Code of the page. The overheads when downloading the whole page are at most 2$\times$; for pages with large amount of binary data like YouTube and AirBnB, the overhead was only 10-13%. Load times for Text/Code only – which are required to actually begin rendering the page for the user – are impacted more strongly, with penalties as high as 3$\times$ and a worst case of about 2$\times$.

What is the computational overhead of BlindBox encryption, and how does this overhead impact page load times? While the encryption costs are not noticeable in the page download times observed over the "typical client" network configuration, we immediately see the cost of encryption overhead when the available link capacity increases to 1Gbps in Figure 4 – at this point, we see a performance overhead of as much as 16$\times$ relative to the baseline SSL download time. For both runs (Figs. 3 and 4), we observed that the CPU was almost continuously fully utilized to transfer data during data transmission. At 20Mbps, the encryption cost is not noticeable as the CPU can continue producing data at around the link rate; at 1Gbps, transmission with BlindBox stalls relative to SSL, as the BlindBox sender cannot encrypt fast enough to keep up with the line rate. This result is unsurprising given the results in Table 2, showing that BlindBox takes 30$\times$ longer to encrypt a packet than standard HTTPS. This overhead can be mitigated with extra cores; while we ran with only one core per connection, tokenization can easily be parallelized.

What is the bandwidth overhead of transmitting encrypted tokens for a typical web page? Minimizing bandwidth overhead is key to client performance: less data transmitted means less cost, faster transfer times, and faster detection times. The bandwidth overhead in BlindBox depends on the number of tokens produced. The number of encrypted tokens varies widely depending on three parameters of the page being loaded: what fraction of bytes are text/code which must



(a) Window-Based Tokenization



(b) Delimiter-Based Tokenization

Figure 5: Bandwidth overhead over top-50 web dataset.

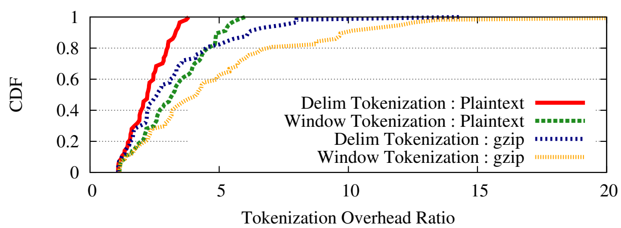


Figure 6: Ratio: transmitted bytes with BlindBox to transmitted bytes with SSL.

be tokenized, how "dense" the text/code is in number of delimiters, and whether or not the web server and client support compression.

Figures 5 (a) and (b) break down transmitted data into the number of text-bytes, binary-bytes, and tokenize-bytes using the window-based and delimiter-based tokenization algorithms (as discussed in §3); the right hand axis shows the overhead of adding tokens over transmitting just the original page data. We measured this by downloading the Alexa top-50 websites [1] and running BlindBox over all page content (including secondary resources loaded through AJAX, callbacks, etc.) The median page with delimited tokens sees a 2.5× increase in the number of bytes transmitted. In the best case, some pages see only a 1.1× increase, and in the worst case, a page sees a 14× overhead. The median page with window tokens sees a 4× increase in the number of bytes transmitted; the worst page sees a 24× overhead. The first observable factor affecting this overhead, as seen in these figures, is simply what fraction of bytes in the original page load required tokenization. Pages consisting mostly of video suffered lower penalties than pages with large amounts of text, HTML, and Javascript because we do not tokenize video.

7.2.3 *Middlebox Performance*

We investigate performance at the middlebox using both micro-benchmarks and overall throughput.

What throughput can BlindBox sustain and how does this compare to standard IDS? When running our BlindBox implementation over synthetic traffic, we measured a throughput of 166Mbps; when running Snort over the same traffic, we measured a throughput of 85Mbps. Hence, BlindBox performed detection twice as fast as Snort, which inspects *unencrypted* traffic. The reason behind this situation is twofold. First, BlindBox reduces all detection to exact matching, pushing all regular expression parsing to a secondary middlebox, invoked rarely. Second, our implementation is built over DPDK-click, a faster packet-capture library than what Snort uses by default. Hence, it is unsurprising that BlindBox performs detection more quickly. Nevertheless, the point of this experiment is not to show that BlindBox is faster than Snort, but instead to demonstrate that BlindBox provides competitive performance to today's deployments.

How does BlindBox compare in detection time against other strawman approaches? While we did not implement a version of BlindBox which relied on our strawmen, we can compare against it using a smaller benchmark. Once again, in Table 2, the FE strawman is seen to be prohibitively impractical: detection over a single packet against a 3000 ruleset takes more than a day.

