Public Key Encryption with Flexible Pattern Matching

Elie Bouscatié, Guilhem Castagnos, Olivier Sanders

Public Key Encryption with Flexible Pattern Matching

Élie Bouscatié[1,2], Guilhem Castagnos[2], and Olivier Sanders[1]

[1] Orange Labs, Applied Crypto Group, Cesson-Sévigné, France
[2] Université de Bordeaux, INRIA, CNRS, IMB UMR 5251, F-33405 Talence, France

Abstract. Many interesting applications of pattern matching (*e.g.* deep-packet inspection or medical data analysis) target very sensitive data. In particular, spotting illegal behaviour in internet traffic conflicts with legitimate privacy requirements, which usually forces users (*e.g.* children, employees) to blindly trust an entity that fully decrypts their traffic in the name of security.

The compromise between traffic analysis and privacy can be achieved through searchable encryption. However, as the traffic data is a stream and as the patterns to search are bound to evolve over time (*e.g.* new virus signatures), these applications require a kind of searchable encryption that provides more flexibility than the classical schemes. We indeed need to be able to search for patterns of variable sizes in an arbitrary long stream that has potentially been encrypted prior to pattern identification. To stress these specificities, we call such a scheme a stream encryption supporting pattern matching.

Recent papers use bilinear groups to provide public key constructions supporting these features [3, 13]. These solutions are lighter than more generic ones (*e.g.* fully homomorphic encryption) while retaining the adequate expressivity to support pattern matching without harming privacy more than needed. However, all existing solutions in this family have weaknesses with respect to efficiency and security that need to be addressed. Regarding efficiency, their public key has a size linear in the size of the alphabet, which can be quite large, in particular for applications that naturally process data as bytestrings. Regarding security, they all rely on a very strong computational assumption that is both interactive and specially tailored for this kind of scheme.

In this paper, we tackle these problems by providing two new constructions using bilinear groups to support pattern matching on encrypted streams. Our first construction shares the same strong assumption but dramatically reduces the size of the public key by removing the dependency on the size of the alphabet, while nearly halving the size of the ciphertext. On a typical application with large patterns, our public key is two order of magnitude smaller than the one of previous schemes, which demonstrates the practicality of our approach. Our second construction manages to retain most of the good features of the first one while exclusively relying on a simple (static) variant of DDH, which solves the security problem of previous works.

Keywords: Pattern Matching · Searchable encryption

1 Introduction

The increasing outsourcing of IT services allows companies to shift the burden of managing their own infrastructure to some third parties but comes with many challenges regarding privacy. Traditional encryption is of no help here as it would prevent these third parties from providing their services. This has led cryptographers to propose countless encryption algorithms that are compatible with some sets of functions, meaning that it is possible to evaluate these functions directly on the ciphertexts, without having to decrypt the latter.

1.1 Related Works

As a rule of thumb, versatile systems supporting a large set of functions (*e.g.* [6, 15]) are the most complex ones, which has led to the design of encryption schemes supporting a very specific function. One of the most prominent examples of this approach is the one of searchable encryption (*e.g.* [1, 4, 12, 18, 22]) where some entities have the ability to decide whether a ciphertext C contains a given pattern (also called keyword) without decrypting C. Put differently, the ciphertext leaks nothing but the presence (or absence) of the pattern. The popularity of this type of encryption stems from the variety of applications that only need the ability to search a pattern (*e.g.*, DPI: deep packet inspection [13, 21], external storage [8], etc) combined with the efficiency of most cryptographic schemes supporting this feature. However, the fact that the latter are all presented as *searchable encryption* schemes does not mean that they are similar. Actually, this is quite the opposite as illustrated, for example, by the construction in [8] and the one in [13].

Here, the differences lie not only in the choice of the security model or the computational assumption underlying the construction, as it is usually the case in cryptography, but also in the ability to index data before encryption. In the case of external storage [8], it indeed seems reasonable to assume that each data of a database can be associated with a set of appropriate keywords that will be processed to ensure efficient queries on the encrypted database. Conversely, the use-case of DPI of Internet traffic [13, 21] can hardly assume indexation of sent data. One should rather assume in this case that the data are encrypted on-the-fly without being able to pre-process them. Moreover, as pointed out in [13], there might be no obvious set of keywords to associate with the transmitted data. Finally, in this case, the searched pattern/keyword can be located anywhere in the encrypted stream, which precludes standard searchable encryption schemes (*e.g.* [1]): We do not want to decide if a ciphertext C encrypts a given pattern W but, instead, if C encrypts a message that contains W as a substring, which is fundamentally different.

In this regard, the case of encrypted traffic, that we study in this paper, is clearly the most complex one. To emphasize the difference with the scenarios compatible with indexation, we will talk of *Stream* Encryption supporting *Pattern Matching* (SEPM).

More specifically, we will consider a simple but versatile use-case where a *receiver* relies on a *service provider* to analyse the encrypted traffic he receives from a *sender*. We assume that this service requires to perform pattern matching on the traffic, which is actually the case for several applications (*e.g.* DPI). As the service provider is not fully trusted[3], we do not want to share the decryption key with it. Instead, the service provider will receive from the receiver specific trapdoors that allows it to detect the presence of some patterns within the encrypted streams.

To rehabilitate standard searchable encryption schemes, the first approach to solve this problem was based on tokenization (*e.g.* [10, 21]), a technique that consists in splitting the stream to encrypt into overlapping substrings of some fixed length ℓ. Each substring S is then encrypted using a searchable encryption scheme whereas trapdoors are issued for patterns W of size ℓ. Thanks to the property of searchable encryption, one can indeed decide if $S = W$, which solves our problem as long as all searched patterns have the same size ℓ. Unfortunately, this approach inherently suffers from at least one of the following downsides, as explained in [13]: lack of expressivity if one considers only one possible substring length ℓ, lack of privacy if one splits the genuine patterns into several subpatterns of the same size or lack of efficiency if one repeats the process for each possible pattern length.

One could solve the expressivity issue by using instead predicate/functional encryption with some additional privacy features, such as *e.g.* anonymous predicate encryption [16] or hidden vector encryption [7]. A symmetric alternative was also recently proposed in [17]. Unfortunately, these solutions inherently require to provide a trapdoor for each possible position of a given pattern within the stream, which is a real problem in our case as the stream can be of any length. As explained in [13], one would then have to define a sufficiently large upper bound on this length and then generate a very large number of trapdoors (*e.g.* 1 billion for a 1 GB stream) for *each* pattern. In the symmetric setting, one could leverage efficient schemes (*e.g.* [17,23]) to argue that each trapdoor can be relatively small but, in this case, a new key (and hence new trapdoors) must be generated for each communication, which quickly becomes cumbersome. More generally, the public key setting seems more suitable in our case as it allows to generate universal trapdoors that can be used to analyse the traffic with any sender.

To circumvent all these problems, the authors of [13] proposed a new approach that allows to search patterns of any size with constant-size trapdoor. Intuitively, the core idea of this scheme is to encrypt the stream character by character by generating group elements whose exponent is $\alpha_b z^i$ where α_b is a secret encoding of the character b and z^i is a secret monomial encoding the position i of this character within the stream. Aggregating these elements leads to polynomials that can be identified with appropriate trapdoors. Unfortunately, these nice features come at the cost of three major weaknesses:

[3] More specifically, the service provider is trusted to provide the requested service but it should only learn the information necessary to carry out its task.

1. The security of [13] requires secrecy of all the elements α_b and z cited above. As the sender needs this information to encrypt the stream, the solution chosen by the authors is to provide the group elements $g^{\alpha_b z^i}$ in the public key for every possible position i and character b in the alphabet. The size of the public key thus significantly increases with the ones of the stream and of the alphabet, which quickly becomes cumbersome.
2. The polynomial construction of the trapdoors uses coefficients that have to be fresh, at least for different occurrences of the same character. The consequence is that the number of pairings needed for a test at some position is linear in the maximum occurrence of a same symbol in the pattern, which significantly increases the computational cost of the pattern detection procedure.
3. The security analysis of [13] was only made under a very strong, ad-hoc interactive assumption (i-GDH) that is likely to be necessary, as explained by the authors.

Very recently, [3] addresses some of the problems above by introducing a fragmentation approach that consists in splitting the stream into non-overlapping fragments $\mathcal{F}_h$ and into other non-overlapping fragments $\overline{\mathcal{F}}_h$ that straddle the former. This technique will be explained in more details in Section 4.1 but intuitively this is done in such a way that any searched pattern is entirely contained by a fragment $\mathcal{F}_h$ or $\overline{\mathcal{F}}_h$. The main advantage of this technique, that can actually be applied to [13] or any similar schemes, is that it reduces the problem of encrypting large strings to the one of encrypting several small fragments, which significantly reduces the size of the public key.

Based on this fragmentation approach, the authors of [3] propose a construction that allows to test the presence of one pattern at one position for a constant cost of 2 pairings. Moreover the dependency of the public key on the length of the string is replaced by a fixed upper bound on the length of the keywords to be searched, which is indeed much smaller in the context of DPI. But this construction uses twice as many ciphertext elements as in [13] and shares several features with it, including the fact that security still relies on the interactive i-GDH assumption and that the public key depends linearly on the size of the alphabet. The authors of [3] also consider the notion of pattern privacy, meaning that the trapdoors should not reveal the corresponding pattern but, as already noted in [13], it is very hard to retain this property for this kind of schemes, which leads to a security model in [3] that seems a bit contrived. Moreover, in the asymmetric setting that we consider here, this property can only be achieved for patterns originating from a high min-entropy set as in [5]. A look at some open-source list of patterns[4] shows that this assumption does not hold, at least for the DPI use-case. In this paper, we will therefore not consider this outlying property that would only make our security model more complex.

[4] *e.g.* https://github.com/coreruleset/coreruleset

1.2 Our Contributions

If we sum up the state-of-the-art of SEPM, there are two main areas of progress: performance and security. We propose to improve both with two related constructions that solve the previous problems one after the other.

Improving efficiency. From the efficiency standpoint, we note that [3] manages to reduce the size of the public key and the complexity of the detection procedure, compared to [13], but at the cost of ciphertexts containing twice as many elements. Moreover, if L is the size of the fragments and $\mathcal{S}$ is the plaintext alphabet (that is, we encrypt strings of characters $b \in \mathcal{S}$) the public key of [3] is essentially of size $L|\mathcal{S}|$, which remains quite important for many use-cases. For example, in the DPI context, it is natural to consider bytestrings which means that $|\mathcal{S}| = 256$. At first sight, it could be tempting to consider smaller alphabets, *e.g.*, bits instead of bytes, but this would lead to larger fragments that would result in a significant expansion of the ciphertext (eight-fold if we use bits instead of bytes) and that would reduce the gain regarding the public key.

In our first construction, we completely depart from the polynomial approach used in [3, 13] to fully leverage the fragmentation approach. More specifically, we note that the geometric basis z^i introduced in [13] and taken over by [3] is no longer required thanks to fragmentation. This allows us to design a new construction that looks more natural and that reduces the size of the ciphertext to nearly half the one in [3]. Interestingly, the resulting ciphertexts are essentially signatures on the characters to encrypt for some aggregatable signature scheme [19]. Intuitively, aggregatability of the signatures will ensure correctness of the construction as one will be able to combine different ciphertext elements to reconstruct (encrypted) patterns that can be tested with the appropriate trapdoors. At the same time, unforgeability of the signatures will ensure non-malleability of the ciphertexts and hence security of the whole construction.

Moreover, thanks to our approach, we can replace the secret character encoding α_b used in previous works by public elements of $\mathbb{Z}_p$ (that act as the signed messages for [19]), which leads to shorter public keys that no longer depend on the size of the alphabet.

Table 1 highlights the benefits of our first construction compared to the state-of-the-art. Although the gain consists in some multiplicative factors that could get lost in a $O(\cdot)$ notation, we stress that these factors have important consequences in practice. For example, if we take over the concrete parameters considered in [3], we show in Section 6 that we end up with a public key of 1.92 MB instead of 247 MB. For real-world applications, there is a significant difference between these two sizes as the latter would probably be impractical for many use-cases.

Improving security. Our first construction only focuses on efficiency but does not consider the issue of previous works regarding security, namely the reliance on interactive ad-hoc assumptions. Actually, it still requires the i-GDH assumption, which is not very satisfying.

In our second construction, we tackle this problem by designing a scheme relying on a static assumption, EXDH, that is a simple variant of the DDH assumption. Actually, this assumption has already been used to construct an e-cash system in [11], which gives more confidence in the hardness of the underlying computational problem.

Here, the main difficulty is to modify our original scheme so as to rely on this static assumption while limiting the impact on the performance. This is particularly difficult because we consider a very strong security model where the adversary is able to query any trapdoor that does not allow to trivially succeed in the security experiment. In particular, we allow the adversary to query trapdoors that match the challenge streams, which makes simulation much harder, as we will explain in Section 4.

