Pattern Matching on Encrypted Streams

Nicolas Desmoulins[1], Pierre-Alain Fouque[2], Cristina Onete[3]
and Olivier Sanders[4]

[1] Orange Labs, Applied Crypto Group, Caen, France
[2] Université de Rennes 1 & Institut Universitaire de France, Rennes, France
[3] Université de Limoges, CNRS UMR 7252, Limoges, France
[4] Orange Labs, Applied Crypto Group, Cesson-Sévigné, France

Abstract. Pattern matching is essential in applications such as deep-packet inspection (DPI), searching on genomic data, or analyzing medical data. A simple task to do on plaintext data, pattern matching is much harder to do when the privacy of the data must be preserved. Existent solutions involve searchable encryption mechanisms with at least one of these three drawbacks: requiring an exhaustive (and static) list of keywords to be prepared before the data is encrypted (like in symmetric searchable encryption); requiring *tokenization*, *i.e.*, breaking up the data to search into substrings and encrypting them separately (*e.g.*, like BlindBox); relying on symmetric-key cryptography, thus implying a token-regeneration step for each encrypted-data source (*e.g.*, user). Such approaches are ill-suited for pattern-matching with evolving patterns (*e.g.*, updating virus signatures), variable searchword lengths, or when a single entity must filter ciphertexts from multiple parties.
In this work, we introduce Searchable Encryption with Shiftable Trapdoors (SEST): a new primitive that allows for pattern matching with universal tokens (usable by all entities), in which keywords of arbitrary lengths can be matched to arbitrary ciphertexts. Our solution uses public-key encryption and bilinear pairings.

In addition, very minor modifications to our solution enable it to take into account regular expressions, such as fully- or partly-unknown characters in a keyword (wildcards and interval/subset searches). Our trapdoor size is at most linear in the keyword length (and independent of the plaintext size), and we prove that the leakage to the searcher is only the trivial one: since the searcher learns whether the pattern occurs and where, it can distinguish based on different search results of a single trapdoor on two different plaintexts.
To better show the usability of our scheme, we implemented it to run DPI on all the SNORT rules. We show that even for very large plaintexts, our encryption algorithm scales well. The pattern-matching algorithm is slower, but extremely parallelizable, and it can thus be run even on very large data. Although our proofs use a (marginally) interactive assumption, we argue that this is a relatively small price to pay for the flexibility and privacy that we are able to attain.

1 Introduction

Learning whether a given pattern occurs in a larger input string (and where exactly that happens) has many applications, such as when searching on genomic data, in deep-packet inspection (DPI), or when delegating searches in databases. In such cases, the entity performing the search, usually called the *gateway*, is only semi-trusted by the owner of the input data. Indeed, in all the three scenarios above, it is of paramount importance to preserve the *privacy* of the input data[5].

Consider the case of a middlebox, such as a virus scan or a firewall. A user who may trust the middlebox to scan its data for viruses might not, in fact, be comfortable revealing the full contents of its data to that middlebox. Similarly, a person might trust a laboratory to check whether their genome contains a particular substring (indicating, *e.g.*, a genetic predisposition to a disease); however, the laboratory should not, in this way, come into possession of that person's full genome. Such concerns have been exacerbated lately by threats of mass-surveillance, following the revelations of Edward Snowden. As a consequence, data *encryption* is slowly becoming an *a priori* pre-requisite for pattern matching.

In cryptography, pattern matching on encrypted data is closely related to Searchable Encryption, either Symmetric [16–18, 32] or Public-Key [9]. Many Searchable Encryption solutions, however, only allow to search for pre-chosen keywords, which are hard-coded in the encrypted input. Searching for a new keyword – not indicated *a priori* – in that same (already encrypted) data would yield a false negative, even if that keyword is, in fact contained in the input data. Correctly matching the new pattern to the data requires that the latter be re-encrypted. Therefore this solution is ill-suited to more dynamic environments, like DPI. We provide a full comparison with related literature, including searchable encryption, in Section 1.2.

Pattern matching with non-static patterns can be achieved through symmetric-key techniques and so-called *tokenization* [31]. In this approach, a sliding-window technique is used to encode keywords of a given, fixed length, which can then be matched by the searcher. This allows searches to be performed for arbitrarily-chosen keywords; however, a disadvantage is that each instantiation requires a new generation of tokens. Moreover, this only works for a fixed keyword length and different ciphertexts are required to handle different pattern sizes. This is less than ideal for many use-cases such as DPI, since for instance SNORT rules [1] include patterns of many different lengths. In this paper, our goal is to improve on this solution, specifically by allowing to search on encrypted data, with patterns that are non-static (flexible), of variable length, and universal (no need to re-tokenize). In particular, we achieve secure pattern-matching on encrypted data with *universal tokens*.

[5] By contrast, in many cases, the patterns themselves may be publicly known.

