BlindIDS: Market-Compliant and Privacy-Friendly Intrusion Detection System over Encrypted Traffic

Sébastien Canard
Orange Labs, France
sebastien.canard@orange.com

Aïda Diop
Orange Labs, France
aida.diop@orange.com

Nizar Kheir
Thales Group, France
nizar.kheir@thalesgroup.com

Marie Paindavoine
Orange Labs, France
marie.paindavoine@orange.com

Mohamed Sabt
DejaMobile, France
mohamed.sabt@dejamobile.com

ABSTRACT

The goal of network intrusion detection is to inspect network traffic in order to identify threats and known attack patterns. One of its key features is Deep Packet Inspection (DPI), that extracts the content of network packets and compares it against a set of detection signatures. While DPI is commonly used to protect networks and information systems, it requires direct access to the traffic content, which makes it blinded against encrypted network protocols such as HTTPS. So far, a difficult choice was to be made between the privacy of network users and security through the inspection of their traffic content to detect attacks or malicious activities.

This paper presents a novel approach that bridges the gap between network security and privacy. It makes possible to perform DPI directly on encrypted traffic, without knowing neither the traffic content, nor the patterns of detection signatures. The relevance of our work is that it preserves the delicate balance in the security market ecosystem. Indeed, security editors will be able to protect their distinctive detection signatures and supply service providers only with encrypted attack patterns. In addition, service providers will be able to integrate the encrypted signatures in their architectures and perform DPI without compromising the privacy of network communications. Finally, users will be able to preserve their privacy through traffic encryption, while also benefiting from network security services. The extensive experiments conducted in this paper prove that, compared to existing encryption schemes, our solution reduces by 3 orders of magnitude the connection setup time for new users, and by 6 orders of magnitude the consumed memory space on the DPI appliance.

AsiaCCS '17, April 2–6, 2017, Abu Dhabi, United Arab Emirates

DOI: http://dx.doi.org/10.1145/3052973.3053013

Keywords

Intrusion detection, network middleboxes, deep packet inspection, decryptable searchable encryption, security, privacy

1. INTRODUCTION

Network intrusion detection and prevention systems are extensively used today for early warning and protection of IT systems against generic attacks and intrusions. They identify unauthorized use, misuse, and abuse of information systems by both system insiders and external attackers [22]. They also supply a wide range of functionalities such as access control through the blocking of unauthorized content and destinations (e.g., IP or domain blacklists), data loss protection, and detection of cyberattacks such as vulnerability exploits, malwares, and botnets. A main feature of these systems is Deep Packet Inspection (DPI). It inspects the content of network packets and tunes decisions, including making alert or even rejecting packets, based on a fine-grained matching against a set of known malicious patterns. Nowadays, security editors operate extensive threat intelligence in order to build and update their comprehensive set of attack signatures, which is further used to conduct DPI and to supply value-added security services. In fact, most security editors put forward their attack signatures databases as a key competitive differentiator, arguing they may cover a wider set of malicious patterns and attacks [13, 5]. They are indeed often benchmarked against each other based on the coverage of their attack signatures, which constitutes the key driver for DPI-based security services - e.g., [26].

However, DPI requires access to the clear-text content of network packets, making it blinded against network encryption (e.g., the use of SSL/TLS protocols). This is a fundamental limitation of DPI, as it opens the door wide to a large number of attacks. According to a recent study, nearly half of malware attacks in 2015 has infiltrated their target networks while using encryption as a cover [27]. This comes in the context of an increasing trend towards network encryption, with 70% of global Internet traffic being encrypted by the end of 2016, and some networks exceeding 80% [24].

To overcome the challenge of network encryption, a common solution called SSL inspection offers to use Man-In-The-Middle (MITM) appliances between the two ends of a network connection. It uses trusted certificates to impersonate the recipient of the originating SSL session, then decrypts and applies DPI over the clear-text traffic con-

tent [14]. Nonetheless, SSL inspection breaks the end-to-end security of the SSL protocol, raising security issues about the privacy of the log data generated by the MITM appliance [15]. It also raises ethical issues. For instance, Google has discovered the unauthorized use of digital certificates issued by an intermediate certificate authority linked to ANSSI, a French cyber security agency operating with French intelligence agencies [23].

So far, a delicate choice was to be made between security and privacy; either preserving end-to-end encryption, or using SSL inspection and DPI [16]. Recently, a new approach called BlindBox, has proposed an encryption protocol that enables for the first time to search for malicious patterns directly over encrypted traffic [25]. BlindBox is based on multi-party computation techniques, such as garbled circuits and oblivious transfer. It allows a middlebox appliance on the network path between the two ends of an encrypted connection to search for malicious patterns with no need to break the end-to-end encryption. This is a key contribution of BlindBox, as it sets a milestone on the long path towards reconciling network security and privacy. Despite its original solution, BlindBox still lacks two important properties that would make quite difficult the wide adoption of this approach.

First, BlindBox requires that the middlebox encrypts the entire set of malicious patterns to be looked for in the traffic using a key derived from the secret session key of each new HTTPS connection. This drastically increases the time for connection setup. In particular, the connection setup time was evaluated in [25] to 97 seconds for a generic HTTPS connection, while using a sample set of 3 thousand detection signatures. The middlebox would also need to encrypt and manage a distinct copy of the entire set of malicious patterns used for detection, for the entire duration of every single HTTPS connection. This requires an excessive memory space at the middlebox (e.g., 512GB for 3K rules and 100 parallel connections), which makes it very difficult to integrate in any real-world deployment. Second, BlindBox requires the security editors to supply their entire set of malicious patterns in clear-text to the service providers in charge of middlebox appliances. Although this makes no difference from a technical standpoint, it does not fit with the delicate balance in the security market ecosystem. Such malicious patterns and their coverage constitute a key business differentiator for any security editor. Hence, it is highly unlikely that security editors will be keen to deliver these valuable assets in clear-text to the middlebox appliances [19].

In this paper, we leverage a decryptable searchable encryption scheme [12], based on public-key cryptography, that addresses these main limitations of the BlindBox approach. The main goal is to provide a technically sound solution that is at the same time (a) privacy-friendly, meaning no access is possible to the clear-text content of encrypted traffic, (b) security-aware, meaning it supports DPI over encrypted traffic, and (c) practical, achieving both performance and real-world market requirements. Extensive evaluations conducted in this paper show that our encryption protocol enhances by several orders of magnitude the performance of BlindBox, including both the connection setup time and the memory space needed by the middlebox to perform its task.

To summarize, the contributions of this paper are threefold:

- a DPI solution that achieves both privacy and security requirements, with a performance several orders of magnitude closer to real-world conditions than other similar encryption schemes such as BlindBox;
- a DPI solution that preserves the privacy of both network traffic and pattern signatures. This provides indeed more privacy assurance to the users, since the middlebox does not even know the patterns it is looking for in the traffic;
- the first complete and formal security model that characterizes an intrusion detection system operating over an encrypted traffic, which contribution may be of independent interest. We also prove that our DPI solution is secure in this model.

The paper is now organized as follows. Section 2 describes the architecture of our solution and the key security and privacy requirements that it aims to achieve. Section 3 reviews related work and summarizes our main contributions. Section 4 introduces our encryption protocol and provides the appropriate security and privacy proofs. Section 5 presents evaluation and experimental results. Finally, section 6 concludes.

2. ARCHITECTURE AND SECURITY

We first describe the main architecture in which we will work, and we then give the required security properties that should be satisfied by our intrusion detection system in the context of encrypted network communications.

2.1 Global Architecture

The encryption protocol that we describe in this paper involves four main roles: security editor (a.k.a rule generator in [25]), service provider (a.k.a. middlebox in [25]), sender and receiver. The sender and receiver roles are interchangeable with respect to our solution. Each end of a secure network connection may play both the sender and receiver roles within the same session. In a standard intrusion detection scenario, either the sender or receiver role may be a malicious entity (e.g., individual attacker, compromised server) that attempts to compromise the other remote and benign entity. Other scenarios including, for example, a malware-infected terminal connecting to a malicious destination, may still be possible. Both the sender and receiver roles in this case are malicious. This is also a scenario that we cover in our encryption protocol, unless when the two entities are able to collaborate together. Although it is technically possible, this scenario may not be considered as a limitation to our approach, since both parties may agree on a secret key through a third channel that is out of reach for both the security editor and the service provider. Hence, such attacks will get unnoticed even in the context of a clear-text network traffic.

The security editor is the entity that is in charge of editing the detection signatures. These signatures may include IP or domain blacklists, such as malware domains, IPs enrolled into botnets, or websites delivering censored or illicit content such as adultery and terrorism. They may also include a logical combination of binary patterns that capture malware samples and vulnerability exploits. Finally, they may include more elaborated regular expressions that characterize fine-grained attacks such as SQL and code injections. Detection signatures are key assets for the security editor.

They require extensive and continuous threat intelligence in order to constantly enhance their coverage so they may capture new threats including zero-day attacks. The security editor role in this paper may belong to a large number of market competitors such as Symantec, TrendMicro, McAfee, BitDefender, and Kaspersky.

On the other hand, service providers are stakeholders that offer network and security services both to end-users and enterprises. They supply middleboxes, both as physical appliances and virtual cloud-based services, and that support multiple security services such as firewalls, proxies, data loss protection, intrusion detection and prevention. Service providers often partner with security editors in order to add detection logic, pattern signatures for instance, into their middleboxes. In the context of this paper, services using deep packet inspection are required to provide the same security guarantees both for encrypted and clear-text network connections.

