


Towards practical intrusion detection system over encrypted traffic*

Sébastien Canard¹ | Chaoyun Li² 

¹Applied Crypto Group, Orange Labs, Caen, France

²Department of Electrical Engineering (ESAT), imec-COSIC, KU Leuven, Leuven, Belgium

Correspondence

Chaoyun Li, Department of Electrical Engineering (ESAT), imec-COSIC, KU Leuven, Leuven, Belgium.
Email: chaoyun.li@esat.kuleuven.be

Funding information

KU Leuven, Grant/Award Number: C16/15/058; Fonds Wetenschappelijk Onderzoek, Grant/Award Number: CyberSecurity Research Flanders VR20192203; Agence Nationale de la Recherche, Grant/Award Number: PRESTO ANR-19-CE39-0011-04, Fonds Wetenschappelijk Onderzoek (FWO) post-doctoral fellow, Grant/Award Number: 1283121N

Abstract

Privacy and data confidentiality are today at the heart of many discussions. But such data protection should not be done at the detriment of other security aspects. In the context of network traffic, intrusion detection system becomes totally blind when the traffic is encrypted, making clients again vulnerable to known attacks. To reconcile security and privacy, BlindBox and BlindIDS are proposed to perform Deep Packet Inspection over an encrypted traffic, based on two different cryptographic techniques. But, on one side, even if BlindBox is quite efficient to detect an anomalous encrypted traffic, it necessitates a very high setup time for clients and servers and does not protect the know-how of Security Editors (SEs) working on detection rules. On the other side, BlindIDS does protect SE's market and does not introduce any latency during setup time, but is definitely not enough efficient for a practical use. Herein, it is shown that the design of a fully efficient and market-compliant intrusion detection system over an encrypted traffic is possible. The system is based on only symmetric cryptography, and permits to encrypt a packet of 1500 bytes in about 6 μ s and to test such packets with 3000 rules in less than 2 μ s.

1 | INTRODUCTION

Once restricted to the most sensitive web traffic, such as online payment, encryption is now widely used. Each year, the share of encrypted web sessions is growing, reaching 68 per cent of overall sessions in 2017 [2]. The surge of encrypting traffic gives cybercriminals an avenue to hide in plain sight. Encryption turns malicious traffic indistinguishable from non-harmful one, preventing, for example, intrusion detection based on deep packet inspection (DPI). Since 2015, Dell security report [3] has warned companies and individuals alike on the danger of non-inspected Internet traffic. In its 2016 report [4], the number of affected users by under-the-radar attacks is estimated to 900 million. The necessity of a responsible encrypted traffic inspection is again stressed in the 2018 report [5]. To alleviate the consequences of such attacks, security editors propose to use proxies that establish a secure connection with the web server on behalf of the client. The proxy is then in a position of decrypting and inspecting the whole traffic, thanks to a set of so-called 'rules' defined by

security editors, and which permits the proxy to differentiate safe from unsafe traffics. However, this approach raises problems of confidentiality, security and privacy.

1.1 | Main existing approaches

In this context, the use of encryption techniques that allow a third party, the proxy, to inspect the encrypted traffic without having to decrypt the contents, is an approach that cannot be overlooked. BlindBox [6] is one of the first to have taken this step towards reconciling privacy and security. It relies on multi-party computations techniques, and in particular, garbled circuits, to enable a middlebox appliance to test security editors' rules by searching for malicious patterns directly on the encrypted contents. As highlighted in Ref. [7], this seminal paper suffers nevertheless from two shortcomings.

First, the set of patterns to be searched has to be encrypted for each pair of sender/receiver, as the encryption key is

*This work is an extended version of the Chapter 11 in the PhD thesis [1] of the second author.

derived from the session key. This induces a high setup time for every Internet connection, incompatible with online detection. Moreover, such set of patterns has to be encrypted under a different key for each parallel connection, which results in a huge memory consumption. Second, to perform the patterns' encryption for each session, the inspecting proxy has to access the patterns in clear. Those patterns are generated by a security editor, and are the core of its added value. Hence, in a competitive market, it is unlikely that the editors will be willing to disclose those patterns to all possible proxies.

The first one was the later treat in Ref. [8] with the PrivDPI system. While Blindbox sends the messages of garbled circuit per rule, PrivDPI only needs to send a few group elements per rule. This is due to a new technique for generating encrypted rules as well as the good idea of reusing intermediate results generated in previous sessions across subsequent sessions.

BlindBox shortcomings were also addressed with the BlindIDS solution [7]. In this proposal, the authors leverage decryptable searchable encryption (DSE), a cryptographic primitive where decryption keys are independent from trapdoors keys, used to perform search. This independence allows security editors to encrypt their patterns once and for all. These encrypted sessions can be used for every parallel session, thus notably reducing both setup time and memory consumption. However, the search operation uses pairings, whose performances are not suitable for online inspection.

We prove here that this is feasible to obtain the best of both solutions: efficient setup, low memory consumption, rules' confidentiality against proxies and efficiency of the whole protocol among a sender, a receiver and a proxy.

1.2 | Our contributions

Herein, we propose the first encrypted traffic inspection system that permits to obtain all the desired properties. Our system reaches the following interesting properties: (i) no public key encryption is involved (except potentially for deriving the session key); (ii) unauthorized actors cannot learn any information about the traffic other than it is malicious or safe (traffic indistinguishability); (iii) detection patterns are encrypted by the security editor, as the set of rules, and never accessible by other parties (rule indistinguishability); (iv) encrypted patterns are valid for all connections and (v) pattern matching is almost as simple as an equality test.

1.2.1 | Solution overview

Our first idea to obtain a truly efficient system is using *symmetric encryption techniques*. Then, all steps, from the encryption/decryption part to the rule generation and the detection algorithm, are based on symmetric cryptography methods.

Let T be a traffic divided into a set of 'tokens', each of them being denoted t_j . The first step of our construction

consists in having a secret key s shared between the Security Editor SE and the senders/receivers S/R. Such key is used by SE, for each pattern r_i related to a rule,¹ to compute the corresponding blinded rules B_i during the generation of the rules procedure (called RuleGen). A sender also makes use of this key to compute a blinded version p_j of each token t_j of the traffic. Using a deterministic algorithm, the detection procedure (named Detect) is then a simple match between the B_i 's and the p_j 's. It is obvious that the Service Provider SP should not know the key s so as not to break the traffic indistinguishability property. Moreover, as the receiver has a priori to come back from p_j to the token, we can here use a pseudorandom permutation (PRP) F . Finally, one problem that may occur is that a sender/receiver having access to the set of blinded rules can perform alone (i.e. without requesting the Service Provider) a brute force attack to break the rule indistinguishability property. We can here easily assume that SE can make use of a specific channel providing confidentiality, authenticity and integrity to send blinded rules to SP.

Action (actor)	Inputs	Actions
RuleGen(SE)	Rules r_i , key s	$B_i = F(s, r_i)$
Send(S)	Tokens t_j , key s	$p_j = F(s, t_j)$
Receive(R)	Traffic p_j , key s	$t_j = F^{-1}(s, p_j)$
Detect(SP)	Rules B_i , traffic p_j	$B_i = p_j?$

The main problem with the above tentative is that the matching between one B_i and one p_j is done by SP during the Detect procedure, so that the latter needs to obtain both. Regarding the traffic and the blinded token p_j , it cannot be sent as it is since it can be used by SE to break the traffic indistinguishability property. We then make use of an encapsulation technique where each p_j is encrypted using a key K shared by SP on one side, and both S and R on the other side. This time, as we need decryption (at least by SP for efficiency reasons), we use a pseudorandom function (PRF) G in counter mode.

Action (actor)	Inputs	Actions
RuleGen(SE)	Rules r_i , key s	$B_i = F(s, r_i)$
Send(S)	Tokens t_j , Keys (s, K)	$p_j = F(s, t_j)$ $q_j = G(K, j) \oplus p_j$
Receive(R)	Traffic q_j , Keys (s, K)	$p_j = q_j \oplus G(K, j)$ $t_j = F^{-1}(s, p_j)$
Detect(SP)	Rules B_i , key K , Traffic q_j	$p_j = G(K, j) \oplus q_j$ $B_i = p_j?$

¹In the sequel, the notions of rules and patterns are considered equivalent. In fact, as explained above, a rule corresponds to the detection of a pattern.

With such system, there is still the problem that the resulting encrypted traffic is deterministically computed, such that the same token p_j results in the same ciphertext q_j . The problem of using a true random value to compute the p_j 's poses, is that matching will be possible only if both parties (SE and S) use the same randomness. To perform that, we introduce a counter which is used with the PRP F . Let C be the maximum number of occurrences of distinct tokens in the traffic. During RuleGen, SE generates C blinded tokens for each rule r_i . During Send, the sender chooses at random a counter $c \in [0, C - 1]$ to compute the blinded tokens (the first token uses c , which one is then incremented each time a new blinded token is computed). In this way, the frequencies of the tokens are hidden. Furthermore, we add a true random salt to G such that the q_j 's are indistinguishable from random values for any adversary without access to K .

Action (actor)	Inputs	Actions
RuleGen(SE)	Rules r_i , key s	$B_{i,k} = F(s, r_i c_k)$
Send(S)	Tokens t_j ,	$p_j = F(s, t_j c)$
	Keys (s, K) ,	$q_j = G(K, \text{salt} + j) \oplus p_j$
	Counter c , Random salt	
Receive(R)	Traffic q_j ,	$p_j = q_j \oplus G(K, \text{salt} + j)$
	Keys (s, K) ,	$t_j c = F^{-1}(s, p_j)$
	Counter c , salt	
Detect(SP)	Rules B_i , key K ,	$p_j = G(K, \text{salt} + j) \oplus q_j$
	Traffic q_j , salt	$B_i = p_j?$

One problem with the above description is that F being a PRP, a fraudulent sender or receiver can inverse the $B_{i,k}$'s to break the rule indistinguishability property. The solution is to replace F by a non-reversible pseudorandom function. But the receiver is no more able to decrypt the received traffic, so that we need to send the traffic twice: one time using the technique given above (replacing the PRP by a PRF, and used to detect unsafe traffic) and the other one using a classical TLS channel and a shared key k (used by the receiver to decrypt the traffic). If the traffic is safe, this moreover permits the receiver to go faster by just decrypting the TLS part. But this also permits the sender to send two different things: one safe fake traffic that is tested by SP and one true corrupted one that will be executed by the receiver. Without any additional trick, this can be detected by the receiver that has all the material to check the difference. But this is done at the detriment of the efficiency. Our idea is to ask SP to hash the encrypted tokens p_j it has obtained during the Detect phase and a similar hash is also computed by the receiver, permitting him to detect such fraud by the sender by a simple equality test.