The searchable strawman is also prohibitively slow: it performs detection over a 1500 byte packet in 157 ms, which is equivalent to no more than 6-7 packets per second. This performance is three orders of magnitude slower than the performance of BlindBox's middlebox. This overhead results from the fact that the searchable strawman must perform an encryption operation over every keyword to perform a comparison against a client token, resulting in a task linear in the number of keywords. In contrast, BlindBox's DPIEnc scheme encrypts the data in such a way that the middlebox can use a fast, pre-computed search tree (which gives a logarithmic search) to match encrypted tokens to rules.

8. RELATED WORK

Related work falls into two categories: insecure proposals, and work on computing on encrypted data.

8.1 Insecure Proposals

Some existing systems mount a man-in-the-middle attack on SSL [28, 27] by installing fake certificates at the middlebox [31, 41]. This enables the middlebox to break the security of SSL and decrypt the traffic so it can run DPI. This breaks the end-to-end security of SSL, and results in a host of issues, as surveyed by Jarmoc [28].

Some proposals allow users to tunnel their traffic to a third party middlebox provider, e.g. Meddle [40], Beyond the Radio [48], and APLOMB [42]. These approaches allow the middlebox owner to inspect/read all traffic. The situation is preferable to the status queue (from the client's perspective) in that the inspector is one with whom the client has a formal/contractual relationship – but, unlike BlindBox, the client still must grant *someone* access to the plaintext traffic. Further, this approach is not preferable to service providers, who may wish to enforce policy on users in the network, e.g., that no hosts within the network are infected with botnet malware.

一些现有的系统通过在中间件安装假证书来对SSL进行中间人攻击。这使得中间盒能够破坏SSL的安全性并解密流量，以便它可以运行DPI。正如Jarmoc [28]调查的那样，这破坏了SSL的端到端安全性，并导致一系列问题。一些建议允许用户将他们的流量通过隧道传输到第三方中间盒提供商，例如，多管闲事[40]，超越无线电[48]，冷静[42]。这些方法允许中间盒所有者检查/读取所有流量。这种情况比状态队列(从客户的角度来看)更好，因为检查员是与客户有正式/合同关系的人——但是，与盲箱不同，客户仍然必须授权某人访问明文流量。此外，这种方法对于服务提供商来说并不可取，服务提供商可能希望对网络中的用户实施策略，例如，网络中没有主机感染僵尸网络恶意软件。

全同态加密和一般函数加密是加密方案，可以计算加密数据的任何函数；因此，原则上它们可以支持深度包检查任务的复杂性。然而，它们不能解决我们的威胁模型中所有期望的安全属性，更重要的是，它们非常慢，目前比未加密的计算至少慢8个数量级。最近的一些系统，如CryptDB 和Mylar，展示了如何有效地支持加密数据上的一些专门计算。然而，这些系统执行的任务不同于中间人所需的任务，不符合我们的威胁模型。已经有大量关于可搜索加密的工作。没有一个可搜索的加密方案提供了一种安全加密规则和支持任意正则表达式的策略，这两者都是BlindBox提供的。此外，现有方案不能提供分组处理所需的性能。例如，BlindBox比使用宋等人的对称密钥可搜索方案的系统快三个数量级。公钥可搜索加密方案，如[19]，甚至更慢，因为它们对每对令牌执行一次加密配对(每次配对需要数百微秒)，以规则内容(规则数量的线性而非对数任务)。

8.2 Computing on Encrypted Data

Fully homomorphic encryption (FHE) [24] and general functional encryption [23, 26] are encryption schemes that can compute any function over encrypted data; hence, they can in principle support the complexity of deep packet inspection tasks. However, they do not address all the desired security properties in our threat model, and more importantly, they are prohibitively slow, currently at least 8 orders of magnitude slower than unencrypted computation [25].

Some recent systems such as CryptDB [37] and Mylar [38] showed how to support some specialized computation efficiently on encrypted data. However, these systems perform different tasks than is needed for middleboxes and do not meet our threat model.

There has been a large amount of work on searchable encryption [46, 29, 19, 15]. No searchable encryption scheme provides a strategy for encrypting the rules securely and for supporting arbitrary regexps, both of which BlindBox provides. Moreover, existing schemes cannot provide the performance required for packet processing. For example, BlindBox is three orders of magnitude faster than a system using the symmetric-key searchable scheme of Song et al. [46]. Public-key searchable encryption schemes, such as [19], are even slower because they perform a cryptographic pairing (which takes hundreds of microseconds per pairing), for every pair of token to rule content (a linear, rather than logarithmic task in the number of rules).