2.2 Security Requirements

Before going into a more formal description, we first give some words on the expected security requirements.

Service Provider (SP) role is handled in our protocol as an *Honest-but-Curious* entity. It applies DPI honestly over the encrypted traffic, using the detection signatures delivered by the security editor. The SP will try, however, to acquire additional information about either or both the traffic content and the malicious patterns represented in the signatures. This is indeed a key contribution of our paper compared to BlindBox, since in our protocol the SP does not know neither the traffic content nor the pattern signatures. The SP is able to blindly detect attacks, without knowing the malicious patterns that it is looking for in the traffic.

Security Editor (SE) role is also handled in our protocol as an *Honest-but-Curious* entity. All detection signatures include true and authentic malicious patterns whose main purpose is exclusively to detect and qualify network attacks. This is a fairly reasonable assumption because SEs will not tarnish their reputation by issuing wrong or misleading signatures. Moreover, wrong signatures that match benign traffic will lead to false positives, which may degrade the quality of service as perceived by end users. However, the SE is curious, as it may try to acquire information about the clear-text content of the traffic. This could be either through direct eavesdropping over the network, or through the SP alerts.

Colluding SE and SP that use our encryption protocol will be able to mount a dictionary attack. Therefore, we consider that the SE and SP will not collaborate together in our security model. We consider this assumption as fairly reasonable because both SE and SP may be curious, but open dishonest behavior will cause them extensive damages in a free market environment.

Colluding sender and receiver can agree on a shared secret using a third channel that is out of reach for both SE and SP. In case of a malicious sender and receiver, our security model does not expect both parties to collaborate. We refer to the example of an infected bot connecting to its command server which is under control of the same attacker. We may split this scenario into two phases. First, an attacker may compromise a benign terminal and recruit it into a malicious botnet network. Second, the infected bot connects to a remote command and control a server that was specifically set by the attacker for this purpose. Unfortunately, when network encryption is used, our security model only detects the first phase. Nevertheless, this is also a limitation to all existing SSL inspection solutions since the infected bot and the command server may agree on a secret coding or encryption in order to hide malicious commands, so as they cannot be covered by clear-text detection signatures.

For the sake of simplicity, we consider in the remainder of this paper that the sender role is malicious, and the receiver is honest. Our encryption protocol applies in the same way in the context of an honest sender and a malicious receiver.

2.3 A New Security Model

Based on the above remarks, we now formally define the security model that represents an ideal intrusion detection system operating over an encrypted traffic. We first model the interactions between actors, and we then describe the three main security properties that need to be satisfied by the detection system.

We consider a set $\mathcal{R}$ of (non-encrypted) rules and a detection algorithm, denoted Detect, taking as input a (non-encrypted) traffic T and the set $\mathcal{R}$. In the sequel, we say that the traffic is malicious iff $\mathsf{Detect}(T, \mathcal{R}) = 0$. Otherwise, the traffic is safe and $\mathsf{Detect}(T, \mathcal{R}) = 1$. In both cases, some auxiliary information aux can be also provided as output to the detection algorithm. We also consider in our model that the detection procedure can output more detailed information (for example the number of malicious patterns that has been recognized[1]).

Model for interactions. An intrusion detection system over an encrypted traffic, denoted π, and played by the four actors Service Editor (SE), Service Provider (SP), Sender (S) and Receiver (R), is composed of five main procedures.

- Setup, on input the security parameter λ, generates the public parameters param of the system, and the potential keys of the actors. When an actor A ($\mathsf{A} \in \{\mathsf{SE}, \mathsf{SP}, \mathsf{S}, \mathsf{R}\}$) manages a cryptographic key, we always consider that there is a key pair $(\mathsf{sk}_\mathsf{A}, \mathsf{pk}_\mathsf{A})$ where sk_A is secret and only known by A, and pk_A is publicly available. The latter key may be empty (in case of a secret key based solution). Such a procedure can be uniquely executed by one actor, or played by several ones (for example, each actor A can create his/her own keys). We consider that all public keys are now included in the parameters param.

- $\mathsf{RuleGen}$, on input the parameters param, the SE secret key sk_SE and a set $\mathcal{R}$ of rules to detect a malicious traffic, outputs a set $\mathcal{B}$ of blinded rules that are then sent to SP.

- Send takes as input the public parameters param, potentially the secret key sk_S of the sender and the public key pk_R of a receiver R, and a traffic T. It outputs an encrypted traffic E for receiver R.

[1]Even if the pattern itself remains unknown by the Service Provider executing the detection algorithm. For example, in our construction, SP knows which trapdoor matches, but does not know the underlying keyword, see Section 5 for details.

Experiment $\mathsf{Exp}^{det}_{\pi,\mathcal{A}}(\lambda)$

$(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathcal{B} \leftarrow \mathsf{RuleGen}(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathcal{R})$;
$E \leftarrow \mathcal{A}(1^\lambda, \mathsf{param})$;
if $\mathsf{Detect}(\mathsf{param}, E, \mathcal{B}) = 1$, then return 0;
$T \leftarrow \mathsf{Receive}(\mathsf{param}, \mathsf{sk}_{\mathsf{R}}, E)$;
if $\mathsf{Detect}(T, \mathcal{R}) = 0$, then return 0.
return 1;

Figure 1: Detection experiment

Experiment $\mathsf{Exp}^{tr-ind}_{\pi,\mathcal{A}}(\lambda)$

$b \leftarrow \{0, 1\}$;
$(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$T_0, T_1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{param})$;
if $\mathsf{type}(T_0, T_1) = 0$, return 0;
$E_b \leftarrow \mathsf{Send}(\mathsf{param}, T_b)$;
$b' \leftarrow \mathcal{A}(E_b)$;
return $(b = b')$;

Figure 2: Traffic indistinguishability experiment

- Detect, on input param, the Service Provider public key $\mathsf{pk}_{\mathsf{SP}}$ (if it exists), an encrypted traffic E and the set $\mathcal{B}$ of blinded rules from SE, outputs a bit $b \in \{0, 1\}$, stating that the underlying traffic T is malicious ($b = 0$) or safe ($b = 1$). It may also return some auxiliary information aux, such as for example the blinded rule that matched. If something goes wrong, it outputs an error message $\perp$.
- Receive is executed by taking on input the parameters param, the receiver's secret key sk_{R} and an encrypted traffic E. It outputs a plain traffic T, or an error message $\perp$.

Remark 1. *After the execution of the procedure* Setup *to generate the public parameters* param *and the keys, we denote* $\mathcal{B} = \mathsf{RuleGen}(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathcal{R})$ *where* $\mathcal{R}$ *is the set of rules, and* $E = \mathsf{Send}(\mathsf{param}, \mathsf{sk}_{\mathsf{S}}, \mathsf{pk}_{\mathsf{R}}, T)$ *where* T *is a traffic. An intrusion detection system over an encrypted traffic is said correct iff*

$$T = \mathsf{Receive}(\mathsf{param}, \mathsf{sk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{S}}, E), \text{ and}$$
$$\mathsf{Detect}(T, \mathcal{R}) = \mathsf{Detect}(\mathsf{param}, E, \mathcal{B}) \text{ (including aux).}$$

Model for security properties. As sketched in the previous section, there are mainly three security properties that should be verified by such a system. We call the first one the detection property, the second one the traffic indistinguishability property, and the last one the signature indistinguishability property.

Detection. Informally speaking, the detection property states that any malicious traffic (that is a traffic considered as malicious when not encrypted) must be detected by the Service Provider in the proposed intrusion detection system over encrypted traffic. This is related to the security-awareness feature, and gives a way to guarantee the correctness of the detection.

More formally, we give the experiment in Figure 1, for an adversary $\mathcal{A}$. On input the parameters, $\mathcal{A}$ outputs an encrypted traffic E such that $\mathsf{Detect}(\mathsf{param}, E, \mathcal{B}) = 1$ (that is stated as safe) while the decrypted version T is malicious (that is $\mathsf{Detect}(T, \mathcal{R}) = 0$).

Then, an intrusion detection system over encrypted traffic π is said *detectable* if for any probabilistic polynomial-time $\mathcal{A}$, there exists a negligible function $\nu(\lambda)$ such that:

$$\mathsf{Succ}^{det}_{\pi,\mathcal{A}}(\lambda) = \Pr\left[\mathsf{Exp}^{det}_{\pi,\mathcal{A}} = 1\right] \leq \nu(\lambda).$$

Traffic indistinguishability. The traffic indistinguishability property informally states that it is not feasible for the Service Provider to learn any information about the traffic, other than it is malicious or safe. We here focus on the privacy-friendly feature, that is no access to the clear-text content is possible.

In a traditional indistinguishability property, the adversary chooses two messages and should not be able to distinguish which of the two is encrypted by the challenger. In our context, we have the problem that the adversary may choose one malicious traffic and one safe traffic so that it will be easy to distinguish which one is used by the challenger, simply by applying the Detect algorithm. Moreover, the detection algorithm can also give to $\mathcal{A}$ some auxiliary information aux about the type of attack, so that it can choose the two messages accordingly. We then introduce the notion of *type* for a traffic, using the following definition.