We nevertheless manage to deal with these various queries with a simple assumption by essentially adding two elements per character in the ciphertext. Regarding the size of the latter, this brings us back to the state-of-the-art [3] but our second construction has two main advantages. Firstly, it retains a short public key that still does not depend on the size of the alphabet. Secondly, it relies on a static assumption, which is a significant improvement over other schemes.

Summary of contributions Table 1 provides a comparison between our constructions and [3, 13] with respect to the main metrics of such schemes. A more detailed complexity analysis can be found in Section 6.

This table shows that our first construction yields significantly shorter public keys while roughly halving the size of the ciphertext compared to [3]. It is done without decreasing the performance of the Test procedure (*i.e.* patterns detection). This is therefore the most suitable solution if one favours efficiency.

Our second construction reports lesser performance (but still better than the state-of-the-art for several metrics) but relies on a static computational assumption, which is noticeable compared to previous constructions. This is the current best solution if one favours security.

		Schemes			
		SEST ([13])	AS^{3}E([3])	Section 4.3	Section 4.4
Public Key	(nb. elements)	$n(\lvert\mathcal{S}\rvert+1)$	$2L(\lvert\mathcal{S}\rvert+1)$	**4L**	**6L**
Ciphertext	(nb. elements)	$2n$	$4n$	$\mathbf{2n+\frac{n}{L}}$	$4n+\frac{n}{L}$
Trapdoor	(nb. elements)	$L+2$	$2L$	$2L$	$3L$
Test	(nb. pairings)	$n(L+2)$	$2n$	$2n$	$3n$
Computational Assumption		interactive	interactive	interactive	**static**

Table 1. Comparison with related works. The scalars $|\mathcal{S}|$ and n denote respectively the number of elements in the plaintext alphabet and the length of the traffic to encrypt. L stands for the length of the longest pattern queried in SEST and for an upper bound on this value in the other schemes.

Outline. In Section 2 we provide the necessary background on bilinear groups along with the description of the computational assumptions used in our paper. Section 3 is dedicated to the syntax and the security model of SEPM. Our constructions are described in Section 4 and then proven secure in Section 5. Finally, we give a complexity analysis in Section 6.

2 Preliminaries

2.1 Bilinear Groups

Our construction requires bilinear groups whose definition is recalled below.

Definition 1. *Bilinear groups are a set of three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p along with a map, called pairing, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that is*

1. *bilinear: for any $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$;*
2. *non-degenerate: for any $(g, \widetilde{g}) \in \mathbb{G}_1 \times \mathbb{G}_2$, $(g, \widetilde{g}) \neq (1_{\mathbb{G}_1}, 1_{\mathbb{G}_2})$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;*
3. *efficient: for any $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$, $e(g, \widetilde{g})$ can be efficiently computed.*

As most recent cryptographic papers, we only consider bilinear groups of prime order with *type 3* pairings [14], meaning that no efficiently computable homomorphism is known between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.2 Decisional Assumptions

We now introduce the decisional assumptions underlying the security of our constructions.

Definition 2 (i-GDH assumption [13]). *Let r, s, t, c and κ be five positive integers and $\mathtt{R} \in \mathbb{Z}_p[X_1, \ldots, X_c]^r, \mathtt{S} \in \mathbb{Z}_p[X_1, \ldots, X_c]^s$ and $\mathtt{T} \in \mathbb{Z}_p[X_1, \ldots, X_c]^t$ be three tuples of multivariate polynomials over $\mathbb{Z}_p$. For any polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_c]$, we say that f is dependent on $< \mathtt{R}, \mathtt{S}, \mathtt{T} >$ if there are $\{a_j\}_{j=1}^s \in \mathbb{Z}_p^s \setminus \{(0, \ldots, 0)\}$, $\{b_{i,j}\}_{i,j=1}^{i=r,j=s} \in \mathbb{Z}_p^{r \cdot s}$ and $\{c_k\}_{k=1}^t \in \mathbb{Z}_p^t$ such that*

$$f \sum_i a_j S^{(j)} = \sum_{i,j} b_{i,j} R^{(i)} S^{(j)} + \sum_k c_k T^{(k)}.$$

Let $\mathcal{O}^{\mathtt{R}}$ (resp. $\mathcal{O}^{\mathtt{S}}$ and $\mathcal{O}^{\mathtt{T}}$) be oracles that, on input $\{\{a^{(k)}_{i_1,\ldots,i_c}\}_{i_1,\ldots,i_c=0}^{d_k}\}_{k=1}^{\kappa}$, add the polynomials $\{\sum_{i_1,\ldots,i_c} a^{(k)}_{i_1,\ldots,i_c} \prod_j X_j^{i_j}\}_{k=1}^{\kappa}$ to $\mathtt{R}$ (resp. $\mathtt{S}$ and $\mathtt{T}$).

Let $(\chi_1, \ldots, \chi_c) \xleftarrow{\$} \mathbb{Z}_p^c$ be a secret vector and $q_{\mathtt{R}}$ (resp. $q_{\mathtt{S}}$ and $q_{\mathtt{T}}$) be the number of queries to $\mathcal{O}^{\mathtt{R}}$ (resp. $\mathcal{O}^{\mathtt{S}}$) (resp. $\mathcal{O}^{\mathtt{T}}$). The i-GDH *assumption states that, given the values $\{g^{R^{(i)}(\chi_1,\ldots,\chi_c)}\}_{i=1}^{r+\kappa q_{\mathtt{R}}}, \{\widetilde{g}^{S^{(i)}(\chi_1,\ldots,\chi_c)}\}_{i=1}^{s+\kappa q_{\mathtt{S}}}$ and $\{e(g, \widetilde{g})^{T^{(i)}(\chi_1,\ldots,\chi_c)}\}_{i=1}^{t+\kappa q_{\mathtt{T}}}$, it is hard to decide whether $\zeta = g^{f(\chi_1,\ldots,\chi_c)}$ or ζ uniform in $\mathbb{G}_1$ if f is independent of $< \mathtt{R}, \mathtt{S}, \mathtt{T} >$.*

This strong assumption has been introduced in [13] and used in a subsequent work [3]. We only use it in our first protocol and show how to replace it by the following static assumption in our second protocol.

Definition 3 (EXDH assumption [11]). *Given $g, g^a, g^{ab}, g^c \in \mathbb{G}_1$ and $\widetilde{g}, \widetilde{g}^a, \widetilde{g}^b \in \mathbb{G}_2$, it is hard to decide whether $\zeta = g^{abc}$ or ζ is uniform in $\mathbb{G}_1$.*

This assumption was used in [11] to construct an e-cash system. In that work, it was called the weak-EXDH assumption because the authors also consider a stronger variant of this assumption. In this paper, we simply call it the EXDH assumption as we only need this weak variant. It only holds for type 3 bilinear groups.

3 Stream encryption supporting pattern matching (SEPM)

Notation. For two integers $a < b$, we let $[\![a, b[\![= \{i \in \mathbb{N} : a \leq i < b\}$, or simply $[\![b[\![$ if $a = 0$. For a finite set S, we use the notation $x \xleftarrow{\$} S$ to say that x is chosen uniformly at random in S.

In this paper, we consider entities exchanging data that are represented as sequences of characters that we call *strings*. These characters may originate from different sets/alphabets (*e.g.* $\{0,1\}$ for bitstrings, $\{0,1\}^8$ for bytestrings, etc) but for sake of simplicity we assume that each of them can be associated with a unique element of $\mathbb{Z}_p$, for some large prime p. For most cases, this mapping ϕ is straightforward, for example:

- $\{0,1\} \xrightarrow{\phi} \mathbb{Z}_p$ with $\phi(b) = b \in \mathbb{Z}_p$
- $\{0,1\}^8 \xrightarrow{\phi} \mathbb{Z}_p$ with $\phi(b_7, \ldots, b_0) = \sum_{i=0}^{7} b_i 2^i \in \mathbb{Z}_p$

In the worst case, it is always possible to define a correspondence table so we can consider strings of elements of $\mathbb{Z}_p$ without loss of generality. Finally, as in previous works (*e.g.* [3,13]), we will consider a wildcard character $\star$ that matches all characters. Therefore, all data considered in this paper are assumed to be strings of characters in $\mathbb{Z}_p \cup \{\star\}$. For a string $W = (w_0, \ldots, w_{\ell-1}) \in (\mathbb{Z}_p \cup \{\star\})^\ell$ of length $\ell \in \mathbb{N}$, we let $\mathsf{supp}(W) = \{j \in [\![\ell[\![: w_j \neq \star\}$.

3.1 Definition

We adapt the syntax and security of SEST [13] by setting an upper bound L on the length of the keywords for which trapdoors may be issued. Contrary to that work, our syntax does not require to define an upper bound on the length of the stream to be encrypted.

A stream encryption scheme that supports pattern matching (SEPM) is defined by 5 algorithms that we call `Setup`, `Keygen`, `Issue`, `Encrypt` and `Test`. The first three of these are run by an entity called the receiver, while `Encrypt` is run by a sender and `Test` by a gateway.

- $\texttt{Setup}(1^\lambda, L)$: This probabilistic algorithm takes as input a security parameter λ and an upper bound L on the length of the keywords for which trapdoors may be issued. It returns the public parameters pp that will be considered as an implicit input of all other algorithms and so will be omitted.
- $\texttt{Keygen}()$: This probabilistic algorithm run by the receiver returns a key pair $(\mathsf{sk}, \mathsf{pk})$. The former value is secret and only known to the receiver, while the latter is public.
- $\texttt{Issue}(W, \mathsf{sk})$: This probabilistic algorithm takes as input the receiver's secret key along with a string $W = (w_0, \ldots, w_{\ell-1}) \in (\mathbb{Z}_p \cup \{\star\})^\ell$ of any size $\ell \leq L$ and returns a trapdoor TD_W.
- $\texttt{Encrypt}(M, \mathsf{pk})$: This probabilistic algorithm takes as input the receiver's public key along with a string $M = (m_0, \ldots, m_{n-1}) \in \mathbb{Z}_p^n$ of any size n and returns a ciphertext C.
- $\texttt{Test}(C, W, \mathsf{TD}_W)$: This deterministic algorithm takes as input a ciphertext C encrypting a string $M = (m_0, \ldots, m_{n-1}) \in \mathbb{Z}_p^n$ of any size n along with a trapdoor TD_W for a string $W = (w_0, \ldots, w_{\ell-1}) \in (\mathbb{Z}_p \cup \{\star\})^\ell$ of any size $\ell \leq L$. It returns the set (potentially empty) $\mathsf{Match} \subset [\![n]\!]$ of all indexes i s.t. for all $k \in \mathsf{supp}(W)$, $w_k = m_{i+k}$.

As in recent schemes, [3, 13], and more generally in searchable encryption, [1, 7], our definition of SEPM does not consider a decryption algorithm: this functionality can easily be added by also encrypting the stream under a conventional encryption scheme. However, decryption could be performed in an SEPM by issuing a trapdoor for all characters of $\mathbb{Z}_p$ and running the $\texttt{Test}$ algorithm on the ciphertext for each of them.

3.2 Security Model

Correctness. As in [1], we divide correctness into two parts. The first one stipulates that the $\texttt{Test}$ algorithm run on (C, W, TD_W) will always return i if W matches M at index i (no false negatives). More formally, this means that, for any string M of size n and any W of length $\ell \leq min(n, L)$:

$$(\forall k \in \mathsf{supp}(W), m_{i+k} = w_k) \\ \Rightarrow \Pr[i \in \texttt{Test}(\texttt{Encrypt}(M, \mathsf{pk}), W, \texttt{Issue}(W, \mathsf{sk}))] = 1,$$

where the probability is taken over the set of key-pairs $(\mathsf{sk}, \mathsf{pk})$.

The second part of the correctness property requires that false positives (*i.e.*, when the $\texttt{Test}$ algorithm returns i despite the fact that W doesn't match M at this position) only occur with negligible probability. More formally, this means that, for any string M of size n and any W of length $\ell \leq min(n, L)$:

$$(\exists k \in \mathsf{supp}(W), m_{i+k} \neq w_k) \\ \Rightarrow \Pr[i \in \texttt{Test}(\texttt{Encrypt}(M, \mathsf{pk}), W, \texttt{Issue}(W, \mathsf{sk}))] = \mu(\lambda)$$

where the probability is taken over the set of key-pairs $(\mathsf{sk}, \mathsf{pk})$ and μ is a negligible function.

Selective Indistinguishability (sIND-CPA). We use the notion of selective indistinguishability defined in [13] which is adapted to be consistent with the slight modifications we introduce in the syntax.