Action(actor)	Inputs	Actions
RuleGen(SE)	Rules r_i , key s	$B_{i,k} = F(s, r_i c_k)$
Send(S)	Tokens t_j ,	$e = \text{TLS}(k, \{t_j\}_j)$,
	Keys (s, K, k) ,	$p_j = F(s, t_j c)$,
	Counter c ,	$q_j = G(K, \text{salt} + j) \oplus p_j$
	Random salt	
Receive(R)	Traffic e ,	$\{t_j\}_j = \text{TLS}^{-1}(k, e)$,
	Keys (s, K, k) ,	$p_j = F(s, t_j c)$,
	Counter c ,	$b_j = \mathcal{H}(p_j)?$
	Salt, hash b_j	
Detect(SP)	Rules B_i ,	$p_j = G(K, \text{salt} + j) \oplus q_j$,
	key K ,	$B_i = p_j?$,
	Traffic q_j ,	$b_j = \mathcal{H}(p_j)$
	Salt	

There is still one important drawback in the current version of the protocol: the Security Editor has to compute a number of encrypted rules which is proportional to the number of couples S/R times the constant C . Our idea to remove the proportionality to the number of couples S/R is to make use of a broadcast encryption scheme BE (see Section 3.1) for the group of all senders and receivers (such group is denoted I in the sequel). Then, SE knows the master secret key mk to manage the group, and each sender (respectively receiver) owns a membership key denoted sk_n (resp. sk'_n) which permits him to retrieve the key s using some header Hdr output by SE during (broadcast) encryption. This way, SE can generate the same encrypted rules for several distinct users.

Action(actor)	Inputs	Actions
RuleGen(SE)	Rules r_i ,	$(s, \text{Hdr}) = \text{BE.Enc}(mk, I)$
Send(S)	Master key mk	$B_{i,k} = F(s, r_i c_k)$
	Tokens t_j ,	$s = \text{BE.Dec}(sk_s, \text{Hdr})$
	Keys (s, K, k) ,	$e = \text{TLS}(k, \{t_j\}_j)$,
	Counter c ,	$p_j = F(s, t_j c)$,
Receive(R)	Random salt	$q_j = G(K, \text{salt} + j) \oplus p_j$
	Traffic e ,	$s = \text{BE.Dec}(sk_r, \text{Hdr})$,
	Keys (sk'_r, K, k) ,	$\{t_j\}_j = \text{TLS}^{-1}(k, e)$,
	Counter c ,	$p_j = F(s, t_j c)$,
Detect(SP)	Salt, hash b_j	$b_j = \mathcal{H}(p_j)?$
	Rules B_i ,	$p_j = G(K, \text{salt} + j) \oplus q_j$
	key K ,	$B_i = p_j?$,
	Traffic q_j ,	$b_j = \mathcal{H}(p_j)$
Salt		

1.2.2 | Implementation

We implemented our system and provide a thorough evaluation over popular web pages, comparing performances with BlindBox and BlindIDS, showing that our work is a significant step for a real-life deployment of privacy-preserving intrusion detection systems. More precisely, the encryption of a packet of 1500 bytes is done in about 6 μ s (compare to 90 μ s for BlindBox and 27 ms for BlindIDS). The detection phase for 3000 rules and one packet necessitates less than 2 μ s (compare to 33 μ s for BlindBox and 74 s for BlindIDS).

1.2.3 | Organization

Herein, in the next section, we first recall and modify a little the security model proposed in [7] for an ideal intrusion detection system over an encrypted traffic. We then give the details of our new protocol in Section 3 and describe our implementation and experimental results in Section 5. The related work is detailed in Section 6, before the conclusion.

2 | SYSTEM ARCHITECTURE AND SECURITY

2.1 | Actors and Architecture

We consider the following actors in the DPI system (following [6, 7]):

- the Security Editor, denoted SE, who is responsible for generating and maintaining a list of malwares' signatures;
- the Service Provider, denoted SP, who searches intrusions in the traffic, using the rules provided by the SE;
- a sender, denoted S, who sends messages over the Internet. The set of all senders is denoted \mathcal{S} .
- a receiver, denoted R, who receives the messages. The set of all receivers is denoted \mathcal{R} .

The SE role is performed by organisations such as McAfee, Symantec and Kaspersky. The detection signatures are the main assets for the SE. The SP, namely, the middlebox in Ref. [6], provides both physical and cloud-based services such as proxies. In most deployments, the SE or the middlebox can read the plain traffic between S and R. Herein, it is aimed to propose IDS using DPI over encrypted traffic which is as good as that over the plain traffic.

2.2 | Main procedures

We here just give some minor modification of the initial model proposed in Ref. [7]. An intrusion detection system over an encrypted traffic, denoted Δ is composed of the following procedures. The main difference is that we more clearly define

the different actors' cryptographic keys, and put them, by default, in the corresponding procedures.

Let T be a traffic. We consider that T is divided into tokens of fixed size, denoted t_j . Moreover, as already said, we consider that a rule corresponds to a pattern to be detected in the traffic, and we then independently talk about either a rule or a pattern in the sequel. The set of rule is then denoted \mathcal{M} and corresponds to a set of patterns to be searched, using equality tests.

- Setup, on input the security parameter λ , generates the public parameters param of the system, and the keys of the different actors, that is sk_{SE} for Security Editor, sk_{SP} for Service Provider, sk_{S} for Sender, and sk_{R} for Receiver.
- RuleGen, on input the parameters param , the SE secret key sk_{SE} and a set \mathcal{M} 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 , the secret key sk_{S} of the sender, and a traffic $T = \{t_j\}_p$, where each t_j is a token of fixed size. It outputs an encrypted traffic E for a receiver R.
- Detect, on input param , the Service Provider private key sk_{SP} , 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 or some additional information for the receiver. 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} , an encrypted traffic E and optionally some additional auxiliary information aux coming from SP. It outputs a plain traffic T , or an error message \perp .

2.2.1 | Correctness

Let param be the output of the Setup procedure. Let \mathcal{B} be a set of blinded rules, as $\mathcal{B} = \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M})$ where \mathcal{M} is a set of rules, and let $E = \text{Send}(\text{param}, \text{sk}_{\text{S}}, T)$ where T is a traffic. An intrusion detection system over an encrypted traffic is said *correct* iff

$$T = \text{Receive}(\text{param}, \text{sk}_{\text{R}}, E, \text{aux}), \text{ and} \\ \text{Detect}(T, \mathcal{M}) = \text{Detect}(\text{param}, \text{sk}_{\text{SP}}, E, \mathcal{B}) \text{ (incl. aux).}$$

2.3 | Security requirements

We now define the expected security for such system. Informally speaking, we consider that the Service Provider is honest-but-curious since it applies the DPI honestly but can try to obtain information about either the users' traffic or the SE's rules. We also use the honest-but-curious paradigm for

the Security Editor as all the rules are considered as true and authentic malicious patterns. But similarly to the SP, the SE may try to acquire information about the clear-text content of the traffic. We, however, do not consider the case where the SP and the SE collude, as they can in this case easily mount a dictionary attack. Finally, we also do not consider a coalition between a sender and a receiver since, as in a non-encrypted traffic, they can easily agree on any shared secret key and any encryption algorithm to add an overlay of encryption so that the detection becomes infeasible. We now go into more formal details.

As shown in Ref. [7], there are mainly three security properties that should be verified by such a system: detection, traffic indistinguishability and rule indistinguishability. We here modify a little of these properties, so as they better suit real needs. We think that these modifications can lead to better secure schemes in the future. The modification has already been sketched in the introduction, and will be detailed for each security property below. All security experiments are given in Figure 1. We consider, for each of them, that the Setup has already been executed as:

$$(\text{param}, \text{sk}_{\text{SE}}, \text{sk}_{\text{SP}}, \text{sk}_{\text{S}}, \text{sk}_{\text{R}}) \leftarrow \text{Setup}(1^\lambda).$$

2.3.1 | Detection

The detection property informally states that any malicious traffic must be detected by the Service Provider. This is close to the above correctness property, but considering that either the sender or the receiver tries to cheat. The related security experiment is given in Figure 1. On input the parameters, \mathcal{A} outputs an encrypted traffic E such that it is stated as safe (i.e. $\text{Detect}(\text{param}, \text{sk}_{\text{SP}}, E, \mathcal{B}) = 1$) while the decrypted version T is malicious (i.e. $\text{Detect}(T, \mathcal{M}) = 0$).

Definition 1 (Detection) An intrusion detection system over encrypted traffic Δ is said detectable if for any probabilistic polynomial-time \mathcal{A} , there exists a negligible function $\nu(\lambda)$ such that:

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

As explained in Ref. [7], we do not consider the case where sender and receiver are both dishonest and collude. No TLS inspection system can treat this case as the sender and the receiver may agree on some secret coding or encryption to hide malicious traffic, in an undetectable way.

2.3.2 | Traffic indistinguishability

The traffic indistinguishability property informally states that it is not feasible for non-authorized actors to learn any information about the traffic, other than it is malicious or safe. Particularly, the SP is assumed to not learn any information of the traffic other than the match of traffic and rules.

In fact, compare to Ref. [7], we consider that the Service Provider SP manages its own private key sk_{SP} such that its role necessitates some knowledge that are not available to other actors. We thus consider two different traffic indistinguishability experiments, depending on the knowledge of the adversary: the secret key sk_{SE} of the Security Editor or the one sk_{SP} of the Service Provider. It is obvious that having access to both keys easily break the traffic indistinguishability property.