Definition 1 (Traffic Type). *Let* T_0 *and* T_1 *be two traffics and let* $\mathcal{R}$ *be a set of rules. We say that* T_0 *and* T_1 *are of the* same type*, denoted* $\mathsf{type}(T_0, T_1) = 1$*, iff*

$$\mathsf{Detect}(\mathsf{param}, T_0, \mathcal{R}) = \mathsf{Detect}(\mathsf{param}, T_1, \mathcal{R}),$$

including the auxiliary information aux.

More formally, we then give the traffic indistinguishability experiment in Figure 2, for an adversary $\mathcal{A}$ having access to both a Receive oracle (given an encrypted traffic E of its choice, $\mathcal{A}$ obtains the related plain traffic E) and the RuleGen oracle (given a set of rules $\mathcal{R}$ of its choice, the adversary gets back $\mathcal{B} \leftarrow \mathsf{RuleGen}(\mathsf{param}, \mathsf{sk}_{SE}, \mathcal{R})$). The adversary first chooses two traffics T_0 and T_1 and, if they have the same type, one of them, T_b is encrypted and given to $\mathcal{A}$. Eventually, $\mathcal{A}$ has to guess the bit b.

Then, an intrusion detection system over encrypted traffic π is said *traffic-indistinguishable* if for any probabilistic polynomial-time $\mathcal{A}$, there exists a negligible function $\nu(\lambda)$ such that:

$$\mathsf{Adv}^{tr-ind}_{\pi,\mathcal{A}}(\lambda) = \left|2 \cdot \Pr\left[\mathsf{Exp}^{tr-ind}_{\pi,A} = 1\right] - 1\right| \leq \nu(\lambda).$$

Rule indistinguishability. Finally, the rule indistinguishability property informally states that it is not feasible for the Service Provider to learn any information about the rules. Here, we treat the market-compliant feature, as it ensures the privacy of the pattern signatures.

Again, we need to handle the fact that the Service Provider can create any traffic of its choice, and make use of the encrypted rules to test them and learn some information. This is a brute-force attack on the signatures, sending a lot of (random) traffic to guess the logic behind the rules. This is a typical test for security solutions and it works for both encrypted and non-encrypted rules: an intrusion detection system over encrypted traffic cannot resist such an attack, and

Experiment $\mathsf{Exp}^{rul-ind}_{\pi,\mathcal{A}}(\lambda)$
$b \leftarrow \{0,1\}$;
$(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathcal{R}_0, \mathcal{R}_1 \leftarrow \mathcal{A}_f(1^\lambda, \mathsf{param})$;
$\mathcal{B}_b \leftarrow \mathsf{RuleGen}(\mathsf{param}, \mathsf{sk}_{\mathsf{SE}}, \mathcal{R}_b)$;
$b' \leftarrow \mathcal{A}_g(\mathsf{sk}_{\mathsf{R}}, \mathcal{B}_b)$;
return $(b = b')$;

Figure 3: Rule indistinguishability experiment

it should not be considered in the model. The main point is that the SP cannot learn any information other than what is provided as output to the Detect algorithm. Our idea is then to use the high-min entropy property, which informally states that the adversary cannot obtain the rule "by chance". More formally, we use the following definition (see e.g., [10]).

DEFINITION 2 (MIN-ENTROPY). *A probabilistic adversary* $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ *has* min-entropy μ *if*

$$\forall \lambda \in \mathbb{N}\ \forall r \in \mathcal{R} : \Pr\left[r' \leftarrow \mathcal{A}_f(1^\lambda, b)\ :\ r' = r\right]\ \leq\ 2^{-\mu(\lambda)}\ .$$

$\mathcal{A}$ *is said to have* high min-entropy *if it has min-entropy* μ *with* $\mu(\lambda) \in \omega(\log \lambda)$.

The experiment related to rule indistinguishability is given in Figure 3, for an adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ with high-min entropy (considering that $\mathcal{A}_f$ and $\mathcal{A}_g$ cannot communicate one to each other, as in e.g., [10]), that can create any traffic of its choice. In a nutshell, the adversary $\mathcal{A}_f$ chooses two sets of rules $\mathcal{R}_0$ and $\mathcal{R}_1$, and one of them is used in the RuleGen procedure. The output $\mathcal{B}_b$ is then given to $\mathcal{A}_g$, that eventually outputs the bit b.

Then, an intrusion detection system over encrypted traffic π is said *rule-indistinguishable* if for any probabilistic polynomial-time $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ having high-min entropy, there exists a negligible function $\nu(\lambda)$ such that:

$$\mathsf{Adv}^{rul-ind}_{\pi,\mathcal{A}}(\lambda) = \left|2 \cdot \Pr\left[\mathsf{Exp}^{rul-ind}_{\pi,A} = 1\right] - 1\right| \leq \nu(\lambda).$$

3. TECHNICAL CONTRIBUTIONS

We review in this section related work on application-level security functions applied to encrypted traffic. We emphasize on the BlindBox approach wich is the closest to our work [25], and then we detail the main technical contributions of our work compared to BlindBox.

3.1 Related Work

Related work includes multiple recent studies that address security requirements in the context of encrypted network traffic. Authors in [16] provide a survey of main contributions on intrusion detection over encrypted traffic. The focus in [16] was exclusively on traffic analysis, using high level features of the traffic, such as packet size, entropy, and application identification. Further statistics and machine learning techniques applied to these features enable to detect specific attack types such as scan attempts [29], denial of service [7], and brute force [8, 9]. These statistical techniques are mostly agnostic to the encryption protocol. They do not support any DPI functionality, and apply in the same way to both encrypted and clear-text traffic.

In [30], authors introduce a framework called QoS2 that enables network middleboxes delivering encrypted content to support performance-oriented capabilities such as content caching. QoS2 extends web servers with the ability to serve mixed-content, including both encrypted and cleartext content within the same connection. The main concept is that content providers can separate private content (to be sent over HTTPS) from other public content that is sent over HTTP. QoS2 prevents man-in-the-middle attacks by generating checksums for all public content and sending these checksums over secure HTTPS connection. Content caching will be further supported by network middleboxes based only on the public content in each connection. Hence, QoS2 limits itself to the public content delivered over regular HTTP, but no security operations such as DPI are possible for the private content. A malicious provider may thus dissimulate malicious content in the private connection, without the possibility for network middleboxes to detect the malicious content through DPI.

In [18], authors propose a deep packet filtering protocol that leverages the Software-Defined-Networking paradigm in order to supply filtering functions over encrypted traffic. In this protocol, the service provider (SP) first informs the sender about the header fields that need to be inspected for each new network connection. Then the sender and the SP run an interactive protocol to encrypt the set of detection rules, where the sender inputs a self-generated key, and the provider inputs the set of filtering rules. The protocol in [18] suffers from two main limitations. First, the filtering rules are encrypted once for every single HTTPS connection, which makes it hard to scale in large real-world deployment scenarios. Second, the SP informs the sender about the header fields that will be inspected by the filtering rules, which requires the SP to have full access to these rules in clear-text. This clearly does not fit with the real-world constraints for the security market ecosystem.

Melis et al. propose in [20, 6] a new solution that enables the privacy-preserving outsourcing of network functions in the cloud. The contribution in [20, 6] is similar to our work by means of protecting the privacy of sensitive security policies against curious service providers. Nonetheless, it only applies in private network environments. It requires an exit gateway for the enterprise network to encrypt clear-text traffic content and to forward it to the cloud-based security function. Our solution is different as it enables to encrypt the traffic in an end-to-end way between the sender and receiver of a network connection. It preserves the privacy of both security policies and the content of network communications.

The closest work to our contribution is BlindBox [25] and its extension Embark that supports its outsourcing to the cloud [17]. BlindBox uses multi-party computation such as garbled circuits and oblivious transfers, and supports DPI through three distinct detection protocols. The first protocol enables the SP to search for patterns or keywords at random locations in the encrypted traffic. The second protocol extends the first one by enabling also to search for patterns at specific offsets. Finally, the third protocol supports probable cause decryption. It allows the SP to decrypt the traffic when it detects suspicious keywords using the first two protocols. To do so, it embeds the decryption key into the trapdoor used for pattern matching, leading thus to a full decryption of the traffic. The SP is then able to operate

full IDS over the clear-text traffic, refining the results of the first two protocols.

3.2 BlindBox Limitations

While the BlindBox solution provides valuable contributions that advance the state of the art on privacy-respectful intrusion detection, it suffers, however, from two main limitations. The first one is on scalability, and the second is its compliance with the market ecosystem for network security solutions.

The encryption protocol implemented by BlindBox requires the SP to engage with both endpoints for a garbled circuit evaluation, once for every single HTTPS connection, and for every single detection rule to be tested on the traffic. In fact, the BlindBox solution leverages the regular SSL handshake in order to generate a symmetric key for encryption, which is indeed unique for every single HTTPS connection. Hence, the SP will need to engage with the sender in order to obtain a secret key that it uses to deterministically encrypt each detection rule provided by the security editor. These rules will be then stored by the SP for DPI purposes during the entire duration of the HTTPS connection. This drastically increases the setup time for every single HTTPS connection, reaching up to 97 seconds according to [25], and so making it impractical under real-world constraints. The memory space available at the SP appliance will also need to grow polynomially according to the number of *(a)* unique endpoints to be protected, *(b)* unique HTTPS connections for every single endpoint, and *(c)* unique detection rules supported by the SP. Clearly this would restrain the usage of BlindBox to small network environments, including a limited number of both endpoints and detection rules.