Informally, this notion requires that no adversary $\mathcal{A}$, having committed to $M^{(0)}$ and $M^{(1)}$ before seeing pk, can decide whether a ciphertext C encrypts $M^{(0)}$ or $M^{(1)}$, even with access to an oracle returning a trapdoor TD_W for any queried string W that does not allow to trivially distinguish these two strings. This is formally defined by the experiment $\mathtt{Exp}^{sind-cpa}_{\mathcal{A}}(1^\lambda, L)$ described in Figure 3.2. Here, $\mathcal{O}\mathtt{Issue}$ returns $\mathsf{TD}_W \leftarrow \mathtt{Issue}(W, \mathsf{sk})$ when queried on $W = (w_0, \ldots, w_{\ell-1})$ with $\ell \leq L$, unless there are $i \in [\![n - \ell[\![$ and $b \in \{0,1\}$ with

$$(\forall k \in \mathsf{supp}(W), m^{(b)}_{i+k} = w_k) \wedge (\exists k \in \mathsf{supp}(W), m^{(1-b)}_{i+k} \neq w_k).$$

This is a natural restriction as TD_W would allow to trivially win this experiment for such W. We nevertheless stress that $\mathcal{O}\mathtt{Issue}$ can be queried with patterns W matching *both* $M^{(0)}$ and $M^{(1)}$. Finally, we require that $M^{(0)}$ and $M^{(1)}$ be of the same size because the corresponding ciphertexts would be trivially distinguishable otherwise. This restriction could however be lifted by using some padding technique to generate constant-size ciphertexts.

$\mathtt{Exp}^{sind-cpa}_{\mathcal{A}}(1^\lambda, L)$

1. $pp \leftarrow \mathtt{Setup}(1^\lambda, L)$
2. $(M^{(0)}, M^{(1)}) \leftarrow \mathcal{A}$, with $M^{(b)} = (m^{(b)}_0, \ldots, m^{(b)}_{n-1})$ for $b \in \{0,1\}$ and $n \in \mathbb{N}$
3. $\mathsf{pk} \leftarrow \mathtt{Keygen}()$
4. $\beta \xleftarrow{\$} \{0,1\}$
5. $C \leftarrow \mathtt{Encrypt}(M^{(\beta)}, \mathsf{pk})$
6. $\beta' \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Issue}}(C, \mathsf{pk})$
7. **If** $\beta = \beta'$ then **return** 1, **else return** 0.

Fig. 1. sIND-CPA Security Game

We define the advantage of an adversary $\mathcal{A}$ in $\mathtt{Exp}^{sind-cpa}_{\mathcal{A}}(1^\lambda, L)$ as

$$\mathtt{Adv}^{sind-cpa}_{\mathcal{A}}(1^\lambda, L) = \left| \Pr[\mathtt{Exp}^{sind-cpa}_{\mathcal{A}}(1^\lambda, L) = 1] - \frac{1}{2} \right|.$$

A stream encryption scheme that is searchable for pattern matching is sIND-CPA secure if this advantage is negligible for any polynomial-time adversary.

4 Our Constructions

Before explaining how our constructions work, we first recall the fragmentation technique introduced in [3] that we slightly simplify for ease of exposition.

4.1 Fragmentation

Let n be the length of the string to be encrypted and $L \geq 2$ be the upper bound on the length of the patterns to search. We set $d_{\mathcal{F}} := L - 1$ and $s_{\mathcal{F}} := 2d_{\mathcal{F}}$. We suppose for simplicity that there exists an integer $n_{\mathcal{F}}$ such that $n = (2n_{\mathcal{F}}+1)d_{\mathcal{F}}$. Note that we can always fulfil this requirement by adding dummy characters to the string to encrypt. See also the remark at the end of this subsection.

For all $h \in [\![n_{\mathcal{F}}[\![$, we call $\mathcal{F}_h = [\![s_{\mathcal{F}}h, s_{\mathcal{F}}(h+1)[\![$ a *fragment* and we call $\overline{\mathcal{F}}_h = [\![s_{\mathcal{F}}h + d_{\mathcal{F}}, s_{\mathcal{F}}(h+1) + d_{\mathcal{F}}[\![$ an *overlined fragment*. Hence, $n_{\mathcal{F}}$ is the *number* of fragments (or overlined ones), $s_{\mathcal{F}}$ is their *length* and $d_{\mathcal{F}}$ is the *offset* between fragments and overlined ones.

A remarkable property of this construction is that for any integer $\ell \leq L$ and any index $i \in [\![n - \ell[\![$, the set of ℓ consecutive integers $[\![i, i + \ell[\![$ is contained in at least an (overlined) fragment.

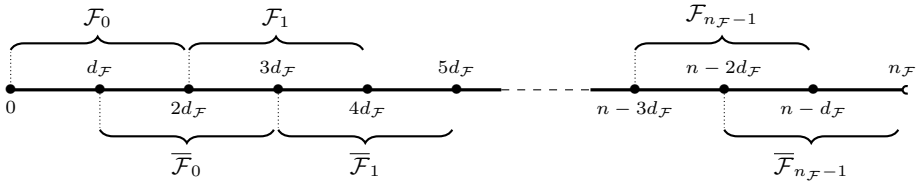

Fig. 2. Fragmentation of $[\![n[\![$ with $n = (2n_{\mathcal{F}} + 1)d_{\mathcal{F}}$

For all $i \in [\![n[\![$, we define $\mathtt{frag}(i), \mathtt{pos}(i), \overline{\mathtt{frag}}(i)$ and $\overline{\mathtt{pos}}(i)$ by

$$\begin{aligned} i &= s_{\mathcal{F}}\mathtt{frag}(i) + \mathtt{pos}(i), \text{ with } 0 \leq \mathtt{pos}(i) < s_{\mathcal{F}} \\ i - L &= s_{\mathcal{F}}\overline{\mathtt{frag}}(i) + \overline{\mathtt{pos}}(i), \text{ with } 0 \leq \overline{\mathtt{pos}}(i) < s_{\mathcal{F}}. \end{aligned}$$

In other words, $(\mathtt{frag}(i), \mathtt{pos}(i))$ is the (quotient, remainder) pair of the euclidean division of i by $s_{\mathcal{F}}$ and so is $(\overline{\mathtt{frag}}(i), \overline{\mathtt{pos}}(i))$ for the division of $i - L$ by $s_{\mathcal{F}}$. Thus, $\mathtt{frag}(i)$ (resp. $\overline{\mathtt{frag}}(i)$) is the index of the fragment that contains i and $\mathtt{pos}(i)$ (resp. $\overline{\mathtt{pos}}(i)$) is the position of i inside $\mathcal{F}_{\mathtt{frag}(i)}$ (resp. $\overline{\mathcal{F}}_{\overline{\mathtt{frag}}(i)}$).

Remarks. A benefit of this fragmentation approach is that one does not need to define a bound on the length of the strings to encrypt. One can indeed encrypt strings of arbitrary length by processing each fragment independently. Conversely, [13] requires to define a maximal length n at the setup phase. Technically, it would be possible in [13] to split the string to encrypt into fragments of size n so as to be able to support strings of any size. Unfortunately, this would harm correctness of the resulting scheme because patterns straddling two fragments would be undetectable. In this respect, the fragmentation approach is perfectly suited to stream encryption.

Another remark is that, with this fragmentation approach, the precise knowledge of n and the number of fragment $n_{\mathcal{F}}$ is not needed in practice to encrypt

the data. Theses values are indeed only necessary for formal definition of our construction so as to correctly index each fragment. As a result one can drop in practice the condition $n = (2n_{\mathcal{F}} + 1)d_{\mathcal{F}}$, and process data as a stream cipher without using dummy characters: one can pause encryption in the middle of a fragment and resume it accordingly. However, for ease of exposition, we will suppose in the following that n is known at encryption time and that $n = (2n_{\mathcal{F}} + 1)d_{\mathcal{F}}$.

4.2 Intuition of our Constructions

As we explain in the introduction, the goal of our paper is twofold: we want to propose a new scheme with a better complexity than the one of [3] but also to rely on a much more reasonable computational assumption. This will be done in two steps. In the first step, we only focus on efficiency and propose a very simple construction that still requires an interactive assumption. In the second step, we show how one can tweak the previous construction to rely on a static assumption without significantly impacting performance.

First Construction. Let us first show how we can simplify the AS^{3}E protocol of Bkakria *et al.* [3] so that the size of the encryption is nearly halved, all other things being equal. In [3], each character m_i is essentially encrypted as $\{C_i, \overline{C}_i, C'_i, \overline{C}'_i\}$ with

- $C_i = (g^{z^{\mathtt{pos}(i)}})^{a_{\mathtt{frag}(i)}}$ and $C'_i = (g^{\alpha'_{m_i}(\alpha_{m_i} z)^{\mathtt{pos}(i)}})^{a_{\mathtt{frag}(i)}}$, where α'_{m_i} and α_{m_i} are secret values representing the character m_i, z is secret and $a_{\mathtt{frag}(i)}$ is a random scalar common to the whole fragment $\mathcal{F}_{\mathtt{frag}(i)}$;
- $\overline{C}_i$ and $\overline{C}'_i$ are generated similarly but for the overlined fragment $\mathcal{F}_{\mathtt{frag}(i)}$ containing i.

This construction is thus clearly reminiscent of [13] where m_i would be encrypted as $C_i = (g^{z^{\mathtt{pos}(i)}})^{a_{\mathtt{frag}(i)}}$ and $C'_i = (g^{\alpha'_{m_i}(z)^{\mathtt{pos}(i)}})^{a_{\mathtt{frag}(i)}}$ if one used fragmentation in the original scheme. However, the use of monomials $(z^{\mathtt{pos}(i)})$ whose degree depends on the position of the character within the stream was necessary in [13] to achieve a specific property, namely the ability to shift trapdoor (that is, a trapdoor can be used at *any* position). As we discuss in the introduction, the fragmentation technique makes this property less interesting. Actually the schemes proposed by Bkakria *et al.* do not achieve this property (they provide a trapdoor for each possible position of the pattern), which questions the interest of keeping the same structure as in [13].

By getting rid of this z element, it is possible to replace, for each fragment $\mathcal{F}_h$, the $s_{\mathcal{F}}$ elements C_i by a single element $C_h = g^{a_h}$ bearing the randomness a_h used for all elements C'_i with $i \in \mathcal{F}_h$ (i.e. $\mathtt{frag}(i) = h$), which roughly halves the size of the ciphertext. We can also simplify this way the elements C'_i by setting $C'_i = (g^{\alpha_{\mathtt{pos}(i),m_i}})^{a_{\mathtt{frag}(i)}}$ where $\alpha_{\mathtt{pos}(i),m_i}$ is a secret scalar encoding both the character m_i and its position $\mathtt{pos}(i)$ within the fragment.

We give the shape of such an encryption for very small fragments. When this technique is used to encrypt a message $M = (m_0, m_1, \ldots, m_{13})$ with fragments of size $s_{\mathcal{F}} = 4$, the sender chooses random elements a_0, a_1, a_2 and $\overline{a}_0, \overline{a}_1, \overline{a}_2$ to encrypt the fragments of M as follows:

$$M = (\overbrace{m_0, m_1, \underbrace{m_2, m_3}_{\overline{a}_0}}^{a_0}, \overbrace{\underbrace{m_4, m_5}_{\overline{a}_1}}^{a_1}, \overbrace{\underbrace{m_6, m_7}_{\overline{a}_1}}^{a_1}, \overbrace{\underbrace{m_8, m_9}_{\overline{a}_2}}^{a_2}, \overbrace{\underbrace{m_{10}, m_{11}}_{\overline{a}_2}}^{a_2}, m_{12}, m_{13}).$$

The resulting ciphertext C is then:

C_0				C_1				C_2					
C'_0	C'_1	C'_2	C'_3	C'_4	C'_5	C'_6	C'_7	C'_8	C'_9	C'_{10}	C'_{11}	`Null`	`Null`
`Null`	`Null`	$\overline{C}'_2$	$\overline{C}'_3$	$\overline{C}'_4$	$\overline{C}'_5$	$\overline{C}'_6$	$\overline{C}'_7$	$\overline{C}'_8$	$\overline{C}'_9$	$\overline{C}'_{10}$	$\overline{C}'_{11}$	$\overline{C}'_{12}$	$\overline{C}'_{13}$
		$\overline{C}_0$		$\overline{C}_1$				$\overline{C}_2$					

Once we have reduced the size of the ciphertext, we focus on the one of the public key, which contained in [3] about $2L(|\mathcal{S}| + 1)$ elements for an alphabet $\mathcal{S}$ of size $|\mathcal{S}|$. As we explain in Section 3, we can associate each character of the alphabet with an element of $\mathbb{Z}_p$. One could then try to set $C'_i = ((g^{\alpha_{\mathsf{pos}(i)}})^{m_i})^{a_{\mathsf{frag}(i)}}$ where $\alpha_{\mathsf{pos}(i)}$ would only encode the position $\mathsf{pos}(i)$ and where $m_i \in \mathbb{Z}_p$ is the character to encrypt, but such a scheme would suffer from malleability. Indeed, by raising C'_i to the power m_j/m_i one could transform a ciphertext encrypting m_i into a ciphertext encrypting m_j and so could use, for example, a legitimate trapdoor for m_j to detect m_i. In other words, a SEPM scheme cannot be secure if it is malleable. Our first construction solves this problem by setting $C'_i = (g^{x_{\mathsf{pos}(i)}}(g^{y_{\mathsf{pos}(i)}})^{m_i})^{a_{\mathsf{frag}(i)}}$ where $x_{\mathsf{pos}(i)}$ and $y_{\mathsf{pos}(i)}$ are secret values specific to the position $\mathsf{pos}(i)$. One can indeed note that C'_i is essentially a PS signature [19] on m_i generated with secret keys $(x_{\mathsf{pos}(i)}, y_{\mathsf{pos}(i)})$. Non-malleability of the ciphertext thus intuitively results from the unforgeability of PS signatures.

