Privacy-Preserving Pattern Matching on Encrypted Data

Anis Bkakria[1], Nora Cuppens[1,2], and Frédéric Cuppens[1,2]

[1] IMT Atlantique, Rennes, France
{firstname.lastname}@imt-atlantique.fr
[2] Polytechnique Montréal, Montréal, Canada
{firstname.lastname}@polymtl.ca

Abstract. Pattern matching is one of the most fundamental and important paradigms in several application domains such as digital forensics, cyber threat intelligence, or genomic and medical data analysis. While it is a straightforward operation when performed on plaintext data, it becomes a challenging task when the privacy of both the analyzed data and the analysis patterns must be preserved. In this paper, we propose new provably correct, secure, and relatively efficient (compared to similar existing schemes) public and private key based constructions that allow arbitrary pattern matching over encrypted data while protecting both the data to be analyzed and the patterns to be matched. That is, except the pattern provider (resp. the data owner), all other involved parties in the proposed constructions will learn nothing about the patterns to be searched (resp. the data to be inspected). Compared to existing solutions, the constructions we propose has some interesting properties: (1) the size of the ciphertext is linear to the size of plaintext and independent of the sizes and the number of the analysis patterns; (2) the sizes of the issued trapdoors are constant on the size of the data to be analyzed; and (3) the search complexity is linear on the size of the data to be inspected and is constant on the sizes of the analysis patterns. The conducted evaluations show that our constructions drastically improve the performance of the most efficient state of the art solution.

Keywords: Searchable encryption · Pattern Matching

1 Introduction

In several application domains such as deep-packet inspection and genomic data analysis, learning the presence of specific patterns as well as their positions in the data are essential. In the previous two use cases, pattern searches are often performed by entities that are not fully trusted by data owners. For instance, in the case of deep-packet inspection (DPI), a company that aims to outsource its network traces to a third party forensic scientist to find indictors of compromise might not be comfortable revealing the full contents of its traces to the forensic scientist. Similarly, in the case of genomic data analysis, a patient that wants to check whether its genome contains particular patterns representing a genetic

predisposition to specific diseases might not be comfortable revealing the full contents of its genome to the laboratory that performs the analysis.

Existing solutions that may be used to overcome the previous problem rely mainly on searchable encryption based techniques [1–6]. Unfortunately, these techniques suffer from at least one of the following limitations. First, the lack of support for pattern-matching with evolving patterns, such as virus signatures which are updated frequently (case of symmetric searchable encryption [2–4]); second, the lack of support for variable pattern lengths (e.g., tokenization-based techniques such as BlindBox [5]); third, the incompleteness of pattern detection methods which yield false negatives (case of BlindIDS [6]); and fourth, the disclosure of detection patterns (case of searchable encryption with shiftable trapdoors [1]). We provide a full comparison with related literature in Section 2.

In this paper, we propose two technically sound constructions: $\mathbf{S^4E}$ supporting pattern matching of adaptively chosen and variable (upper bounded) lengths patterns on secret key encrypted streams, and $\mathbf{AS^3E}$ supporting pattern matching of adaptively chosen and variable (upper bounded) lengths patterns on public key encrypted streams. Both $\mathbf{S^4E}$ and $\mathbf{AS^3E}$ ensure that (1) both the data owner and the third-party entity performing pattern matching operations will learn nothing about the searched patterns except their lengths, (2) both the pattern provider and the third-party entity that is going to perform pattern matching will learn nothing about the data to be analyzed except the presence or the absence of the set of unknown patterns (i.e., the third-party entity will not have access to patterns plaintexts), (3) the third-party entity will be able to perform pattern matching correctly over the data to be analyzed. From a practical point of view, our construction has some interesting properties. First, the size of the ciphertext depends only on the size of the plaintext (it is independent of the sizes and the number of analysis patterns). Second, the size of the issued trapdoors is independent of the size of the data to be analyzed. Third, the search complexity depends only on the size of the data to be analyzed and is constant on the size of the analysis patterns. The two constructions we propose in this paper are – to our knowledge – the first constructions to provide all previously mentioned properties without using costly and complex cryptographic scheme such as fully homomorphic encryption. The conducted evaluations show that the two proposed constructions improve by up to four orders of magnitude the performance of the most efficient state of the art solution SEST [1].

The paper is organized as follows. Section 2 reviews related work and details the main contributions of our work. Section 3 presents the assumptions under which our schemes achieve provable security. The intuition behind the proposed constructions is presented in Section 4. Section 5 and 6 formalize our S^4E and AS^3E primitives and provide their security results. In Sections 7 and 8, we discuss the complexity of our constructions and provide experimental results. Finally, section 9 concludes.

2 Related Work

One possible solution for pattern matching over encrypted traffic is to use techniques that allow evaluation of functions over encrypted data. Generic approaches such as fully homomorphic encryption (FHE) [7, 9] and functional encryption (FE) [8] are currently impractical due to their very high complexities.

Several searchable encryption (SE) techniques have been proposed for keyword searching over encrypted data [2–4]. The main idea is to associate a trapdoor with each keyword to allow searching for these keywords within a given encrypted data. Ideally, an entity which does not have access to the plaintext and encryption key should learn nothing about the plaintext except the presence or the absence of the keyword. For most existing SE techniques, searches are performed on keywords that have been pre-chosen by the entity encrypting the data. Such approaches are more suitable for specific types of searches, such as database searches in which records are already indexed by keywords, or in the case of emails filtering in which flags such as "urgent" are used. Unfortunately, SE techniques become useless when the set of keywords cannot be known before encryption. This is usually the case for messaging application and Internet browsing traffic where keywords can include expressions that are not sequences of words *per se* (e.g., /chamjavanv.inf?aapf/login.jsp?=). The two constructions we propose in this paper offer better search flexibility as, even after the plaintext has been encrypted, they can allow arbitrarily chosen keywords to be searched without re-encryption.

To overcome the previous limitations, tokenization-based approaches have been proposed. In [5], the authors propose BlindBox, an approach that splits the data to be encrypted into fragments of the same size l and encrypts each of those fragments using a searchable encryption scheme where each fragment will represent a keyword. Nevertheless, this solution suffers from two limitations: (1) it is useful only if all the searchable keywords have the same length l. Obviously the previous condition is seldom satisfied in real-world applications that requires pattern matching (e.g., DPI). If we want to use this approach with keyword of different lengths $\mathcal{L}$, we should for each $l_i \in \mathcal{L}$, split the data to be encrypted into fragments of size l_i and encrypt them, which quickly becomes bulky. (2) The proposed approach may easily cause false negatives since, even if the keyword is of size l (the size of each fragment), it cannot be detected if it straddles two fragments. Recently, in [6], Canard et al. proposed BlindIDS – a public key variant of the BlindBox approach [5] that additionally ensures keywords indistinguishability. That is, the entity that is going to search over the encrypted data will lean nothing about the keywords. Unfortunately, BlindIDS suffers from the same limitations as BlindBox. The two constructions we propose in this paper address the main drawbacks of these tokenization-based techniques since they allow for arbitrary trapdoors to be matched against the encrypted data, without false negatives or false positives.

Several approaches [10–12] proposed solutions for substring search over encrypted data based on secure multi-party computation. Unfortunately, to offer

pattern matching operation, these solutions require often several interactions between the searcher and the data encrypter.

As pointed out in [1], anonymous predicate encryption (e.g., [13]) or hidden vector encryption [14] may provide a convenient solution for pattern matching over encrypted data. However, in order to search a pattern p of length l on a data of length n, the searcher should obtain $n - l$ keys to be able to check the presence of p on every possible offset of the data, which is clearly a problem when dealing with large datasets.

One of the most interesting techniques for pattern matching over encrypted traffic is the searchable encryption with shiftable trapdoor (SEST) [1]. The proposed construction relies on public-key encryption and bilinear pairings to overcome most of the limitations of previously mentioned techniques. It allows for patterns of arbitrary lengths to be matched against the encrypted data, without false negatives or false positives. This improvement comes at the cost of the practicability of the technique. In fact, the proposed schema requires a public key of size linear to the size of the data to be encrypted (a public key of $\simeq 8000$ GB is required for encrypting 1GB of data). Moreover, the trapdoor generation technique used by the SEST leaks many information (such as, the number of different characters, the maximum number of occurrences of a character) about the patterns to be searched. Furthermore, the number of pairings needed for testing the presence of a keyword in an offset of the data depends on the maximum number of occurrences of the characters contained in the keyword. This makes the proposed technique quite inefficient when used for bit level matching. By contrast, for testing the presence of a pattern in encrypted data, our constructions require a constant number of pairings in the size of the pattern (see Section 7 for more details). This makes our constructions more efficient when matching long keywords at bit level.

As we have seen, many different approaches can be used to address pattern matching over encrypted data. To give better understanding of the benefits of the two approaches we propose in this paper compared to existing ones, we provide in Table 1 a comparative overview of their asymptotic complexities, and their ability to ensure the security properties we are aiming to provide. Note that we only consider BlindBox (a symmetric searchable encryption-based solution), BlindIDS (an asymmetric searchable encryption-based solution), Predicate Encryption/Hidden Vector Encryption and the SEST approach. Other approaches, as explained before, require data re-encryption each time a new keyword is considered [2–4], induce higher complexity [7–9], require interactivity [10–12] or ensure weaker privacy level [4].

According to the Table 1, the two constructions we propose in this paper (S^4E and AS^3E) are the only primitives that simultaneously enable arbitrary trapdoors (with upper bounded keyword size), provides a correct keyword detection, and ensures the privacy of the used trapdoors.

In Table 1, (✓) is used to denote that a property is provided under specific conditions. AS^3E ensures trapdoor's privacy for patterns of high-min entropy (see Section 6 for more details). In addition, both S^4E and AS^3E support pattern

	Primitives					
	BlindBox	BlindIDS	PE/HVE	SEST	S^4E	AS^3E
Number of Trapdoors	$O(s \cdot q)$	$O(q)$	$O(n \cdot q)$	$O(q)$	$O(q)$	$O(q)$
Public Parameters size	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(l_i)$	$O(1)$
Encryption keys size	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(l_i)$	$O(l_i)$
Ciphertext size	$O(n \cdot L)$	$O(n \cdot L)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Number of trapdoors	$O(q)$	$O(q)$	$O(n \cdot q)$	$O(q)$	$O(q)$	$O(q)$
Search complexity	$q \cdot \log(q)$ comparisons	q pairings	$q \cdot n$ pairings	$2 \times \prod_1^q l_i \cdot n$ pairings	$2 \cdot q \cdot n$ pairings	$2 \cdot q \cdot n$ pairings
Arbitrary trapdoors	✗	✗	✓	✓	(✓)	(✓)
Trapdoor's privacy	✗	(✓)	✗	✗	✓	(✓)
Correctness (no false positives)	✗	✗	✓	✓	✓	✓

Table 1: Complexity and ensured security properties comparison between related work and our primitive. The scalars n, q, l_i, L, s denotes respectively the length of the traffic to encrypt, the number of pattern to be searched, the length of each pattern, the number of different lengths among the q patterns to be searched and the number of data encrypters. We used (✓) to denote that the property is provided under specific conditions.

matching of arbitrary but upper bounded lengths patterns. As we show in Section 7, we stress that in both S^4E and AS^3E, increasing the upper bound size of patterns affects only the size of the trapdoor generated for each pattern. The size of later increases linearly with the increase of the size of the former.

The two constructions we propose do not require very large public parameters, secret key or very large public keys as SEST and PE/HVE. Moreover, their search complexities is lower than SEST by a factor of l_i (the length of the pattern w_i to be searched), since they are constant in the size of the pattern to be searched. Therefore, the proposed constructions are an interesting middle way which provides the best of PE/HVE and SEST while ensuring patterns' privacy. Their only limitation compared to PE/HVE and SEST is the upper bounded size of patterns to be searched that should be fixed before the data encryption, which we believe to be a reasonable price to pay to achieve all the other features.

3 Security Assumption

In this section, we describe the security assumptions under which our two constructions S^4E and AS^3E achieve provable security.

Definition 1 (Bilinear Maps). *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be three finite cyclic groups of large prime order p. We assume that there is an asymmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that, for all $a, b \in \mathbb{Z}_p$ the following conditions hold:*

– *For all* $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{a \cdot b}$
– *For all* $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, $e(g^a, \widetilde{g}^b) = 1$ *iff* $a = 0$ *or* $b = 0$
– $e(\cdot, \cdot)$ *is efficiently computable*

As in [1], the security of the proposed constructions hold as long as $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism exists between $\mathbb{G}_1$ and $\mathbb{G}_2$ in either directions. In the sequel, the tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot, \cdot))$ is refereed to as a bilinear environment.