9. DISCUSSION

Before concluding, we discuss future directions and implications of BlindBox.

New Security Models. In our extended paper [43], we provide cryptographic proofs that the BlindBox system respects the exact match and probable cause privacy models. We believe our new privacy models are useful. Both approaches raise the bar substantially from the current "man-in-the- middle" model, which de-facto breaks any trust in SSL altogether. Nevertheless, BlindBox allows the middlebox to know when a keyword from an attack rule matches the traffic even though the entire rule (with all its keywords contents) does not match. In our experiments, almost all keyword detections that *didn't* result in the detection of a complete rule were for generic substrings (e.g. onmouseover), which most likely do not leak a user's personal data.

ISP Adoption. In enterprises and private networks, BlindBox provides a good trade-off between the desires of users (who want privacy, and may want processing) and the network administrator (who wants to deploy processing primarily, and is willing to respect privacy if able to do so). Hence, deploying BlindBox is aligned with both parties' interests. However, in ISPs, sales of user data to marketing and analytics firms are a source of revenue – hence, an ISP has an incentive *not* to deploy BlindBox. Consequently, deployment in ISPs is likely to take place either under legislative requirement through privacy laws, or through a change in incentives. The growing trend of middlebox processing being offered as a "service" – which users opt in to and pay for – may offer an incentive for an ISP to deploy BlindBox with the promise of a new revenue stream directly from the user.

Client Adoption. BlindBox proposes a new end-to-end encryption protocol to replace HTTPS altogether. A truly ideal solution would require no changes at the endpoints – indeed, the success of middlebox deployments is partly due to the fact that middleboxes can be simply "dropped in" to the network. Unfortunately, existing HTTPS encryption algorithms use strong encryption schemes, which do not support *any* functional operations and cannot be used for our task; hence one must change HTTPS. Nonetheless, we believe that, in the long run, a change to HTTPS to allow inspection of encrypted traffic can be generic enough to support a wide array of middlebox applications, and not just the class of middleboxes in BlindBox. We believe these benefits will merit widespread "default" adoption in end host software suites.

10. CONCLUSION

In this paper, we presented BlindBox, a system that resolves the tension between security and DPI middlebox functionality in networks. To the best of our knowledge, BlindBox is the first system to enable Deep Packet Inspection over encrypted traffic without requiring decryption of the underlying traffic. BlindBox supports real DPI applications such as IDS, exfiltration detection, and parental filtering. BlindBox performs best over long-running, persistent connections using SPDY-like or tunneled protocols. Using BlindBox Detect, a middlebox running BlindBox can perform detection on a single core at 186Mbps – competitive with many deployed IDS implementations.

We envisage that BlindBox is the first step towards a *general protocol* to resolve the tension between encryption and all categories of middleboxes. BlindBox currently supports middleboxes for DPI filtering only, however, we believe that the general blueprint BlindBox provides – computation over encrypted traffic – can be extended to implement other middlebox capabilities, including caches, protocol accelerators, compression engines.

11. ACKNOWLEDGMENTS

We thank the anonymous reviewers of the SIGCOMM PC for their valuable comments on this work. We especially thank our shepherd Mike Walfish for his detailed and helpful feedback. Vern Paxson coined the term "probable cause" for one of our privacy models. Kay Ousterhout gave many comments on early revisions of this work. McAfee Stonesoft, Lastline, and Palo Alto networks met with us to discuss state-of-the-art IDS today, and Stonesoft and Lastline provided us with metadata about their commercial rulesets. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1106400. We also thank Intel Research for their generous funding and technical feedback on this work.