Second, encrypting the detection rules once for every new HTTPS connection requires the SP to have direct access to these rules in clear-text. This is clearly inadequate with the delicate balance in the security market ecosystem, since the security editors (SE) will be very reluctant to share their detection rules with the service providers. Using the BlindBox solution, one possibility to address this limitation is for the SP to engage inline with the SE at each new connection setup in order for the SE to encrypt the rules on behalf of the SP. The whole set of encrypted rules will be then delivered by the SE to the SP in order to implement the DPI service. Although this solution seems to be feasible from a technical standpoint, it will add a huge latency during connection setup because the entire set of encrypted rules will need to be shared between the SE and SP over the network.

Finally, it is worth noticing that the full IDS functionality claimed by [25] works with an embedded decryption key which is revealed in case of a match. However, this technique does not allow the SP to have a full IDS functionality. The regular expressions are indeed not evaluated on the whole traffic, but only on already-suspicious communications. As such, it only helps in limiting the number of false positive, but does not reduce the number of false negative.

The Embark system [17] does not treat these limitations. It only aims to extend the initial BlindBox solution with the ability to securely outsource network middleboxes to the cloud.

3.3 Proposed Solutions

Our system offers to encrypt the malicious patterns used for detection *only once* for all the HTTPS connections that are delivered through the SP. This is a fundamental contribution of this paper compared to BlindBox because of the following two reasons. Firstly, it drastically decreases the connection setup time, by several orders of magnitude, since it will be no longer required for the SP to delay the connection setup until it encrypts the entire set of malicious patterns used for detection. It also decreases the memory space required to perform DPI by the SP, being dependent only on the number of detection rules, but not on the number of endpoints and/or the number of concurrent HTTPS connections. Secondly, it enables to outsource the rule encryption to the SE, that will be able to encrypt (a.k.a. to protect) its detection rules, and then to deliver them only encrypted to the SP.

Sure one can argue that the use of a same encrypted ruleset for detection may reveal the equality of matched patterns across different connections. The security provider, by comparing the matched rules over a large set of network connections (similar to brute-force attacks), may be able to recover some detection patterns, which may partially compromise the market-compliance property. While such brute-force attacks could be possible at least in theory, they apply both in the context of encrypted and clear-text network connections. Therefore, we do not consider them as a limitation to our solution since they are not inherent to our encryption protocol. Moreover, our protocol provides stronger privacy guarantees because the security provider does not even know the detection patterns it is looking for in the encrypted traffic.

To do so, our solution leverages a Decryptable Searchable Encryption (DSE) scheme that is based on public-key cryptography [12]. Our protocol enables the SE to generate a public-key pair $(\mathsf{pk}_\mathsf{R}, \mathsf{sk}_\mathsf{R})$, and a trapdoor key. The trapdoor key is used by the SE to encrypt the malicious detection patterns and to obtain trapdoors. These are further sent to the SP in order to apply DPI. Using the same DSE scheme [12], the sender encrypts the traffic using the public key of the receiver, and the SP performs DPI over the encrypted traffic using the trapdoors generated by the SE. While the use of a public-key cryptography comes with an increasing decryption overhead on the receiver, extensive evaluation performed in section 5 shows that this is a fairly reasonable price to pay in case of short HTTPS connections, which is common for most popular web sites on the Internet.

4. OUR DSE SOLUTION

Our solution is based on the so-called Decryptable Searchable Encryption (DSE) cryptographic tool [12], which is a pairing-based scheme. In this section, we briefly introduce bilinear maps and DSE, and then we describe the way we use the DSE protocol to enable intrusion detection over encrypted traffic.

4.1 Bilinear Environment

First, let us recall the notion of bilinear environment.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$ be three finite multiplicative abelian groups of large prime order q. Let g_1 be a generator of $\mathbb{G}_1$ and g_2 be a generator of $\mathbb{G}_2$. We assume that there exists an asymmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$, such that for all $a, b \in \mathbb{Z}_q$:

1. $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$,
2. $e(g_1^a, g_2^b) = 1$ iff $a = 0$ or $b = 0$,

3. $e(\cdot,\cdot)$ is efficiently computable.

In the sequel, the tuple $(q, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_t, e(\cdot,\cdot))$ is referred to as a bilinear environment.

4.2 Decryptable Searchable Encryption

Formal definition. A Decryptable Searchable Encryption scheme is a public-key encryption with keyword search (a.k.a. PEKS) with the additional possibility for the owner of a specific secret key sk to decrypt the ciphertext. Informally, the keyword search property states that given a ciphertext c and a keyword w, this is feasible to test whether the ciphertext matches the keyword or not. This ability is obvious for the owner of the decryption key, but can also be transferred to anyone under the form of a keyword-specific trapdoor $T(w)$. A DSE scheme is composed of the following procedures.

- KeyGen is the key generation algorithm which takes as input a security parameter λ and outputs a public key pk, a decryption private key sk and a trapdoor secret key tk.
- Enc is the encryption procedure and takes as input a keyword w and the public key pk. It outputs a ciphertext c related to w and pk.
- TrapGen is the generation of a trapdoor, which takes as input a keyword w and the trapdoor key tk and outputs a trapdoor $T(w)$.
- Test is a testing algorithm. It takes as input a ciphertext c and a trapdoor $T(w)$ and outputs 1 if c is an encryption of w and 0 otherwise.
- Dec is the decryption procedure. It takes as input a ciphertext c and the private key sk. It outputs the related keyword w.

As detailed in [12], a DSE scheme should verify a particular form of indistinguishability against Chosen Ciphertext Attacks (CCA). This property is related to an experiment such that given the public key pk, the adversary outputs two different keywords w_0 and w_1 and, after having randomly chosen a bit $b \in \{0,1\}$, the challenger outputs c_b as the ciphertext related to w_b. Eventually, the adversary outputs a bit b' and wins if $b' = b$. Throughout the experiment, the adversary may send queries $w \notin \{w_0, w_1\}$ to a trapdoor derivation oracle and queries $c \neq c_b$ to a decryption oracle. We say that a DSE satisfies *indistinguishability under a chosen-plaintext attack* if for every legitimate adversary, the advantage to win this experiment is negligible.

A DSE scheme. In [12], the authors propose the following efficient construction. Let $(q, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_t, e(.,.))$ be a bilinear environment and let F, G, H be three hash functions that are each modeled as a random oracle.

- $\mathsf{KeyGen}(1^\lambda)$. This procedure randomly chooses $x, x' \in \mathbb{Z}_q$ and computes $y = g_1^x$ and $y' = g_1^{x'}$. We have $\mathsf{sk} = x$, $\mathsf{tk} = x'$ and $\mathsf{pk} = (y, y')$.
- $\mathsf{Enc}(w, \mathsf{pk})$. This procedure chooses $r \in \mathbb{Z}_q$ and computes $c_1 = g_1^r$, $(s_1, s_2) = G(y^r)$, $c_2 = s_1 \oplus w$, $c_3 = g_1^{s_2}$, $u = e(y'^{s_2}, F(w))$ and $c_4 = H(u) + r \pmod q$. The ciphertext is $c = (c_1, c_2, c_3, c_4)$.
- $\mathsf{TrapGen}(w, \mathsf{tk})$. This procedure outputs the trapdoors $T(w) = F(w)^{x'}$.
- $\mathsf{Test}(c, T(w))$. This procedure first computes the value $u = e(c_3, T(w))$ and $r = c_4 - H(u) \pmod q$. If $c_1 \neq g_1^r$, it returns 0. Otherwise, it computes $s = y^r$, $(s_1, s_2) = G(s)$ and $w = c_2 \oplus s_1$. If $c_3 \neq g_1^{s_2}$, it returns 0. Otherwise 1 is returned.
- $\mathsf{Dec}(c, \mathsf{sk})$. This procedure computes $s = c_1^x$, $(s_1, s_2) = G(s)$ and $w = c_2 \oplus s_1$. If $c_3 \neq g_1^{s_2}$, it returns $\perp$. Otherwise, it computes $u = e(y'^{s_2}, F(w))$, $r = c_4 - H(u) \pmod q$ and checks whether $c_1 = g_1^r$. If this condition is satisfied, it returns w. Otherwise $\perp$ is returned.

4.3 Overview

Based on the above DSE tool, we now provide a solution for detecting a malicious content over an encrypted traffic. For example, in the case of an IDS in charge of filtering out malicious content, the protocol should be set up in a way that the Service Provider (SP) can detect malicious content, and if the traffic is not compromised, SP should be able to forward the encrypted traffic to the receiver R without getting any information.

At first, we consider that the traffic between a sender S and a receiver R is encrypted using a DSE (in particular, a modified version of the one described in the previous section, as we will see below). This gives us the main desired properties: the confidentiality of the traffic and the possibility to detect a malicious traffic by using equality tests.

Adequacy to security market. This way, we also obtain an additional property: the Service Provider (SP) does not know the searched pattern as it only knows the trapdoor $T(w)$, and not the keyword w itself. But when considering the efficient instantiation given in [12], we have to face the problem that when the Test procedure is successful, it also permits to output the matching keyword, as one can compute $w = c_2 \oplus s_1$. We then modify the above scheme to achieve our goal. In our protocol below, we compute c_4 as $c_4 = H(u) + a \pmod q$ where $a \in \mathbb{Z}_q^*$ belongs to the public key given by SE. Using such trick, there is no way for the SP to obtain information about the keywords, even when a match is found, as we will see in security proofs. We are thus compliant with the current security market, contrary to the BlindBox solution.