In both cases, we deal with the problem that the adversary may choose, in the indistinguishability experiment, one malicious traffic and one safe traffic so that it will be easy for him to distinguish which one is used by the challenger, using the Detect algorithm or some auxiliary information. We then reuse the notion of *type* [7]: two traffics T_0 and T_1 are of the *same type*, denoted $\text{type}(T_0, T_1) = 1$, iff

$\text{Exp}_{\Delta, \mathcal{A}}^{\text{det}}(\lambda)$ $\begin{aligned} & \mathcal{B} \leftarrow \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M}); \\ & E \leftarrow \mathcal{A}(\text{param}, \text{sk}_{\text{S}}, \text{sk}_{\text{R}}); \\ & \text{if } \text{Detect}(\text{param}, \text{sk}_{\text{SP}}, E, \mathcal{B}) = 1, \text{ return } 0; \\ & T \leftarrow \text{Receive}(\text{param}, \text{sk}_{\text{R}}, E); \\ & \text{if } \text{Detect}(T, \mathcal{M}) = 0, \text{ return } 0; \\ & \text{return } 1. \end{aligned}$	$\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda)$ $\begin{aligned} & b \leftarrow \mathcal{S}\{0, 1\}; \\ & (T_0, T_1, \text{aux}_{\mathcal{A}}) \leftarrow \\ & \quad \mathcal{A}^{\text{Send, RuleGen}}(\text{sk}_{\text{SP}}, \text{param}); \\ & \text{if } \text{type}(T_0, T_1) = 0, \text{ return } 0; \\ & E_b \leftarrow \text{Send}(\text{param}, T_b); \\ & b' \leftarrow \mathcal{A}^{\text{Send, RuleGen}}(E_b, \text{aux}_{\mathcal{A}}); \\ & \text{return } (b = b'). \end{aligned}$	$\text{Exp}_{\Delta, \mathcal{A}}^{\text{se-tr-ind}}(\lambda)$ $\begin{aligned} & b \leftarrow \mathcal{S}\{0, 1\}; \\ & (T_0, T_1, \text{aux}_{\mathcal{A}}) \leftarrow \\ & \quad \mathcal{A}^{\text{Send, RuleGen}}(\text{sk}_{\text{SE}}, \text{param}); \\ & \text{if } \text{type}(T_0, T_1) = 0, \text{ return } 0; \\ & E_b \leftarrow \text{Send}(\text{param}, T_b); \\ & b' \leftarrow \mathcal{A}^{\text{Send, RuleGen}}(E_b, \text{aux}_{\mathcal{A}}); \\ & \text{return } (b = b'). \end{aligned}$
$\text{Exp}_{\Delta, \mathcal{A}}^{\text{hme-rul-ind}}(\lambda)$ $\begin{aligned} & b \leftarrow \mathcal{S}\{0, 1\}; \\ & (\mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathcal{A}_f(\text{param}, \text{sk}_{\text{SP}}, \text{sk}_{\text{S}}, \text{sk}_{\text{R}}); \\ & \mathcal{B}_b \leftarrow \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M}_b); \\ & b' \leftarrow \mathcal{A}_g(\mathcal{B}_b); \\ & \text{return } (b = b'). \end{aligned}$	$\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-rul-ind}}(\lambda)$ $\begin{aligned} & b \leftarrow \mathcal{S}\{0, 1\}; \\ & (\mathcal{M}_0, \mathcal{M}_1, \text{aux}_{\mathcal{A}}) \leftarrow \mathcal{A}^{\text{Send}}(\text{param}, \text{sk}_{\text{SP}}); \\ & \mathcal{B}_b \leftarrow \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M}_b); \\ & b' \leftarrow \mathcal{A}^{\text{Send}}(\mathcal{B}_b, \text{aux}_{\mathcal{A}}); \\ & \text{return } (b = b'). \end{aligned}$	

FIGURE 1 Security experiments

$$\text{Detect}(\text{param}, T_0, \mathcal{M}) = \text{Detect}(\text{param}, T_1, \mathcal{M})$$

including the auxiliary information aux, and where \mathcal{M} is a set of rules.

More formally, we give in Figure 1 two different experiments, $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda)$ and $\text{Exp}_{\Delta, \mathcal{A}}^{\text{se-tr-ind}}(\lambda)$, for an adversary \mathcal{A} having access to both a Send oracle (given a plain traffic T of its choice, \mathcal{A} obtains the related encrypted traffic E) and the RuleGen oracle (given a set of rules \mathcal{M} of its choice, the adversary gets back $\mathcal{B} \leftarrow \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M})$). To emphasize the adversary's power, it is denoted as $\mathcal{A}^{\text{Send, RuleGen}}$ in the two experiments. 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 .

We moreover have more restrictions on the adversary \mathcal{A} in $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda)$. We assume that \mathcal{A} does not query the Send oracle with traffic containing tokens in T_0 or T_1 ; otherwise, the traffic indistinguishability is trivially broken since the Send encrypts traffics deterministically up to a counter which is not exponentially large. Also, \mathcal{A} does not chooses tokens in T_0 or T_1 as rules to query RuleGen; otherwise, the detection functionality allows \mathcal{A} to trivially distinguish the T_0 and T_1 by the pattern of matching. This is a common restriction for searchable encryption security definitions (e.g., see the MBSE security in [6]).

Definition 2 (Traffic indistinguishability) An intrusion detection system over encrypted traffic Δ is considered traffic-indistinguishable if for any probabilistic polynomial-time \mathcal{A} , there exists a negligible function $\nu(\lambda)$ such that

$$\text{Adv}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda) = |2 \cdot \Pr[\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}} = 1] - 1| \leq \nu(\lambda),$$

and

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

2.3.3 | Rule indistinguishability

The rule indistinguishability property states that it is not feasible to learn any information about the rules. In fact, contrary to [7], we consider two different kinds of rule indistinguishability.

High-min entropy rule indistinguishability

We remark that if the adversary is a Sender or a Receiver, then it can create any valid traffic of its choice, and make use of the encrypted rules to test them and learn some information. In this case, we make use of the high min-entropy property, stating that [9] a probabilistic adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ has *min-entropy* μ if

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

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

This restriction may limit the number of rules we can manage since for some of them, part of the information can be publicly known, as for example, 'bad' domain names for URL blacklists (see also Section 5.2).

Service provider rule indistinguishability

If the adversary has no access to such secrets (i.e. sk_S, sk_R), then it may want to obtain some information about the underlying rules, but has no more restrictions on the entropy. We argue that such an adversary may be complementary to the previous one, and that an intrusion detection system over the encrypted traffic should be resistant to both kinds of attacks.

Both experiments are given in Figure 1, for (i) an adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ being able to create any traffic and with high min-entropy (see e.g. [9] for details) for $\text{Exp}_{\Delta, \mathcal{A}}^{\text{hme-rul-ind}}(\lambda)$, and (ii) a standard adversary \mathcal{A} for $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-rul-ind}}(\lambda)$, that has access to a Send oracle, denoted by $\mathcal{A}^{\text{Send}}$, giving on output an encrypted traffic from a plain payload. The adversary \mathcal{A} (\mathcal{A}_f) chooses two sets of rules \mathcal{M}_0 and \mathcal{M}_1 , and one of them is used in the RuleGen procedure. The output \mathcal{B}_b is then given to \mathcal{A} (\mathcal{A}_g), that eventually outputs the bit (b).

Notice that the relation between Send and RuleGen is symmetric if we compare $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-rul-ind}}(\lambda)$ to $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda)$. Thus, similar to the $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-tr-ind}}(\lambda)$, we assume that the adversary in $\text{Exp}_{\Delta, \mathcal{A}}^{\text{sp-rul-ind}}(\lambda)$ does not query RuleGen with any rules in \mathcal{M}_0 or \mathcal{M}_1 and not query Send with traffic containing rules in \mathcal{M}_0 or \mathcal{M}_1 .

Definition 3 (Rule indistinguishability) An intrusion detection system over encrypted traffic Δ 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 both

$$\text{Adv}_{\Delta, \mathcal{A}}^{\text{hme-rul-ind}}(\lambda) = |2 \cdot \Pr[\text{Exp}_{\Delta, \mathcal{A}}^{\text{hme-rul-ind}} = 1] - 1| \leq \nu(\lambda)$$

and

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

3 | DETAILS OF OUR PROTOCOL

In this section, we first give the main cryptographic building blocks we need for our construction, before giving the details of the latter.

3.1 | Cryptographic building blocks

At first, we present the main cryptographic building blocks we will need. We also give the related security requirements that will be useful in our security proofs. Let λ be a security parameter.

3.1.1 | Pseudorandom function

A function $F: \{0,1\}^s \times \{0,1\}^\ell \rightarrow \{0,1\}^n$ is a pseudorandom function (PRF) if

- given a key $K \in \{0,1\}^s$ and an input $M \in \{0,1\}^\ell$, one can efficiently compute $F(K, M)$;
- in a nutshell, an adversary against a PRF should not be able to distinguish the output of F from the uniform distribution \mathcal{U} . More formally, for any probabilistic polynomial-time \mathcal{A} , there exists a negligible function $\nu(\lambda)$ such that

$$\text{Adv}_{F,\mathcal{A}}^{\text{prf}}(\lambda) = |2 \cdot \Pr[\text{Exp}_{F,\mathcal{A}}^{\text{prf}} = 1] - 1| \leq \nu(\lambda).$$

where $\text{Exp}_{F,\mathcal{A}}^{\text{prf}}$ is given in Figure 2, in which \mathcal{U} is the uniform distribution, and where \mathcal{A} is given access to an oracle which on input a message M , outputs $F(K, M)$.

Note that F can be implemented as a keyed hash function such as *SHA-256*.

In our construction, we need another property for the used PRF. More precisely, in the rule indistinguishability experiment, the adversary knows the key K , which does not permit us to rely on the above pseudorandomness, for obvious reasons. We then consider the case of a fixed-key PRF and we require the one-wayness of the resulting function, against an adversary

having access to the key K . More formally, for any probabilistic polynomial-time \mathcal{A} , there exists a negligible function $\nu(\lambda)$ such that

$$\text{Adv}_{F,\mathcal{A}}^{\text{ow-prf}}(\lambda) = \Pr[\text{Exp}_{F,\mathcal{A}}^{\text{ow-prf}} = 1] \leq \nu(\lambda).$$

where $\text{Exp}_{F,\mathcal{A}}^{\text{ow-prf}}$ is given in Figure 2.

It is commonly believed that the keyed *SHA-256* verifies such property (see e.g. [10] for some comments on that point). Moreover, as a hash function, *SHA-256* can also be treated as a random oracle. Using both the one-wayness of the keyed *SHA-256* and the random oracle model, we will be able to prove that our scheme provides rule indistinguishability against fraudulent senders and receivers (in the high min-entropy setting, see Section 2.3.3).