1.1 Our contributions

We opt for a solution in a public-key setting (which immediately achieves universality for our patterns). The gateway will be able to search for *keywords* on encrypted data using *trapdoors* that are unforgeable. More specifically, our construction can support pattern matching for keywords that can be adaptively chosen and which can have variable lengths. Moreover, the size of the trapdoors corresponding to those keywords does not depend on the length of the input data (our trapdoors are short, even when we are searching in very large input data). We support regular expressions, such as the presence of wildcards or matching encrypted input to general data-subsets. Thus, our solution is well suited to deep packet inspection or delegated searches on medical data.

Intuitively, in our construction we *project* each coordinate of the plaintext S (and then of the keyword W) on a geometric basis consisting of some values z^i, for $i = 0, \ldots, |S| - 1$. We prevent malleability of trapdoors by embedding the exact order of the bits of W into a polynomial, which cannot be forged without the secret key. A fundamental part of the searching algorithm that we propose is the way in which the middlebox will be able to *shift* from one part of the ciphertext to another, when searching for a match with W. Thus, our scheme can be viewed as an anonymous predicate encryption scheme where one could derive the secret keys for $(*, w_1, \ldots, w_\ell, *, \ldots, *)$,...,$(*, \ldots, *, w_1, \ldots, w_\ell)$ from the secret key for $(w_1, \ldots, w_\ell, *, \ldots, *)$.

Such changes require the definition of a new primitive that we call Searchable Encryption with Shiftable Trapdoors (SEST). We provide a formal security model for the latter, which ensures that even a malicious gateway knowing trapdoors $\mathsf{td}_{W_1}, \ldots, \mathsf{td}_{W_q}$ does not learn any information from an encrypted string S beyond the presence of the keyword W_k in S, for $k \in [1, q]$.

Our construction is – to our knowledge – the first SEST scheme, and thus can be taken as a proof-of-concept construction. We guarantee the desired properties by only using asymmetric prime order bilinear groups (*i.e.* a set of 3 groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ along with an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$) for which very efficient implementations have been proposed (*e.g.* [7]). Encryption of plaintexts S only requires operations in the group $\mathbb{G}_1$, while detection of the keyword W is done by performing pairings. The former operation requires only the public key while the latter additionally needs the corresponding trapdoor; only the trapdoor-issuing algorithm requires the corresponding secret key.

We are able to allow for pattern-matching when some of the contents of the keywords are either fully-unknown, *i.e.*, wildcards, or partially-unknown, *i.e.*, in an interval. Searches for such regular expressions remain fully-compatible with our original solution. In the first case, the only difference is that when issuing the trapdoor, instead of fully randomizing it we choose special randomness – equal to 0 – for the "coefficients" of the polynomial that we project the wildcards or unknown subsets to. For the scenario of partially-known trapdoors, we require a more complex key-generation process since we use different values on which to (uniformly) project the unclear values to. These will be used in the trapdoor generation step, ensuring that if a partially-known input is used, that coefficient of the trapdoor will still "vanish".

In particular, our pattern-matching algorithm is very similar to that of Rabin-Karp and consequently, we can use it to solve similar problems. In addition to the previous use-cases, our technique can also be used to perform 2D pattern matching in images, or searching subtrees in rooted, labelled trees. However, note that due to the privacy-preserving goal of our work, we cannot benefit from many of the tricks used by Rabin-Karp, thus yielding a scheme with limited efficiency.

We also analyze how well our scheme performs when applied to DPI. We implemented our scheme to search for all the SNORT rules in input data of varying sizes. Even for large data, the encryption algorithm is very efficient. Moreover, while the testing (pattern matching) step scales less well with increasing input-data size, that particular step is highly parallelizable, and thus the running time can be much reduced.

Impact and limitations. Our scheme allows for a flexible searchable encryption mechanism, in which encrypters do not have to embed a list of possible keywords into their ciphertexts. Moreover, we also provide a great deal of flexibility with respect to searching for keywords of arbitrary lengths. In this sense, our technique allows for searchable encryption with *universal tokens*, which can be used in deep-packet inspection, applications on genomic and medical data, or matching subtrees in labelled trees.

One limitation of our scheme is the size of our public keys. We require a public key of size linear in the size of the plaintext to be encrypted (which is potentially very large). This is mostly due to the need to shift the ciphertext each time in order to detect the presence of the keyword. We also require a large ciphertext, consisting of a number of elements that is again linear in the size of the plaintext; however, the same inefficiency is inherent also to solutions such as BlindBox [31], in which we must encrypt many "windows" of the data, of same size. Finally, the search of a keyword of size ℓ in a plaintext of size n requires at least $2(n - \ell + 1)$ pairing computations.