Some of the security proofs of the proposed constructions, given in the full version of this paper [22], rely partially on showing that given a number of pattern trapdoors, the adversary will be unable to distinguish a new valid trapdoor from a random element. Thus, the leakage can be bounded only by considering the adversary's query to the issuing oracle. Hence, either we considerably reduce the maximum length of the patterns to be searched (≤ 30), which allow to define a GDH instance providing all public parameters, the trapdoors for all possible patterns, and the challenge elements. Or we use an interactive variant of the GDH assumption to offer flexibility to the simulator by allowing the elements $g^{R^{(i)}(x_1, \cdots, x_c)}$, $\widetilde{g}^{S^{(i)}(x_1, \cdots, x_c)}$, and $e(g, \widetilde{g})^{T^{(i)}(x_1, \cdots, x_c)}$ of the GDH assumption [19] to be queried to specific oracles.

So, we prove the security of the proposed constructions under an interactive assumption. That is, we use a slightly modified General Diffie-Hellman (GDH) problem assumption [19] to allow the adversary to request the set of values on which the reduction will break the GDH assumption. This interactive aspect of the GDH instance we are considering reduces slightly the security of the construction we are proposing. However, this interactive assumption makes possible the definition of quite efficient constructions with interesting properties. First, the size of the ciphertext depends only on the size of the plaintext (it is independent of the sizes and the number of the analysis patterns). Second, the size of the issued trapdoors is independent of the size of the data to be searched. Third, the search complexity depends only on the size of the data and is constant on the sizes of the patterns to be matched. Attaining all previously mentioned properties while protecting both the data to be analyzed and the patterns to be matched and being able to handle arbitrary analysis pattern query is not obvious and may justify the use of such an interactive assumption.

Definition 2 (independence [19]). *Let* p *be some large prime,* r*,* s*,* t*,* c*, and* k *be five positive integers and* $R \in \mathbb{F}_p[X_1, \cdots, X_c]^r$*,* $S \in \mathbb{F}_p[X_1, \cdots, X_c]^s$*, and* $T \in \mathbb{F}_p[X_1, \cdots, X_c]^t$ *be three tuples of multivariate polynomials over* $\mathbb{F}_p$*. Let* $R^{(i)}$ *,* $S^{(i)}$ *and* $T^{(i)}$ *denote respectively the* i*-th polynomial contained in* R*,* S*, and* T*. For any polynomial* $f \in \mathbb{F}_p[X_1, \cdots, X_c]$*, we say that* f *is dependent on* $< R, S, T >$ *if there exist constants* $\{\vartheta_j^{(a)}\}_{j=1}^{s}$*,* $\{\vartheta_{i,j}^{(b)}\}_{i=1,j=1}^{i=r,j=s}$*,* $\{\vartheta_k^{(c)}\}_{k=1}^{t}$ *such that*

$$f \cdot (\sum_j \vartheta_j^{(a)} \cdot S^{(j)}) = \sum_{i,j} \vartheta_{i,j}^{(b)} \cdot R^{(i)} \cdot S^{(j)} + \sum_k \vartheta_k^{(c)} T^{(k)}$$

We say that f *is independent of* $< R, S, T >$ *if* f *is not dependent on* $< R, S, T >$*.*

Definition 3 (i-GDH assumption). *Let p be some large prime, r, s, t, c, and k be five positive integers and $R \in \mathbb{F}_p[X_1, \cdots, X_c]^r$, $S \in \mathbb{F}_p[X_1, \cdots, X_c]^s$, and $T \in \mathbb{F}_p[X_1, \cdots, X_c]^t$ be three tuples of multivariate polynomials over $\mathbb{F}_p$. Let $\mathcal{O}^r$, (resp. $\mathcal{O}^s$ and $\mathcal{O}^t$) be oracle that, on input $\{\{a^{(k)}_{i_1,\cdots,i_c}\}_{i_j=0}^{d_k}\}_k$, adds the polynomials $\{\sum_{i_1,\cdot,i_c} a^{(k)}_{i_1,\cdot,i_c} \prod_j X_j^{i_j}\}_k$ to R (resp. S and T).*

Let $(x_1, \cdots, x_c)$ be secret vector and q_r (resp. q_s) (resp. q_t) be the number of queries to $\mathcal{O}^r$ (resp. $\mathcal{O}^s$) (resp. $\mathcal{O}^t$). The i-GDH assumption states that, given $\{g^{R^{(i)}(x_1,\cdots,x_c)}\}_{i=1}^{r+k\cdot q_r}$, $\{\widetilde{g}^{S^{(i)}(x_1,\cdots,x_c)}\}_{i=1}^{s+k\cdot q_s}$, and $\{e(g,\widetilde{g})^{T^{(i)}(x_1,\cdots,x_c)}\}_{i=1}^{t+k\cdot q_t}$, it is hard to decide whether (i) $U = g^{f(x_1,\cdots,x_c)}$ or U is random and (ii) $U' = \widetilde{g}^{f(x_1,\cdots,x_c)}$ or U' is random if f is independent of $< R, S, T >$.

As argued in [1], the hardness of the i-GDH problem depends on the same argument as the GDH problem which has already been proven in the generic group model [19]. That is, as long as the challenge polynomial that we denote f is independent of $< R, S, T >$, an adversary cannot distinguish $g^{f(x_1,\cdots,x_c)}$ (resp. $\widetilde{g}^{f(x_1,\cdots,x_c)}$) from a random element of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$). The definition method of the content of the sets R,S, and T (by assumption or by the queries to oracles) does not fundamentally change the proof.

4 The Intuition

The intuition behind the proposed constructions relies on two observations. First, the number of analysis patterns is often very small compared to the quantity of data that are going to be analyzed, e.g., in a deep packet inspection scenario, the number of patterns provided by the SNORT intrusion detection system is 3734 [20]. Second, the sizes of the detection patterns are also very small compared to the size of the traces to be analyzed (e.g., the largest pattern size used by SNORT is 364 Bytes).

For a data with alphabet Σ, the proposed constructions associate each element σ of Σ with a secret encoding $(\alpha'_\sigma, \alpha_\sigma)$. They fragment the sequence of symbols that represents the data $\mathcal{B}$ as described in the Figure 1 in which Φ represents the number of symbols (i.e., the size) of each fragment and p_{max} represents the largest number of symbols in a pattern. To allow the matching of patterns at any possible offset of the data to be searched, in the proposed constructions, we require that $\Phi \geq 2 \cdot (p_{max} - 1)$. In the rest of the paper, we will use $\{x_i\}_{i=a}^{i=b}$ to denote the set of elements x_i, $i \in [a, b]$ and $|\mathcal{B}|$ to denote the number of symbol (i.e., the size) that compose $|\mathcal{B}|$.

As illustrated by the Figure 1, the sequence of symbols $\mathcal{B}$ is fragmented into $2 \times \eta - 1$ fragments $\{F_i, \overline{F}_j\}_{i=0,j=0}^{i=\eta-1,j=\eta-2}$ where $\eta = |\mathcal{B}|/\Phi$ (for simplicity we will suppose that $|\mathcal{B}|$ is a multiple of Φ). Each $F_i, i \in [0, \eta - 1]$, contains the symbols at indexes $[i \cdot \Phi, (i+1) \cdot \Phi - 1]$, while $\overline{F}_i, i \in [0, \eta - 2]$, contains the symbols at indexes $[(i+1) \cdot \Phi - p_{max} - 1, (i+1) \cdot \Phi + p_{max} - 1]$ of $\mathcal{B}$.

Given an $i \in [0, |\mathcal{B}| - 1]$, in the rest of this paper, we will denote by i_F the index of i inside the fragment F where $F \in \{F_0, \cdots, F_{\eta-1}, \overline{F}_0, \cdots, \overline{F}_{\eta-2}\}$. If $i \notin F$, i_F is not defined. Formally, assuming that $F = [a, b]$:

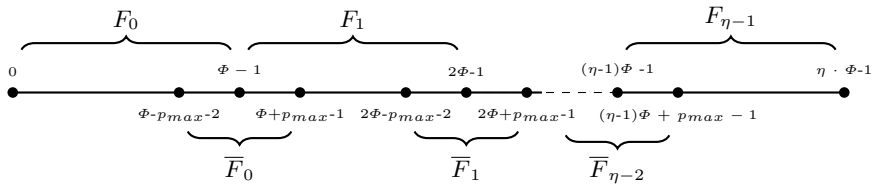


Fig. 1: Fragmentation approach

$$i_F = \begin{cases} i \bmod a & \text{if } i \in F \\ \text{not defined} & \text{otherwise} \end{cases}$$

A trapdoor for a pattern $w = \sigma_{w,0} \cdots \sigma_{w,l-1}$ will be associated with a set of polynomials $\{V_i = v_i \sum_{k=0}^{l-1} \alpha'_{\sigma_{w,k}} \cdot \alpha_{\sigma_{w,k}}^{k+i} \cdot z^k\}_{i=0}^{i=\Phi-l}$ where v_i is a random secret scalar used to prevent new trapdoor forgeries and z a random scalar belonging to the secret key $\mathcal{K}_s$. The trapdoor generated for w consists then in the elements $\{\widetilde{g}^{V_i}, \widetilde{g}^{v_i}\}_{i=0}^{i=\Phi-l}$. Each of the previous elements will be used to check the presence of w at a specific index of the previously constructed fragments.

Meanwhile, the encryption of each symbol σ_i is the tuple $\mathcal{C}_i = \{C_i, C'_i, \overline{C}_i, \overline{C'}_i\}$ that depends on the fragment in which the index i of σ_i in $\mathcal{B}$ belongs. If it belongs to F_ϵ (resp. $\overline{F}_\epsilon$) then C_i and C'_i (resp. $\overline{C}_i$ and $\overline{C'}_i$) contain the encryption of σ_i regarding the index i_{F_ϵ} of i in F_ϵ (resp. the index $i_{\overline{F}_\epsilon}$ of i in $\overline{F}_\epsilon$).

Then, if we want to test the presence of w at the index i, if i belongs to F_ϵ (resp. $\overline{F}_\epsilon$), then we compare the bilinear mapping results of the elements $C_{i_{F_\epsilon}}$, $\widetilde{g}^{v_{i_{F_\epsilon}}}$ (resp. $\overline{C}_{i_{\overline{F}_\epsilon}}$, $\widetilde{g}^{v_{i_{\overline{F}_\epsilon}}}$) and $C'_{i_{F_\epsilon}}$, $\widetilde{g}^{V_{i_{F_\epsilon}}}$ (resp. $\overline{C'}_{i_{\overline{F}_\epsilon}}$, $\widetilde{g}^{V_{i_{\overline{F}_\epsilon}}}$). If w is not present, then the bilinear mapping results will be random-looking elements of $\mathbb{G}_T$ which will be useless to the adversary for learning any information about the plaintext and/or the content of the tested pattern.

5 S^{4}E Construction

In this section, we propose S^{4}E, a construction that supports pattern matching of adaptively chosen and variable (upper bounded) lengths patterns on secret key encrypted streams. Before formalizing S^{4}E, we present a use-case scenario on which S^{4}E can be useful.

5.1 Usage Scenario

To cope with new and sophisticated cybercrime threats, new threat intelligence platforms such as [18] are relying on the collaboration between different involved entities that include, on one side, companies, organizations, and individuals that are targeted by cyber attacks, and on the other, security editors that are in charge of defining and providing strategies for effectively detect and prevent

cyber attacks. To be useful, such platforms should, on one hand, be fueled by data owners, i.e., companies, organizations, and individuals that agree to share the traces (e.g., network and operating system traces) of the cyber attacks that they have suffered. On the other hand, the platform should allow the security editors to analyze (e.g., search specific patterns) and correlate the traces that are shared by the data owners. The considered threat intelligence platform is often managed by non-fully trusted third-party service provider (SP) which provides the required storage space and computation power with affordable cost.

Unfortunately, both data owners (i.e., attack traces owners) and security editors are still very reluctant for adopting such kind of threat intelligence platforms because of two main reasons. First, the traces to be shared contain often highly sensitive information that may raise serious security and/or business threats when disclosed to non-fully trusted third parties (e.g, SP). Second, the shared traces analysis rely mainly on techniques that use pattern matching for inspecting and detecting malicious behaviors. Those analysis patterns are the result of extensive threat intelligence conducted by security editors. They are often put forward as a key competitive differentiator arguing that they can cover a wider set of malicious behaviors. Thus, security editors are typically reluctant to share their analysis patterns with non-fully trusted third-parties.