3.1.2 | Hash function

We also need a cryptographically secure hash function \mathcal{H} , that is *collision resistant*, *resistant to pre-image* and *resistant to second pre-image*.

3.1.3 | Broadcast encryption

As shown in Refs. [11, 12], Broadcast Encryption (BE) schemes [13] can be used to enforce some access control in the multi-user setting. Most of the time, and this will be the case here, a classical symmetric key encryption scheme should be added to such broadcast encryption scheme. A broadcast encryption BE can be described by the following procedures.

- Setup, on input the security parameter λ , it generates the public parameters param of the system, a master secret key msk .
- Extract, on input the parameters param , the key msk and a user's index i outputs the key sk_i of user i .
- Enc takes as input the public parameters param , the master key msk and a set of indices \mathcal{I} . It outputs a header Hdr and a key $\text{K} \in \mathcal{K}$.
- Dec, on input param , a header Hdr and a secret key sk_i for $i \in \mathcal{I}$, such procedure outputs the key K .

Regarding security, a specific form of indistinguishability should be defined on the key K [14]. The corresponding experiment $\text{Exp}_{\text{BE},\mathcal{A}}^{\text{be-ind}}$ is given in Figure 2, where \mathcal{A} has access to an Enc oracle on input \mathcal{I} (getting on output (Hdr, K)) and to a Dec oracle on input (Hdr, i) (having access to the output key K). We say that a broadcast encryption scheme BE is *indistinguishable* if for any probabilistic polynomial-time \mathcal{A} , there exists a negligible function $\nu(\lambda)$ such that

$$\text{Adv}_{\text{BE},\mathcal{A}}^{\text{be-ind}}(\lambda) = |2 \cdot \Pr[\text{Exp}_{\text{BE},\mathcal{A}}^{\text{be-ind}} = 1] - 1| \leq \nu(\lambda).$$

$\text{Exp}_{F,\mathcal{A}}^{\text{prf}}(\lambda)$ <p> $b \leftarrow \mathcal{S}\{0, 1\};$ $K \leftarrow \{0, 1\}^s;$ For $i = 1, 2, \dots, q$ do $(M_i, \text{aux}_A) \leftarrow \mathcal{A}(1^\lambda);$ if $b = 0$, then $R_i \leftarrow \mathcal{U}(1^n);$ if $b = 1$, then $R_i = F(K, M_i);$ $b' \leftarrow \mathcal{A}(R_1, \dots, R_q, \text{aux}_A);$ return $(b = b')$. </p>	$\text{Exp}_{F,\mathcal{A}}^{\text{ow-prf}}(\lambda)$ <p> $K \leftarrow \{0, 1\}^s;$ $M \leftarrow \{0, 1\}^\ell;$ $R \leftarrow F(K, M);$ $\tilde{M} \leftarrow \mathcal{A}(K, R);$ return $(M = \tilde{M})$. </p>
$\text{Exp}_{\text{BE},\mathcal{A}}^{\text{be-ind}}(\lambda)$ <p> $b \leftarrow \mathcal{S}\{0, 1\};$ $(\text{param}, \text{msk}) \leftarrow \text{Setup}(1^\lambda);$ $(\mathcal{I}, \text{aux}_A) \leftarrow \mathcal{A}(\text{param});$ if $b = 0$, then $\text{K} \leftarrow \mathcal{K};$ if $b = 1$, then $(\text{Hdr}, \text{K}) = \text{Enc}(\text{param}, \text{msk}, \mathcal{I});$ $b' \leftarrow \mathcal{A}(\text{K}, \text{aux}_A);$ return $(b = b')$. </p>	

FIGURE 2 Building blocks security experiments

3.2 | Description

Let F, G be secure PRFs, both with parameters s and ℓ . Let \mathcal{H} be a cryptographically secure hash function. Finally, let BE be a secure broadcast encryption. All details are given in Section 3.1. Let \mathcal{S} (resp. \mathcal{R}) be the set of senders (respective receivers) and let I be a set of indices related to $\mathcal{S} \cup \mathcal{R}$.

We now give the details of each step of our intrusion detection system over an encrypted traffic.

- Setup
 - SE first executes $\text{mk} \leftarrow \text{BE.Setup}(1^\lambda)$. It then computes $\text{sk}_n \leftarrow \text{BE.Extract}(\text{mk}, n)$ for each element of $\mathcal{S} \cup \mathcal{R}$ and sends the result (in a secure way) to the corresponding actor. After that, it computes $(\text{Hdr}, s) \leftarrow \text{BE.Enc}(\text{mk}, I)$. We assume that $s \in \{0,1\}^s$. Finally, SE defines the integer C as the maximum number of occurrences of distinct tokens in the traffic.

During a particular session between a sender S (with index $n \in I$) and a receiver R (with index $\tilde{n} \in I$), a few more things are executed by the actors.

- A key $K \leftarrow \{0,1\}^s$ for the PRF G is generated and secretly shared by SP, S and R .
- A key k for a TLS protocol is also generated and secretly shared by S and R .

At the end, we have

$$\text{param} = (C, \text{Hdr}), \text{sk}_{\text{SE}} = (\text{mk}), \text{sk}_{\text{SP}} = (K), \\ \text{sk}_S = \text{sk}_R = (\text{sk}_n, K, k)$$

- RuleGen
 - For each rule $r_i \in \mathcal{M}$, and for each $c_k \in [0, C - 1]$, the Security Editor SE computes $B_{i,k} \leftarrow F(s, r_i \| c_k)$. Then SE sends the set $\mathcal{B} = \{B_{i,k}\}_{i,k}$ to SP.
- Send
 - On input Hdr and a packet payload T , S first computes $s \leftarrow \text{BE.Dec}(\text{Hdr}, \text{sk}_n)$.
 - S then chooses $c \leftarrow$ and salt $\leftarrow \{0,1\}^\ell$. Next, the packet payload is parsed into a set of unique tokens $\{t_j\}_j$. For each t_j , S computes

$$p_j = F(s, t_j \| c) \quad (1)$$

$$q_j = G(K, \text{salt} + j) \oplus p_j, \quad (2)$$

where distinct token has a counter c , and it is incremented by one modulo C when the token repeats.

- Finally, S encrypts the whole packet payload with TLS key k , and obtain e . It then sends $E = (\{q_j\}_j, e, c, \text{salt})$ to SP.
- Detect
 - Receiving the encrypted traffic $E = (\{q_j\}_j, e, c, \text{salt})$ from S , SP computes $p_j = G(K, \text{salt} + j) \oplus q_j$ for each received q_j . If there is a match between one p_j and one $B_{i,k} \in \mathcal{B}$, then the encrypted token is marked as malicious and SP generates an alert. Otherwise, the token is marked as legitimate. If all tokens in the packet are legitimate, SP

redirects to R the TLS encrypted packet payload and the values c and salt. SP additionally hashes the set $\{p_j\}_j$ to obtain the value \tilde{h} , and gives such auxiliary information aux to R .

- Receive
 - On input e, c, salt and its secret key sk_R , the receiver R decrypts the traffic e using the TLS protocol and the key k . After that, it generates the p_j 's in the same way as S using $\text{sk}_{\tilde{n}}, c$ and salt.
 - R computes the hash value \tilde{h} of the obtained p_j 's for verification. If it matches with \tilde{h} , R accepts the traffic. Otherwise, S is considered as malicious and R outputs \perp .

3.3 | Security analysis

We show that the proposed scheme has the detection property and traffic and rule indistinguishability properties.

3.3.1 | Detection

We first prove the detection property of the proposed scheme.

Theorem 1 *Our scheme is detectable if \mathcal{H} is collision-resistant.*

Proof. As explained in Section 2.3, we do not consider the case where S and R collude to break the detection property. The case where the sender is honest is obviously achieved, so that we here only consider the case of a dishonest sender.

We then consider a successful adversary \mathcal{A} against the detection property. According to the detection experiment $\text{Exp}_{\Delta, \mathcal{A}}^{\text{det}}$ in Figure 1, the adversary \mathcal{A} sends $E = (\{q_j\}_j, e, c, \text{salt})$ to SP such that:

1. e is the ciphertext of a malicious traffic T' under TLS key k , which means that there exists $t'_j \in T'$ and c_0 such that $p'_{j_0} = F(s, t'_j \| c_0) \in \mathcal{B}$;
2. The Receive procedure does not output \perp , which means that the honest receiver R can first retrieve the set $\{t'_j\}_j$ using e and k , then compute, for all j , $\tilde{p}'_j = F(s, t'_j \| c)$, and finally $\mathcal{H}(\tilde{p}'_j) = \mathcal{H}(G(K, \text{salt} + j) \oplus q_j)$ in hash verification (otherwise, the traffic is rejected, see Section 3.2); and
3. The detection procedure Detect, on input q_j outputs one for all j , which means in particular that $G(K, \text{salt} + j) \oplus q_j \notin \mathcal{B}$.

As \mathcal{B} and s are considered as upright, the first and second points show that $\tilde{p}'_{j_0} \in \mathcal{B}$. This together with the second point implies that $\tilde{p}'_{j_0} \neq G(K, \text{salt} + j_0) \oplus q_{j_0}$. It follows from the second point that \mathcal{A} has obtained a collision of \mathcal{H} , which happens with negligible probability provided that \mathcal{H} is collision-resistant.

We briefly estimate the probability of false positive. If a false positive happens, then we have $F(s, r_i \| c) = F(s, t_j \| c)$ for some $r_i \neq t_j$. That is, we have a collision of $F(s)$. However, the

probability of obtaining a collision of $F(s, \cdot)$ is negligible since the F is implemented as the hash function *SHA-256*. This implies that the probability of false positive is negligible.

3.3.2 | Traffic indistinguishability

We now prove that our scheme is traffic indistinguishable: it verifies both traffic indistinguishability against a malicious service provider (*sp-tr-ind*) and against a malicious security editor (*se-tr-ind*). We thus prove the two following theorems. We assume in the sequel that the TLS protocol is secure and do not consider the value e in our proof. One can simply add the advantage of breaking TLS, which can obviously be considered as negligible.