In this regard, it seems logical that the security of our first construction relies on a strong computational assumption (PS signatures were essentially proven in the generic group model). Following [3, 13], we indeed prove security under the i-GDH assumption from [13], which is not really satisfactory. The goal of our second construction is to retain as much as possible the core idea (and thus the efficiency) of our new protocol while relying on a more reasonable assumption.

Second Construction. To understand why the previous construction is unlikely to rely on a static assumption, we need to briefly explain how its `Test` procedure works. As we have explained, a ciphertext element C'_i encrypting a character m_i at index i is a group element $g^{a_{\mathsf{frag}(i)}} \in \mathbb{G}_1$ raised to a power $x_{\mathsf{pos}(i)} + m_i y_{\mathsf{pos}(i)}$. By multiplying these C'_i together for $i \in \mathcal{I}$, where $\mathcal{I}$ is a subset of a fragment $\mathcal{F}_h$, we get the $C_h \in \mathbb{G}_1$ element raised to the power $\sum_{i\in\mathcal{I}}(x_{\mathsf{pos}(i)} + m_i y_{\mathsf{pos}(i)})$. By providing a mirror element in $\mathbb{G}_2$, that is, an element $\widetilde{g}^{\sum_{i\in\mathcal{I}}(x_{\mathsf{pos}(i)}+m_i y_{\mathsf{pos}(i)})}$ for some $\widetilde{g} \in \mathbb{G}_2$, we can easily check if the ciphertexts

$\{C'_i\}_{i\in\mathcal{I}}$ encrypt $\{m_i\}_{i\in\mathcal{I}}$ thanks to the bilinearity of the pairing. Of course, there are still several issues to address (we in particular need to prevent trapdoor forgeries) but the core idea remains the same.

The problem we face with such a construction is to deal with any trapdoor query in the security proof. The constraints we place on the $\mathcal{O}\texttt{Issue}$ oracle in Section 3.2 are indeed very mild so we must be able to generate trapdoors for almost all possible patterns. Moreover, as our scheme has public keys, these trapdoors must be valid since the adversary could test them on patterns that it has encrypted itself. Concretely, this means that, in our proof, our simulator must be able to generate the elements $\widetilde{g}^{\sum_{i\in\mathcal{I}}(x_{\mathsf{pos}(i)}+m_i y_{\mathsf{pos}(i)})}$ for almost all possible values of m_i.

Clearly, we would like some static assumption providing each pair $\{\widetilde{g}^{x_{\mathsf{pos}(i)}}, \widetilde{g}^{y_{\mathsf{pos}(i)}}\}$ separately. Unfortunately, this cannot work in our case. Indeed, the proof uses a standard hybrid strategy where, at each step, the element $C'_{i^*} = (g^{x_{\mathsf{pos}(i^*)}}(g^{y_{\mathsf{pos}(i^*)}})^{m_i})^{a_{\mathsf{frag}(i^*)}}$ encrypting the i^*-th character is replaced by a random element. Given $\{\widetilde{g}^{x_{\mathsf{pos}(i^*)}}, \widetilde{g}^{y_{\mathsf{pos}(i^*)}}\}$, one could trivially detect this substitution because the ciphertext also contains $g^{a_{\mathsf{frag}(i^*)}}$. This is why our first construction, along with [3, 13], uses the i-GDH assumption that is tailored to this kind of schemes. This interactive assumption indeed provides an oracle that can answer any trapdoor query by providing exactly the requested element $\widetilde{g}^{\sum_{i\in\mathcal{I}}(x_{\mathsf{pos}(i)}+m_i y_{\mathsf{pos}(i)})}$. This way, the simulation is perfect without having to worry about how these elements are computed concretely.

As the pair $\{\widetilde{g}^{x_{\mathsf{pos}(i^*)}}, \widetilde{g}^{y_{\mathsf{pos}(i^*)}}\}$ must remain unknown, a better strategy is to generate the pairs $\{\widetilde{g}^{x_{\mathsf{pos}(i)}}, \widetilde{g}^{y_{\mathsf{pos}(i)}}\}$, for $i \neq i^*$, in such a way that the sum $\sum_{i\in\mathcal{I}}(x_{\mathsf{pos}(i)} + m_i y_{\mathsf{pos}(i)})$ can be computed without the knowledge of $x_{\mathsf{pos}(i^*)}$ and $y_{\mathsf{pos}(i^*)}$. More concretely, this means that the pairs $(x_{\mathsf{pos}(i)}, y_{\mathsf{pos}(i)})$, for $i \neq i^*$, must be able to cancel $(x_{\mathsf{pos}(i^*)}, y_{\mathsf{pos}(i^*)})$ and so should be generated from the same secret value (let us call it A) defining an instance of the computational problem we have to solve. Unfortunately, here again, we pay the price of the strong security model we consider in Section 3.2.

Indeed, as we allow the adversary to query trapdoors for patterns matching the challenge ciphertext (contrarily to, *e.g.*, [20]), all the ciphertext elements, except C'_{i^*}, must be well formed. This means that it should be possible to essentially compute $g^{Aa_{\mathsf{frag}(i^*)}}$ to generate C'_i, for $i \neq i^*$ but, in the meantime, it should be impossible to distinguish $g^{Aa_{\mathsf{frag}(i^*)}}$ from randomness to ensure the validity of our hybrid argument in position i^*.

To address this problem, without weakening our security model, we choose to slightly modify our trapdoors by randomizing them with two different random values s_1 and s_2. Concretely, our trapdoors will be of the form

$$\widetilde{g}^{\sum_{i\in\mathcal{I}}[s_1(x_{\mathsf{pos}(i)}+m_i y_{\mathsf{pos}(i)})+s_2 z_{\mathsf{pos}(i)}]},$$

for some new scalars $z_{\mathsf{pos}(i)}$ that will be defined by our public key. The only price to pay is an increase in the size of the ciphertext that must now contain two elements per position to match these two random values.

Intuitively, these two scalars will provide enough flexibility to cancel the elements $x_{\mathsf{pos}(i^*)}$ and $y_{\mathsf{pos}(i^*)}$ without falling back on the previous problem. More

specifically, they will allow us to consider a slightly more complex computational problem where $A = ab$, for some secret a and b, which allows us to construct $(x_{\mathsf{pos}(i)}, y_{\mathsf{pos}(i)})$ from a or b but not $A = ab$. This way, the challenge ciphertext can be simulated without making the underlying computational problem trivial. Moreover, the latter (called EXDH assumption, see Section 2) remains a simple variant of the DDH assumption, which gives more confidence in its hardness, in particular because it was already used in a previous paper [11] to design an e-cash system.

In the end, our second construction manages to be proven under a static assumption at the cost of a small increase in the ciphertext and trapdoors sizes, compared to our first contribution. We believe this is a significant improvement over the state-of-the-art [3, 13] that required a tailored assumption.

4.3 Our First Protocol

- $\mathtt{Setup}(1^\lambda, L)$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$ be the description of type 3 bilinear groups. This algorithm selects $g \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\}$, $\widetilde{g} \in \mathbb{G}_2 \backslash \{1_{\mathbb{G}_2}\}$ and returns as public parameters $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \widetilde{g}, d_{\mathcal{F}} := L - 1, s_{\mathcal{F}} := 2d_{\mathcal{F}})$.

- $\mathtt{Keygen}()$: This algorithm chooses $x_k, y_k \xleftarrow{\$} \mathbb{Z}_p$ for all $k \in [\![s_{\mathcal{F}}]\!]$ and returns $\mathsf{sk} := \{(x_k, y_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}$ and $\mathsf{pk} := \{(g^{x_k}, g^{y_k})\}_{k \in [\![s_{\mathcal{F}}]\!]}$.

- $\mathtt{Encrypt}(M, \mathsf{pk})$: This algorithm parses M as $(m_0, \ldots, m_{n-1}) \in \mathbb{Z}_p^n$ and pk as $\{(X_k, Y_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}$, selects $a_h, \overline{a}_h \xleftarrow{\$} \mathbb{Z}_p$ for all $h \in [\![n_{\mathcal{F}}]\!]$, where $n_{\mathcal{F}}$ is defined as in Section 4.1, *i.e.*, $n = (2n_{\mathcal{F}} + 1)d_{\mathcal{F}}$, and returns the ciphertext $C := \{\{C_h, \overline{C}_h\}_{h \in [\![n_{\mathcal{F}}]\!]}, \{(C'_i, \overline{C}'_i)\}_{i \in [\![n]\!]}\}$ generated as follows:

$C_h := g^{a_h}$, for $h \in [\![n_{\mathcal{F}}]\!]$	$\overline{C}_h := g^{\overline{a}_h}$, for $h \in [\![n_{\mathcal{F}}]\!]$
For $i \in [\![n - d_{\mathcal{F}}]\!]$ **:**	**For** $i \in [\![d_{\mathcal{F}}, n]\!]$ **:**
$C'_i := (X_{\mathsf{pos}(i)}(Y_{\mathsf{pos}(i)})^{m_i})^{a_{\mathsf{frag}(i)}}$	$\overline{C}'_i := (X_{\overline{\mathsf{pos}}(i)}(Y_{\overline{\mathsf{pos}}(i)})^{m_i})^{\overline{a}_{\overline{\mathsf{frag}}(i)}}$
For $i \in [\![n - d_{\mathcal{F}}, n]\!]$ **:**	**For** $i \in [\![d_{\mathcal{F}}]\!]$ **:**
$C'_i := \mathtt{Null}$	$\overline{C}'_i := \mathtt{Null}$

- $\mathtt{Issue}(W, \mathsf{sk})$: On $W = (w_0, \ldots, w_{\ell-1}) \in (\mathbb{Z}_p \cup \{\star\})^\ell$, $\mathsf{sk} = \{(x_k, y_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}$, $\ell \leq L$, it runs:

  For $\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]$**:**

  $s \xleftarrow{\$} \mathbb{Z}_p$, $\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_{\mathcal{F}}-1}) := (\overbrace{\star, \ldots, \star}^{\delta}, w_0, \ldots, w_{\ell-1}, \overbrace{\star, \ldots, \star}^{s_{\mathcal{F}}-\ell-\delta})$

  $S := s \sum_{k \in \mathsf{supp}(\widehat{W})} (x_k + y_k \widehat{w}_k)$, $\mathsf{td}_{W,\delta} := \{\widetilde{g}^s, \widetilde{g}^S\}$

  Return $\mathsf{TD}_W := \{\mathsf{td}_{W,\delta}\}_{\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]}$

- $\mathtt{Test}(C, W, \mathsf{TD}_W)$: This algorithm uses $\mathsf{TD}_W = \{\mathsf{td}_{W,\delta}\}_{\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]}$ to test whether the string $W \in (\mathbb{Z}_p \cup \{\star\})^\ell$ matches the message M encrypted by C as follows:

$\mathsf{Match} := \emptyset$
For $i \in [\![n-\ell]\!]$**:**
  If $[\![i, i+\ell[\![\subset \mathcal{F}_{\mathtt{frag}(i)}$**:**
    Get the trapdoor element $\mathsf{td}_{W,\mathtt{pos}(i)} = \{T_1, T_2\}$ from TD_W
    If $e\left(\prod_{k \in \mathsf{supp}(W)} C'_{i+k}\ ,\ T_1\right) = e\big(C_{\mathtt{frag}(i)}, T_2\big)$**:**
      $\mathsf{Match} := \mathsf{Match} \cup \{i\}$
  Else: `#now we know that` $[\![i, i+\ell[\![\subset \overline{\mathcal{F}}_{\overline{\mathtt{frag}}(i)}$
    Get the trapdoor $\mathsf{td}_{W,\overline{\mathtt{pos}}(i)} = \{T_1, T_2\}$ from TD_W ;
    If $e\left(\prod_{k \in \mathsf{supp}(W)} \overline{C}'_{i+k}\ ,\ T_1\right) = e\Big(\overline{C}_{\overline{\mathtt{frag}}(i)}, T_2\Big)$**:**
      $\mathsf{Match} := \mathsf{Match} \cup \{i\}$
Return Match