The S^4E construction can be used to overcome the previous two limitations by building a platform that is (1) market compliant meaning that both the data owner and the third-party entity performing the pattern matching operations will learn nothing about the patterns to be used by security editors for analyzing the shared traces (as proved by Theorem 4), and (2) privacy-friendly, signifying that (2.1) the third-party entity performing pattern matching will learn nothing about the shared data except the presence or the absence of a set of unknown analysis patterns, and (2.2) the pattern provider will learn no more than the indexes on which the searched pattern exists (as proved by Theorem 2).

5.2 Architecture

The architecture considered for the S^4E construction involves three parties: the data owner (DO) representing the entity that holds the data to be analyzed (e.g., the network traces in the case of DPI), the pattern provider (PP) representing the entity that supplies the patterns that will be matched, and the service provider (SP) are stakeholders that offer computation infrastructures that will be used to perform the pattern matching operations on the data to be analyzed. To test the presence of a pattern on DO's data, PP starts by generating collaboratively with DO a trapdoor for the pattern to be matched. Then, PP sends the generated trapdoor to SP who performs the matching operation and notifies the PP with the results (i.e., the presence of the patterns as well as their corresponding positions in the DO's data).

5.3 Security Requirements and Hypothesis

PP, DO, and SP are considered in S^{4}E as *Honest-but-curious* entities. First, we expect PP to provide valid patterns allowing an effective analysis of DO's shared data. This a fairly reasonable assumption since a pattern provider (e.g., a security editor in the case of DPI or a laboratory in the case of genomic data analysis) will not defile its reputation by issuing incorrect or misleading analysis patterns. Otherwise, this will result in many false positives, which may considerably degrade the quality of the analyses that will be provided to the DO. Nevertheless, we expect the PP to be curious: it may try to derive information about the analyzed data by accessing the data analyzed by the SP and/or the pattern matching results returned by the SP.

Second, we suppose that SP will perform the pattern matching operations honestly over the DO's data using the analysis patterns provided by PP. However, we suppose that SP may try to learn additional information about either or both the DO's outsourced data and the analysis patterns provided by PP. In addition, we assume that the SP that may try to create values by analyzing other third-parties data using the set of patterns provided by PP for the analysis of DO's outsourced data.

Third, we suppose DO to follow honestly the S^{4}E protocol. However, we expect that he/she may try to learn additional information about the patterns provided by PP for analyze his/her data.

In addition, we suppose that (i) SP and PP will not collude to learn more information about the traffic, and (ii) SP and DO will not collude to learn more information about the patterns to be searched. We believe that these two last assumptions are fairly reasonable since, in a free market environment, an open dishonest behavior will result in considerable damages for involved entities.

Finally, we require S^{4}E to provide correct results. That is, (1) any part of DO's data that matches one of PP's patterns when not encrypted must be matched by S^{4}E (no false negatives), and (2) we require that for any traffic that does not match any of the PP's analysis patterns when not encrypted, the probability that S^{4}E returns a false positive is negligible.

5.4 Definition of S^{4}E

S^{4}E is defined using five algorithms that we denote **Setup**, **Keygen**, **Encrypt**, **Issue**, and **Test**. The first three algorithms are performed by DO, the **Issue** algorithm is performed collaboratively by DO and PP, and the **Test** algorithm is performed by SP.

- **Setup**$(1^\lambda, \Phi, p_{max})$ is a probabilistic algorithm that takes an input a security parameter λ, the fragmentation size to be used Φ, and the maximum size of a pattern p_{max}. It returns the public parameters $params$.
- **Keygen**$(params, \Sigma)$ is a probabilistic key generation algorithm that takes as input the public parameters $params$ and a finite set Σ representing the alphabet to be used for representing the data to be searched and the pattern

to be matched. It outputs a secret key $\mathcal{K}_s$ and a trapdoor generation key $\mathcal{K}_t$. The latter will be sent to PP using a secure channel.
- **Encrypt**$(params, \mathcal{K}_s, \mathcal{B})$ is a probabilistic algorithm that takes as input the public parameters $params$, the secret key $\mathcal{K}_s$, and a finite sequence (string) of elements $\mathcal{B}$ of Σ of size n. It returns a ciphertext $\mathcal{C}$.
- **Issue**$(params, \mathcal{K}_s, \mathcal{K}_t, w)$ is a probabilistic algorithm executed interactively between PP and DO. It takes as input the public parameters $params$, the secret key $\mathcal{K}_s$, the trapdoor generation key $\mathcal{K}_t$, and w – a sequence of elements of Σ of length smaller or equal to p_{max}, and returns a trapdoor td_w.
- **Test**$(params, \mathcal{C}, td_w)$ is a deterministic algorithm that takes as input the public parameters $params$, a ciphertext $\mathcal{C}$ encrypting a sequence of m elements $\mathcal{B} = \sigma_0 \cdots \sigma_{m-1}$ of Σ, and the trapdoor td_w for the sequence of Σ's elements of length l, $w = \sigma_{w,0} \cdots \sigma_{w,l-1}$. This algorithm is executed interactively between PP and SP. The former provides the trapdoor td_w and the latter executes the algorithm and returns the set of indexes $\mathcal{I} \subset \{0, m-l-1\}$ where for each $i \in \mathcal{I}$, $\sigma_i \cdots \sigma_{i+l-1} = \sigma_{w,0} \cdots \sigma_{w,l-1}$ to PP.

We note that the sizes of the elements defined in the previous algorithms, i.e., the size of the data to be analyzed $\mathcal{B}$, the size of the pattern to be searched w, and the largest analysis pattern size p_{max} refer to the number of symbols of Σ that compose each element. In addition, we note that S^{4}E does not consider a decryption algorithm since there is no need for decrypting the outsourced data. However, we stress that a decryption feature can be straightforwardly performed by issuing a trapdoor for all characters $\sigma \in \Sigma$ and running the Test algorithm on the encrypted data for each of them.

5.5 S^{4}E's Security Requirements

As said in Section 5.3, there are mainly 4 security requirements that should be satisfied by our construction: Trace indistinguishability for both PP and SP, pattern indistinguishability for both DO and SP, trapdoor usefulness (i.e., the trapdoors are useful only to search DO's data), and the correctness property.

In the following, we use the game-based security definition proposed in [1] for trace indistinguishability by adapting the standard notion of IND-CPA which requires that no adversary $\mathcal{A}$ (e.g., PP or SP), even with an access to an oracle $\mathcal{O}^s$ that issues a trapdoor td_{p_i} for any adaptively chosen pattern p_i, can decide whether an encrypted trace contains T_0 or T_1 as long as the trapdoors $\{td_{p_i}\}$ issued by $\mathcal{O}^s$ do not allow trivial distinction of the traces T_0 and T_1. We note that we consider the quite standard *selective* security notion [16]. This notion requires the adversary to choose and commit T_0 and T_1 at the beginning of the experiment, before seeing $params$.

Definition 4 (Data indistinguishability). *Let λ be the security parameter, Σ be the alphabet to be used, $\mathcal{A}$ be the adversary and $\mathcal{C}$ be the challenger. We consider the following game that we denote $Exp^{S^4E\_D\_IND\_CPA}_{\mathcal{A},\beta}$:*

(1) Setup: $\mathcal{C}$ *executes* $Setup(1^\lambda, \Phi, p_{max})$ *to generate params and the algorithm* $Keygen(params, \Sigma)$ *to generate the keys* $\mathcal{K}_s$ *and* $\mathcal{K}_t$*. Then it sends params to the adversary.*

(2) Query: $\mathcal{A}$ *can adaptively ask* $\mathcal{O}^s$ *for the trapdoor* td_{w_i} *for any pattern* $w_i = \sigma_{i,0} \cdots \sigma_{i,l_i-1}$ *where* $\sigma_{i,j} \in \Sigma$*. We denote* $\mathcal{W}$ *the set of patterns submitted by* $\mathcal{A}$ *to* $\mathcal{O}^s$ *in this phase.*

(3) Challenge: Once $\mathcal{A}$ *decides that Phase (2) is over, it chooses two data streams* $T_0 = \sigma^*_{0,0} \cdots \sigma^*_{0,m-1}$ *and* $T_1 = \sigma^*_{1,0} \cdots \sigma^*_{1,m-1}$ *and sends them to* $\mathcal{C}$*.*

(a) If $\exists w = \sigma_0 \cdots \sigma_{l_i} \in \mathcal{W}$*,* $k \in \{0,1\}$*, and* j *such that:*

$$\sigma^*_{k,j} \cdots \sigma^*_{k,j+l_i} = \sigma_0 \cdots \sigma_{l_i} \neq \sigma^*_{1-k,j} \cdots \sigma^*_{1-k,j+l_i} \quad \textit{then return 0.}$$

(b) $\mathcal{C}$ *chooses a random* $\beta \in \{0,1\}$*, creates* $\mathcal{C} = Encrypt(param, \mathcal{K}_s, T_\beta)$*, and sends it to* $\mathcal{A}$*.*

(4) Guess. $\mathcal{A}$ *outputs the guess* β'*.*

(5) Return $(\beta = \beta')$*.*

We define $\mathcal{A}$*'s advantage by* $Adv^{Exp^{S^4E\_D\_IND\_CPA}_{\mathcal{A},\beta}}(\lambda) = |Pr[\beta = \beta'] - 1/2|$*.* S^4E *is said to be data indistinguishable if* $Adv^{Exp^{S^4E\_D\_IND\_CPA}_{\mathcal{A},\beta}}(\lambda)$ *is negligible.*

We note that in the previous definition, the restriction used in phase (3)(a) ensures that if one of the data streams T_k contains a pattern $w_i \in \mathcal{W}$ in the position j, then this is also the case for T_{1-k}. If such a restriction is not used, $\mathcal{A}$ will trivially win the game by running $Test(params, \mathcal{C}, td_{w_i})$.

We want to be able to evaluate the advantage of the SP for using the issued trapdoors to analyze other third-parties' data (i.e., data that are not provided and encrypted by DO). Since encrypted data and trapdoors should be created using the same secret key $\mathcal{K}_s$ (the trapdoor generation key $\mathcal{K}_t$ is created using $\mathcal{K}_s$), such an advantage is equivalent to the ability of the SP to forge valid DO's encrypted data.

Definition 5 (Encrypted Data Forgery). *Let* λ *be a security parameter,* Σ *be the alphabet to be used,* $\mathcal{A}$ *be the adversary,* $\mathcal{C}$ *be the challenger,* $\mathcal{O}^s$ *be an oracle that issues a trapdoor for any adaptively chosen pattern, and* $\mathcal{O}^r$ *be an oracle that encrypts any adaptively chosen data. We consider the following* $Exp^{S^4E\_EDF}_{\mathcal{A}}$ *game:*

(1) Setup: $\mathcal{C}$ *executes* $Setup(1^\lambda, \Phi, p_{max})$ *to generate params and the algorithm* $Keygen(params, \Sigma)$ *to generate the keys* $\mathcal{K}_s$ *and* $\mathcal{K}_t$*. Then it sends params to the adversary.*

(2) Query:

- $\mathcal{A}$ *can ask* $\mathcal{O}^s$ *for issuing the trapdoor* td_{w_i} *for any adaptively chosen pattern* $w_i = \sigma_{i,1} \cdots \sigma_{i,l_i}$ *where* $\sigma_{i,j} \in \Sigma$*. We denote* $\mathcal{W}$ *the set of patterns submitted by* $\mathcal{A}$ *to* $\mathcal{O}^s$ *in this phase.*
- $\mathcal{A}$ *can adaptively ask* $\mathcal{O}^r$ *to create* $\mathcal{C}^T = Encrypt(params, \mathcal{K}_s, T)$*. We denote* $\mathcal{T}$ *the set of datasets encrypted by the* $\mathcal{O}^r$*.*

(3) Forgery: The adversary chooses the dataset $T^* \notin \mathcal{T}$ *such that* T^* *contains* w $(w \in \mathcal{W})$ *at index* i *and forges the encrypted dataset* C^{T^*} *of* T^* *.*

We define $\mathcal{A}$*'s advantage of winning the game* $Exp_{\mathcal{A}}^{S^4E\text{-}EDF}$ *by* $Adv^{Exp_{\mathcal{A}}^{S^4E\text{-}EDF}}(\lambda) = Pr[i \in Test(params, C^{T^*}, td_w)]$*.* S^4E *is said to be encrypted data forgery secure if* $Adv^{Exp_{\mathcal{A}}^{S^4E\text{-}EDF}}(\lambda)$ *is negligible.*

The following definition formalizes the patterns indistinguishability property for SP. That is, we evaluate the advantage of the SP to decide whether a trapdoor encrypts the pattern w_0^* or w_1^* even with an access to an oracle $\mathcal{O}^s$ that issues a trapdoor for any adaptively chosen pattern.