Theorem 2 *Our scheme is traffic-indistinguishable against malicious service provider with*

$$\text{Adv}_{\mathcal{A}}^{\text{sp-tr-ind}}(\lambda) \leq 2(\text{Adv}_{\text{BE}, \mathcal{A}}^{\text{be-ind}}(\lambda) + \text{Adv}_{F, \mathcal{A}}^{\text{prf}}(\lambda)).$$

Proof. Assume that the adversary \mathcal{A} knows SP's secret key $\text{sk}_{\text{SP}} = K$ and has access to Send and RuleGen oracles. The adversary \mathcal{A} outputs two plain traffics T_0 and T_1 . According to a bit b , an encrypted traffic is generated, as $E_b = \text{Send}(\text{param}, T_b)$, where $T_b = \{t_j^{(b)}\}_j$.

Game 0. This is the original attack game, where the encrypted traffic E_b is composed of (i) the set $\{q_j^{(b)} = G(K, \text{salt} + j) \oplus p_j^{(b)}\}_j$ where each $p_j^{(b)} = F(s, t_j^{(b)} \| c_j)$ and $s = \text{BE.Dec}(\text{Hdr}, \text{sk}_n)$, (ii) the used random counter c_0 and salt, and (iii) the TLS ciphertext e (not considered). The adversary \mathcal{A} eventually outputs a bit b' . Since \mathcal{A} knows K , it can obtain the $p_j^{(b)}$'s as $p_j^{(b)} = G(K, \text{salt} + j) \oplus q_j^{(b)}$.

Let \mathcal{S}_0 be the event that the $b = b'$. Then we have

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{sp-tr-ind}}(\lambda) &= |2\Pr[\text{Exp}_{\mathcal{A}}^{\text{sp-tr-ind}}(\lambda) = 1] - 1| \\ &= |2\Pr[\mathcal{S}_0] - 1|. \end{aligned}$$

Game 1. We modify Game 0 by replacing the broadcast key s , output in $s = \text{BE.Dec}(\text{Hdr}, \text{sk}_n)$, by a random value in \mathcal{H} . Let \mathcal{S}_1 be the event that the $b = b'$ in Game 1. We can describe a distinguisher between Game 0 and Game 1, which exactly corresponds to the broadcast encryption indistinguishability experiment given in Figure 2 (as \mathcal{A} has no access to mk , nor s). Then

$$|\Pr[\mathcal{S}_1] - \Pr[\mathcal{S}_2]| = \text{Adv}_{F, \mathcal{A}}^{\text{prf}}(\lambda). \quad (4)$$

Game 2. Recall that \mathcal{A} is not allowed to query Send with any traffic which contains any $t_j^{(b)}$ or query the RuleGen with $t_j^{(b)}$. Furthermore, the $p_j^{(b)}$'s are distinct due to the varying counter c_j for the identical $t_j^{(b)}$, as in Equation (1). This enables us to transform Game 1 to Game 2 in which we replace the

$p_j^{(b)}$'s with truly random values in $\mathcal{H}(1^\lambda)$. Let \mathcal{S}_2 be the event that the $b = b'$ in Game 2. We can again describe a distinguisher between Game 1 and Game 2, which exactly corresponds to the pseudorandomness of a PRF, as described in Figure 2. Then

$$|\Pr[\mathcal{S}_1] - \Pr[\mathcal{S}_2]| = \text{Adv}_{F, \mathcal{A}}^{\text{prf}}(\lambda). \quad (4)$$

Here, the counter c does not change anything since, in the PRF security experiment, the input M is known to the adversary, as c (and the plain traffic) in our game.

At the end of this game, the traffic is then composed of (i) the set of random values $\{p_j^{(b)}\}_j$ and (ii) the used random counter c_0 and salt. That is, \mathcal{A} can only get b' by random guess. Obviously, $\Pr\mathcal{S}_2 = 1/2$.

Using additionally the results given in Equations (3) and (4), we finally have

$$\text{Adv}_{\mathcal{A}}^{\text{sp-tr-ind}}(\lambda) \leq 2(\text{Adv}_{\text{BE}, \mathcal{A}}^{\text{be-ind}}(\lambda) + \text{Adv}_{F, \mathcal{A}}^{\text{prf}}(\lambda)),$$

which concludes the proof.

Theorem 3 *Our scheme is traffic-indistinguishable against any adversary without the knowledge of sk_{SE} with*

$$\text{Adv}_{\mathcal{A}}^{\text{se-tr-ind}}(\lambda) = 2 \text{Adv}_{G, \mathcal{A}}^{\text{prf}}(\lambda).$$

Proof. We prove the result on adversaries who know SE's secret key $\text{sk}_{\text{SE}} = \text{mk}$ and has access to Send and RuleGen oracles. It outputs two plain traffics T_0 and T_1 . According to a bit b , an encrypted traffic is generated, as $E_b = \text{Send}(\text{param}, T_b)$, where $T_b = \{t_j^{(b)}\}_j$.

Game 0. This is the original attack game, where the encrypted traffic E_b is composed of (i) the set $\{q_j^{(b)} = G(K, \text{salt} + j) \oplus p_j^{(b)}\}_j$, where each $p_j^{(b)} = F(s, t_j^{(b)} \| c_j)$ and $s = \text{BE.Dec}(\text{Hdr}, \text{sk}_n)$, (ii) the used random counter c_0 and salt, and (iii) the TLS ciphertext e (not considered). The adversary \mathcal{A} eventually outputs a bit b' . Since \mathcal{A} knows mk , it knows the value s . Hence, \mathcal{A} can compute $p_j^{(0)}$ and $p_j^{(1)}$.

Let \mathcal{S}_0 be the event that the $b = b'$. Then we have

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{se-tr-ind}}(\lambda) &= |2\Pr[\text{Exp}_{\mathcal{A}}^{\text{se-tr-ind}}(\lambda) = 1] - 1| \\ &= |2\Pr[\mathcal{S}_0] - 1|. \end{aligned}$$

Game 1. We modify Game 0 by replacing the $G(K, \text{salt} + j)$ with by the output $f_\lambda(\text{salt} + j)$ for given salt, where the function f_λ chosen uniformly at random in the set of all functions mapping l -bit strings to n -bit strings. Let \mathcal{S}_1 be the event that the $b = b'$ in Game 1. Since K is unknown to \mathcal{A} and salt is randomly chosen for any new query, one can describe a distinguisher between Game 0 and Game 1, which exactly

corresponds to the pseudorandomness of the PRF G , as described in Figure 2. Then

$$|\Pr[S_0] - \Pr[S_1]| = \text{Adv}_{G,\mathcal{A}}^{\text{prf}}(\lambda).$$

At the end of this game, the traffic is then (removing the TLS ciphertext e) composed of (i) the set $\{q_j^{(b)} = f_\lambda(\text{salt} + j) \oplus p_j^{(b)}\}_j$ given as above, and (ii) the used random counter c_0 and salt. That is, the traffic is encrypted with a one-time pad. Obviously, $\Pr S_1 = 1/2$, and then

$$\text{Adv}_{\mathcal{A}}^{\text{se-tr-ind}}(\lambda) = 2 \text{Adv}_{G,\mathcal{A}}^{\text{prf}}(\lambda),$$

which concludes the proof.

3.3.3 | Rule indistinguishability

We now prove that our scheme is rule indistinguishable: it verifies both rule indistinguishability in the basic setting ($sp - rul - ind$) and in the high min-entropy one ($bme - rul - ind$). We thus prove the two following theorems.

Theorem 4 *Our scheme is rule-indistinguishable in the basic setting with*

$$\text{Adv}_{\mathcal{A}}^{\text{sp-rul-ind}}(\lambda) \leq 2(\text{Adv}_{\text{BE},\mathcal{A}}^{\text{be-ind}}(\lambda) + \text{Adv}_{F,\mathcal{A}}^{\text{prf}}(\lambda)).$$

Proof. Assume that the adversary \mathcal{A} knows SP's secret key $\text{sk}_{\text{SP}} = K$ and has access to the RuleGen and Send oracle. It outputs two sets of rules \mathcal{M}_0 and \mathcal{M}_1 . According to a bit b , a set \mathcal{B}_b of blinded rules is generated, as $\mathcal{B}_b = \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M}_b)$.

Game 0. This is the original attack game, where the blinded rules are given by the set $\mathcal{B}_b = \{B_{i,k}^{(b)}\}_{i,k}$ with $B_{i,k}^{(b)} = F(s, r_i^{(b)} \| c_k)$ with $r_i^{(b)} \in \mathcal{M}_b$ and $c_k \in [0, C - 1]$. The adversary \mathcal{A} eventually outputs a bit b' . Let S_0 be the event that the $b = b'$. Then we have

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{sp-rul-ind}}(\lambda) &= |2\Pr[\text{Exp}_{\mathcal{A}}^{\text{sp-rul-ind}}(\lambda) = 1] - 1| \\ &= |2\Pr[S_0] - 1|. \end{aligned}$$

As the adversary has chosen \mathcal{M}_0 and \mathcal{M}_1 , and since it has access to the Send oracle giving on input a known payload $T = \{t_j\}_j$ it simply has to generate a payload permitting to learn some information about the rules that are truly related to \mathcal{B}_b .

In fact, as for Theorem 2, we can first define Game 1 in which we replace the broadcast key s , output in $s = \text{BE.Dec}(\text{Hdr}, \text{sk}_n)$, by a random value in \mathcal{H} .

Notice that \mathcal{A} does not query RuleGen or Send oracle with the $r_i^{(b)}$'s. Moreover, the $B_{i,k}^{(b)}$'s are distinct due to the varying counter c_k . Then, we can transform Game 1 to Game 2

in which we replace the $B_{i,k}^{(b)}$'s with truly random values in $\mathcal{H}(1^\lambda)$. As the blinded rules are no more related to input rules in \mathcal{M}_b , this is obvious that $\Pr S_2 = 1/2$.

We then have

$$\text{Adv}_{\mathcal{A}}^{\text{sp-rul-ind}}(\lambda) \leq 2(\text{Adv}_{\text{BE},\mathcal{A}}^{\text{be-ind}}(\lambda) + \text{Adv}_{F,\mathcal{A}}^{\text{prf}}(\lambda)),$$

which concludes the proof.