Correctness. We first show that if M contains a pattern W at position i, then i is necessarily contained in the subset returned by $\mathtt{Test}(C, W, \mathsf{TD}_W)$. Here, we assume that $i \in [\![n-\ell[\![$ is such that $[\![i, i+\ell[\![\subset \mathcal{F}_{\mathtt{frag}(i)}$. Otherwise, we would have $[\![i, i+\ell[\![\subset \overline{\mathcal{F}}_{\overline{\mathtt{frag}}(i)}$ and adapting the following argument to this case would be straightforward.
The `Issue` algorithm ensures that, at some point, a trapdoor element $\mathsf{td}_{W,\mathtt{pos}(i)} = \{T_1, T_2\}$ was generated for

$$\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_{\mathcal{F}}-1}) := (\overbrace{\star, \ldots, \star}^{\mathtt{pos}(i)}, w_0, \ldots, w_{\ell-1}, \overbrace{\star, \ldots, \star}^{s_{\mathcal{F}}-\ell-\mathtt{pos}(i)}).$$

To show that the index i is added by `Test` in Match, we must show that the pairing equation is satisfied. By non-degeneracy of the pairing, this is equivalent to showing that the following equation on the exponents of $e(g, \widetilde{g})$ holds:

$$s \sum_{k \in \mathsf{supp}(W)} a_{\mathtt{frag}(i+k)}(x_{\mathtt{pos}(i+k)} + y_{\mathtt{pos}(i+k)} m_{i+k}) = a_{\mathtt{frag}(i)} s \sum_{k \in \mathsf{supp}(\widehat{W})} x_k + y_k \widehat{w}_k$$

As $[\![i, i+\ell[\![\subset \mathcal{F}_{\mathtt{frag}(i)}$, we have for all $k \in \mathsf{supp}(W)$, $\mathtt{frag}(i+k) = \mathtt{frag}(i)$ and $\mathtt{pos}(i+k) = \mathtt{pos}(i)+k$. Thus after simplification, we have to show the equivalent equation:

$$\sum_{k \in \mathsf{supp}(W)} x_{\mathtt{pos}(i)+k} + y_{\mathtt{pos}(i)+k} m_{i+k} = \sum_{k \in \mathsf{supp}(\widehat{W})} x_k + y_k \widehat{w}_k. \tag{1}$$

Formally, the fact that W is contained at index $i \in [\![n-\ell[\![$ in M means that $m_{i+k} = w_k$ for all $k \in \mathsf{supp}(W)$. Hence the LHS of (1) is equal to

$$\sum_{k \in \mathsf{supp}(W)} x_{\mathtt{pos}(i)+k} + y_{\mathtt{pos}(i)+k} w_k.$$

As $\widehat{w}_{\mathsf{pos}(i)+k} = w_k$ for all $k \in \mathsf{supp}(W)$, we get that this sum equals

$$\sum_{k \in \mathsf{supp}(W)} x_{\mathsf{pos}(i)+k} + y_{\mathsf{pos}(i)+k} \widehat{w}_{\mathsf{pos}(i)+k}.$$

Finally, we note that $\mathsf{supp}(\widehat{W}) = \{\mathsf{pos}(i) + k\}_{k \in \mathsf{supp}(W)}$. We can then re-index the sum above to get

$$\sum_{k \in \mathsf{supp}(\widehat{W})} x_k + y_k \widehat{w}_k,$$

which proves (1). Thus the pairing equality holds and `Test` returns a set containing i. In other words, there is no false negative in our system.

Now, let us assume that W is *not* contained in M at position i. If `Test` returns a set containing i, then the reasoning above implies that we would have:

$$\sum_{k \in \mathsf{supp}(W)} x_{\mathsf{pos}(i)+k} + y_{\mathsf{pos}(i)+k} m_{i+k} = \sum_{k \in \mathsf{supp}(W)} x_{\mathsf{pos}(i)+k} + y_{\mathsf{pos}(i)+k} w_k,$$

which means that:

$$\sum_{k \in \mathsf{supp}(W)} y_{\mathsf{pos}(i)+k}(m_{i+k} - w_k) = \sum_{\substack{k \in \mathsf{supp}(W) \\ m_{i+k} \neq w_k}} y_{\mathsf{pos}(i)+k}(m_{i+k} - w_k) = 0. \tag{2}$$

Since M does not contain W at position i, there exists at least one $k \in \mathsf{supp}(W)$ such that $w_k \neq m_{i+k}$ so the last sum above is not empty. As the $\{y_k\}_{k \in [\![s_{\mathcal{F}} [\![}$ are chosen uniformly at random independently of M and W, equation (2) holds with negligible probability $1/p$ (the probability that a non-zero linear form evaluates to 0). This means that we can also dismiss the occurrence of false positives.

Note that one could consider a stronger model of correctness, where an adversary intends to bypass the detection system. In this case, as the public key contains the g^{y_k}'s, the adversary gains access to some information on the y_k's which are thus not independent of M and W and the above reasoning fails. However, one could easily transform an adversary managing to find a message M and a pattern W such that equation (2) holds, into an algorithm that solve the discrete logarithm problem. As a result, we will have this stronger notion of correctness under the discrete logarithm assumption in $\mathbb{G}_1$.

4.4 Our Second Protocol

- `Setup`$(1^\lambda, L)$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$ be the description of type 3 bilinear groups. This algorithm selects $g \in \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\}$, $\widetilde{g} \in \mathbb{G}_2 \backslash \{1_{\mathbb{G}_2}\}$ and returns as public parameters $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \widetilde{g}, d_{\mathcal{F}} := L - 1, s_{\mathcal{F}} := 2d_{\mathcal{F}})$.

- `Keygen`(): This algorithm chooses $x_k, y_k, z_k \xleftarrow{\$} \mathbb{Z}_p$ for all $k \in [\![s_{\mathcal{F}} [\![$ and returns $\mathsf{sk} = \{(x_k, y_k, z_k)\}_{k \in [\![s_{\mathcal{F}} [\![}$ and $\mathsf{pk} = \{(g^{x_k}, g^{y_k}, g^{z_k})\}_{k \in [\![s_{\mathcal{F}} [\![}$.

- $\texttt{Encrypt}(M, \mathsf{pk})$: This algorithm parses M as $(m_0, \ldots, m_{n-1}) \in \mathbb{Z}_p^n$ and pk as $\{(X_k, Y_k, Z_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}$, selects $a_h, \overline{a}_h \xleftarrow{\$} \mathbb{Z}_p$ for all $h \in [\![n_{\mathcal{F}}]\!]$, where $n_{\mathcal{F}}$ is defined as in Section 4.1, *i.e.*, $n = (2n_{\mathcal{F}} + 1)d_{\mathcal{F}}$, and returns the ciphertext $C = \{\{C_h, \overline{C}_h\}_{h \in [\![n_{\mathcal{F}}]\!]}, \{(C'_{i,1}, C'_{i,2}, \overline{C}'_{i,1}, \overline{C}'_{i,2})\}_{i \in [\![n]\!]}\}$ generated as follows:

  $C_h = g^{a_h}$, for $h \in [\![n_{\mathcal{F}}]\!]$
  For $i \in [\![n - d_{\mathcal{F}}]\!]$ **:**
    $C'_{i,1} = (X_{\mathsf{pos}(i)} (Y_{\mathsf{pos}(i)})^{m_i})^{a_{\mathsf{frag}(i)}}$
    $C'_{i,2} = (Z_{\mathsf{pos}(i)})^{a_{\mathsf{frag}(i)}}$
  For $i \in [\![n - d_{\mathcal{F}}, n]\!]$ **:**
    $C'_{i,1} = C'_{i,2} = \texttt{Null}$

  $\overline{C}_h = g^{\overline{a}_h}$, for $h \in [\![n_{\mathcal{F}}]\!]$
  For $i \in [\![d_{\mathcal{F}}, n]\!]$ **:**
    $\overline{C}'_{i,1} = (X_{\overline{\mathsf{pos}}(i)} (Y_{\overline{\mathsf{pos}}(i)})^{m_i})^{\overline{a}_{\overline{\mathsf{frag}}(i)}}$
    $\overline{C}'_{i,2} = (Z_{\overline{\mathsf{pos}}(i)})^{\overline{a}_{\overline{\mathsf{frag}}(i)}}$
  For $i \in [\![d_{\mathcal{F}}]\!]$ **:**
    $\overline{C}'_{i,1} = \overline{C}'_{i,2} = \texttt{Null}$

- $\texttt{Issue}(W, \mathsf{sk})$: On $W = (w_0, \ldots, w_{\ell-1}) \in (\mathbb{Z}_p \cup \{\star\})^{\ell}$, $\mathsf{sk} = \{(x_k, y_k, z_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}$, $\ell \leq L$, it runs:

  For $\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]$**:**
    $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$, $\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_{\mathcal{F}}-1}) := (\overbrace{\star, \ldots, \star}^{\delta}, w_0, \ldots, w_{\ell-1}, \overbrace{\star, \ldots, \star}^{s_{\mathcal{F}} - \ell - \delta})$
    $S = s_1 \sum_{k \in \mathsf{supp}(\widehat{W})} [x_k + y_k \widehat{w}_k] + s_2 \sum_{k \in \mathsf{supp}(\widehat{W})} z_k$, $\mathsf{td}_{W,\delta} = \{\widetilde{g}^{s_1}, \widetilde{g}^{s_2}, \widetilde{g}^{S}\}$
  Return $\mathsf{TD}_W = \{\mathsf{td}_{W,\delta}\}_{\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]}$

- $\texttt{Test}(C, W, \mathsf{TD}_W)$: This algorithm uses $\mathsf{TD}_W = \{\mathsf{td}_{W,\delta}\}_{\delta \in [\![s_{\mathcal{F}} - \ell + 1]\!]}$ to test whether the string $W \in (\mathbb{Z}_p \cup \{\star\})^{\ell}$ matches the message M encrypted by C as follows:

  $\mathsf{Match} := \emptyset$
  For $i \in [\![n - \ell]\!]$**:**
    If $[\![i, i + \ell [\![\subset \mathcal{F}_{\mathsf{frag}(i)}$**:**
      Get the trapdoor element $\mathsf{td}_{W,\mathsf{pos}(i)} = \{T_1, T_2, T_3\}$ from TD_W
      If $e\left(\prod_{k \in \mathsf{supp}(W)} C'_{i+k,1} \ , \ T_1\right) \cdot e\left(\prod_{k \in \mathsf{supp}(W)} C'_{i+k,2} \ , \ T_2\right) = e\big(C_{\mathsf{frag}(i)}, T_3\big)$**:**
        $\mathsf{Match} = \mathsf{Match} \cup \{i\}$
    Else: `#now we know that` $[\![i, i + \ell [\![\subset \overline{\mathcal{F}}_{\mathsf{frag}(i)}$
      Get the trapdoor $\mathsf{td}_{W,\overline{\mathsf{pos}}(i)} = \{T_1, T_2, T_3\}$ from TD_W ;
      If $e\left(\prod_{k \in \mathsf{supp}(W)} \overline{C}'_{i+k,1} \ , \ T_1\right) \cdot e\left(\prod_{k \in \mathsf{supp}(W)} \overline{C}'_{i+k,2} \ , \ T_2\right) = e\Big(\overline{C}_{\overline{\mathsf{frag}}(i)}, T_3\Big)$**:**
        $\mathsf{Match} = \mathsf{Match} \cup \{i\}$
  Return Match

The correctness of this protocol is similar to the one of the first protocol.

5 Security Analysis

The security of our protocols is stated by the following theorem, proved in this section.

Theorem 1.

- *The scheme described in section 4.3 is* sIND-CPA *secure under the* i-GDH *assumption.*
- *The scheme described in section 4.4 is* sIND-CPA *secure under the* EXDH *assumption.*

5.1 Proof Strategy

The proof of the theorem above follows the same strategy for both protocols but will rely on very different arguments according to the construction. Let $M^{(0)} = (m_0^{(0)}, \ldots, m_{n-1}^{(0)})$ and $M^{(1)} = (m_0^{(1)}, \ldots, m_{n-1}^{(1)})$ be the two strings returned by $\mathcal{A}$ at the beginning of the game. Our proof uses a sequence of games to argue that the advantage of $\mathcal{A}$ is negligible. This is a standard hybrid argument, in which at each game hop we randomize another element of the challenge ciphertext. However, due to the peculiarities of the fragmentation technique, we will have to consider the following two sets:

$$\mathcal{I}_{\neq} = \{i \in [\![n - d_{\mathcal{F}}[\![: m_i^{(0)} \neq m_i^{(1)}\} \text{ and } \overline{\mathcal{I}}_{\neq} = \{i \in [\![d_{\mathcal{F}}, n[\![: m_i^{(0)} \neq m_i^{(1)}\}.$$

In this proof we will denote the elements of $\mathcal{I}_{\neq}$ (resp. $\overline{\mathcal{I}}_{\neq}$) as $\{i_1, \ldots, i_{|\mathcal{I}_{\neq}|}\}$ (resp. $\{i'_1, \ldots, i'_{|\overline{\mathcal{I}}_{\neq}|}\}$). For $j = 1, \ldots, |\mathcal{I}_{\neq}|$, we let $\mathcal{I}_{\neq}^{(j)} = \{i_1, \ldots, i_j\}$ and $\mathcal{I}_{\neq}^{(0)} = \emptyset$. We define $\overline{\mathcal{I}}_{\neq}^{(j)}$ similarly. Finally, to harmonize the proofs of our protocols, we will introduce the notation $\mathbf{C}'_i$, for $i \in [\![n - d_{\mathcal{F}}[\![$, where:

- $\mathbf{C}'_i = [C'_i]$ in our first protocol;
- $\mathbf{C}'_i = [C'_{i,1}, C'_{i,2}]$ in our second protocol.