Definition 6 (Pattern Indistinguishability to SP). *Let* λ *be the security parameter,* Σ *be the alphabet to be used,* $\mathcal{A}$ *be the adversary and* $\mathcal{C}$ *the challenger. We consider the following game that we denote* $Exp_{\mathcal{A}_{SP},\beta}^{S^4E\text{-}P\text{-}IND\text{-}CPA}$*:*

(1) Setup: $\mathcal{C}$ *executes* $Setup(1^\lambda, \Phi, p_{max})$ *to generate params and the algorithm* $Keygen(params, \Sigma)$ *to generate the keys* $\mathcal{K}_s$ *and* $\mathcal{K}_t$*. Then it sends params to the adversary.*
(2) Observation: $\mathcal{A}$ *may observe the ciphertext* $\mathcal{C}^{T_i}$ *of a set of (unknown) traces* $T_i \in \mathcal{T}$*.*
(3) Query: $\mathcal{A}$ *can adaptively ask* $\mathcal{O}^s$ *for the trapdoor* td_{w_i} *for any pattern* $w_i = \sigma_{i,1} \cdots \sigma_{i,l_i}$ *where* $\sigma_{i,j} \in \Sigma$*. We denote by* $\mathcal{W}$ *the set of patterns submitted by* $\mathcal{A}$ *to* $\mathcal{O}^s$ *in this phase.*
(4) Challenge: Once $\mathcal{A}$ *decides that Phase (2) is over, it chooses two patterns* $w_0^* = \sigma_{0,0}^* \cdots \sigma_{0,l}^*$ *and* $w_1^* = \sigma_{1,0}^* \cdots \sigma_{1,l}^*$ *such that* $w_0^*, w_1^* \notin \mathcal{W}$ *and sends them to* $\mathcal{C}$*. If* $\exists T \in \mathcal{T}$ *such that* $w_0^* \in T$ *or* $w_1^* \in T$ *then return 0. Otherwise,* $\mathcal{C}$ *chooses a random* $\beta \in \{0, 1\}$*, creates* $td_{w_\beta^*}$*, and sends it to* $\mathcal{A}$*.*
(5) Guess:
  - $\mathcal{A}$ *may try to forge the ciphertext of chosen date and uses the Test algorithm to try to find out the chosen value of* β*.*
  - $\mathcal{A}$ *outputs the guess* β'*.*
(6) Return $(\beta = \beta')$*.*

We define the advantage of the adversary $\mathcal{A}$ *for winning* $Exp_{\mathcal{A}_{SP},\beta}^{S^4E\text{-}P\text{-}IND\text{-}CPA}$ *by* $Adv^{Exp_{\mathcal{A}_{SP},\beta}^{S^4E\text{-}P\text{-}IND\text{-}CPA}}(\lambda) = |Pr[\beta' = \beta] - 1/2|$*.* S^4E *is said to be pattern indistinguishable to SP if* $Adv^{Exp_{\mathcal{A}_{SP},\beta}^{S^4E\text{-}P\text{-}IND\text{-}CPA}}(\lambda)$ *is negligible.*

In addition, we aim to evaluate the advantage of DO for deciding whether a trapdoor encrypts the patterns w_0^* or w_1^* even with an access to an oracle $\mathcal{O}^s$ that plays the role of PP and perform the issue algorithm for any adaptively chosen pattern. The following definition formalizes the pattern indistinguishably property for DO.

Definition 7 (Pattern Indistinguishability to DO). *Let* λ *be the security parameter,* Σ *be the alphabet to be used,* $\mathcal{A}$ *be the adversary and* $\mathcal{C}$ *the challenger. We consider the following game that we denote* $Exp_{\mathcal{A}_{DO},\beta}^{S^4E\text{-}P\text{-}IND\text{-}CPA}$*:*

(1) Setup: $\mathcal{C}$ *executes* $Setup(1^\lambda, \Phi, p_{max})$ *to generate params and the algorithm* $Keygen(params, \Sigma)$ *to generate the keys* $\mathcal{K}_s$ *and* $\mathcal{K}_t$*. Then it sends params to the adversary.*
(2) Query: $\mathcal{A}$ *can ask* $\mathcal{O}^s$ *to play the role of PP in the issue algorithm for any adaptively chosen pattern* $w_i = \sigma_{i,1} \cdots \sigma_{i,l_i}$ *where* $\sigma_{i,j} \in \Sigma$*. We denote by* $\mathcal{W}$ *the set of patterns chosen by* $\mathcal{A}$ *in this phase.*
(3) Challenge: Once $\mathcal{A}$ *decides that Phase (2) is over, it chooses two patterns* $w_0^* = \sigma_{0,0}^* \cdots \sigma_{0,l}^*$ *and* $w_1^* = \sigma_{1,0}^* \cdots \sigma_{1,l}^*$ *such that* $w_0^*, w_1^* \notin \mathcal{W}$ *and sends them to* $\mathcal{C}$*. The latter chooses a random* $\beta \in \{0,1\}$*, and plays the role of PP in the issue algorithm to generate a trapdoor for* w_β^**.*
(4) Guess: $\mathcal{A}$ *outputs the guess* β'*.*
(5) Return $(\beta = \beta')$*.*

We define the advantage of the adversary $\mathcal{A}$ *for winning* $Exp^{S^4E\_P\_IND\_CPA}_{\mathcal{A}_{DO},\beta}$ *by* $Adv^{Exp^{S^4E\_P\_IND\_CPA}_{\mathcal{A}_{DO},\beta}}(\lambda) = |Pr[\beta' = \beta] - 1/2|$*.* S^4E *is said to be pattern indistinguishable to DO if* $Adv^{Exp^{S^4E\_P\_IND\_CPA}_{\mathcal{A}_{DO},\beta}}(\lambda)$ *is negligible.*

We say that S^{4}E provides pattern indistinguishability if it is pattern indistinguishable to both DO and SP.

Definition 8 (S^{4}E Correctness). *Let* $\mathcal{B} = \sigma_0, \cdots \sigma_{m-1}$ *and* $w = \sigma_{w,0}, \cdots \sigma_{w,l-1}$ *be respectively the data to be analyzed and the pattern to be matched.* S^4E *is correct iff the following conditions hold:*

(i) $Pr[i \in Test(params, Encrypt(params, \mathcal{B}, \mathcal{K}_s), Issue(params, \mathcal{K}_s, \mathcal{K}_t, w))] = 1$ *if* $\mathcal{B}$ *contains* p *at index* i*.*
(ii) $Pr[i \in Test(params, Encrypt(params, \mathcal{B}, \mathcal{K}_s), Issue(params, \mathcal{K}_s, \mathcal{K}_t, w))]$ *is negligible if* $\mathcal{B}$ *does not contain* w *at index* i*.*

Condition (i) of the previous definition ensures that the Test algorithm used by S^{4}E produces no false negatives. Condition (ii) ensures that false positives (i.e., the case in which Test algorithm returns i notwithstanding the fact that $\sigma_i \cdots \sigma_{i+l-1} \neq \sigma_{w,0} \cdots \sigma_{w,l-1}$) only occur with negligible probability.

5.6 A trivial Protocol

A trivial attempt for defining a construction that ensures all of the security requirements we defined in Section 5.3 would consist of modifying the most efficient state of the art solution SEST [1] towards a secret key based-construction as described in the following algorithms. The Setup, Keygen, and Encrypt algorithms are to be performed by the DO. The Issue algorithm will be performed collaboratively by the DO and the PP, while the Test algorithm will be performed by the SP.

– **Setup**$(1^\lambda, n)$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot))$ be a bilinear environment. This algorithm selects $g \xleftarrow{\$} \mathbb{G}_1, \widetilde{g} \xleftarrow{\$} \mathbb{G}_2$ and returns $params \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot), g, \widetilde{g}, n)$.

– **Keygen(**$params$**,**Σ**)**: On input of the alphabet Σ, this algorithm selects $z \xleftarrow{\$} \mathbb{Z}_p$ and $\{\alpha_\sigma \xleftarrow{\$} \mathbb{Z}_p\}_{\sigma\in\Sigma}$, computes and adds $\{g^{z^i}\}_{i=0}^{i=n-1}$ to $params$ (required for proving the trace indistinguishability property). It returns the secret key $\mathcal{K}_s = \{z, \{\alpha_\sigma\}_{\sigma\in\Sigma}\}$.
– **Encrypt(**$params, \mathcal{B}, \mathcal{K}_s$**)**: To encrypt $\mathcal{B} = \sigma_1 \cdots \sigma_n$, this algorithm chooses $a \xleftarrow{\$} \mathbb{Z}_p$ and returns $\mathcal{C} = \{C_i, C'_i\}_{i=0}^{n-1}$ where $C_i = g^{a\cdot z^i}$ and $C'_i = g^{a\cdot\alpha_{\sigma_i}\cdot z^i}$.
– **Issue(**$params, w, \mathcal{K}_s$**)** issues a trapdoor td_w for a pattern $w = \sigma_{w,0}, \cdots, \sigma_{w,l-1}$ of length $l \leq n$ as described in Algorithm 1. We denote by L the array that will be used to store random scalars that will be used to encode each symbol of the pattern w, and by $\mathcal{I}$ the array of sets representing the indices of symbols in w that are encoded using the same random scalar. Actually, a random scalar can be re-used as long as it has not been used to encode the same symbol. That is, $L[i]$ is the random scalar to use with the (imperatively distinct) symbols at indices $\mathcal{I}_i$ of w.

Input: $\mathcal{K}_s, params, w = \sigma_{w,0}, \cdots \sigma_{w,l-1}$
Output: td_w
$td_w = \emptyset$, $V = 0, c = 0$
$L[i] = 0$ for all $i \in [0, l-1]$
$Ind[\sigma] = 0$ for all $\sigma \in \Sigma$
foreach $i \in [0, l-1]$ **do**
  if $L[Ind[\sigma_{w,i}]] = 0$ **then**
    $L[c] \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{I}_c = \{i\}$, $c = c + 1$
  else
    $\mathcal{I}_{Ind[\sigma_{w,i}]} = \mathcal{I}_{Ind[\sigma_{w,i}]} \cup \{i\}$
  end
  $V = V + z^i \cdot \alpha_{\sigma_{w,i}} \cdot L[Ind[\sigma_{w,i}]]$
  $Ind[\sigma_{w,i}] = Ind[\sigma_{w,i}] + 1$
end
$td_w = \{c, \{\mathcal{I}_j\}_{j=0}^{j=c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{j=c-1}, \widetilde{g}^V\}$

Algorithm 1: Issue

– **Test(**$params, \mathcal{C}, td_w$**)** checks whether the encrypted data $\mathcal{C}$ contains w by parsing td_w as $\{c, \{\mathcal{I}_j\}_{j=0}^{j=c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{j=c-1}, \widetilde{g}^V\}$ and $\mathcal{C}$ as $\{C_i, C'_i\}_{i=0}^{n-1}$, and checking for all $j \in [0, n-l]$ if the following equation holds:

$$\prod_{t=0}^{c-1} e(\prod_{i\in\mathcal{I}_t} C'_{j+i}, \widetilde{g}^{L[t]}) = e(C_j, \widetilde{g}^V)$$

We can prove the correctness, the data indistinguishability, and encrypted data unforgeability properties by following the same strategies as in [22] (Sections A.1, A.2, and A.3). Unfortunately, this construction inherits the three main limitations of the SEST construction. First, the size of the public parameters $params$ is linear to the size of the data to be analyzed (which may be very large). Second, the pattern indistinguishability requirement cannot be satisfied since the

Issue algorithm (Algorithm 1) leaks many information (such as, the number of different symbols and the maximum number of occurrences of a symbol) about the pattern to be matched. Third, searching the presence of a pattern w is linear to the maximum number of occurrences of each symbol in w, which makes this construction impractical for matching small alphabet based patterns (e.g., bit, or hexadecimal patterns).