Theorem 5 *In the random oracle model, our scheme is rule-indistinguishable in the high min-entropy setting with*

$$\text{Adv}_{\mathcal{A}}^{\text{bme-rul-ind}}(\lambda) \leq \text{Adv}_{\text{SHA-256},\mathcal{A}}^{\text{ow-prf}}(\lambda) + 2^{-\mu(\lambda)+1}.$$

Proof. Assume that the adversary \mathcal{A} knows sender and receiver's secret keys $\text{sk}_S = \text{sk}_R = (\text{sk}_m, K, k)$. Hence, \mathcal{A} knows the key s to the PRF F . It outputs two sets of rules \mathcal{M}_0 and \mathcal{M}_1 . According to a bit b , a set \mathcal{B}_b of blinded rules is generated, as $\mathcal{B}_b = \text{RuleGen}(\text{param}, \text{sk}_{\text{SE}}, \mathcal{M}_b)$.

Game 0. This is the original attack game, where the blinded rules are given by the set $\mathcal{B}_b = \{B_{i,k}^{(b)}\}_{i,k}$ with $B_{i,k}^{(b)} = F(s, r_i^{(b)} \| c_k)$ with $r_i^{(b)} \in \mathcal{M}_b$ and $c_k \in [0, C - 1]$. The adversary \mathcal{A} eventually outputs a bit b' . Let S_0 be the event that the $b = b'$. Then we have

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{bme-rul-ind}}(\lambda) &= |2\Pr[\text{Exp}_{\mathcal{A}}^{\text{bme-rul-ind}}(\lambda) = 1] - 1| \\ &= |2\Pr[S_0] - 1|. \end{aligned}$$

In this case, the adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ can create any traffic of its choice, since it has access to the sender's key. But, as we fall into the high min-entropy setting, \mathcal{A}_f and \mathcal{A}_g cannot communicate with each other, and \mathcal{A}_g has no chance to obtain one element in \mathcal{M}_b 'by chance'.

Game 1. We modify Game 0 to Game 1 by adding an abort when the adversary makes use of a rule included in \mathcal{M}_b . Let S_1 be the event that the $b = b'$ in Game 1. Obviously, the difference between S_1 and S_0 is given by the high min-entropy (see Section 2.3), and thus we have

$$|\Pr[S_0] - \Pr[S_1]| \leq 2^{-\mu(\lambda)},$$

where $\mu(\lambda) \in \omega(\log \lambda)$ according to the high min-entropy property of the rule set.

After Game 1, considering keyed SHA-256 for the PRF F , the aim of the adversary \mathcal{A} is to distinguish, among the unknown sets \mathcal{M}_0 and \mathcal{M}_1 , which of the two has been used to compute $\mathcal{B}_b = \{\text{SHA-256}(s, r_i^{(b)} \| c_k)\}_{i,k}$. Applying the high min-entropy, \mathcal{A} has no way to find one of the $r_i^{(b)}$ by chance. Using the technique introduced by Bellare and Rogaway [5], we can prove by contradiction and in the random oracle model, that we can use such distinguisher \mathcal{A} to break the one-wayness of the keyed SHA-256

(see Section 3.1). For this purpose, we construct a machine that is given a key K and a value $R = \text{SHA-256}(K, M)$ for an unknown input M . We set s as the key K , and embed the challenge R in the set \mathcal{B}_b that is sent back to \mathcal{A} . Our machine then watches for random oracle queries that \mathcal{A} makes related to SHA-256 . If there is one such query (K, M) for which $R = \text{SHA-256}(K, M)$, then the machine outputs M . As in [5], we argue that \mathcal{A} has no advantage in distinguishing \mathcal{M}_0 and \mathcal{M}_1 in case that \mathcal{A} does not ask for the such image. So our machine wins with non-negligible probability, which permits us to conclude that

$$|2\Pr[S_1] - 1| = \text{Adv}_{\mathcal{A}}^{\text{Game1}}(\lambda) = \text{Adv}_{\text{SHA-256}, \mathcal{A}}^{\text{ow-prf}}(\lambda),$$

and also conclude our proof.

4 | EXTENSIONS OF PROPOSED PROTOCOL

Our primary scheme in Section 3.2 performs a single keyword match over encrypted traffic. This section presents some extensions to support more sophisticated rules which enable limited IDS and then discusses the support of regular expressions.

4.1 | Supporting limited IDS

4.1.1 | Detecting rules with attributes

IDS's rules typically contain some attributes of suspicious keywords. They can be categorised as a packet field or a range of positions in the packet payload. For these rules, single keyword matching can no longer work. A direct solution is to reveal the attributes to Service Provider SP as illustrated in BlindBox [6]. However, even revealing the inspection positions indicated in the rule can threaten the privacy of endpoints since the SP could deduce which applications or protocols the endpoint runs [12].

Construction (Sketch)

To ensure strong protection on the rules and traffic, we adapt the rule and token generation methods, where the rule attributes are concatenated to the keywords. Note that the attribute and the keywords together are sent to PRFs, it is protected while the detection can still function correctly. To be more specific, the rules and tokens are generated by

$$F(s, r_i || str || c_k) \text{ and } F(s, t_j || str || c)$$

in RuleGen and Send respectively, where str represents any attribute of rule r_i , for example, 'http_header'. Another example is the offset information of keywords in the payload. In this case, all possible positions in a certain range should be

given the value of str . Consequently, the SP can detect traffic which matches rules with attributes.

4.1.2 | Detecting rules with multiple conditions

Many malicious behaviours can only be identified under multiple conditions. Hence, some detection rules check multiple tokens simultaneously to reduce false positives [15]. The trivial approach of attaching the rule id to its action would leak too much information on the traffic and rule. Instead, secret sharing-based encryption can be employed. The reader is referred to [12] for more details.

4.2 | Support of regular expressions

In general, our scheme cannot support more sophisticated rules, that is, regular expressions. BlindBox proposed a workaround, where the decryption key is accessible in case of a match on the patterns [6]. The Service Provider SP then decrypts the content and evaluates the regular expression on clear-text. This does not reduce the false negative rate, that poses the higher threat, but only the false positive rate, and at the cost of the privacy.

In [12], an alternative approach is proposed in the use case where the *admin server* plays the role of a Security Editor SE and the SP is semi-honest. The idea is that SP sends the warning and the suspicious packets to SE for the security consideration, who will enforce the endpoints to hand over the TLS key for decryption, or it will stop the connection. In this way, the decryption is performed by the trusted admin server (SE) instead of the untrusted SP.

As a result, general-purpose IDS over encrypted data mostly focus on pattern matching. Moreover, experiments in [12] show that only a very small portion of packets are matched which require the support of sophisticated rules, for example, less than 1% in instruction detection traffic dumps with over 30 million packets.

5 | IMPLEMENTATION AND VALIDATION

In this section, we focus on the implementation and validation of our approach. We evaluate the performances of our protocol and compare the obtained results with the classical HTTPS and the two main existing results: BlindBox and BlindIDS.

5.1 | Implementation details

5.1.1 | Implementation environment

We implemented our protocol given in Section 3.2 on an Intel (R) Xeon(R) with a E5-2637 processor with four cores running at 3.70 GHz, in C language running in a 64-bit Linux OS.

5.1.2 | Cryptographic choices

To implement our protocol, we have chosen the keyed hash function HMAC-*SHA-256* as a pseudorandom function and *SHA-256* again as a hash function. This gives us the output size of 256 bits for F and G respectively (which corresponds to the size of the p_j 's and the q_j 's). The size of the keys s and K is moreover defined as 128 bits, as prescribed by most of government agencies. For all these functions, we have used the LibTomCrypt library.

Regarding the broadcast encryption, we have chosen to use the basic LSD scheme [16], which is an improvement of the SD scheme [17] in terms of key manipulation. We have considered a group of 2^{32} users (corresponding to 4.3 billions, which seems to be enough in practice). Following [16], the decryption procedure necessitates 31 executions of a pseudorandom function and each user has to memorise about 180 keys (128-bit length).

In our benchmarks, we only consider the decryption phase. In fact, the key generation, extraction and encryption ones are done by the Security Editor during an off-line phase, and are then of less importance to compare our solution with related work. Assuming that the group of users may have evolved since the last connection of a user, we moreover take into account the time needed to execute this decryption procedure at each Send procedure.

5.2 | Functional tests

We consider the same framework as BlindBox and BlindIDS regarding rules and traffic. At first, the traffic we use comes from the ICTF dataset (<https://ictf.cs.ucsb.edu/>) which contains network traces collected during an 'International Capture The Flag' event.

Regarding the rules, we also refer to the same public datasets for detection functionalities related to (i) malwares [18, 19], (ii) parental control [20] and (iii) general rules [15]. For all these rules' datasets, our solution can be applied using comparison and perfect matches with either the URL header field [18, 20] or hexadecimal strings or text keywords contained in the entire packet [15, 19].

In all cases, the rules we can manage are based on some static contents, and we are obviously not capable to manage general regular expression. In the first case, we support most of the proposed entries. Contrary to [6, 7], we, however, consider that 100% is not really possible since there are some rules containing some URL blacklists that can be easily obtained. For example, if the adversary is both the sender and the receiver, this can be used to break the rule indistinguishability property and obviously does not fall into the 'high-min entropy' setting. In the second case, as we cannot manage regular expressions, we can only manage three-fourths of the proposed entries (see bib24[6, 7] for details).

Regarding the ability of our solution to detect attacks, as the structure of our solution is exactly the same as in the BlindBox one [7], as we use the same tokenisation strategy

applied to the same dataset, it is obvious that we can achieve the same accuracy as in BlindBox and BlindIDS (again, see Refs. [6, 7] for details). More precisely, there are two tokenisation algorithms. At first, *window-based tokenisation* produces fixed-length tokens: for every offset in the traffic, the sender creates a token of a fixed length. Then, *delimiter-based tokenisation* gives variable length tokens: each token starts and ends before or after a specific delimiter such as a punctuation, a spacing, or a special symbol.

Using this framework, we compare our solution with standard HTTPS and with the two main existing solutions, namely BlindBox [6] and BlindIDS [7]. This comparison is done based on the setup and encryption time on sender/receiver's side, key size for sender/receiver, detection time on the Service Provider's one and RAM usage for the Service Provider. We provide several results based on the size of the considered traffic and the number of rules that have been edited by the Security Editor.