This way, we can refer to the first element of $\mathbf{C}'_i$ as $\mathbf{C}'_i[1]$. We define similarly $\overline{\mathbf{C}}'_i$. We can now define the following sequence of games.

- $\mathbf{game}_{0,1}$ denotes the $\mathtt{Exp}_{\mathcal{A}}^{sind-cpa}$ game, as described in algorithm 3.2;
- for $j = 1, \ldots, |\mathcal{I}_{\neq}|$:
  - $\mathbf{game}_{j-1,2}$, which is the same game as $\mathbf{game}_{j-1,1}$ except that, for the second protocol, $\mathbf{C}'_{i_j}[2]$ is replaced by a random element of $\mathbb{G}_1$;
  - $\mathbf{game}_{j,1}$, which is the same game as $\mathbf{game}_{j-1,1}$ except that *all elements* of $\mathbf{C}'_{i_j}$ are replaced by random elements of $\mathbb{G}_1$.
- for $j = |\mathcal{I}_{\neq}| + 1, \ldots, |\mathcal{I}_{\neq}| + |\overline{\mathcal{I}}_{\neq}|$:
  - $\mathbf{game}_{j-1,2}$, which is the same game as $\mathbf{game}_{j-1,1}$ except that $\overline{\mathbf{C}}'_{i'_{j-|\mathcal{I}_{\neq}|}}[2]$ is replaced by a random element of $\mathbb{G}_1$;

- $\mathbf{game}_{j,1}$, which is the same game as $\mathbf{game}_{j-1,1}$ except that *all elements* of $\overline{\mathbf{C}}'_{i'_{j-|\mathcal{I}_{\neq}|}}$ are replaced by random elements of $\mathbb{G}_1$.

In the case of our first protocol, one can note that $\mathbf{game}_{j,1}$ and $\mathbf{game}_{j,2}$ are the same.

Let $\mathbb{S}_j$ be the probability of success of $\mathcal{A}$ in $\mathbf{game}_{j,1}$. We can write :

$$\begin{aligned}\mathtt{Adv}^{sind-cpa}_{\mathcal{A}}(1^\lambda, n) &= \left|\Pr[\mathtt{Exp}^{sind-cpa}_{\mathcal{A}}(1^\lambda, n) = 1] - \frac{1}{2}\right| \\ &\leq \sum_{j=1}^{|\mathcal{I}_{\neq}|+|\overline{\mathcal{I}}_{\neq}|} \left|\mathbb{S}_{j,1} - \mathbb{S}_{j-1,1}\right| + \left|\mathbb{S}_{|\mathcal{I}_{\neq}|+|\overline{\mathcal{I}}_{\neq}|,1} - \frac{1}{2}\right|\end{aligned}$$

Ultimately, in the last game, the challenge ciphertext contains no information about $m_i^{(\beta)}$, for all i such that $m_i^{(0)} \neq m_i^{(1)}$. Thus, an adversary playing this game can only succeed with probability $\frac{1}{2}$ and we then have $|\mathbb{S}_{|\mathcal{I}_{\neq}|+|\overline{\mathcal{I}}_{\neq}|,1} - \frac{1}{2}| = 0$.

We conclude this proof using the following theorems.

Theorem 2. *For our first construction,* $|\mathbb{S}_{j,1} - \mathbb{S}_{j-1,1}|$ *is negligible under the* i-GDH *assumption, for all* $j \in \{1, \ldots, |\mathcal{I}_{\neq}| + |\overline{\mathcal{I}}_{\neq}|\}$.

Theorem 3. *For our second construction,* $|\mathbb{S}_{j,1} - \mathbb{S}_{j-1,1}|$ *is negligible under the* EXDH *assumption, for all* $j \in \{1, \ldots, |\mathcal{I}_{\neq}| + |\overline{\mathcal{I}}_{\neq}|\}$.

We only give proofs of these theorems for $j = 1, \ldots, |\mathcal{I}_{\neq}|$ as these proofs readily extend to the cases $j = |\mathcal{I}_{\neq}| + 1, \ldots, |\mathcal{I}_{\neq}| + |\overline{\mathcal{I}}_{\neq}|$.

In these proofs, to simplify notations, we let $i^* := i_j$ be the j-th index of $\mathcal{I}_{\neq}$, and we let $\widehat{M} = (\widehat{m}_0, \ldots, \widehat{m}_{s_{\mathcal{F}}-1})$ be the substring of $M^{(\beta)}$ corresponding to the fragment containing i^*.

5.2 Proof of Theorem 2

In our simulation, we set an upper bound q on the number of trapdoor queries that the adversary is allowed to make. The i-GDH instance from which we make our reduction has $c = 2n_{\mathcal{F}} + (q+2)s_{\mathcal{F}}$ variables called

$$\{\{a_h, \overline{a}_h\}_{h \in [\![n_{\mathcal{F}}]\!]}, \{(x_k, y_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}, \{s_t\}_{t \in [\![q s_{\mathcal{F}}]\!]}\}$$

and a secret evaluation $(\chi_1, \ldots, \chi_c) \xleftarrow{\$} \mathbb{Z}_p^c$ of these variables.

Initially, $\mathtt{R} = \{\{a_h, \overline{a}_h\}_{h \in [\![n_{\mathcal{F}}]\!]}, \{(x_k, y_k)\}_{k \in [\![s_{\mathcal{F}}]\!]}\}, \mathtt{S}, \mathtt{T} = \emptyset$ and

$$f = a_{\mathtt{frag}(i^*)}(x_{\mathtt{pos}(i^*)} + y_{\mathtt{pos}(i^*)}\widehat{m}_{\mathtt{pos}(i^*)}).$$

The simulator has oracle access to $\mathcal{O}^{\mathtt{R}}, \mathcal{O}^{\mathtt{S}}$ and $\mathcal{O}^{\mathtt{T}}$ to add $\kappa = 2$ polynomials at a time to these sets. At any moment, the simulator knows the elements in the current set $\{g^{R(\chi_1,\ldots,\chi_c)}, \widetilde{g}^{S(\chi_1,\ldots,\chi_c)}, e(g,\widetilde{g})^{T(\chi_1,\ldots,\chi_c)}\}_{R \in \mathtt{R}, S \in \mathtt{S}, T \in \mathtt{T}}$.
For some polynomial P we say

the simulator uses $\mathcal{O}^{\mathtt{R}}$ to get $g^{P(\chi_1,\ldots,\chi_c)}$

to say that it uses $\mathcal{O}^{\mathtt{R}}$ to add the polynomial P to $\mathtt{R}$ and so now it knows $g^{P(\chi_1,\ldots,\chi_c)}$ (resp. $\widetilde{g}^{P(\chi_1,\ldots,\chi_c)}$).
Likewise, for some polynomials P, Q we say

the simulator uses $\mathcal{O}^{\mathtt{S}}$ to get $\widetilde{g}^{P(\chi_1,\ldots,\chi_c)}$ and $\widetilde{g}^{Q(\chi_1,\ldots,\chi_c)}$

to say that it uses $\mathcal{O}^{\mathtt{S}}$ to add the polynomials $\{P, Q\}$ to $\mathtt{S}$ so now it knows $g^{P(\chi_1,\ldots,\chi_c)}$ and $\widetilde{g}^{Q(\chi_1,\ldots,\chi_c)}$).
In the description of our simulator, we use the names of a variable $a_h, \overline{a}_h, x_k, y_k$ or s_t for its secret random evaluation χ_j by abuse of notation while in the proof of independency we really consider them as variables.
Finally, the simulator knows the i-GDH challenge ζ.

Key Generation. The simulator implicitly defines the secret key as $\mathsf{sk} = \{(x_k, y_k)\}_{k\in[\![s_{\mathcal{F}}[\![}$ by setting the public key to $\mathsf{pk} = \{(g^{x_k}, g^{y_k})\}_{k\in[\![s_{\mathcal{F}}[\![}$ as the polynomials x_k, y_k are initially in $\mathtt{R}$.

Trapdoor Generation. The adversary can make at most q trapdoor queries to our simulator. To generate a trapdoor TD, the simulator has to generate at most $s_{\mathcal{F}}$ trapdoor elements td. Let $\widehat{W}$ be the fragment-sized pattern corresponding to the t-th trapdoor element td for some $t \leq s_{\mathcal{F}}q$. The simulator uses $\mathcal{O}^{\mathtt{S}}$ to get $\widetilde{g}^{s_t}$ and $\widetilde{g}^{S_t}$ where

$$S_t = s_t \sum_{k\in\mathsf{supp}(\widehat{W})} (x_k + y_k\widehat{w}_k)$$

and sets $\mathsf{td} = \{\widetilde{g}^{s_t}, \widetilde{g}^{S_t}\}$.

Challenge Generation. The simulator sets the challenge cyphertext as follows:

- $C_h = g^{a_h}$ and $\overline{C}_h = g^{\overline{a}_h}$ for $h \in [\![n_{\mathcal{F}}[\![$ as the polynomials $a_h, \overline{a}_h$ are initially in $\mathtt{R}$.
- it uses $\mathcal{O}^{\mathtt{R}}$ to get valid $C'_i = g^{a_{\mathtt{frag}(i)}(x_{\mathtt{pos}(i)}+y_{\mathtt{pos}(i)}m_i^{(\beta)})}$ for $i \notin \mathcal{I}_{\neq}^{(j)}$
- $C'_i \xleftarrow{\$} \mathbb{G}_1$ for $i \in \mathcal{I}_{\neq}^{(j-1)}$ and $C'_{i^*} = \zeta$
- it uses $\mathcal{O}^{\mathtt{R}}$ to get valid $\overline{C}'_i = g^{\overline{a}_{\overline{\mathtt{frag}}(i)}(x_{\overline{\mathtt{pos}}(i)}+y_{\overline{\mathtt{pos}}(i)}m_i^{(\beta)})}$ for $i \in [\![d_{\mathcal{F}}, n[\![$.

If $\zeta = g^f$, then C'_{i^*} is a valid element and the simulator is playing $\mathbf{game}_{j-1,1}$. Else, C'_{i^*} is a random element from $\mathbb{G}_1$ and the simulator is playing game $\mathbf{game}_{j,1}$. An adversary able to distinguish $\mathbf{game}_{j-1,1}$ from $\mathbf{game}_{j,1}$ is thus able to break the i-GDH assumption if the polynomial $f = a_{\mathtt{frag}(i^*)}(x_{\mathtt{pos}(i^*)} + y_{\mathtt{pos}(i^*)}\widehat{m}_{\mathtt{pos}(i^*)})$ is independent from the sets $\mathtt{R}, \mathtt{S}$ and $\mathtt{T}$ (after all the queries made by the simulator), which remains to prove.

Proof of Independence. This is done by showing that

$$a_{\mathtt{frag}(i^*)}(x_{\mathtt{pos}(i^*)} + y_{\mathtt{pos}(i^*)}\widehat{m}_{\mathtt{pos}(i^*)}) \sum_j b_j S^{(j)} = \sum_{i,j} c_{i,j} R^{(i)} S^{(j)} + \sum_k d_k T^{(k)}$$

implies $b_j = 0$ for $j = 0, \ldots, |\mathtt{S}| - 1$.