Scalability. To achieve our second goal, we use another trick which consists in observing that the trapdoor and decryption keys are totally independents in the Furh-Paillier efficient construction [12]. Then, one can independently compute the public key y related to the "encryption" phase and the public key y' related to the "test" phase. The consequence is that the Security Editor SE can compute its own trapdoor key $\mathsf{tk} = x'$ (and the corresponding public key $y' = g_1^{x'}$) without needing to know the decryption key $\mathsf{sk} = x$ (and the corresponding public key $y = g_1^x$) of a receiver. Then, we can manage several key pairs $(x_i, y_i)_i$ such that $x_i \in_R \mathbb{Z}_q^*$ and $y_i = g_1^{x_i}$: one for each Receiver R_i. This way, we easily obtain scalability as the SE only publishes one single set of trapdoors to permit the SP to detect malicious traffic for as many endpoints as it wants. Again, this is much better than the BlindBox solution, both regarding

the time complexity of the setup, and the space complexity for SP.

Overview of the protocol. Our solution BlindIDS-DSE exploits this double property of DSE to build a deep packet inspection protocol over encrypted data using the cryptographic blocks mentioned in [12]. Upon receiving or generating the traffic and prior to encryption, the Sender (S) applies a tokenization algorithm to the traffic. There are two tokenization algorithms that generates different performances at detection time. The tokenization algorithm produces fixed-length (window-based tokenization) or variable-length (delimiter-based) keywords, as in [25].

BlindIDS-DSE protocol is composed of four agents, each implementing different modules acting in the detection system. The protocol runs as follows.

- **System setup** (Setup)
  - RG runs the DSE key generation algorithm to generate the secret trapdoor key $\mathsf{sk}_{\mathsf{SE}}$, and the related public key pk_{SE}.
  - Independently, each receiver R runs the DSE key generation protocol in order to generate a public key pair $(\mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}})$.
- **Rule generation** (RuleGen)
  - SE uses the trapdoor generation procedure with keywords of fixed-length in order to produce trapdoors $T(w)$ for each keyword w related to attacks.
  - SE sends the trapdoors to SP for detection.
- **Sending preparation** (Send)
  - At first, SP establishes connection with both S and R.
  - S produces fixed-length tokens from the traffic, and, using the specific receiver R's public key, encrypts each token to which it appends its position in the payload (in order for the reverse tokenization algorithm to be able to reconstruct the traffic), using the DSE encryption algorithm Enc.
  - S sends the encrypted tokens to R.
- **Detection** (Detect)
  SP runs equality tests for each encrypted token and entries in a tree data structure containing trapdoors. If there is a match for all keywords in the rule, SP's detection module outputs 0, else it outputs 1. Accordingly, SP either drops the packet, or sends the tokens to R, depending on the security policy enforced by SP.
- **Validation and packet reconstitution** (Receive)
  R receives the set of encrypted tokens and runs the reverse tokenization algorithm in order to reconstruct the message, using the decryption procedure Dec. Based on the tokenization procedure that is implemented by the sender, the position of each decrypted keyword may be forwarded with the encrypted token in order to enable the receiver to reverse the tokenization algorithm and to reconstruct the traffic in clear-text.

4.4 Detailed Description

Let $(q, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_t, e(.,.))$ be a bilinear environment and let F, G, H be three hash functions (modeled as random oracles).

- Setup. The system setup consists in two independent steps of key generation.
  - The SE executes the $\mathsf{DSE.KeyGen}(1^\lambda)$ procedure, keeps only the trap generation key $\mathsf{tk} = x' \leftarrow \mathbb{Z}_q$. It publishes $\mathsf{pk}_{\mathsf{RG}} = g_1^{x'}$, along with some random string $a \in \mathbb{Z}_q^*$.
  - The receiver R also executes $\mathsf{DSE.KeyGen}(1^\lambda)$ to obtain its secret decryption key $\mathsf{sk}_R = x \leftarrow \mathbb{Z}_q$. It publishes the associated public key $\widetilde{\mathsf{pk}_{\mathsf{R}}} = g_1^x$.
  - The public key for a receiver R is the pair $\mathsf{pk}_{\mathsf{R}} = (\mathsf{pk}_{\mathsf{SE}}, \widetilde{\mathsf{pk}_{\mathsf{R}}})$.
- RuleGen. For each keyword w_i to be searched for in the traffic, the SE executes $\mathsf{TrapGen}(w_i, \mathsf{tk})$ of the DSE scheme. It returns $T_i = F(w_i)^{x'}$. The SE then sends the set of traps $\mathcal{T} = \{T_1, ..T_l\}$ to the SP.
- Send.
  - S splits the traffic into tokens $t_1, \ldots, t_n$.
  - For each token t_i, S draws r_i uniformly at random in $\mathbb{Z}_q$ and executes $\mathsf{Enc}(t_i, \mathsf{pk}_{\mathsf{R}})$ of our modified DSE scheme. This procedure computes the following:

$$\begin{aligned} c_{1,i} &= g_1^{r_i}; \\ (s_1, s_2)_i &= G(\widetilde{\mathsf{pk}_{\mathsf{R}}}^{r_i}); \\ c_{2,i} &= s_{1,i} \oplus t_i; \\ c_{3,i} &= g_1^{s_{2,i}}; \\ u_i &= e(\mathsf{pk}_{\mathsf{SE}}^{s_{2,i}}, F(t_i)); \\ c_{4,i} &= H(u_i) + a \mod q. \end{aligned}$$

    The ciphertext for each token t_i is the quadruplet $c_i = (c_{1,i}, c_{2,i}, c_{3,i}, c_{4,i})$.
  - The encrypted traffic E is the collection of the ciphertexts for each token. S sends E to R.
- Detect. When the Service Provider intercepts an encrypted traffic E between S and R, it executes the procedure $\mathsf{DSE.Test}(c_i, T_j)$ on each ciphertext to look for matching signatures. The Test procedure includes of the following steps.
  - The SP first computes the value $u_i = e(c_{3,i}, T_j)$, and then computes $a' = c_{4,i} - H(u_i) \mod q$.
  - If $a \neq a'$, it returns 1. This means that t_i and r_j are different and so that the signature does not match with the traffic. Otherwise, it returns 0.

  If the procedure returns 0 the Service Provider generates an alert. It may also return the auxiliary information $\mathsf{aux} = T_i$, which is the trapdoor that matched. Otherwise, the traffic is forwarded to R.
- Receive. For each ciphertext, R executes $\mathsf{Dec}(c_i, \mathsf{sk}_{\mathsf{R}})$ of the modified DSE scheme, made of the following steps.

– R computes $s_i = c_{1,i}^x$ and use G to retrieve the pair $(s_{1,i}, s_{2,i})$. Then, it calculates $t_i = c_{2,i} \oplus s_{1,i}$.

– If $c_{3,i} \neq g_1^{s_{2,i}}$, the procedure returns $\perp$.

– Else, it computes $u_i = e(\mathsf{pk}_{\mathsf{SE}}^{s_{2,i}}, F(t_i))$.

– If $H(u_i) - c_{4,i} \neq a$, then return $\perp$.

– It returns t_i.

While our protocol is primarily intended for pattern-matching, we can use the same secret-key embedding technique as BlindBox for evaluating regular expressions on suspicious traffic. See [25] for details.

4.5 Security

Following the security model, we have the three following theorems, where π is our intrusion detection system over encrypted traffic. Proofs are given in Appendix A.

THEOREM 1. *Our scheme π is detectable provided that there is no collision in the trapdoor generation function.*

We prove the indistinguishability under two assumptions: the computational Diffie-Hellman problem (CDH) and the GDDHE assumption, the latter one being introduced in [11]. For the sake of simplicity, we give a slightly informal version of the GDDHE assumption.

DEFINITION 3 (CDH). *Let g, g^a, g^b be three elements in $\mathbb{G}$, a probabilistic polynomial-time adversary has a negligible probability to compute g^{ab}.*

DEFINITION 4 (((P, Q, f) − GDDHE)). *Let s, n be positive integers and $P, Q \in \mathbb{F}_p[X_1, \ldots, X_n]^s$ be two s-tuples of n-variate polynomials over $\mathbb{F}_p$. Let $f \in \mathbb{F}_p[X_1, \ldots, X_n]$ a polynomial which is linearly independent from P and Q. Given $H(x_1, \ldots, x_n) = (g^{P(x_1,\ldots,x_n)}, \tilde{g}^{Q(x_1,\ldots,x_n)}) \in \mathbb{G}_1^s \times \mathbb{G}_2^s$ and $T \in \mathbb{G}_T$, a probabilistic polynomial-time adversary has a negligible probability to successfully decide if $T = g_T^{f(x_1,\ldots,x_n)}$.*

THEOREM 2. *Our scheme π is traffic-indistinguishable under the* CDH *and the* GDDHE *assumptions in the random oracle model.*

THEOREM 3. *Our scheme π is rule-indistinguishable for rules of high min-entropy, in the random oracle model.*

5. EXPERIMENTS

This section introduces our experimental setup, including the implementation details and evaluation of our encryption protocol. It aims at evaluating the functionalities and performance of our solution, and to compare them with similar state of the art solutions such as BlindBox.