5.3 | Performance comparison

We can now evaluate the performance of our solution. The result is given in Table 1, together with our comparison with related work. Note that the source codes of [6, 7] are not publicly available, so we only compare our results with the one reported in their article. Our implementation environment is equivalent to the one used in BlindIDS, and quite close to the one in BlindBox. Our purpose is not to give an exact comparison but to compare the orders of magnitude.

In Table 1, we use the figures in Ref. [6] for HTTPS and BlindBox and take the figures in [7] for BlindIDS. We do not give the figures for detection overhead of HTTPS (n.a. in Table 1) since the standard HTTPS cannot perform intrusion detection over encrypted traffic.

These figures definitely show that our solution is very performing and better than related works in all aspects.

5.3.1 | Connection setup

As for BlindIDS, our solution does not impact the setup time for a connection, while the BlindBox one depends on the number of rules to be tested, to generate the garbled circuits.

5.3.2 | Sender/receiver cost

Regarding the sender/receiver side, we should study at first the encryption/decryption time. As shown in Table 1, our approach reduces by three orders of magnitude the time to encrypt/decrypt the traffic, compare to BlindIDS solution. This is due to the fact that we only manipulate symmetric cryptographic techniques. In comparison with BlindBox, one can see that the bigger the traffic size, the better will be our solution. Even if the broadcast decryption phase is expensive, it is done only once, whatever the size of the traffic. Finally, it is

TABLE 1 Performance and comparison of our solution with the standard HTTPS technique, the BlindBox, and the BlindIDS solutions

	Description	HTTPS	BlindBox	BlindIDS	Our solution
Connection time (Sender/Receiver)	Setup (1 keyword)	73 ms	595 ms	73 ms	73 ms
	Setup (3K rules)	73 ms	183 s	73 ms	73 ms
	Encrypt/Decrypt (128 bits)	13 ns	9.6 μ s	729 μ s	240 ns
	Encrypt/Decrypt (1500 bytes)	3 μ s	960 μ s	27 ms	6.5 μ s
Detection time (Service provider)	1 rule, 1 token	n.a.	20 ns	691 μ s	10 ns
	1 rule, 1 packet	n.a.	7.9 μ s	41.3 ms	980 ns
	3K rules, 1 token	n.a.	137 ns	700 ms	16 ns
	3K rules, 1 packet	n.a.	30 μ s	74 s	1.5 μ s
RAM usage (Service provider)	1 rule, 1 connection	n.a.	16.72 MB	0.2 KB	0.1 KB
	3K rules, 1 connection	n.a.	50.16 GB	0.58 MB	0.3 MB
	1 rule, 100 connections	n.a.	175 MB	0.2 KB	0.1 KB
	3K rules, 100 connections	n.a.	512 GB	0.58 MB	0.3 MB

obvious that we cannot compare to the HTTPS but we demonstrate here that we are not so far coming near the same order of magnitude.

Secondly, we should consider bandwidth cost and throughput, which correspond to key figures related to client performance. To measure how competitive our solution is, we compare the load time of it with the ones of BlindBox and BlindIDS, for some popular websites. In fact, in our case, we only have to consider that the traffic is sent approximately twice: first being encrypted using TLS, and second being encrypted (using PRFs) in the q_j 's. Then, as shown in [7], it necessitates about 97s to encrypt/decrypt a Twitter page of 284 KB using BlindBox, while BlindIDS can do that in about 5s. A CNN webpage (131 KB) necessitates 2.3s using BlindIDS and again 97s using BlindBox. A Facebook page (74 KB) is loaded in about 1s using BlindIDS and 97s using BlindBox. Using our solution, the resulting time for all these websites is less than 100 ms, which is very close to the result of the current standard HTTPS protocol!

5.3.3 | Detection

We also evaluate and compare the overhead for the Service Provider during detection. We then measure the memory space and the time required to perform detection, according to the number of detection rules (from 1 rule to 3K rules) and the size of the network connections (from 1 token of 128 bits to 1 packet of 1500 bytes). Again, our performances regarding the time needed to test all the rules is quite similar to the one of BlindBox. Even if it seems that our figures are a little bit better, both implementations have not been done in an optimised manner, and as the one from BlindBox has been done in 2015, several optimizations are certainly available today. However, compared to BlindIDS, we are widely more performing, up to seven orders of magnitude for 3K rules and a packet of 1500 bytes! Again, symmetric cryptography is definitely better than

pairing based asymmetric cryptography in terms of performance.

On RAM usage, we are much better than BlindBox and we are close to BlindIDS. This is due to the fact that we have similarly replaced the garbled circuits of BlindBox by some trapdoors that are derived from the malicious keywords in the detection rules: for each rule, there is one single version of its blinded version. As for BlindIDS, the difference is due to the fact that we are using symmetric cryptography techniques, compared to the bilinear pairing setting of BlindIDS.

5.4 | Real-life deployment

Similar to related works, it remains a lot of work from our 'practical' results to a real-life deployment. In this section, we give some potential solutions to several concrete problems related to such potential deployment of our solution.

5.4.1 | Management of parameter C

The parameter C determines how many times the same rule will be encrypted and how many times the Service Provider will check for the same rule. Therefore it directly ties to the cost of detection, even though being limited, as shown in Table 2. It has also an important impact on the RAM usage since it turns out to be linearly dependent.

If C is large, there will be a lot of blinded rules and the detection procedure will be costly, both being in $\mathcal{O}(C|\mathcal{M}|)$. If C is short, detection time will be short but the risk is that several users encrypt the same token t_j in the payload with the same value $c \in \mathbb{Z}_C$. The Service Provider will then obtain the information that the same token has been encrypted in two different payloads, then breaking the traffic indistinguishability property. We have a trade-off to manage, which strongly depends on the real-life setting in which such solution is deployed, depending on

the number of rules, the number of users and the amount of traffic to be managed by the Service Provider.

Except increasing the parameter C , one solution to decrease the risk of breaking the traffic indistinguishability is for the Security Editor to regularly edit a new set of blinded rules (from the same set \mathcal{R} of rules). This obviously cannot be done by changing the value C . One option is to replace the set $[0, C - 1]$ by a randomly chosen subset of size C in a much larger set (e.g., \mathbb{Z}_q for a very large integer q). Another option (that can be used together with the previous one) is for the Security Editor to execute again the broadcast encryption procedure $\text{BE.Enc}(mk, I)$ to get a fresh couple (Hdr, s) . The header is broadcast to all users and the secret s is used to produce a new set of blinded rules.

5.4.2 | Key management

One main difference of our system compare to existing ones is the fact that we are using symmetric key cryptography. As we have seen, the advantage is that our performances are remarkably better than those of BlindBox and BlindIDS. The main downside regarding this choice is that a real-world deployment should have to pay more attention on the way keys should be managed. More precisely, the same key needs to be shared among a lot of people: the key s among SE, senders and receivers, and the key K among SP, senders and receivers. The compromising of such keys may lead to the deterioration of the security properties. In short, our security is more sensitive to coalitions between the different actors.

5.4.3 | Intranet versus internet deployment

In practice, there are two ways to deploy such kind of solution.

On one hand, one single entity can manage both the client and the server. For example, in case of an intranet, a company can explicitly instal to its employees' PCs a dedicated plugin or extension permitting the latter to make use of the encryption scheme described herein. As the company also manages the servers for the intranet services (related to vacations, travels, declaration of worked hours, etc.), it can also implement this encryption algorithm on those dedicated servers. The company can finally manage a proxy to perform the detection procedure, as described in the previous sections. If one of the two parties (PC or intranet servers) does not make use of the right encryption algorithm (because it has been corrupted by an external

attacker), the other will detect it, and refuse the connection, so as to remain safe. If both parties apply our encryption mechanism, then the proxy will necessarily detect an intrusion.

On the other hand, if one single entity cannot manage both parties of a communication (client and server), we need to first standardise the whole encryption and detection mechanism (potentially at both ISO and IETF sides) and then provide the relevant implementation (modifying the cipher suites in e.g. TLS/SSL for both clients and servers) so that it can largely be deployed. Then, a server can for example refuse any traffic not implementing this traffic encryption solution, and the client can also refuse to connect to any server not accepting those algorithms, so that the detection is always performed, keeping the whole system safe.

5.4.4 | Additional meta-data treatment

As a complement, there are some existing methods to perform intrusion detections by making use of the meta-data, that are most of the time not encrypted. In fact, both approaches are compatible, and complementary. In a real-life deployment, the proxy can act in two steps: (i) perform an intrusion detection using the non-encrypted meta-data, and (ii) perform a second intrusion detection in the encrypted data, using our method.

6 | RELATED WORK

This section reviews some related work on intrusion detection over encrypted traffic. We focus on works most relevant to us: *multi-party computation*-based BlindBox, *public key searchable encryption*-based BlindIDS, pattern matching on encrypted streams, *searchable symmetric encryption schemes* and finally hardware-based solution EVE. To compare with the two closest works, BlindBox and BlindIDS, we also give at the end of this section a comparison in terms of security and functionality.

6.1 | Main papers

6.1.1 | BlindBox, Embark and PrivDPI

The BlindBox paper [6] proposes three distinct detection protocols supporting DPI over encrypted traffic. They all support equality tests between the encrypted traffic and the rules defined by the Security Editor. In the third solution, the Service Provider can also retrieve the decryption key embedded into the trapdoor used for equality test, permitting a full decryption of the traffic, and then the possibility for SP to operate a full IDS (but with no more confidentiality).

Those solutions are all based on garbled circuits and oblivious transfers. The idea is to execute a garbled circuit evaluation for each TLS connection, and for each detection rule to be tested. The secret key used to encrypt the traffic is secretly embedded into the garbled circuit by the sender and the SP deterministically encrypts each pattern to be tested using that key

TABLE 2 Performance of detection for different values of the C parameter

Parameter C		100	1000	10,000	100,000
Detection time	1 rule, 1 token	10 ns	10 ns	10 ns	10.2 ns
	3K rules, 1 packet	1.4 μ s	1.5 μ s	2.3 μ s	7.5 μ s
RAM usage	1 rule, 1 token	10 B	0.1 KB	1KB	10 KB
	3K rules, 1 packet	30 KB	0.3 MB	3MB	30 MB

(but without knowing it, using oblivious transfer techniques). Then, the garbled circuit is executed in a 2-Party protocol, and finally outputs the decision on the safeness of the traffic.