Since $\mathtt{T} = \emptyset$, we may already remove the last sum. Since the factor $a_{\mathtt{frag}(i^*)}$ only appears in the set $\mathtt{R}$ and more specifically as the $\mathtt{frag}(i^*)$-th element of the initial set $\{a_h\}_{h \in n_{\mathcal{F}}}$ and in the outputs of $\mathcal{O}^{\mathtt{R}}$, we can discard the other terms in the right hand side of the equation (and divide each member by $a_{\mathtt{frag}(i^*)}$). We reformulate the remaining coefficients as $b_{\mathtt{pos}(i^*),t}, b'_{\mathtt{pos}(i^*),t}, c_t, c'_t, b_{k,t}$ and $b'_{k,t}$ for $k \in [\![s_{\mathcal{F}} [\![\backslash \{\mathtt{pos}(i^*)\}$ and $1 \leq t \leq q_{\mathtt{S}}$ so the previous equality can be written as:

$$(x_{\mathtt{pos}(i^*)} + y_{\mathtt{pos}(i^*)}\widehat{m}_{\mathtt{pos}(i^*)}) \sum_{t=1}^{q_{\mathtt{S}}} \left[b_{\mathtt{pos}(i^*),t} s_t + b'_{\mathtt{pos}(i^*),t} S_t \right] =$$

$$= \sum_{t=1}^{q_{\mathtt{S}}} \left[c_t s_t + c'_t S_t \right] - \sum_{\substack{k=0 \\ k \neq \mathtt{pos}(i^*)}}^{s_{\mathcal{F}}-1} \left[(x_k + y_k \widehat{m}_k) \sum_{t=1}^{q_{\mathtt{S}}} \left[b_{k,t} s_t + b'_{k,t} S_t \right] \right].$$

This equation can also be written as:

$$\sum_{k=0}^{s_{\mathcal{F}}-1} \left[(x_k + y_k \widehat{m}_k) \sum_{t=1}^{q_{\mathtt{S}}} \left[b_{k,t} s_t + b'_{k,t} S_t \right] \right] = \sum_{t=1}^{q_{\mathtt{S}}} \left[c_t s_t + c'_t S_t \right]$$

and we show that if it holds, then $b_{\mathtt{pos}(i^*),t} = b'_{\mathtt{pos}(i^*),t} = 0$ for $t = 1, \ldots, q_{\mathtt{S}}$. Let's fix $1 \leq t \leq q_{\mathtt{S}}$. If we only keep the terms in s_t, we get :

$$\sum_{k=0}^{s_{\mathcal{F}}-1} \left[(x_k + y_k \widehat{m}_k)(b_{k,t} s_t + b'_{k,t} S_t) \right] = c_t s_t + c'_t S_t. \tag{3}$$

-We show that $\boxed{b'_{\mathtt{pos}(i^*),t} = 0}$: Keeping only the terms in equation (3) with total degree 2 in $\{x_k\}_{k \in [\![s_{\mathcal{F}} [\![}$ shows that:

$$\sum_{k=0}^{s_{\mathcal{F}}-1} x_k b'_{k,t} S_t = 0.$$

Simplifying by S_t in this equality shows that $\sum_{k=0}^{s_{\mathcal{F}}-1} x_k b'_{k,t} = 0$ and by independance of the variables $\{x_k\}_{k \in [\![s_{\mathcal{F}} [\![}$, we have $b'_{k,t} = 0$ for all $k \in [\![s_{\mathcal{F}} [\![$ and in particular, $b'_{\mathtt{pos}(i^*),t} = 0$.

-We show that $\boxed{b_{\mathtt{pos}(i^*),t} = 0}$: If we focus on the terms in equation (3) with total degree 1 in $\{x_k, y_k\}_{k \in [\![s_{\mathcal{F}} [\![}$, we get:

$$\sum_{k=0}^{s_{\mathcal{F}}-1} \left[(x_k + y_k \widehat{m}_k)(b_{k,t} s_t) \right] = c'_t S_t.$$

As $S_t = s_t \sum_{k \in \mathsf{supp}(\widehat{W})} (x_k + y_k \widehat{w}_k)$, where $\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_\mathcal{F}-1})$ is the t-th fragment-sized pattern processed by our simulator, this means, after simplifying by s_t:

$$\sum_{k=0}^{s_\mathcal{F}-1} [(x_k + y_k \widehat{m}_k) b_{k,t}] = c'_t \sum_{k \in \mathsf{supp}(\widehat{W})} (x_k + y_k \widehat{w}_k). \tag{4}$$

Keeping only the terms in $\{x_k\}_{k \in [\![s_\mathcal{F} [\![}$ in equation (4) shows that:

$$\sum_{k=0}^{s_\mathcal{F}-1} x_k b_{k,t} = c'_t \sum_{k \in \mathsf{supp}(\widehat{W})} x_k.$$

The independence of the variables $\{x_k\}_{k \in [\![s_\mathcal{F} [\![}$ shows that, for all $k \in [\![s_\mathcal{F} [\![$,

$$b_{k,t} = \begin{cases} c'_t & \text{if } k \in \mathsf{supp}(\widehat{W}), \\ 0 & \text{if not.} \end{cases}$$

We study the two following cases to conclude the proof:

- **if** $\boxed{c'_t = 0 \textbf{ or } \mathsf{pos}(i^*) \notin \mathsf{supp}(\widehat{W})}$, then we can already conclude that $b_{\mathsf{pos}(i^*),t}$=0;
- **else,** $\boxed{c'_t \neq 0 \textbf{ and } \mathsf{pos}(i^*) \in \mathsf{supp}(\widehat{W})}$ and we show a contradiction with the natural restriction placed on patterns in this game.

Indeed, in this last case we can rewrite equation (4) as:

$$c'_t \sum_{k \in \mathsf{supp}(\widehat{W})} (x_k + y_k \widehat{m}_k) = c'_t \sum_{k \in \mathsf{supp}(\widehat{W})} (x_k + y_k \widehat{w}_k).$$

We simplify by c'_t and keep the terms in $\{y_k\}_{k \in [\![s_\mathcal{F} [\![}$:

$$\sum_{k \in \mathsf{supp}(\widehat{W})} y_k \widehat{m}_k = \sum_{k \in \mathsf{supp}(\widehat{W})} y_k \widehat{w}_k.$$

The independence of the variables $\{y_k\}_{k \in [\![s_\mathcal{F} [\![}$ shows that, in this case, $\widehat{m}_k = \widehat{w}_k$ for all $k \in \mathsf{supp}(\widehat{W})$. This concretely means that $M^{(\beta)}$ contains W. However, we also have $\mathsf{pos}(i^*) \in \mathsf{supp}(\widehat{W})$. As, by definition of $i^* \in \mathcal{I}_{\neq}$, $m_{i^*}^{(\beta)} \neq m_{i^*}^{(1-\beta)}$, this means that $M^{(1-\beta)}$ does *not* contain W, which contradicts the restriction placed on patterns. This last case thus cannot occur, which concludes our proof.

5.3 Proof of Theorem 3

In the case of our second protocol, we need to proceed in two steps by using the intermediate games $\mathbf{game}_{j-1,2}$.

Lemma 1. *The difference* $|\mathbb{S}_{j-1,2} - \mathbb{S}_{j-1,1}|$ *is negligible under the* EXDH *assumption.*

Proof. Let $(g, g^a, g^{ab}, g^c, \zeta, \widetilde{g}, \widetilde{g}^a, \widetilde{g}^b) \in \mathbb{G}_1^5 \times \mathbb{G}_2^3$ be a EXDH instance.

Key Generation. The simulator generates random scalars $\{(u_k, v_k, v'_k, t_k)\}_{k\in[\![s_{\mathcal{F}}[\![}$ and implicitly sets the secret key $\mathsf{sk} = \{(x_k, y_k, z_k)\}_{k\in[\![s_{\mathcal{F}}[\![}$ with, for all $k \in [\![s_{\mathcal{F}}[\![$,

$$x_k = v_k + au_k\widehat{m}_k$$
$$y_k = v'_k - au_k$$

$$z_k = t_k \text{ if } k \neq \mathsf{pos}(i^*) \text{ and } z_{\mathsf{pos}(i^*)} = t_{\mathsf{pos}(i^*)} + ab.$$

Indeed, the simulator is able to compute the corresponding public key pk using g^a and g^{ab}. Note that the distribution of this public key is identical to the distribution of a regular public key.

Trapdoor Generation. To generate a trapdoor element $\mathsf{td}_{W,\delta} = \{T_1, T_2, T_3\}$ for a keyword W and an offset $\delta \in [\![s_{\mathcal{F}} - \ell + 1[\![$, the simulator sets

$$\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_{\mathcal{F}}-1}) := (\underbrace{\star, \ldots, \star}_{\delta}, w_0, \ldots, w_{\ell-1}, \underbrace{\star, \ldots, \star}_{s_{\mathcal{F}}-\ell-\delta})$$

and proceeds as follows:

- Case 1: $\widehat{w}_{\mathsf{pos}(i^*)} = \star$

  The simulator chooses $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$ and returns $T_1 = \widetilde{g}^{s_1}, T_2 = \widetilde{g}^{s_2}$

$$T_3 = \left(\prod_{k\in\mathsf{supp}(\widehat{W})} (\widetilde{g}^{x_k}(\widetilde{g}^{y_k})^{\widehat{w}_k})^{s_1}\right)\left(\prod_{k\in\mathsf{supp}(\widehat{W})} (\widetilde{g}^{z_k})^{s_2}\right).$$

  This last element T_3 can be computed from $\widetilde{g}^a$ as done for the public key. As $\mathsf{pos}(i^*)$ is not in the support of $\widehat{W}$, we do not need the element $\widetilde{g}^{ab}$ (which is not provided in the EXDH challenge).
- Case 2: $\widehat{w}_{\mathsf{pos}(i^*)} \neq \star$

  Let $J = \{k \in \mathsf{supp}(\widehat{W}) : \widehat{w}_k \neq \widehat{m}_k\}$. The condition on issued trapdoors and the definition of $i^* \in \mathcal{I}_{\neq}$ imply that this set is not empty, as seen at the end of proof of Theorem 2.

  The simulator selects $r, s_2 \xleftarrow{\$} \mathbb{Z}_p$ and implicitely sets

$$s_1 = -bs_2\left(\sum_{k\in J} u_k(\widehat{m}_k - \widehat{w}_k)\right)^{-1} + r.$$

  For all $k \in [\![s_{\mathcal{F}}[\![$, u_k is uniformly distributed and the view of the adversary is independent of these variables: they only appear in x_k and y_k where they are perfectly masked by v_k and v'_k. As a result, one has $\sum_{k\in J} u_k(\widehat{m}_k - \widehat{w}_k) = 0$ with negligible probability $1/p$.
  Then, the simulator returns $T_1 = \widetilde{g}^{s_1}, T_2 = \widetilde{g}^{s_2}$ using $\widetilde{g}^b$ and

$$T_3 = \left(\prod_{k\in\mathsf{supp}(\widehat{W})} (\widetilde{g}^{s_1})^{v_k + v'_k\widehat{w}_k}\right)\left(\prod_{k\in J} (\widetilde{g}^a)^{ru_k(\widehat{m}_k - \widehat{w}_k)}\right)\left(\prod_{k\in\mathsf{supp}(\widehat{W})} \widetilde{g}^{s_2 t_k}\right)$$

using $\widetilde{g}^a$ and $\widetilde{g}^b$.

Developing $s_1 \sum_{k\in\mathsf{supp}(\widehat{W})} x_k + y_k\widehat{w}_k + s_2 \sum_{k\in\mathsf{supp}(\widehat{W})} z_k$ shows that T_3 is correctly generated. In particular the term $\widetilde{g}^{ab}$ in $\widetilde{g}^{z_{\mathsf{pos}(i^*)}}$ cancels out thanks to the definition of x_k, y_k and s_1.
Moreover, the trapdoor element is well distributed as s_1, s_2 are well distributed.

Challenge Generation. The simulator generates the challenge ciphertext as follows:

- $C_h = \begin{cases} g^{a_h} \text{ with } a_h \xleftarrow{\$} \mathbb{Z}_p & \text{for all } h \in [\![n_{\mathcal{F}}[\![\setminus \{\mathtt{frag}(i^*)\} \\ g^c & \text{for } h = \mathtt{frag}(i^*) \end{cases}$

- $C'_{i,1} = \begin{cases} (g^{x_{\mathsf{pos}(i)}}(g^{y_{\mathsf{pos}(i)}})^{m_i^{(\beta)}})^{a_{\mathtt{frag}(i)}} & \text{for all } i \in [\![n - d_{\mathcal{F}}[\![\setminus (\mathcal{F}_{\mathtt{frag}(i^*)} \cup \mathcal{I}_{\neq}^{(j-1)}) \\ (g^c)^{v_{\mathsf{pos}(i)} + v'_{\mathsf{pos}(i)} m_i^{(\beta)}} & \text{for all } i \in \mathcal{F}_{\mathtt{frag}(i^*)} \setminus \mathcal{I}_{\neq}^{(j-1)} \\ \xleftarrow{\$} \mathbb{G}_1 & \text{for all } i \in \mathcal{I}_{\neq}^{(j-1)} \end{cases}$

- $C'_{i,2} = \begin{cases} (g^{z_{\mathsf{pos}(i)}})^{a_{\mathtt{frag}(i)}} & \text{for all } i \in [\![n - d_{\mathcal{F}}[\![\setminus (\mathcal{F}_{\mathtt{frag}(i^*)} \cup \mathcal{I}_{\neq}^{(j)}) \\ (g^c)^{t_{\mathsf{pos}(i)}} & \text{for all } i \in \mathcal{F}_{\mathtt{frag}(i^*)} \setminus \mathcal{I}_{\neq}^{(j)} \\ \xleftarrow{\$} \mathbb{G}_1 & \text{for all } i \in \mathcal{I}_{\neq}^{(j-1)} \\ (g^c)^{t_{\mathsf{pos}(i^*)}\zeta} & \text{for } i = i^* \end{cases}$

- all the overlined elements of C as in $\mathtt{Encrypt}(M^{(\beta)}, \mathsf{pk})$.