5.1 Implementation

We implemented our protocol on an Intel(R) Xeon(R) with a E5-1620 CPU with 4 cores running at 3.70GHz under a 64-bit Linux OS. We used the optimal Ate library [21], written in C, over their default 254-bits Barreto-Naehrig curve. We implemented our protocol in Java 8, using the Java API provided by [21] and built with Java Standard Edition u112. We used SHA 256 as hash function, and Java SecureRandom class to generate random numbers.

Dataset	Entries	Supported entries	%
Malware blocklists [1]	1,250	1,250	100%
URL blacklists [3]	4,546,341	4,546,341	100%
Yara rules [4]	256	198	77.3%
Snort comm. rules [2]	3,467	2,606	75%

Figure 4: Datasets evaluated using our solution

5.2 Evaluation

We evaluate in this paragraph the functionalities and performance of our solution. The functional evaluation leverages the ability of our protocol to detect attacks using standard detection signatures such as malware blocklists, content filtering and data leak protection, parental control, and generic Snort IDS rules. The performance evaluation leverages the overhead at the client-server side, and at the detection appliance supplied by the service provider. It also leverages the ability of our solution to scale under real world network configurations and traffic rates.

5.2.1 Functional evaluation

Overall, our solution provides similar functional properties as for Blindbox, while preserving in the same time the privacy of detection rules with respect to the service provider. We refer to multiple public datasets in order to validate the detection functionalities of our solution. These datasets, as summarized in Figure 4, provide a wide range of functionalities such as malware domains lists [1], parental control through detection of illicit ads, pornography, and violent or terrorist content [3], Yara rules providing malware detection [4], and Snort community rules [2].

First, we evaluated the ability of our solution to implement the detection rules supplied by each separate dataset. The malware blocklists dataset [1] provides a public list of fully qualified URLs that supply malware infected content. The URL blacklists [3] dataset has a similar structure, but covers a wider range of illicit content including also parental control. Both URL datasets may be applied through comparison and perfect match with the URL header fields for outgoing HTTP requests. Our solution, using both delimiter- and window-based tokenization, perfectly matches these URLs over encrypted traffic, providing a 100% detection rate. The Yara rules dataset [4] has a different structure. It provides DPI rules that search for hexadecimal strings, text keywords, and generic regular expressions that characterize malware infected content in traffic payload. The public Yara documentation provides general guidelines to create Yara rules, and best practices to enhance detection performance. In this scope, it deprecates the use of full regular expressions, unless when necessary. Regular expressions are inherently slow, and so the Yara project recommends replacing them where possible with strings search, along with jumps and wild-cards. Our Yara dataset includes 256 distinct rules, where 198 rules (77.3%) include only keyword searches that can be fully detected over encrypted traffic using our solution. Remaining rules include regular expressions that require direct access to the traffic. These rules are indeed supported by our solution using the decryption property, which allows the SP to decrypt a suspect connection when it contains malicious content detected using keyword search. Snort rules [2] have similar properties as for Yara rules, and so we obtained almost the same results using Snort rules as for Yara rules. Using our DSE proto-

col, we were able to evaluate 2,606 rules that do not contain full regular expressions, out of the 3,467 initial Snort rules (75%). The remaining rules are supported by our solution, the same as for Yara rules, using the embedded decryption property.

Second, we evaluated our ability to detect all attacks and malicious content that are covered by the detection rules in our public dataset. Since our DSE protocol perfectly matches rule keywords with encrypted traffic tokens, as discussed in section 4.5, our ability to detect these attacks solely depends on the tokenization strategy, the same as for the BlindBox solution. To confirm this hypothesis, we conducted the same experiment as in the BlindBox paper [25], using the same ICTF dataset [28]. This dataset includes network traces collected during a *capture the flag* exercise, including multiple teams that each had the task of maintaining a set of services such that they remain available and uncompromised throughout the contest. The results of our experiment confirmed our initial hypothesis. We achieved almost a similar detection accuracy as in BlindBox [25] (because for this experiment, our DSE protocol uses the same tokenization strategy applied to the same dataset), including 96.5% of the attack keywords and 98.3% of the attack rules that would have been detected with Snort.

5.2.2 Performance evaluation

This section evaluates the performance of our solution from the perspective of the different actors. First, it leverages the overhead on the sender and receiver when tokenizing and encrypting data using our DSE protocol. Second, it leverages the overhead on the SP, including the time and memory required to implement our detection protocol.

Our performance results, including also a benchmark with the standard SSL inspection technique and the BlindBox solution [25], are summarized in figure 5. As an overall assessment, our solution drastically reduces, by 6 orders of magnitude compared to BlindBox, the memory space required on the DPI appliance, making it much closer to real world configurations. It also reduces, by 3 orders of magnitude, the setup time for new HTTPS connections. The setup time in our solution does not depend on the number of detection rules as in BlindBox, and so it is similar to the connection time for standard HTTPS connections. On the other hand, and because of using a public-key encryption protocol, our solution achieves a lower performance for data encryption compared to BlindBox. Using our solution, a sender would need 27ms to encrypt the content of a network packet with an MTU of 1500 bytes. A typical web page such as CNN would take in average 2.3s to be loaded using our solution, whereas the same page would take nearly 97s to be loaded using BlindBox (mostly because of the connection setup time). We consider this additional overhead for data encryption as a reasonable price to pay in order to reduce the overall setup time for new HTTPS connections. Hence, our solution is much more adapted to short and medium-lived connections, which is the case for most standard web pages on the Internet. In the remaining of this section, we discuss more in details all these findings.

Overhead on the sender and receiver. The overhead on the sender and receiver covers the time for connection setup and encryption before the data is shared over the network.

Connection setup time. Since we use a public-key encryption protocol that does not initially requires an SSL handshake, the setup time is the same as for SSL/TLS connections, as shown in figure 5. Compared to BlindBox, the setup time using our solution does not depend on the detection rules supplied by the SE. Our DSE protocol enables the SE to generate a trapdoor key associated with its public-key pair $(\mathsf{pk}_\mathsf{R}, \mathsf{sk}_\mathsf{R})$. The trapdoor key is used only once by the SE to encrypt the keywords in every new detection rule. The trapdoors are further delivered by the SE to the SP, with no participation from the sender and receiver. Hence, our solution decreases by six orders of magnitude compared to BlindBox the setup time when using 3 thousand detection rules. More interestingly, it adds no overhead for connection setup compared to the standard SSL/TLS protocol.

Data encryption time. This time, however, is longer using our solution than both standard SSL/TLS and BlindBox. For each token included in the traffic, it corresponds to the time for encrypting this token using our DSE protocol, and that we evaluated to $729\mu s$ in average for tokens of 128 bits length. The overhead for data encryption is primarily due to the use of the DSE protocol, which uses public-key cryptography, as opposed to the symmetric-key cryptography as used in SSL/TLS and BlindBox. To better qualify the impact on end-user experience, we loaded multiple popular web sites such as CNN, Facebook, Twitter, BBC, and Bank-of-America, and we evaluated the average overhead, in terms of page load time, compared to both SSL/TLS and BlindBox. The results of our experiment are summarized in figure 6. Our solution adds a significant overhead compared to the standard HTTPS protocol because of its long data encryption time. This data encryption time for our solution is rapidly compensated with the very long connection setup time when using BlindBox. In the overall, the average loadtime using our solution remains acceptable for short and medium-sized web pages, but it increases considerably for websites supplying large content.

Overhead on the service provider. We evaluate the overhead for the SP by measuring the memory space that needs to be available, as well as the time required to perform detection, according to the number of detection rules and the size of network connections.

Detection Time. The only significant overhead for using our solution compared to BlindBox is the detection time on the DPI appliance. Using a ruleset of 3 thousand rules, each rule including in average 3 trapdoor tokens (nearly 10 thousand tokens in total, which is the same as the experimental setup used to evaluate the BlindBox solution in [25]), it takes almost 74 seconds for the SP to apply DPI over an encrypted packet. Compared to the 97s for connection setup using BlindBox, our solution still reduces by 25% the overall overhead (including both setup and detection time). However, it still does not scale well for realtime intrusion detection in large networks. This is mainly because of the `Test` procedure for our DSE protocol. It tests, using a cryptographic operation, each trapdoor in the detection ruleset against each ciphertext in the network traffic. It is thus different from the `Detect` procedure implemented by BlindBox, which searches for perfect matching between the encrypted rules and the encrypted tokens.