12. REFERENCES

[1] Alexa Top Sites. http://www.alexa.com/topsites.
[2] DPDK: Data Plane Development Kit. http://dpdk.org/.
[3] Emerging Threats: Open Source Signatures. https://rules.emergingthreats.net/open/snort-2.9.0/rules/.
[4] McAfee Network Security Platform. http://www.mcafee.com/us/products/network-security-platform.aspx.
[5] OT Extension library. https://github.com/encryptogroup/OTExtension.
[6] Palo Alto Networks. https://www.paloaltonetworks.com/.
[7] Qosmos Deep Packet Inspection and Metadata Engine. http://www.qosmos.com/products/deep-packet-inspection-engine/.
[8] Radisys R220 Network Appliance. http://www.radisys.com/products/network-appliance/.
[9] Snort. https://www.snort.org/.
[10] ssldump. http://www.rtfm.com/ssldump/.
[11] Symantec | Enterprise. http://www.symantec.com/index.jsp.
[12] The GnuTLS Transport Layer Security Library. http://www.gnutls.org/.
[13] University of Toulouse Internet Blacklists. http://dsi.ut-capitole.fr/blacklists/.
[14] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *Proc. ACM CCS*, 2013.
[15] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and Efficiently Searchable Encryption. In *Proc. IACR CRYPTO*, 2007.
[16] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient Garbling from a Fixed-Key Blockcipher. In *Proc. IEEE S&P*, 2013.
[17] BlueCoat. Comparing Explicit and Transparent PRoxies. https://bto.bluecoat.com/webguides/proxysg/security_first_steps/Content/Solutions/SharedTopics/Explicit_Transparent_Proxy_Comparison.htm.
[18] BlueCoat. SSL Encrypted Traffic Visibility and Management. https://www.bluecoat.com/products/ssl-encrypted-traffic-visibility\-and-management.
[19] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. IACR EUROCRYPT*, 2004.
[20] S. Cheshire and M. Krochmal. NAT Port Mapping Protocol (NAT-PMP). RFC 6886, Apr. 2013.
[21] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: A Scalable Fault Tolerant Network Manager. In *Proc. USENIX NSDI*, 2011.
[22] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Commun. ACM*, 28(6):637–647, June 1985.
[23] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. IEEE FOCS*, 2013.
[24] C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *Proc. ACM STOC*, 2009.
[25] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic Evaluation of the AES Circuit. In *Proc. IACR CRYPTO*, 2012.
[26] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable Garbled Circuits and Succinct Functional Encryption. In *Proc. ACM STOC*, 2013.
[27] L.-S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing Forged SSL Certificates in the Wild. In *Proc. IEEE S&P*, 2014.
[28] J. Jarmoc. SSL/TLS Interception Proxies and Transitive Trust. *Presentation at Black Hat Europe*, 2012.
[29] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic Searchable Symmetric Encryption. In *Proc. ACM CCS*, 2012.
[30] J. Katz, A. Sahai, and B. Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In *Proc. IACR EUROCRYPT*, 2008.
[31] A. Kingsley-Hughes. Gogo in-flight Wi-Fi serving spoofed SSL certificates. *ZDNet*, 2015.
[32] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.
[33] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptol.*, 22:161–188, April 2009.
[34] M. Naor and B. Pinkas. Oblivious Transfer with Adaptive Queries. In *Proc. IACR CRYPTO*, 1999.
[35] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The Cost of the "S" in HTTPS. In *Proc. ACM CoNeXT*, 2014.
[36] V. Paxson. Bro: A System for Detecting Network Intruders in Real-time. *Comput. Netw.*, 31(23-24):2435–2463, Dec. 1999.
[37] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proc. ACM SOSP*, 2013.
[38] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan. Building Web Applications on Top of Encrypted Data using Mylar. In *Proc. USENIX NSDI*, 2014.
[39] M. O. Rabin. How to Exchange Secrets with Oblivious Transfer. TR-81, Aiken Computation Lab, Harvard University http://eprint.iacr.org/2005/187.pdf, 1981.
[40] A. Rao, J. Sherry, A. Legout, W. Dabbout, A. Krishnamurthy, and D. Choffnes. Meddle: Middleboxes for Increased Transparency and Control of Mobile Traffic. In *Proc. CoNEXT Student Workshop*, 2012.
[41] Runa. Security vulnerability found in Cyberoam DPI devices (CVE-2012-3372). *Tor Project Blog*, 2012.
[42] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing As a Cloud Service. In *Proc. ACM SIGCOMM*, 2012.
[43] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. Cryptology ePrint Archive, Report 2015/264, 2015. http://eprint.iacr.org/.
[44] Shira Levine. Operators look to embed deep packet inspection (DPI) in apps; Market growing to $2B by 2018. Infonetics Research. http://www.infonetics.com/pr/2014/2H13-Service-Provider-DPI-Products-Market-Highlights.asp.
[45] G. J. Silowash, T. Lewellen, J. W. Burns, and D. L. Costa. Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection. Technical Report CMU/SEI-2013-TN-012.
[46] D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *Proc. IEEE S&P*, 2000.
[47] The Snort Project. Snort users manual, 2014. Version 2.9.7.
[48] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson. Beyond the Radio: Illuminating the Higher Layers of Mobile Networks. In *Proc. ACM MobiSys*, 2015.
[49] G. Vigna. ICTF Data. https://ictf.cs.ucsb.edu/#/.
[50] A. C. Yao. How to Generate and Exchange Secrets. In *Proc. IEEE FOCS*, 1986.
[51] K. Zetter. The Feds Cut a Deal With In-Flight Wi-Fi Providers, and Privacy Groups Are Worried. *Wired Magazine*, 2014.