Note that either $\zeta = g^{abc}$ and the game is $\mathbf{game}_{j-1,1}$ as $C'_{i^*,2}$ is well-formed or ζ is random and the game is $\mathbf{game}_{j-1,2}$. Any adversary able to distinguish these two games can then be used against the EXDH assumption.

Lemma 2. *The difference* $|\mathbb{S}_{j,1} - \mathbb{S}_{j-1,2}|$ *is negligible under the* EXDH *assumption.*

Proof. Let $(g, g^a, g^{ab}, g^c, \zeta, \widetilde{g}, \widetilde{g}^a, \widetilde{g}^b) \in \mathbb{G}_1^5 \times \mathbb{G}_2^3$ be a EXDH instance.

Key Generation. The simulator generates random scalars $\{v_k, y_k, t_k\}_k \in [\![s_{\mathcal{F}}[\![$ and implicitly sets the secret key $\mathsf{sk} = \{(x_k, y_k, z_k)\}_{k\in[\![s_{\mathcal{F}}[\![}$ with, for all $k \in [\![s_{\mathcal{F}}[\![$,

$$x_k = v_k \text{ if } k \neq \mathsf{pos}(i^*) \text{ and } x_{\mathsf{pos}(i^*)} = v_{\mathsf{pos}(i^*)} + ab,$$
$$z_k = t_k \text{ if } k \neq \mathsf{pos}(i^*) \text{ and } z_{\mathsf{pos}(i^*)} = t_{\mathsf{pos}(i^*)} + a.$$

Indeed, the simulator is able to compute the public key pk associated with this secret key by using g^a and g^{ab}. Note that the distribution of this public key is identical to the distribution of a regular public key.

Trapdoor Generation. To issue a trapdoor element $\mathsf{td}_{W,\delta} = \{T_1, T_2, T_3\}$ for a keyword W and an offset $\delta \in [\![s_{\mathcal{F}} - \ell + 1 [\![$, the simulator sets

$$\widehat{W} = (\widehat{w}_0, \ldots, \widehat{w}_{s_{\mathcal{F}}-1}) := (\underbrace{\star, \ldots, \star}_{\delta}, w_0, \ldots, w_{\ell-1}, \underbrace{\star, \ldots, \star}_{s_{\mathcal{F}}-\ell-\delta})$$

and proceeds as follows:

- Case 1: $\widehat{w}_{\mathsf{pos}(i^*)} = \star$
  The simulator acts exactly as in the protocol because the elements from the EXDH instance are only involved in $x_{\mathsf{pos}(i^*)}$ and $z_{\mathsf{pos}(i^*)}$.
- Case 2: $\widehat{w}_{\mathsf{pos}(i)^*} \neq \star$

  The simulator selects $r, s_1 \xleftarrow{\$} \mathbb{Z}_p$ and implicitly sets $s_2 := -bs_1 + r$.
  Then, it returns $T_1 = \widetilde{g}^{s_1}$, $T_2 = \widetilde{g}^{s_2}$ using $\widetilde{g}^b$ and

$$T_3 = \left(\prod_{k \in \mathsf{supp}(\widehat{W})} \widetilde{g}^{s_1(v_k + y_k \widehat{w}_k)} \right) \left(\prod_{k \in \mathsf{supp}(\widehat{W})} (\widetilde{g}^{s_2})^{t_k} \right) (\widetilde{g}^a)^r \text{ using } \widetilde{g}^a \text{ and } \widetilde{g}^b.$$

  Developping $s_1 \sum_{k \in \mathsf{supp}(\widehat{W})} [x_k + y_k \widehat{w}_k] + s_2 \sum_{k \in \mathsf{supp}(\widehat{W})} z_k$ shows that T_3 is correctly generated. Moreover, the trapdoor element is well distributed as s_1, s_2 are well distributed.

Challenge Generation. The simulator generates the challenge ciphertext as follows :

- $C_h = \begin{cases} g^{a_h} \text{ with } a_h \xleftarrow{\$} \mathbb{Z}_p & \text{for all } h \in [\![n_{\mathcal{F}} [\![\setminus \{\mathtt{frag}(i^*)\} \\ g^c & \text{for } h = \mathtt{frag}(i^*) \end{cases}$

- $C'_{i,1} = \begin{cases} (g^{x_{\mathsf{pos}(i)}} (g^{y_{\mathsf{pos}(i)} m_i^{(\beta)}})^{a_{\mathtt{frag}(i)}} & \text{for all } i \in [\![n - d_{\mathcal{F}} [\![\setminus (\mathcal{F}_{\mathtt{frag}(i^*)} \cup \mathcal{I}_{\neq}^{(j)}) \\ (g^c)^{v_{\mathsf{pos}(i)} + y_{\mathsf{pos}(i)} m_i^{(\beta)}} & \text{for all } i \in \mathcal{F}_{\mathtt{frag}(i^*)} \setminus \mathcal{I}_{\neq}^{(j)} \\ \xleftarrow{\$} \mathbb{G}_1 & \text{for all } i \in \mathcal{I}_{\neq}^{(j-1)} \\ (g^c)^{y_{\mathsf{pos}(i^*)} m_{i^*}^{(\beta)}} \zeta & \text{for } i = i^* \end{cases}$

- $C'_{i,2} = \begin{cases} (g^{z_{\mathsf{pos}(i)}})^{a_{\mathtt{frag}(i)}} & \text{for all } i \in [\![n - d_{\mathcal{F}} [\![\setminus (\mathcal{F}_{\mathtt{frag}(i^*)} \cup \mathcal{I}_{\neq}^{(j)}) \\ (g^c)^{t_{\mathsf{pos}(i)}} & \text{for all } i \in \mathcal{F}_{\mathtt{frag}(i^*)} \setminus \mathcal{I}_{\neq}^{(j)} \\ \xleftarrow{\$} \mathbb{G}_1 & \text{for all } i \in \mathcal{I}_{\neq}^{(j)} \end{cases}$

- all the overlined elements of C as in $\mathtt{Encrypt}(M^{(\beta)}, \mathsf{pk})$.

Note that either $\zeta = g^{abc}$ and the game is $\mathbf{game}_{j-1,2}$ as $C'_{i^*,1}$ is well-formed or ζ is random and the game is $\mathbf{game}_{j,1}$. Any adversary able to distinguish these two games can then be used against the EXDH assumption.

6 Complexity Analysis

Table 1 in Section 1.2 provides a comparison on some specific metrics with two relevant constructions of the state-of-the-art, namely [13] and [3]. We here provide a more comprehensive performance assessment of our constructions that we only compare to [3] as the latter outperforms [13].

6.1 Space Complexity

In this part, we focus on the size of the different elements involved in SEPM constructions. To have a common metric, we implement our bilinear groups using the BLS12-381 curve [9], yielding 48-Bytes (compressed) elements of $\mathbb{G}_1$, 96-Bytes (compressed) elements of $\mathbb{G}_2$ and 572-Bytes elements of $\mathbb{G}_T$. To provide a fair comparison, we select the same parameters as in [3] and thus consider the encryption of 1GB bytestrings where any pattern of size at most 10KB (*i.e.* $\mathrm{L} = 10000$) can be searched. The results are presented in Table 2. One can note that the results for [3] differ from those provided in the original paper. This is due in part to the use of Barreto-Naehrig curves [2] in [3] that are now deprecated. Regarding the size of the public key, the difference also stems from an error in [3] as the authors do not take into account the $|\mathcal{S}|$ factor in their computations. For bytestrings, we have $|\mathcal{S}| = 256$, which is quite significant.

Table 2 highlights the difference between our constructions and the one in [3], in particular regarding the size of the public key where ours are about 100 times smaller. Our first construction also halves the size of the ciphertext but the latter remains large. Improving this characteristic while retaining the nice features of SEPM is an open problem.

	Schemes		
	AS^{3}E ([3])	Section 4.3	Section 4.4
Public Key	247 MB	1.92 MB	2.88 MB
Ciphertext	192 GB	96 GB	192 GB
Trapdoor	1.92 MB	1.92 MB	2.88 MB

Table 2. Comparison with the state of the art

6.2 Computational Complexity

We now focus on the computational cost of the `Encrypt`, `Issue` and `Test` procedures by providing in Table 3 an estimation of the number of operations required to perform them. We set n as the length of the message to encrypt and L as the bound on the size of searchable patterns. As the treatment of wildcard and non-wildcard characters strongly differs in our `Test` procedure, we assume that the searched pattern contains c non-wildcard characters.

In our case, the encryption can be speeded up by (pre-)computing the 2^8 elements $\{(Y_k)^b\}_{k\in[\![s_{\mathcal{F}}]\!],b\in[\![2^8]\!]}$ and use the results to directly generate the ciphertext elements. Compared to the naive protocol description in Sections 4.3 and 4.4, this saves $2n$ exponentiations.

	Schemes		
	AS^{3}E ([3])	Section 4.3	Section 4.4
`Encrypt`	$4n\mathbf{e}_1$	$\left(4n+\frac{n}{L}\right)\mathbf{e}_1+n\mathbf{m}_1$	$\left(6n+\frac{n}{L}\right)\mathbf{e}_1+n\mathbf{m}_1$
`Issue`	$2L\mathbf{e}_2$	$2L\mathbf{e}_2$	$3L\mathbf{e}_2$
`Test`	$nc\mathbf{m}_1+2n\mathbf{P}$	$nc\mathbf{m}_1+2n\mathbf{P}$	$2nc\mathbf{m}_1+3n\mathbf{P}+n\mathbf{m}_T$

Table 3. Comparison with the state of the art. For $i\in\{1,2,T\}$, $\mathbf{m}_i$ (resp. $\mathbf{e}_i$) stands for one multiplication (resp. exponentiation) in $\mathbb{G}_i$ and $\mathbf{P}$ for one pairing.

Our comparison shows that the performance of all these schemes is very similar and essentially requires a few exponentiations in $\mathbb{G}_1$ to encrypt one byte and 2 pairings per byte for detections. The concrete performance will obviously depend on the devices performing these computations. We nevertheless note that, for all these schemes, these computations are embarrassingly parallelizable.

Acknowledgements

The second author was supported by the French ANR ALAMBIC project ANR-16-CE39-0006. The third author is grateful for the support of the ANR through project ANR-19-CE39-0011-04 PRESTO and project ANR-18-CE-39-0019-02 MobiS5.

References

1. Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, 21(3):350–391, July 2008.
2. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.
3. Anis Bkakria, Nora Cuppens, and Frédéric Cuppens. Privacy-preserving pattern matching on encrypted data. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 191–220. Springer, Heidelberg, December 2020.
4. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 506–522. Springer, Heidelberg, May 2004.

5. Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 461–478. Springer, Heidelberg, August 2013.
6. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.
7. Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, Heidelberg, February 2007.
8. Raphael Bost. $\Sigma o \phi o \varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1143–1154. ACM Press, October 2016.
9. Sean Bowe. BLS12-381: New zk-SNARK Elliptic Curve Construction. `https://electriccoin.co/blog/new-snark-curve/`, 2017.
10. Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine, and Mohamed Sabt. BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 561–574. ACM Press, April 2017.
11. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical (full version of the PKC 2015 paper). *IACR Cryptol. ePrint Arch.*, 2014:785, 2014.
12. Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88. ACM Press, October / November 2006.
13. Nicolas Desmoulins, Pierre-Alain Fouque, Cristina Onete, and Olivier Sanders. Pattern matching on encrypted streams. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 121–148. Springer, Heidelberg, December 2018.
14. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
15. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
16. Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Journal of Cryptology*, 26(2):191–224, April 2013.
17. Shangqi Lai, Xingliang Yuan, Shifeng Sun, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, and Dongxi Liu. Practical encrypted network traffic pattern matching for secure middleboxes. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021.
18. Iraklis Leontiadis and Ming Li. Storage efficient substring searchable symmetric encryption. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*, SCC '18, page 3–13. Association for Computing Machinery, 2018.
19. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.
20. Saeed Sedghi, Peter van Liesdonk, Svetla Nikova, Pieter H. Hartel, and Willem Jonker. Searching keywords with wildcards on encrypted data. In Juan A. Garay

and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 138–153. Springer, Heidelberg, September 2010.
21. Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, *SIGCOMM 2015*, pages 213–226, 2015.
22. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.
23. Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 763–780. ACM Press, October 2018.