As shown in the previous section and in Ref. [7], this process should be done at each TLS connection, and then drastically increases the setup time (97 s according to Ref. [6]). Moreover, the memory space needed is proportional to the number of (i) unique receivers to be protected, (ii) unique TLS connections, and (iii) unique detection rules. Thus, even if the BlindBox authors show that such garbled circuits and oblivious transfer techniques can be very efficiently implemented, this is definitely not enough efficient for a practical privacy-friendly IDS over an encrypted traffic. BlindBox is not scalable, and has serious limitation compare to the market ecosystem for network security solutions.

The Embark system [21], defined by the authors as an extension of BlindBox, does not treat these limitations. It only provides a solution to securely outsource network middleboxes to the cloud.

Very recently, the PrivDPI system [8] has been proposed to improve BlindBox. It provides a more practical variant of the encrypted rule generation, while keeping the same level of security and privacy.

6.1.2 | BlindIDS

BlindIDS [7] takes a different approach. It offers to encrypt the patterns *only once* for all the TLS connections, using public key cryptography, and more specifically decryptable searchable encryption [22] (DSE). The idea is for the sender to encrypt the traffic using the DSE, to determine which traffic can be decrypted by the receiver. In parallel, the Security Editor provides to the Service Provider one trapdoor for each pattern to be tested on the encrypted traffic, using the testing procedure of the DSE scheme.

Then, BlindIDS improves the BlindBox scheme in two aspects. At first, the connection setup time is constant since all trapdoors are computed only once for all TLS traffic. Similarly, the memory space required to perform DPI only depends on the number of detection rules, and no longer on the number of receivers nor the number of concurrent TLS connections. Another positive consequence is that the Service Provider no longer knows the detection patterns it is searching in the encrypted traffic, due to the properties of the DSE. However, the use of a public-key cryptography, and especially pairings, comes with an increasing decryption overhead on the receiver side. Then, the BlindIDS is still not enough for a real-world use.

6.1.3 | Pattern matching on encrypted streams

Recently, Desmoulins et al. [23] have proposed a pattern matching system over encrypted streams. They introduce a new kind of searchable encryption that manages so-called 'shiftable trapdoors'. While BlindBox and BlindIDS only permits to detect patterns that perfectly match one substrings, this

solution permits to detect a pattern even if it straddles two substrings. This solution then permits to manage many more rules than related work, but the detection procedure is about 10 times less efficient than the one of BlindIDS.

6.1.4 | Searchable symmetric encryption

Searchable symmetric encryption (SSE) enables a user to encrypt data in such a way that it can later generate search tokens to send as queries to the storage server [11]. An immediate application of SSE is to the design of searchable cryptographic cloud storage systems. Considering the efficiency of underlying symmetric key primitives, SSE seems a promising alternative to public key searchable encryption schemes used in BlindIDS [7] and in Ref. [23]. However, directly employing the existing SSE protocol [11, 24] cannot meet all our design requirements. To be specific, SP needs to match the encrypted traffic from S and encrypted rules from SE. To this end, S and SE should respectively encrypt data and generate search trapdoors for the given keywords. In either single-user or multi-user SSE schemes, for example, SSE-1 and MSSE in [11], the two actors S and SE share exactly the same secret keys for data encryption and trapdoor generation. Hence the SE can break the traffic indistinguishability trivially. Therefore, the existing SSE scheme cannot be directly used in our case.

6.1.5 | EVE

Very recently, a new proposal was suggested in [25]. The proposed framework and the implemented EVE platform have been built to manage a secure and practical middlebox that handles encrypted traffic. For this purpose, the authors make use of a combination of hardware-based trusted execution (using the Intel-SGX component) and software security technology. The main idea is to give a copy of the traffic to such hardware that can decrypt it, execute the rules on the plain traffic, and give the result. The authors newt improve the performances by applying several software based tricks. Indeed, the result does not rely on cryptographic security but rather on the fact that it is not feasible to break the security of a hardware (embedded keys are secure, and the execution environment is safe), which may be seen as quite questionable (see e.g. this recent paper [26]).

6.2 | Functionality and security comparison

Regarding the above 'categories' of solutions, our construction can be seen as an improvement of the BlindIDS concept, that is making use of decryptable searchable encryption, but in the symmetric key setting, using techniques from searchable symmetric encryption.

If we now focus on the functionalities and security requirements, there are some differences between BlindBox, BlindIDS and our solution that we now detail.

- *Privacy-friendly* means that no access is possible to the plaintext related to encrypted traffic: this property is similarly verified by the three solutions, except with the third protocol proposed in BlindBox, for which the service provider is allowed to decrypt the whole traffic when it detects suspicious tokens when executing the two first protocols.
- *Security-aware* means that the solution supports DPI over encrypted traffic. This property is satisfied by the three solutions. Furthermore, extensions of our original protocol presented in Section 4 can achieve at least the same functionalities as Protocols II in BlindBox. Discussions on the support of full IDS functionality have been given in Section 4.2.
- *Market-compliant* means that each party (Security Editor and Service Provider) preserves its own know-how without revealing it to the other parties. This property is not verified in BlindBox as the Service Provider is required to have a direct plain access to the SE rules², which is definitely not market-compliant, since the SE will be very reluctant to share their detection rules with SPs. In contrast, both BlindIDS and our solution succeed in verifying this property. Our solution naturally necessitates the Service Provider to manage a secret key, while this is not the case for BlindIDS, but this can be added quite easily by managing an additional encryption layer with a Service Provider key.
- *Security level*: all the three solutions reach the same level of security. The only exception is that the BlindBox does not achieve the rule indistinguishability against the Service Provider property, as explained above.

7 | CONCLUSION

We have provided a new approach to intrusion detection over an encrypted traffic. While our general framework is close to BlindIDS, the fact that we make use of the symmetric cryptography makes things far more efficient, and comparable to the state-of-the-art BlindBox in terms of encryption and detection time. It is today possible to obtain the best of all existing solutions in one system: efficient setup, low memory consumption, rules' confidentiality against service providers and efficiency of the whole protocol. The current drawback of our solution is that we need to manage the counter $c < C$, which asks the Security Editor to provide C 'trapdoors' for each rule. The way to prevent the use of such trick can be very good in the future.

ORCID

Chaoyun Li  <https://orcid.org/0000-0001-9917-3419>

REFERENCES

1. Li, C.: New methods for symmetric cryptography. Ph.D. thesis, KU Leuven (2020)

2. SonicWall cyber threat report. <https://www.sonicwall.com/fr-fr/news/sonicwall-cyber-threat-report-2018/> (2018)
3. Dell security Annual threat report. (2015). https://www.bitpipe.com/detail/RES/1431453319_167.html
4. Dell security annual threat report, (2016). <http://www.netthreat.co.uk/assets/assets/dell-security-annual-threat-report-2016-white-paper-197571.pdf>
5. Dell security annual threat report, (2018). <https://www.dell.com/learn/us/en/vn/press-releases/2017-04-20-dell-end-user-security-survey-highlights-unsafe-data-security-practices-in-the-workplace>
6. Sherry, J., et al.: BlindBox: deep packet inspection over encrypted traffic. In: ACM SIGCOMM 2015, London, pp. 213–226 (2015)
7. Canard, S., et al.: BlindIDS: market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In: AsiaCCS 2017, Abu Dhabi, pp. 561–574 (2007)
8. Ning, J., et al.: PrivDPI: privacy-preserving encrypted traffic inspection with reusable obfuscated rules. In: ACM Conference on Computer and Communications Security 2019, London, pp. 1657–1670 (2019)
9. Bellare, M., et al.: Deterministic encryption: Definitional equivalences and constructions without random oracles. In: CRYPTO 2008, Santa Barbara, CA, USA, pp. 360–378 (2008)
10. NIST Special Publication (SP): Rev. 1: Recommendation for applications using approved hash algorithms (800-1072012)
11. Curtmola, R., et al.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security 2006, Alexandria, pp. 79–88 (2006)
12. Yuan, X., et al.: Privacy-preserving deep packet inspection in outsourced middleboxes. In: IEEE INFOCOM, San Francisco, pp. 1–9 (2016)
13. Fiat, A., Naor, M.: Broadcast encryption. In: CRYPTO 1993, Santa Barbara, CA, USA, pp. 480–491 (1993)
14. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. In: CRYPTO 2005, Santa Barbara, CA, USA, pp. 258–275 (2005)
15. Snort. (2016). <https://www.snort.org/downloads/>
16. Halevy, D., Shamir, A.: The LSD broadcast encryption scheme. In: CRYPTO 2002, Santa Barbara, CA, USA, pp. 47–60 (2002)
17. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for Stateless receivers. In: Electronic Colloquium on Computational Complexity (ECCC), number 043 (2002)
18. Malware domain list. (2016). <https://www.malwaredomainlist.com/mdl.php>
19. Yara rules repository, <https://github.com/Yara-Rules/rules> (2016)
20. URL blacklist. (2016). <http://www.urlblacklist.com/?sec=home>
21. Lan, C., et al.: Embark: securely outsourcing middleboxes to the cloud. In: NSDI 2016, Santa Clara, pp. 255–273 (2016)
22. Fuhr, T., Paillier, P.: Decryptable searchable encryption. In: ProvSec 2007, Wollongong, Australia, pp. 228–236 (2007)
23. Desmoulins, N., et al.: Pattern matching on encrypted streams. In: ASIACRYPT (1) 2018, Brisbane, QLD, Australia, pp. 121–148 (2018)
24. Cash, D., et al.: Dynamic searchable encryption in very large databases: data structures and implementation. In: NDSS 2014, San Diego (2004)
25. Han, J., et al.: A secure middlebox framework for enabling visibility over multiple encryption protocols. IEEE/ACM Trans. Netw. 1–14 (2020)
26. Chen, Z., et al.: VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface, vol. 21. USENIX Security. to appear
27. Yara rules repository, <https://github.com/Yara-Rules/rules> (2016)

How to cite this article: Canard S, Li C. Towards practical intrusion detection system over encrypted traffic. *IET Inf. Secur.* 2021;15:231–246. <https://doi.org/10.1049/ise2.12017>

²The obfuscated rule encryption technique used in BlindBox is only used to protect the set of rules w.r.t. users, but not the Service Provider.