5.7 The S^{4}E's Protocol

- **Setup**$(1^\lambda, \Phi, p_{max})$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot))$ be a bilinear environment. This algorithm selects $g \xleftarrow{\$} \mathbb{G}_1, \widetilde{g} \xleftarrow{\$} \mathbb{G}_2$, chooses Φ such that $\Phi \geq 2\cdot(p_{max}-1)$, and returns $params \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot), g, \widetilde{g}, p_{max}, \Phi)$.
- **Keygen**$(params, \Sigma)$: On input of the alphabet Σ, this algorithm selects $z \xleftarrow{\$} \mathbb{Z}_p$, $\{\alpha'_\sigma \xleftarrow{\$} \mathbb{Z}_p, \alpha_\sigma \xleftarrow{\$} \mathbb{Z}_p\}_{\sigma\in\Sigma}$, $r \xleftarrow{\$} \mathbb{Z}_p$, and computes and adds $\{g^{z^i}\}_{i=0}^{i=\Phi-1}$ to $params$. It returns the secret key $\mathcal{K}_s = \{r, z, \{\alpha'_\sigma, \alpha_\sigma\}_{\sigma\in\Sigma}\}$ and the trapdoor generation key $\mathcal{K}_t = \{\widetilde{g}^{r\cdot\alpha'_\sigma\cdot\alpha_\sigma^i\cdot z^j}\}_{i=0,j=0,\sigma\in\Sigma}^{i=\Phi-1,j=p_{max}-1}$ which will be sent to PP using a secure channel.
- **Encrypt**$(params, \mathcal{B}, \mathcal{K}_s)$: it starts by fragmenting $\mathcal{B} = \sigma_0, \cdots \sigma_{m-1}$ into $\{F_i, \overline{F}_j\}_{i=0,j=0}^{i=\eta-1,j=\eta-2}$ where $F_i = [i\cdot\Phi, (i+1)\cdot\Phi - 1]$ and $\overline{F}_j = [(j+1)\cdot\Phi - p_{max} - 2, (j+1)\cdot\Phi + p_{max} - 1]$. It chooses $a_k \xleftarrow{\$} \mathbb{Z}_p$ for each $k \in [0, \eta-1]$ and $\overline{a}_k \xleftarrow{\$} \mathbb{Z}_p$ for each $k \in [0, \eta-2]$ and returns $\mathcal{C} = \{C_i, \overline{C}_i, C'_i, \overline{C'}_i\}_{i=0}^{m-1}$ as described in the following algorithm.

Input: $params, \mathcal{B} = \sigma_0, \cdots \sigma_{m-1}$, $\mathcal{K}_s$, $\{F_i, a_i, \overline{F}_j, \overline{a}_j\}_{i=0,j=0}^{i=\eta-1,j=\eta-2}$
Output: $\mathcal{C} = \{C_i, \overline{C}_i, C'_i, \overline{C'}_i\}_{i=0}^{m-1}$
$\mathcal{C} \leftarrow \emptyset$
foreach $i \in [0, m-1]$ **do**
  $\epsilon \leftarrow i/\Phi$ #find the fragment F_ϵ to which i belongs
  $C_i \leftarrow g^{a_\epsilon\cdot\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{F_\epsilon}}}$, $C'_i \leftarrow g^{a_\epsilon\cdot z^{i_{F_\epsilon}}}$
  if $\epsilon > 0$ ***and*** $i \in \overline{F}_{\epsilon-1}$ **then**
    $\overline{C}_i \leftarrow g^{\overline{a}_{\epsilon-1}\cdot\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{\overline{F}_{\epsilon-1}}}}$, $\overline{C'}_i \leftarrow g^{\overline{a}_{\epsilon-1}\cdot z^{i_{\overline{F}_{\epsilon-1}}}}$
  else if $\epsilon < \eta-1$ ***and*** $i \in \overline{F}_\epsilon$ **then**
    $\overline{C}_i \leftarrow g^{\overline{a}_\epsilon\cdot\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{\overline{F}_\epsilon}}}$, $\overline{C'}_i \leftarrow g^{\overline{a}_\epsilon\cdot z^{i_{\overline{F}_\epsilon}}}$
  else
    $\overline{C}_i \leftarrow$ Null, $\overline{C'}_i \leftarrow$ Null
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_i, C'_i, \overline{C}_i, \overline{C'}_i\}$
end

Algorithm 2: Encrypt

- **Issue**$(params, \mathcal{K}_s, \mathcal{K}_t, w)$ issues a trapdoor td_w for the sequence of symbols $w = \sigma_{w,0}, \cdots, \sigma_{w,l-1}$ of length $l < p_{max}$ as described in the following:

- PP generates $\{v_i \xleftarrow{\$} \mathbb{Z}_p\}_{i=0}^{i=\Phi-l-1}$, uses $\mathcal{K}_t$ to compute

$$\left\{\left(\prod_{j=0}^{l-1} \widetilde{g}^{r\cdot\alpha'_{\sigma_{w,j}}\cdot\alpha^{i+j}_{\sigma_{w,j}}\cdot z^j}\right)^{v_i}\right\}_{i=0}^{\Phi-l-1} = \left\{\widetilde{g}^{v_i\cdot r\sum\limits_{j=0}^{l-1}\alpha'_{\sigma_{w,j}}\cdot\alpha^{i+j}_{\sigma_{w,j}}\cdot z^j}\right\}_{i=0}^{\Phi-l-1}$$

  and sends it to DO.
- DO computes

$$\left\{\left(\widetilde{g}^{v_i\cdot r\sum\limits_{j=0}^{l-1}\alpha'_{\sigma_{w,j}}\cdot\alpha^{i+j}_{\sigma_{w,j}}\cdot z^j}\right)^{-r}\right\}_{i=0}^{\Phi-l-1} = \left\{\widetilde{g}^{v_i\sum\limits_{j=0}^{l-1}\alpha'_{\sigma_{w,j}}\cdot\alpha^{i+j}_{\sigma_{w,j}}\cdot z^j}\right\}_{i=0}^{\Phi-l-1}$$

  and sends it to PP.
- PP computes $td_w = \{\widetilde{g}^{V_i}, \widetilde{g}^{v_i}\}_{i=0}^{\Phi-l-1}$ with $V_i = v_i \sum\limits_{j=0}^{l-1} \alpha'_{\sigma_{w,j}} \cdot \alpha^{i+j}_{\sigma_{w,j}} \cdot z^j$

– **Test**$(params, C, td_w)$ tests whether the encrypted data $\mathcal{C}$ contains w using the following algorithm. It returns the set $\mathcal{I}$ of indexes i in which w exists in $\mathcal{C}$.

Input: $\mathcal{C} = \{C_i, \overline{C}_i, C'_i, \overline{C'}_i\}_{i=0}^{m-1}, td_w = \{V_i, v_i\}_{i=0}^{i=\Phi-l-1}$
Output: $\mathcal{I}$
$\mathcal{I} \leftarrow \emptyset$
foreach $i \in [0, m-1]$ **do**
  $\epsilon \leftarrow i/\Phi$ #find the fragment F_ϵ to which i belongs
  if $i \in F_\epsilon \cap \overline{F}_\epsilon$ **then**
    if $e(\prod_{j=0}^{l-1} \overline{C}_{i+j}, \widetilde{g}^{v_{i_{\overline{F}_\epsilon}}}) = e(\overline{C'}_i, \widetilde{g}^{V_{i_{\overline{F}_\epsilon}}})$ **then**
      $\mathcal{I} \leftarrow \mathcal{I} \cup i$
    end
  else
    if $e(\prod_{j=0}^{l-1} C_{i+j}, \widetilde{g}^{v_{i_{F_\epsilon}}}) = e(C'_i, \widetilde{g}^{V_{i_{F_\epsilon}}})$ **then**
      $\mathcal{I} \leftarrow \mathcal{I} \cup i$
    end
  end
end

Algorithm 3: Test

We note here that the size of the ciphertext produced by the Encrypt algorithm does not depend on the set of patterns to be used but depends only on the size of data to be encrypted. In addition, our Issue and Test algorithms allow to search an arbitrary (upper bounded size) and unforgeable (without the knowledge of the secret key $\mathcal{K}_s$) patterns. The sizes of those trapdoors do not depend on the size of the data to be encrypted but only on the size of the data fragment (around the double of the maximum size of a pattern). Finally, we underline that the elements $\{\widetilde{g}^{v_i}\}_{i=0}^{\Phi-l-1}$ of a trapdoor td_w will not be accessible to the DO, since the trapdoor is to be used only between PP and SP in the Test algorithm to match the pattern w on the encrypted data.

5.8 S^{4}E's Security Results

In this section, we prove that the S^{4}E construction described in Section 5.7 provides the security requirements we described in Section 5.3. The proofs of the following theorems are provided in the full version [22].

Theorem 1. *S^4E is correct.*

Theorem 2. *S^4E is trace indistinguishable under the i-GDH assumption.*

Theorem 3. *S^4E is encrypted data forgery secure under the i-GDH assumption.*

Theorem 4. *S^4E is pattern indistinguishable under the i-GDH assumption.*

6 AS^{3}E Construction

The S^{4}E construction, introduced in Section 5, allows for pattern matching on symmetrically encrypted data. In this section we show that the data fragmentation approach we propose in Section 4 can also be used to build AS^{3}E: a pattern matching of upper bounded length keywords on asymmetrically encrypted stream. In particular, we show in Section 7 that considering the same system and threat model as the most efficient state of the art solution SEST [1], AS^{3}E is far more practical than SEST as it reduces (1) considerably the size of public keys and (2) slightly the search complexity while increasing the size of ciphertext only by a factor of 2.

6.1 Architecture

AS^{3}E involves four roles: Pattern Provider (PP), Service Provider (SP), a *sender*, and a *receiver*. PP and SP are the same two entities we used in the S^{4}E construction. That is, PP is the entity that supplies the patterns that will be searched, and the Service Provider SP are stakeholders that offer computation infrastructures that will be used to perform pattern matching operations on the data to be analyzed. The role *sender* is used to represent the entities that are going to generate the data that is going to be analyzed (e.g., a website that provides web contents). The role *receiver* represents the entities that will receive and process the traffic sent by the *sender*. The *receiver* and the *sender* roles are interchangeable. That is, within the same secure network connection session, each end-point may play both the *sender* and the *receiver* roles. In this context, we suppose that the *receiver* want to analyze the data (e.g., to detect malicious contents) to be sent by the *sender* before using it. In AS^{3}E, we require that the *sender* and the *receiver* will not collaborate together, otherwise, they could use a secure channel that is out of reach for the SP. This scenario should not be considered as a limitation of AS^{3}E since, in such scenario pattern matching cannot be provided by SP even in the context of a plaintext traffic.

6.2 Security Requirements and Hypothesis

We consider the same hypothesis for the two entities PP and SP as in our S^4E construction. That is, PP and SP are considered to be *honest-but-curious* entities. Specifically, PP is supposed to provide valid patterns that allow SP to effectively analyze the data generated by the *sender* while SP is supposed to perform correctly the matching between the patterns provided by PP and the *sender*'s data. Nevertheless, we expect PP and SP to be curious as the former may try to learn information about the sender's data and the latter may try to get additional information about both the patterns provided by PP and the sender's data.

Moreover, we expect the *receiver* to be *honest-but-curious*. That is, he/she will correctly follow AS^3E's protocol. However, he/she may try to learn more information about the patterns that are provided by PP.

In addition, we suppose that the *receiver* and SP will not collude to learn more information about the patterns provided by PP. Otherwise, they could easily mount a dictionary attack. Again, we believe that this last assumption is fairly reasonable since an open dishonest behavior will result in considerable damages for both entities.

Finally, as in S^4E, the pattern matching functionality provided by AS^3E should be correct in a way that (1) any traffic that matches a least one of the analysis patterns provided by PP when not encrypted must be detected as malicious traffic by our construction, and (2) the probability that our construction returns a false positive for any traffic that does not match any of the PP's analysis patterns when not encrypted is negligible.

6.3 Definition of AS^3E

Similarly to the S^4E construction, we used five algorithms to define our construction: **Setup**, **Keygen**, **Encrypt**, **Issue**, and **Test**. The algorithms Setup and Keygen are performed by the entity playing *receiver* role. The Issue algorithm is performed collaboratively by the *receiver* and the PP. The Encrypt algorithm is performed by the *sender* while the Test algorithm is performed by SP.