Role	Description	SSL inspection	BlindBox	Our solution
Sender/ Receiver	Setup (1 keyword)	73ms	588ms	73ms
	Setup (3K Rules)	73ms	97s	73ms
	Encrypt (128 bits)	13ns	69ns	729μs
	Encrypt (1500 bytes)	3μs	90μs	27ms
Service provider *(Detection time)*	1 Rule, 1 Token	Not applicable	20ns	691μs
	1 Rule, 1 Packet	Not applicable	5μs	41.3ms
	3K Rules, 1 Token	Not applicable	137ns	700ms
	3K Rules, 1 Packet	Not applicable	33μs	74s
Service provider *(RAM usage)*	1 Rule, 1 Connection	Not applicable	1.75MB	0.2KB
	3K Rules, 1 Connection	Not applicable	5.12GB	0.58MB
	1 Rule, 100 Connections	Not applicable	175MB	0.2KB
	3K Rules, 100 Connections	Not applicable	512GB	0.58MB

Figure 5: Performance of our solution during connection setup and detection, and benchmark with the standard SSL inspection technique and the BlindBox solution

Website	Size	HTTPS	BlindBox	Our solution
CNN	131KB	0.073	97.008	2.373
Facebook	74KB	0.073	97.004	1.073
Twitter	284KB	0.073	97.017	5.073
BBC	196KB	0.073	97.011	3.573
BoA	74KB	0.073	97.004	1.073

Figure 6: Loadtime (in seconds) for some popular websites

However, by removing the overhead for connection setup on the sender, and partially shifting this overhead to the detection procedure on the SP side, our solution provides a significant advantage compared to BlindBox. First, the SP may use additional heuristics, such as domain or IP reputation, in order to identify possibly suspicious flows, and subsequently to inspect only encrypted connections towards risky destinations. Moreover, the SP may use load balancing to distribute computation over multiple servers, which may not be feasible for the sender. Our solution is also well suited for offline usage, during investigation and post-intrusion forensics. Although this functionality may be at least theoretically supported using BlindBox, the SP would need to store and manage the garbled circuits generated for each single HTTPS connection. This adds a significant overhead compared to the contribution in this paper. Our solution enables the SE to generate the trapdoors only once for every new detection rule. These trapdoors are further applied by the SP to all encrypted connections that are inspected by the DPI appliance. Finally, our solution is better suited than BlindBox for investigation and post-intrusion forensics also because it allows the SP to retroactively test newly supplied detection rules by the SE. These rules may capture zero-day attacks that were yet unknown at the time when the encrypted connection had occurred. The SE can indeed encrypt these new rules using its trapdoor key and deliver them to the SP. This functionality is not supported using BlindBox because it requires the sender to prepare a garbled circuit and to send it to the SP, which indeed may not be feasible in the context of post-intrusion forensics.

Memory usage. While the detection time for the SP is much longer using our DSE protocol than for BlindBox, our approach drastically decreases the memory space that needs to be available on the DPI appliance. This is mainly because of replacing the garbled circuits used in BlindBox by generic trapdoors that are derived from the malicious keywords in the detection rules. In fact the garbled circuits used in BlindBox are prepared by the sender for every single HTTPS connection. Each garbled circuit has a size of 599KB, and needs be stored by the SP during the entire duration of this connection. Therefore, the memory space required on the DPI appliance will grow linearly with respect to both the number of rules and the number of concurrent HTTPS connections. As shown in the table of figure 5, the memory space required to store garbled circuits for 100 concurrent HTTPS connections, and for 3,000 detection rules, is evaluated to 512GB RAM.

However, using our solution, the SE generates a trapdoor only once for every single detection keyword. The trapdoors are further applied by the SP for all the encrypted connections that are inspected by the DPI appliance. Each trapdoor has a unique size of 508 bits, which in turn does not depend on the size of the encryption key. The required memory space does no longer depend on the number of concurrent connections, but only on the number of detection rules, the same as for all DPI appliances used for clear-text intrusion detection. In case of 100 concurrent HTTPS connections, and for 3,000 detection rules, our BlindIDS solution requires only 0.58MB RAM.

6. CONCLUSION

In this paper, we presented BlindIDS, a new system that operates Deep Packet Inspection (DPI) directly over encrypted traffic. We formally introduced our solution based on a security model that represents an ideal intrusion detection system over encrypted traffic. Then we provided appropriate security proofs in order to validate the main features of our system. To the best of our knowledge, BlindIDS is the first system that bridges the gap between network security and privacy, while also preserving the delicate balance in the security market ecosystem. It enables security editors and service providers to securely collaborate in order to provide value added security services that also preserve the confidentiality of end users' data. Our solution is beneficial for everyone: users will preserve their privacy, security editors will be able to protect their distinctive attack signatures, and service providers will be able to deliver intrusion detection services without affecting the privacy of end user traffic. We made a prototype implementation of BlindIDS, and we performed extensive evaluation in order to evaluate the functionality and performance of this solution. Our experiments show that, compared to similar state of the art solutions such as [25], BlindIDS enhances by several orders of magnitude both the connection setup time and the resources required to perform DPI on the security appliance.

7. REFERENCES

[1] Malware domain list. https://www.malwaredomainlist.com/mdl.php, 2016.

[2] Snort. https://www.snort.org/downloads/, 2016.

[3] Url blacklist. http://www.urlblacklist.com/?sec=home, 2016.

[4] Yara rules repository. https://github.com/Yara-Rules/rules, 2016.

[5] M. and Markets. Threat intelligence security market by solution - global forecast to 2020. In *MarketsandMarkets report TC 3591*, 2015.

[6] H. J. Asghar, L. Melis, C. Soldani, E. D. Cristofaro, M. A. Kaafar, and L. Mathy. Splitbox: Toward efficient private network function virtualization. In *workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 7–13, August 2016.

[7] M. Augustin and A. Balaz. Intrusion detection with early recognition of encrypted application. In *IEEE Conference on Intelligent Engineering Systems (INES)*, June 2011.

[8] M. Barati, A. Abdullah, R. Mahmod, N. Mustapha, and N. I. Udzir. Feature selection for ids in encrypted traffic using genetic algorithm. In *International Conference on Computing and Informatics (ICCI)*, pages 279–285, 2013.

[9] M. Barati, A. Abdullah, N. I. Udzir, M. Behzadi, R. Mahmod, and N. Mustapha. Intrusion detection system in secure shell traffic in cloud environment. In *Journal of Computer Science*, volume 10, 2014.

[10] M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 360–378, 2008.

[11] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 440–456, 2005.

[12] T. Fuhr and P. Paillier. Decryptable searchable encryption. In *Provable Security*, volume 4784, pages 228–236, 2007.

[13] R. Holland, S. Balaouras, and J. Blackborow. The state of the cyberthreat intelligence market. In *Forrester report*, 2015.

[14] L.-S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged ssl certificates in the wild. In *IEEE Symposium on Security and Privacy*, 2014.

[15] J. Jarmoc. Ssl interception proxies and transitive trust. In *Black Hat Europe*, 2012.

[16] T. Kovanen, G. David, and T. Hamalainen. Survey: Intrusion detection systems in encrypted traffic. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, volume 9870 of *LNCS*, pages 281–293, 2016.

[17] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu. Embark: Securely outsourcing middleboxes to the cloud. In *Usenix NSDI*, 2016.

[18] Y.-H. Lin, S.-H. Shen, M.-H. Yang, D.-N. Yang, and W.-T. Chen. Privacy-preserving deep packet filtering over encrypted traffic in software-defined networks. In *IEEE Conference on Communications (ICC)*, 2016.

[19] R. McMillan and K. Pratap. Market guide for security threat intelligence services. In *Gartner report (G00259127)*, 2014.

[20] L. Melis, H. J. Asghar, E. D. Cristofaro, and M. A. Kaafar. Private processing of outsourced network functions: Feasibility and constructions. In *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 39–44, March 2016.

[21] S. MITSUNARI. A fast implementation of the optimal ate pairing over bn curve on intel haswell processor. Cryptology ePrint Archive, Report 2013/362, 2013. http://eprint.iacr.org/2013/362.

[22] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. In *IEEE Network*, volume 8, pages 26–41, 1994.

[23] P. Paganini. French government anssi responsible of a mitm against google ssl-tls. In *Security Affairs magazine*, 2013.

[24] Sandvine. Encrypted internet traffic: A global internet phenomena spotlight. In *Sandvine report on Global Internet Phenomena*, 2016.

[25] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[26] T. Skybakmoen, J. Pathak, B. Venkateswaran, M. Spanbauer, and B. Walder. Breach detection systems comparative report. In *NSS Labs Security Value Map*, 2016.

[27] B. Stricker. Uncovering hidden threats within encrypted traffic: A study of north america & emea. In *A10 and Ponemon institute report*, 2016.

[28] G. Vigna. The uc santa barbara ictf competition. https://ictf.cs.ucsb.edu/#/, 2016.

[29] A. Yamada, Y. Miyake, and K. Takemori. Intrusion detection for encrypted web accesses. In *Advanced Information Networking and Applications Workshops*, 2007.

[30] Z. Zhou and T. Benson. Towards a safe playground for https and middleboxes with qos2. In *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, 2015.

APPENDIX

A. SECURITY PROOFS

A.1 Detection Property

PROOF. We consider a successful adversary against the detection property of our scheme. Based on the detection experiment given in Section 2.3, this implies that there exists a keyword w^* such that:

1. SE has published in $\mathcal{B}$ the trapdoor $T^* = F(w^*)^{x'}$;
2. S output a valid ciphertext $c^* = (c_1^*, c_2^*, c_3^*, c_4^*)$;
3. the detection test outputs 1;
4. the decryption of c^* gives back w^*.