- **Setup**$(1^\lambda, \Phi, p_{max})$ is a probabilistic algorithm that takes as input a security parameter λ, the fragmentation size to be used Φ, and the maximum size of a pattern p_{max}. It returns the public parameters $params$ which will be an implicit input to all other algorithms.
- **Keygen**(Σ) is a probabilistic algorithm that takes as input a finite set of symbols Σ representing the alphabet (e.g., bit symbols, byte symbols) used to represent the data to be analyzed. It returns the keys $\mathcal{K}_s$, $\mathcal{K}_p$, and $\mathcal{K}_t$, where $\mathcal{K}_s$ is private and known only to the *receiver*, $\mathcal{K}_t$ is know only to PP, and $\mathcal{K}_p$ is public.
- **Encrypt**$(\mathcal{B}, \mathcal{K}_p)$ is a probabilistic algorithm that takes as input the data to be encrypted $\mathcal{B}$ along with the public key $\mathcal{K}_p$ and returns a ciphertext $\mathcal{C}$.

– **Issue**$(\mathcal{K}_s, \mathcal{K}_t, w)$ is a probabilistic algorithm performed collaboratively by the *receiver* and the PP. It takes as input the *receiver*'s private key $\mathcal{K}_s$, the trapdoor generation key $\mathcal{K}_t$, and a pattern w of length l ($l \leq p_{max}$) and returns a trapdoor td_w.
– **Test**$(\mathcal{C}, td_w)$ is a deterministic algorithm that takes as input a ciphertext $\mathcal{C}$ encrypting a data stream $\mathcal{B}$ along with a trapdoor td_w for a pattern w and returns the set of indexes at which the pattern w occurs in $\mathcal{B}$.

Similarly to the S^{4}E construction, we omit the decryption algorithm in the previous description since we focus mainly on providing arbitrary universal [3] pattern matching over encrypted traffic. The decryption functionality can be easily added by encrypting the data stream $\mathcal{B}$ under a conventional encryption scheme.

6.4 Security Model

For the AS^{3}E construction, there are mainly three security requirements that should be satisfied: the traffic indistinguishability to SP and PP, the pattern indistinguishability to SP and the *receiver*, and the correct detection requirements. We note that, similarly to our S^{4}E construction, we consider the *selective* security notion [16]. In the following, we denote by $\mathcal{O}^s$ a trapdoor-issuing oracle that can be queried to create a trapdoor for any pattern.

The following definition states that it is not feasible for the SP or PP to learn any information about the content of the traffic more than the presence or the absence of the patterns to be matched.

Definition 9 (Trace indistinguishability). *Let λ be the security parameter, Σ be the alphabet to be used, $\mathcal{A}$ be the adversary and $\mathcal{C}$ be the challenger. We consider the following game that we denote $Exp^{AS^3E\_T\_IND\_CPA}_{\mathcal{A},\beta}$:*

(1) Setup: $\mathcal{C}$ executes $Setup(1^\lambda, \Phi, p_{max})$ to generate params and $Keygen(\Sigma)$ to generate $\mathcal{K}_s$, $\mathcal{K}_t$, and $\mathcal{K}_p$. Then it sends params, $\mathcal{K}_p$, and $\mathcal{K}_t$ to $\mathcal{A}$.
(2) Query: $\mathcal{A}$ can adaptively query $\mathcal{O}^s$ to create a trapdoor td_{w_i} for any adaptively chosen pattern $w_i = \sigma_{i,0} \cdots \sigma_{i,l_i-1}$ where $\sigma_{i,j} \in \Sigma$. We denote $\mathcal{W}$ the set of patterns submitted by $\mathcal{A}$ to $\mathcal{O}^s$ in this phase.
(3) Challenge: Once $\mathcal{A}$ decides that Phase (2) is over, it chooses two data streams $T_0 = \sigma^*_{0,0} \cdots \sigma^*_{0,m-1}$ and $T_1 = \sigma^*_{1,0} \cdots \sigma^*_{1,m-1}$ and sends them to $\mathcal{C}$.
(a) If $\exists w = \sigma_0 \cdots \sigma_l \in \mathcal{W}$, $k \in \{0,1\}$, and j such that:

$$\sigma^*_{k,j} \cdots \sigma^*_{k,j+l} = \sigma_0 \cdots \sigma_l \neq \sigma^*_{1-k,j} \cdots \sigma^*_{1-k,j+l} \quad \textit{then return 0.}$$

(b) $\mathcal{C}$ chooses a random $\beta \in \{0,1\}$, creates $\mathcal{C} = Encrypt(T_\beta, \mathcal{K}_p)$, and sends it to $\mathcal{A}$.

[3] The trapdoor generated collaboratively by the *receiver* and PP can be used to analyze any *sender*'s data that is sent to the *receiver*

(4) Guess. $\mathcal{A}$ outputs the guess β'.
(5) Return ($\beta = \beta'$).

We define $\mathcal{A}$'s advantage by $Adv^{Exp^{AS^3E\_T\_IND\_CPA}_{\mathcal{A},\beta}}(\lambda) = |Pr[\beta = \beta'] - 1/2|$. AS^3E is data indistinguishable if $Adv^{Exp^{AS^3E\_T\_IND\_CPA}_{\mathcal{A},\beta}}(\lambda)$ is negligible.

The pattern indistinguishability property informally requires that it is not feasible for an adversary (the SP or the *receiver*) to learn any information about the detection patterns. Since our construction is a public-key based scheme, we need to take into consideration the fact that an adversary can create any traffic of its choice using the public key $\mathcal{K}_p$. In this case, an adversary can mount a brute force attack on PP's patterns by adaptively creating as much traffic as needed to understand the logic behind them. However, a pattern matching-based solution over plaintext or public-key encryption ciphertext cannot resist such an attack, and therefore, it should not be considered in the security model of AS^{3}E. Hence, for AS^{3}E, the pattern indistinguishability property requires that the adversary $\mathcal{A}$ will not learn more information than what is provided as output to the Test algorithm. Formally, we use the high-min entropy property [17] which informally states that $\mathcal{A}$ cannot obtain the patterns "by chance".

Definition 10 (min-entropy). *Given a set of detection patterns $\mathcal{W}$,and a random bit $\beta \in \{0,1\}$. A probabilistic adversary $\mathcal{A} = (\mathcal{A}_f, \mathcal{A}_g)$ has min-entropy μ if*

$$\forall \lambda \in \mathbb{N}, \forall w \in \mathcal{W}, \forall \beta : Pr[w' \leftarrow \mathcal{A}(\lambda, \beta) : w = w'] \leq 2^{-\mu(\lambda)}$$

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

In the experiment $Exp^{AS^3E\_P\_IND}_{\mathcal{A}_{SP}=(\mathcal{A}_f,\mathcal{A}_g),\beta}$ (Definition 11), we define the security notion $AS^3E\_P\_IND$ for an adversary $\mathcal{A}_{SP} = (\mathcal{A}_f, \mathcal{A}_g)$ ($\mathcal{A}_f$ and $\mathcal{A}_g$ are non colluding entities, as in e.g., [17, 6]) with high-min entropy, that can create any traffic of its choice.

Definition 11 (Pattern indistinguishability to SP). *Let λ be the security parameter, Σ be the alphabet to be used, $\mathcal{A}_{SP} = (\mathcal{A}_f, \mathcal{A}_g)$ be the adversary and $\mathcal{C}$ be the challenger. We consider the following game that we denote $Exp^{AS^3E\_P\_IND}_{\mathcal{A}_{SP}=(\mathcal{A}_f,\mathcal{A}_g),\beta}$:*

(1) Setup: $\mathcal{C}$ executes $Setup(1^\lambda, \Phi, p_{max})$ to generate params and $Keygen(\Sigma)$ to generate $\mathcal{K}_s$, $\mathcal{K}_t$, and $\mathcal{K}_p$. Then it sends params and $\mathcal{K}_p$ to $\mathcal{A}_{SP}$.
(2) Query: $\mathcal{A}_{SP}$ can adaptively query $\mathcal{O}^s$ to create a trapdoor td_{w_i} for any pattern $w_i = \sigma_{i,1} \cdots \sigma_{i,l_i}$ where $\sigma_{i,j} \in \Sigma$. We denote by $\mathcal{W}$ the set of patterns submitted by $\mathcal{A}_{SP}$ to $\mathcal{O}^s$ in this phase.
(3) Challenge: Once $\mathcal{A}_{SP}$ decides that Phase (2) is over, $\mathcal{A}_f$ chooses two patterns $w_0^* = \sigma_{0,0}^* \cdots \sigma_{0,l}^*$ and $w_1^* = \sigma_{1,0}^* \cdots \sigma_{1,l}^*$ such that $w_0^*, w_1^* \notin \mathcal{W}$ and sends them to $\mathcal{C}$. $\mathcal{C}$ chooses a random $\beta \in \{0,1\}$, creates $td_{w_\beta^*}$, and sends it to $\mathcal{A}_g$.
(4) Guess: $\mathcal{A}_g$ outputs the guess β'.
(5) Return ($\beta = \beta'$).

We define $\mathcal{A}$'s advantage by $Adv^{Exp^{AS^3E\_P\_IND}_{\mathcal{A}_{SP}=(\mathcal{A}_f,\mathcal{A}_g),\beta}}(\lambda) = |Pr[\beta = \beta'] - 1/2|$. AS^3E is said to be pattern indistinguishable to SP if for any probabilistic polynomial-time $\mathcal{A}_{SP} = (\mathcal{A}_f, \mathcal{A}_g)$ having high-min entropy, $Adv^{Exp^{AS^3E\_P\_IND}_{\mathcal{A}_{SP}=(\mathcal{A}_f,\mathcal{A}_g),\beta}}(\lambda)$ is negligible.

In addition, since the Issue algorithm is performed interactively between the *receiver* and PP, we aim to evaluate the advantage of the *receiver* to decide whether a trapdoor encrypts w_0^* or w_1^* even with an access to an oracle O^s that plays the role of a PP and performs the Issue algorithm for any adaptively chosen pattern. The following definition formalizes the pattern indistinguishability property for the *receiver*.

Definition 12 (Pattern Indistinguishability to the *receiver*). *Let λ be the security parameter, Σ be the alphabet to be used, $\mathcal{A}$ be the adversary and $\mathcal{C}$ the challenger. We consider the following game that we denote $Exp^{AS^3E\_P\_IND\_CPA}_{\mathcal{A}_R,\beta}$:*

(1) Setup: $\mathcal{C}$ executes $Setup(1^\lambda, \Phi, p_{max})$ to generate params and $Keygen(\Sigma)$ to generate $\mathcal{K}_s$, $\mathcal{K}_p$, and $\mathcal{K}_t$. Then it sends params, $\mathcal{K}_s$, $\mathcal{K}_p$, and $\mathcal{K}_t$ to the adversary.
(2) Query: $\mathcal{A}$ can use $\mathcal{O}^s$ as a PP in the Issue algorithm to create a trapdoor for any adaptively chosen pattern $w_i = \sigma_{i,1} \cdots \sigma_{i,l_i}$ where $\sigma_{i,j} \in \Sigma$. We denote by $\mathcal{W}$ the set of patterns chosen by $\mathcal{A}$ in this phase.
(3) Challenge: Once $\mathcal{A}$ decides that Phase (2) is over, it chooses two patterns $w_0^* = \sigma_{0,0}^* \cdots \sigma_{0,l}^*$ and $w_1^* = \sigma_{1,0}^* \cdots \sigma_{1,l}^*$ such that $w_0^*, w_1^* \notin \mathcal{W}$ and sends them to $\mathcal{C}$. $\mathcal{C}$ chooses a random $\beta \in \{0,1\}$, and plays the role of PP in the issue algorithm to generate collaboratively with $\mathcal{A}$ a trapdoor for w_β^*.
(4) Guess: $\mathcal{A}$ outputs the guess β'
(5) Return ($\beta = \beta'$).

We define the advantage of the adversary $\mathcal{A}$ for winning $Exp^{AS^3E\_P\_IND\_CPA}_{\mathcal{A}_R,\beta}$ by $Adv^{Exp^{AS^3E\_P\_IND\_CPA}_{\mathcal{A}_R,\beta}}(\lambda) = |Pr[\beta' = \beta] - 1/2|$. AS^3E is said to be pattern indistinguishable for the receiver if $Adv^{Exp^{AS^3E\_P\_IND\_CPA}_{\mathcal{A}_R\beta}}(\lambda)$ is negligible.

Finally, the pattern matching correctness property is formally defined in the following Definition.

Definition 13 (Correctness). *Given a data stream T and a pattern w. AS^3E is correct iff the following conditions hold:*

(i) $Pr[i \in Test(Encrypt(T, \mathcal{K}_p), Issue(\mathcal{K}_s, \mathcal{K}_t, w))] = 1$ if T contains w at index i.
(ii) $Pr[i \in Test(Encrypt(T, \mathcal{K}_p), Issue(\mathcal{K}_s, \mathcal{K}_t, w))]$ is negligible if T does not contain w at index i.

6.5 The protocol

- **Setup**$(1^\lambda, \Phi, p_{max})$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot))$ be a bilinear environment. This algorithm selects $g \xleftarrow{\$} \mathbb{G}_1, \widetilde{g} \xleftarrow{\$} \mathbb{G}_2$ and returns $params \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e(\cdot,\cdot), g, \widetilde{g}, \Phi, p_{max})$.
- **Keygen**(Σ): On input of the alphabet Σ, this algorithm chooses Φ such that $\Phi \geq 2 \cdot (p_{max} - 1)$, selects $z \xleftarrow{\$} \mathbb{Z}_p$, $\{\alpha'_\sigma \xleftarrow{\$} \mathbb{Z}_p, \alpha_\sigma \xleftarrow{\$} \mathbb{Z}_p\}_{\sigma\in\Sigma}$, and $r \xleftarrow{\$} \mathbb{Z}_p$, computes and sets the public key $\mathcal{K}_p = \{g^{z^i}, g^{\alpha'_\sigma \cdot (\alpha_\sigma \cdot z)^i}\}_{i=0,\sigma\in\Sigma}^{i=\Phi-1}$, the private key $\mathcal{K}_s = \{r, \alpha_\sigma, \alpha'_\sigma, z\}_{\sigma\in\Sigma}$, and the trapdoor generation key $\mathcal{K}_t = \{\widetilde{g}^{r\cdot\alpha'_\sigma\cdot\alpha_\sigma^i\cdot z^j}\}_{i=0,j=0,\sigma\in\Sigma}^{i=\Phi-1,j=p_{max}-1}$. It sends $\mathcal{K}_t$ to PP.
- **Encrypt**$(\mathcal{B}, \mathcal{K}_p)$ fragments $\mathcal{B} = \sigma_1, \cdots \sigma_m$ into $\{F_i, \overline{F}_j\}_{i=0,j=0}^{i=\eta-1,j=\eta-2}$ where $F_i = [i\cdot\Phi+1, (i+1)\cdot\Phi]$ and $\overline{F}_j = [(j+1)\cdot\Phi - p_{max} - 1, (j+1)\cdot\Phi + p_{max}]$. It chooses $a_k \xleftarrow{\$} \mathbb{Z}_p$ for each $k \in [0, \eta-1]$ and $\overline{a}_k \xleftarrow{\$} \mathbb{Z}_p$ for each $k \in [0, \eta-2]$ and returns $\mathcal{C} = \{C_i, \overline{C}_i, C'_i, \overline{C'}_i\}_{i=1}^m$ as described in the following algorithm.

Input: $\mathcal{B} = \sigma_1, \cdots \sigma_m$,$\mathcal{K}_p$, $\{F_i, a_i, \overline{F}_j, \overline{a}_j\}_{i=0,j=0}^{i=\eta-1,j=\eta-2}$
Output: $\mathcal{C} = \{C_i, \overline{C}_i, C'_i, \overline{C'}_i\}_{i=1}^m$
$\mathcal{C} \leftarrow \emptyset$
foreach $i \in [1, m]$ **do**
  $\epsilon \leftarrow i/\Phi$ #find the fragment F_ϵ to which i belongs
  $C_i \leftarrow g^{a_\epsilon \cdot \alpha'_{\sigma_i} \cdot (\alpha_{\sigma_i}\cdot z)^{i_{F_\epsilon}}}$, $C'_i \leftarrow g^{a_\epsilon \cdot z^{i_{F_\epsilon}}}$
  # $g^{\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{F_\epsilon}}}$ and $g^{z^{i_{F_\epsilon}}}$ are retrived from $\mathcal{K}_p$
  if $\epsilon > 0$ ***and*** $i \in \overline{F}_{\epsilon-1}$ **then**
    $\overline{C}_i \leftarrow g^{\overline{a}_{\epsilon-1}\cdot\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{\overline{F}_{\epsilon-1}}}}$, $\overline{C'}_i \leftarrow g^{\overline{a}_{\epsilon-1}\cdot z^{i_{\overline{F}_{\epsilon-1}}}}$
  else if $\epsilon < \eta - 1$ ***and*** $i \in \overline{F}_\epsilon$ **then**
    $\overline{C}_i \leftarrow g^{\overline{a}_\epsilon\cdot\alpha'_{\sigma_i}\cdot(\alpha_{\sigma_i}\cdot z)^{i_{\overline{F}_\epsilon}}}$, $\overline{C'}_i \leftarrow g^{\overline{a}_\epsilon\cdot z^{i_{\overline{F}_\epsilon}}}$
  else
    $\overline{C}_i \leftarrow$ Null, $\overline{C'}_i \leftarrow$ Null
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_i, C'_i, \overline{C}_i, \overline{C'}_i\}$
end

Algorithm 4: Encrypt

- **Issue**$(\mathcal{K}_s, \mathcal{K}_t, w)$ issues a trapdoor td_w for the sequence of symbols $w = \sigma_{w,0}, \cdots, \sigma_{w,l-1}$ of length $l < p_{max}$. AS^{3}E uses the same Issue algorithm as S^{4}E except that *DO* will be replaced by the *receiver*.
- **Test**$(\mathcal{C}, td_w)$ tests whether the encrypted traces $\mathcal{C}$ contains the sequence of symbols w. It returns the set $\mathcal{I}$ of indexes i in which w exists in $\mathcal{C}$. The Test algorithm is the same as described for the S^{4}E construction (Algorithm 3).

6.6 AS^{3}E Security Results

This section presents the security results of AS^{3}E. The proofs of the following theorems are given in the full version of this paper [22].

Theorem 5. *AS^3E is correct.*

Theorem 6. *AS^3E is trace indistinguishable under the i-GDH assumption.*

Theorem 7. *AS^3E is pattern-indistinguishable to SP for patterns of high min-entropy under the i-GDH assumption.*

Theorem 8. *AS^3E is pattern-indistinguishable to the receiver under the i-GDH assumption.*

7 The complexity

We evaluate the practicability of S^4E and AS^3E regarding several properties: the sizes of the public parameters for S^4E, public keys for AS^3E, the trapdoor generation key, the ciphertext, the trapdoor, and the encryption and search complexities. Let Φ be the size of a fragment, p_{max} be the maximum size of a pattern, n be the total number of symbols in the data to be analyzed. Note that S^4E and AS^3E share the same sizes for the ciphertext, the trapdoor generation key, the trapdoors, and the same complexities for trapdoor generation, encryption, and search operations.

The size of the public parameters used in S^4E: The public parameters *params* used in the S^4E construction contain Φ elements of $\mathbb{G}_1$ which represents $32 \times \Phi$ bytes using Barreto-Naehrig (BN) [15].

The size of the public keys used in AS^3E: The public key $\mathcal{K}_p$ used in the S^4E construction contains $2 \times \Phi$ elements of $\mathbb{G}_1$ which represents $64 \times \Phi$ bytes using BN. We underline that the size of the required public key is independent of the size of the data to be analyzed n and depends only on the maximum size of a pattern p_{max} ($n \gg \Phi \geq 2 \times (p_{max} - 1)$). Hence, compared to the most efficient state of the art solution SEST, AS^3E reduces considerably the size of the required public key. To illustrate, if we suppose that 1G of data is to be analyzed using a set of patterns, each composed of at most 10000 bytes, SEST requires a public key of size $32 \times (1 + 256) \times 10^9$ bytes $\simeq$ 8000 GB while AS^3E requires a public key of size 20000×64 bytes $\simeq$ 1.20 MB.

The size of the pattern generation key $\mathcal{K}_t$: For both S^4E and AS^3E, $\mathcal{K}_t$ contains $\Phi \times p_{max} \times |\Sigma|$ elements of $\mathbb{G}_2$. A key allowing to generate trapdoors for a binary pattern of length $l \leq 1000$ will have a size equals to 128 MB.

The size of the ciphertext: In the worst case (i.e., $\Phi = 2 \times (p_{max} - 1)$), each symbol is represented by 4 elements of $\mathbb{G}_1$. Thus, encrypting n symbols requires $128 \times n$ bytes, while SEST produces a ciphertext of size $64 \times n$ bytes using BN.

Trapdoor's size: A trapdoor is composed of $2 \times (\Phi - p_{max})$ elements of $\mathbb{G}_2$ which represents $64 \times (\Phi - p_{max})$ bytes using BN.

Trapdoor generation complexity. Generating a trapdoor for a pattern of length l ($l \leq p_{max}$), as described in the Issue algorithm, requires $(\Phi - l) \times (2l + 2)$

exponentiations and $4l(\Phi - l)$ multiplications in $\mathbb{G}_2$.

The upper bound size of patterns: The upper bound size p_{max} of the patterns that can be searched by S^{4}E and AS^{3}E depends Φ ($p\_max = \Phi/2 - 1$). Increasing p_max will increase linearly the trapdoor's sizes and generation complexity. However, it will not affect any of the other properties of S^{4}E and AS^{3}E.

Encryption complexity According to the Encrypt algorithm (Algorithm 2), in the worst case (i.e., $\Phi = 2 \times (p_{max} - 1)$), encrypting a sequence of n symbols using S^{4}E requires $10 \times n$ exponentiations in $\mathbb{G}_1$. In case in which n is large (i.e., $n \gg \Phi$ and $n \gg |\Sigma|$), the previous complexity can be reduced by pre-computing $\{g^{\alpha'_\sigma \cdot (\alpha_\sigma \cdot z)^i}, g^{z^i}\}_{i=0,\sigma\in\Sigma}^{i=\Phi-1}$. Then for each symbol to encrypt, the encryptor needs only to perform four exponentiations: $(g^{\alpha'_\sigma \cdot (\alpha_\sigma \times z)^{i F_\epsilon}})^{a_\epsilon}$, $(g^{z^{i F_\epsilon}})^{a_\epsilon}$, $(g^{\alpha'_\sigma \cdot (\alpha_\sigma \times z)^{i \overline{F}_\epsilon}})^{\overline{a}_\epsilon}$, and $(g^{z^{i \overline{F}_\epsilon}})^{\overline{a}_\epsilon}$which reduces the overall complexity to $\Phi \times |\Sigma| + 4 \times n$ exponentiations in $\mathbb{G}_1$. As for AS^{3}E, encrypting a sequence of n symbols requires $2 \times n$ exponentiations in $\mathbb{G}_1$.

Search complexity: According to the Test algorithm (Algorithm 3), searching a pattern of size l on a sequence of symbols of size n requires $nl - l^2$ multiplications on the group $\mathbb{G}_1$ and $2 \times (n - l)$ pairings. In fact, the Test algorithm verifies the presence of a pattern (using its associated trapdoor) in each possible offset in the data to be analyzed. Let us denote by s_0 and s_1 the two sequences of symbols of length l to be analyzed to check the presence of a pattern in offsets 0 and 1 respectively of the fragment F_i (resp. $\overline{F}_i$). Checking the presence of the pattern in the offset 0 requires the computation of $\prod_{i=0}^{l-1} C_i$ (resp. $\prod_{i=0}^{l-1} \overline{C}_i$) while checking the presence of the pattern in offset 1 requires the computation of $\prod_{i=0}^{l-1} C_{i+1}$ (resp. $\prod_{i=0}^{l-1} \overline{C}_{i+1}$). Obviously, for the offset 1, we can avoid the recomputation of $\prod_{i=1}^{l-1} C_i$ since it has already been computed for the offset 0. Following the previous observation, searching a pattern of length l on a sequence of symbols of length n requires only n multiplications and n divisions on the group $\mathbb{G}_1$, and $2 \times (n - l)$ pairings. Considering the fact that $l \ll n$, we can upper bound the search complexity by n multiplications, n divisions and $2n$ pairings. Finally, we note that pairing operations can be implemented very efficiently [21] and that our Test procedure is highly parallelizable.

8 Empirical Evaluation

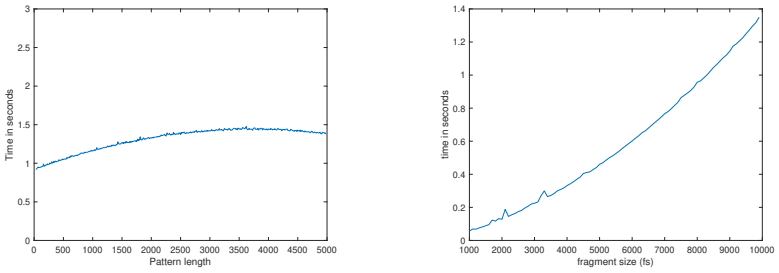
In this section, we experimentally evaluate the performance of S^{4}E and AS^{3}E [4]. We implement the two constructions using the RELIC cryptographic library

[4] We note that the goals of this section is to (1) provide a more concrete estimations of the different operations used by S^{4}E and AS^{3}E and (2) show that S^{4}E and AS^{3}E are more practical than SEST. Particularly, we do not claim that S^{4}E and AS^{3}E are practical enough to perform pattern matching on very large data streams.