In particular, the second point implies that:

$$\begin{aligned} c_3^* &= g_1^{s_2^*}, \\ H(u^*) - c_4^* &= a, \\ c_1^* &= g_1^{r^*}, \end{aligned}$$

where

$$\begin{aligned} s^* &= (c_1^*)^x, \\ (s_1^*, s_2^*) &= G(s^*), \\ t^* &= c_2^* \oplus s_1^*, \\ u^* &= e(\mathsf{pk}_{\mathsf{RG}}^{s_2^*}, F(t^*)). \end{aligned}$$

The third point implies that $c_4^* - H(\tilde{u}) \neq a$ with $\tilde{u} = e(c_3^*, T^*)$. We then obtain that

$$\begin{aligned} e(c_3^*, T^*) &\neq e(\mathsf{pk}_{\mathsf{RG}}^{s_2^*}, F(t^*)), \\ e(c_3^*, T^*) &\neq e(g_1^{x' \cdot s_2^*}, F(t^*)), \\ e(c_3^*, T^*) &\neq e(g_1^{s_2^*}, F(t^*)^{x'}). \end{aligned}$$

As $T^* = F(w^*)^{x'}$ and since, in the detection experiment, we have $\mathsf{Detect}(T, \mathcal{R}) = 0$, which is equivalent to $t^* = w^*$. Then

$$e(c_3^*, T^*) \neq e(g_1^{s_2^*}, T^*).$$

But since the ciphertext is correct, the equality $c_3^* = g_1^{s_2^*}$ holds (see above). Then

$$e(c_3^*, T^*) \neq e(c_3^*, T^*),$$

which is unconditionally infeasible.

We finally evaluate the probability of false positive, that is a legitimate token t that would match a keyword w, with $t \neq w$. That implies that:

$$\begin{aligned} e(\mathsf{pk}_{\mathsf{SE}}^{s_2}, F(t)) &= e(c_3, F(w)^{x'}), \\ e(g_1^{x' \cdot s_2}, F(t)) &= e(g_1^{s_2}, F(w)^{x'}), \\ e(g_1, F(t))^{x' \cdot s_2} &= e(g_1, F(w))^{s_2 \cdot x'}. \end{aligned}$$

We finally obtain that $F(w) = F(t)$, that is a collision in the hash function F, which happens with negligible probability. This concludes the proof. □

A.2 Traffic Indistinguishability Property

Proof. We prove Theorem 2 with an hybrid argument. We construct a simulator $\mathcal{S}$ for the Send procedure against which an adversary $\mathcal{A}$ against the $tr - ind$ experiment has advantage 1/2. In order to obtain this, we want the output of $\mathcal{S}$ to be perfectly random.

We can assume that the messages sent by $\mathcal{A}$ to the challenger do not match any rule. Indeed, if they match the same rule, indistinguishability is meaningless, and if they match different rules, indistinguishability is trivially broken, though one-wayness remains.

Game G_0. This game is the original game, which means that $\mathcal{S}$ follows the Send procedure. The adversary chooses T_0 and T_1 and the challenger encrypts one of them. The adversary's view is the following encrypted traffic, where b is the bit chosen by the challenger.

$$\begin{aligned} c_{1,b} &= g_1^{r_b}; \\ c_{2,b} &= s_{1,b} \oplus T_b; \\ c_{3,b} &= g_1^{s_{2,b}}; \\ c_{4,b} &= H(u_b) + a \mod q. \end{aligned}$$

Throughout the sequence of games, we denote S_i the probability that the bit b' output by $\mathcal{A}$ is b for game G_i. We have:

$$\mathsf{Adv}_{\pi,\mathcal{A}}^{tr-ind}(\lambda) = \left| 2 \cdot \Pr\left[\mathsf{Exp}_{\pi,A}^{tr-ind} = 1 \right] - 1 \right| = |2 \cdot S_0 - 1|.$$

We stress that c_1 does not depend on T_b and is fully randomized as r_b is a random element of $\mathbb{Z}_q$.

Game G_1. We then prove that c_2 is also a random element in $\mathbb{Z}_q$ (in the random oracle model). As the computation of c_2 corresponds to a one-time pad, our work is to prove that $s_1$is a random element. In fact, s_1 is the output of the hash function G, modelled here as a random oracle. The only possibility for the adversary is then to obtain s_1 by a request to the random oracle. The input of G is $\widetilde{\mathsf{pk}_{\mathsf{R}}}^{r_b}$, which means that the adversary can recover s_1 if he is able to compute $g_1^{x \cdot r_b}$ from $c_1 = g_1^{r_b}$ and $\widetilde{\mathsf{pk}_{\mathsf{R}}} = g_1^x$, which exactly corresponds to an instance of the CDH problem.

However, an adversary $\mathcal{B}$ against CDH cannot directly use $\mathcal{A}$ to win the game. One possibility could have been to make us of the bilinear map but, as we use an asymmetric pairing, he cannot verify which of $\mathcal{A}$'s requests to the random oracle is indeed the answer to the CDH challenge. Thus, denoting g_G the number of $\mathcal{A}$'s requests to the random oracle G, for all adversary $\mathcal{B}$ against CDH, we obviously have:

$$|\Pr(S_0) - \Pr(S_1)| \leq \frac{1}{q_G} \mathsf{Adv}_{\mathcal{B}}^{\mathsf{CDH}}(\lambda).$$

Game G_2. We now focus on c_3. In fact, as $c_3 = g_1^{s_2}$, this step can be treated as previously since the only way for the adversary to use c_3 to find b is to obtain s_2, making the right request to the random oracle G. It is then obvious to prove that for all adversary $\mathcal{B}$ against CDH, we have:

$$|\Pr(S_1) - \Pr(S_2)| \leq \frac{1}{q_G} \mathsf{Adv}_{\mathcal{B}}^{\mathsf{CDH}}(\lambda).$$

Game G_3**.** We finally focus on c_4. In this game, we replace $u = e(\mathsf{pk}_{\mathsf{SE}}^{s_{2,i}}, F(t_i))$ in the computation of c_4 by a random uniform value u^*. We argue that the adversary has to break an instance of the GDDHE problem to see the difference. We thus have to show that we truly fall into a GDDHE instance. In fact, the adversary has access to elements of the form of $g_1^{P(r_b, s_2, x', x)}$ combining $c_1, c_2, \widetilde{\mathsf{pk}_{\mathcal{R}}}$ and $\mathsf{pk}_{\mathsf{SE}}$. The resulting polynomial P has monomials of maximal degree 1, as the exponents r_b, s_2 and x' are kept secret. $\mathcal{A}$ has also access to elements in the form of $g_2^{Q(\alpha_i, x')}$. Here, the α_i's are the (unknown) discrete logarithms of $F(t_i)$ that $\mathcal{A}$ can compute for any tokens of his choice, provided that those tokens has the same type as the ones sent to the challenger. As the α_i's are unknown, each monomial of Q has degree at most 2, corresponding to elements of the form $g_t^{s_2 \cdot x' \cdot \alpha_b}$. However, $\mathcal{A}$ has to compute an element in $\mathbb{G}_t$ which has the form $g_t^{f(s_2, x', \alpha_b)}$. The exponent $f(s_2, x', \alpha_b) = s_2 \cdot x' \cdot \alpha_b$ is a monomial of degree 3 which is obviously linearly independent of P and Q. As such, the view of the adversary is truly an instance of the GDDHE assumption. Let $\mathcal{C}$ be an

adversary against the GDDHE assumption. As previously, for all $\mathcal{C}$ and denoting q_H the number of $\mathcal{A}$'s requests to the random oracle, we have:

$$|\Pr(S_2) - \Pr(S_3)| \leq \frac{1}{q_H}\mathsf{Adv}_{\mathcal{C}}^{\mathsf{GDDHE}}(\lambda).$$

Now, S's output is a perfectly random value and we have $S_3 = 1/2$.

Thus, we have:

$$\mathsf{Adv}_{\pi,\mathcal{A}}^{tr-ind}(\lambda) \leq \frac{2}{q_G}\mathsf{Adv}_{\mathcal{B}}^{\mathsf{CDH}}(\lambda) + \frac{1}{q_H}\mathsf{Adv}_{\mathcal{C}}^{\mathsf{GDDHE}}(\lambda),$$

so the adversary's advantage is negligible. □

A.3 Rule Indistinguishability Property

Proof. We consider an adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ having access to the public parameters, including the SE public key $\mathsf{pk}_{\mathsf{SE}} = g_1^{x'}$. The first part of the adversary $\mathcal{A}_f$ outputs two keywords w_0 and w_1, following a high min-entropy distribution, and the challenger executes RuleGen on input w_b for a secret bit $b \in \{0, 1\}$. Such procedure outputs $T_b = F(w_b)^{x'}$. The resulting T_b is given to the second part of the adversary, $\mathcal{A}_g$. We remember that $\mathcal{A}_f$ and $\mathcal{A}_g$ cannot communicate one with each other, since that would allow them to trivially break the rule indistinguishability property.

Note that the output of T_b implicitly fixes the output of the random oracle F for the input w_b.

Then, according to the rule indistinguishability game, the adversary $\mathcal{A}_g$ is allowed to create any encrypted traffic of its choice, using any keyword w of its choice, executing the Send procedure of the scheme π. At each execution, the adversary should ask for the random oracle F on input the keyword w. If it asks for w_b, then it becomes easy for $\mathcal{A}_g$ to determine whether $b = 0$ or 1, by using the bilinear property of e, as $e(g_1, T_b) = e(\mathsf{pk}_{SE}, F(w_b))$. But this happens only with negligible probability $2^{-\mu(\lambda)}$, with $\mu(\lambda) \in \omega(\log \lambda)$, since the rule set has high-min entropy. Otherwise, the adversary has to distinguish between the triplet $(g_1, g_1^{x'}, h_0^{x'})$ and $(g_1, g_1^{x'}, h_1^{x'})$, with unknown h_0 and h_1 (since corresponding to unknown outputs of F), which is obviously unconditionally infeasible since the adversary has not enough material to have a conclusion better than random guessing. □