[21] over the 254-bits BN curve [5]. For all conducted experiments, we used real network traces as the data to be encrypted and analyzed, and we (pseudo) randomly generated the analysis patterns to be searched. In addition, since in both S^4E and AS^3E, the encryption and the trapdoor generation algorithms are to be performed by entities (data owners in case of S^4E or data sender in case of AS^3E) which may not have a large computation power, we run both the trapdoor generation and the encryption algorithms tests on an Amazon EC2 instance (a1.2xlarge) running Linux with an Intel Xeon E5-2680 v4 Processor with 8 vCPU and 16 GB of RAM. In contrast, as the search operations are performed by SP which is supposed to have a large computation power, we run search experiments on an Amazon EC2 instance (m5.24xlarge) running Linux with an Intel Xeon E5-2680 v4 Processor with 96 vCPU and 64 GB of RAM.

In our empirical evaluation, we aim to quantify the following characteristics of the proposed constructions:

- The time required to generate a trapdoor and its corresponding size as a function of the size of the largest analysis pattern p_{max} that can be searched.
- The time taken to encrypt a data stream as a function of its size (i.e. the size of the sequence of symbols that composed the data to be encrypted), the fragmentation size Φ and the size of the considered alphabet.
- The time needed to perform a pattern matching query as a function of the size of the data to be queried and the size of the patterns to be searched.



(a) as a function of the number of symbols in the pattern to be searched

(b) as a function of Φ

Fig. 2: Trapdoor generation time

Trapdoor generation. Fig. 2 describes the time required for issuing a trapdoor for a pattern w as a function of its length (Fig. 2 (a)) as well as the size

[5] The objective behind the usage of the 254-bits BN is to consider the same elliptic curve as in the implementation of the SEST construction. We note that the pairings over the 254-bits BN curve provides almost 100-bit security level.

Φ of data a data fragment (Fig. 2 (b)). According to our experiments, issuing a trapdoor for a pattern of 5000 symbols take 1.4 second. In addition, the sizes of the generated trapdoors are relatively small (256 KB for a pattern of 4000 symbols and a fragmentation size of 10000 symbols).

Encryption time. According to Section 7, the duration of an encryption operation depends mainly on the number of symbols in the data to be encrypted n but also on the fragmentation size Φ and the size $|\Sigma|$ of the considered alphabet Σ. Table 2 reports the time needed to encrypt a data stream fragmented in chunks, each containing 1000 bits ($\Phi = 1000$ and $\Sigma = \{0, 1\}$), as a function of the data stream length n.[6].

Data Length (bytes)	**Time (seconds)**
1000	0.031
3000	0.097
5000	0.158
10000	0.371
30000	1.01
100000	3.0355

Table 2: Encrypting time as a function of n

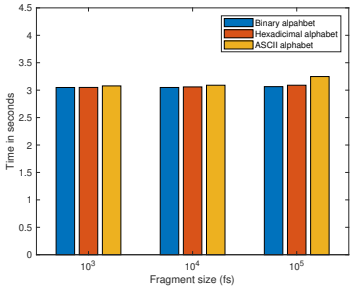


Fig. 3: Time required for encrypting 10^5 symbols as a function of Φ and Σ

As we noted in Section 4, the fragmentation size Φ and the considered alphabets are important parameters in our construction. The former directly influences the size of the largest analysis pattern that can be searched over the encrypted data since the bigger the size of the fragments are, the bigger the size of the supported analysis patterns could be. The latter parameter determines the type of search that can be performed by our construction. In Fig. 3, we compute the time required for the encryption of a data stream composed of 10^5 symbols as a function of the fragmentation size Φ and the type of the considered symbols. We consider three types of alphabets: binary, hexadecimal, and base 256 (i.e., ASCII alphabet) where each symbol is represented respectively in 1, 4 and 8 bits. For Φ, we consider 3 different fragment sizes: 10^3, 10^4, and 10^5 symbols.

As illustrated in Fig. 3, the time required for encrypting a dataset composed of 10^5 symbols increases only by a factor of 0.02 (from 3,04 to 3,2 seconds) when increasing the size of the fragments by a factor of 100 (from 10^3 to 10^5) and increasing the size of the considered alphabet by a factor of 128 (from a base 2 alphabet where $\Sigma = \{0, 1\}$ to a base 256 alphabet where $\Sigma = \{0, 1, \cdots, 255\}$). The previous results show that the increase of the size of supported patterns and the size of the considered alphabet affects very little the encryption time

[6] Encryption time would be roughly 8 times slower with a single-threaded execution.

required by the proposed constructions.

Search time. As shown in Section 7, the complexity of the search operation depends mainly on the number of encrypted symbols n that compose the data to be analyzed. Fig. 4 describes the time required for searching a pattern as a function of the number of encrypted symbols in the data to be analyzed.

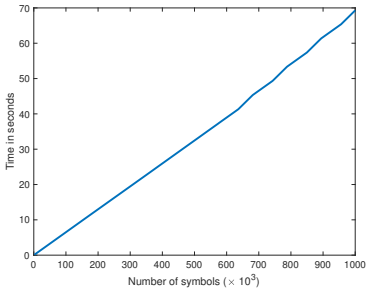
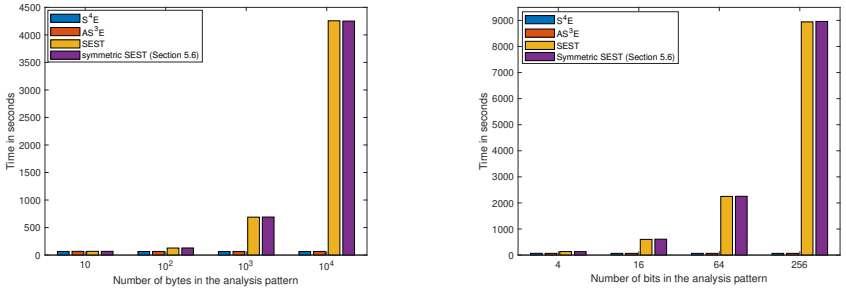


Fig. 4: Time required for searching a pattern as a function of the number of encrypted symbols in the data to be analyzed

The conducted evaluations show that the average search throughput of our construction is 139078 symbols per second with a multi-threaded implementation [7]. Thus, if an ASCII (resp. binary) alphabet is considered, the search throughput is 139 KB (resp. Kb) per second.



(a) byte-level search ($\Sigma = \{0, \cdots, 255\}$) (b) bit-level search ($\Sigma = \{0, 1\}$)

Fig. 5: Timing comparison for testing the presence of a pattern in a string of 10^7 symbols as a function of the pattern size

[7] search time would be roughly 96 times slower with a single-threaded execution.

Fig. 5 (a) (resp. Fig. 5 (b)) compares the time needed for both our and the SEST (both its asymmetric [1] and symmetric (Section 5.6) variants) constructions to test the presence of a pattern of bytes (resp. of bits) in a 10 MB (resp. Mb) dataset as a function of the length of the pattern to be searched. In both bit and byte searches, our construction drastically reduces the search time compared to SEST. This is because that our Test algorithm is constant on the size and on the content of the searched pattern which is not the case for SEST.

9 Conclusion

In this work, we introduced two new provably correct and secure constructions S^4E and AS^3E. S^4E (resp. AS^3E) supporting pattern matching of adaptively chosen and variable (upper bounded) lengths patterns on secret key (resp. public key) encrypted streams. The proposed constructions have several interesting properties. First, they ensure data and pattern indistinguishability meaning that the entity that is going to perform pattern matching will learn nothing about the patterns to be searched as well as the data to be inspected, except the presence or the absence of a set of "unknown" patterns (since the entity charged to perform pattern matching will not have access to the patterns plaintexts). Second, the size of the ciphertext is linear to the size of the plaintext and is constant on the sizes and the number of analysis patterns. Third, the size of the issued trapdoors is constant on the size of the data to be analyzed. Finally, the search complexity is linear to the size of the trace and is constant on the size of the analysis patterns. The proposed constructions can be useful for other application scenarios such as subtrees search and searching of structured data.

To prove the security of the two proposed schemes, we used a slightly modified GDH assumption where the adversary is allowed to choose on which input to play the GDH instance. This relatively minor modification of the GDH assumption allow to define constructions that offer an interesting compromise between the secure and quite costly solutions and the fast and unsecure solution where the data has to be decrypted by the third-party entity that performs the pattern matching.

References

1. Desmoulins, N., Fouque, P. A., Onete, C., & Sanders, O. (2018, December). Pattern Matching on Encrypted Streams. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 121-148). Springer, Cham.
2. Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2011). Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security, 19(5), 895-934.
3. Kamara, S., Moataz, T., & Ohrimenko, O. (2018, August). Structured encryption and leakage suppression. In Annual International Cryptology Conference (pp. 339-370). Springer, Cham.
4. Chase, M., & Shen, E. (2015). Substring-searchable symmetric encryption. Proceedings on Privacy Enhancing Technologies, 2015(2), 263-281.

5. Sherry, J., Lan, C., Popa, R. A., & Ratnasamy, S. (2015). Blindbox: Deep packet inspection over encrypted traffic. ACM SIGCOMM Computer communication review, 45(4), 213-226.
6. Canard, S., Diop, A., Kheir, N., Paindavoine, M., & Sabt, M. (2017, April). Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (pp. 561-574). ACM.
7. Gentry, C., & Boneh, D. (2009). A fully homomorphic encryption scheme (Vol. 20, No. 09). Stanford: Stanford University.
8. Boneh, D., Sahai, A., & Waters, B. (2011, March). Functional encryption: Definitions and challenges. In Theory of Cryptography Conference (pp. 253-273). Springer, Berlin, Heidelberg.
9. Lauter, K., López-Alt, A., & Naehrig, M. (2014, September). Private computation on encrypted genomic data. In International Conference on Cryptology and Information Security in Latin America (pp. 3-27). Springer, Cham.
10. Hazay, C., & Lindell, Y. (2010). Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. Journal of cryptology, 23(3), 422-456.
11. Gennaro, R., Hazay, C., & Sorensen, J. S. (2016). Automata evaluation and text search protocols with simulation-based security. Journal of Cryptology, 29(2), 243-282.
12. Troncoso-Pastoriza, J. R., Katzenbeisser, S., & Celik, M. (2007, October). Privacy preserving error resilient DNA searching through oblivious automata. In Proceedings of the 14th ACM conference on Computer and communications security (pp. 519-528). ACM.
13. Katz, J., Sahai, A., & Waters, B. (2013). Predicate encryption supporting disjunctions, polynomial equations, and inner products. Journal of cryptology, 26(2), 191-224.
14. Boneh, D., & Waters, B. (2007, February). Conjunctive, subset, and range queries on encrypted data. In Theory of Cryptography Conference (pp. 535-554). Springer, Berlin, Heidelberg.
15. Barreto, P. S., & Naehrig, M. (2005, August). Pairing-friendly elliptic curves of prime order. In International Workshop on Selected Areas in Cryptography (pp. 319-331). Springer, Berlin, Heidelberg.
16. Canetti, R., Halevi, S., & Katz, J. (2003, May). A forward-secure public-key encryption scheme. In International Conference on the Theory and Applications of Cryptographic Techniques (pp. 255-271). Springer, Berlin, Heidelberg.
17. Bellare, M., Boldyreva, A., & O'Neill, A. (2007, August). Deterministic and efficiently searchable encryption. In Annual International Cryptology Conference (pp. 535-552). Springer, Berlin, Heidelberg.
18. MISP - Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing, https://www.misp-project.org/, 23 12 2011.
19. Xavier Boyen. The uber-assumption family. In *International Conference on Pairing-Based Cryptography*, pages 39–56. Springer, 2008.
20. Snort Rules. https://www.snort.org/, Accessed: 2019-08-35.
21. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
22. Anis Bkakria, Nora Cuppens, and Frédéric Cuppens. Pattern matching on encrypted data. Cryptology ePrint Archive, Report 2020/422, 2020. https://eprint.iacr.org/2020/422.