Furthermore, we are only able to prove the security of our construction under an interactive assumption, unless we severely restrict the size n of the message space. Indeed, we need an assumption which offers enough flexibility to provide shiftable trapdoors for all possible keywords except the one that allow trivial distinction of the encrypted string. We modify the GDH assumption [8] in a minimal way, to allow the adversary to request the values on which the reduction will break this assumption. We could remove the need for this flexibility, by, for instance reducing the value of n so that the simulator could guess the strings targeted by the adversary but this strongly limits the applications of our construction.

We argue that despite this interactive assumption, the intrinsic value of our construction lies in its flexibility, namely in the fact that we are able to search for arbitrary keywords. This significantly improves existing solutions of, *e.g.*, detecting viruses on encrypted traffic over HTTPS [24, 25, 31].

Moreover, we emphasize that we achieve this high level of flexibility without using complex (and costly) cryptographic tools such as fully homomorphic encryption. We simply need pairings which have become quite standard in cryptography and which can be implemented very efficiently [7]. We therefore argue that our scheme, when compared to solutions providing the same features (see Section 1.3 for more details), offers a practical improvement over the state of the art.

1.2 Related work

How searchable encryption works. In searchable encryption (SE) [9,16–18,32], any party that is given a trapdoor $\mathtt{td}_W$ associated with a keyword W is able to search for that keyword within a given ciphertext. The ideal privacy guarantee required is that searching reveals nothing else on the underlying plaintext (other than the presence or absence of the keyword). Routing encrypted emails, querying encrypted database or running an antivirus on encrypted traffic are typical applications which require such a functionality.

In general, SE searches are usually performed by the middlebox on keywords that have been pre-chosen by the party encrypting the ciphertexts (*i.e.*, the encrypter). In particular, an encrypted string containing W can be detected by the middlebox knowing $\mathtt{td}_W$ only if the sender has selected W as a keyword and has encrypted it using the SE scheme. Such approaches are still suitable for some types of database searches (in which documents are already indexed by keywords), or in the case of emailing applications – for which natural keywords can be the sender's identity, the subject line, or flags such as "urgent". Unfortunately, in cases such as messaging applications, or just for common Internet browsing, the keywords are much harder to find, and can include expressions that are not sequences of words *per se*, but rather something of the kind "http://www.example.com/index.php?username=1".

Our solution allows for better flexibility in terms of searching for arbitrarily-chosen keywords, even after the plaintext has been encrypted and sent. In fact, it is not even necessary that the encrypter be the same person as the party which issues the trapdoors. This makes our solution much better suited to DPI scenarios, whereas SE is typically better suited to database searches.

Tokenization. The solution proposed in [31] to search keywords of length ℓ is to split the string $S = s_0 \dots s_{n-1}$ into $[s_0 \dots s_{\ell-1}]$, $[s_1 \dots s_\ell]$, $\dots$, $[s_{n-\ell} \dots s_{n-1}]$ and then to encrypt each of these substrings using a searchable encryption scheme (the substrings are thus the keywords associated with S). However, this solution has a drawback: it works well if all the searchable keywords $W_1, \dots, W_q$ have the same length but this is usually not the case. In the worst case, if all searchable keyword W_k are of different length ℓ_k, the sender will have, for each $k \in [1, q]$, to split S in substrings of size ℓ_k and encrypt them, which quickly becomes cumbersome. One solution could be to split the searchable keywords W_k into smaller keywords of the same length $\ell_{min} = min_k(\ell_k)$. For example, if $\ell_{min} = 3$ the searchable keyword "execute" could be split into "exe", "cut" and "ute" for which specific trapdoors would be issued. Unfortunately, this severely harms privacy since these smaller keywords will match many more strings S. Moreover, repeating this procedure for every keyword W_k will allow the gateway to receive trapdoors for a large fraction of the set of strings of length ℓ_{min} and so to recover large parts of S with significant probability.

We note that Canard *et al.* [14] recently proposed a public key variant of the Blindbox [31] approach which therefore suffers from the same limitations. Moreover, their performance corresponds to the "delimiter-based" version of their protocol that consists in splitting a string $s = s_0 \dots s_{n-1}$ into t substrings $[s_0 \dots s_{n_1-1}]$, $[s_{n_1} \dots s_{n_2-1}]$, ..., $[s_{n_{t-1}} \dots s_{n-1}]$ which are then independently encrypted using searchable encryption. While this dramatically reduces complexity, we stress that this only allows to detect patterns that perfectly match one of the substrings. In particular, a pattern cannot be detected if it straddles two substrings.

By contrast, our scheme addresses the main drawback of this tokenization technique: we allow for universal trapdoors of arbitrary length to be matched against the encrypted data, without false negatives or positives. This comes at a cost in performance; however, we show in our implementation that our scheme remains practical.

Generic evaluation of functions on ciphertexts. Evaluation of functions over encrypted data is a major topic in cryptography, which has known very important results over the past decade. Generic solutions (*e.g.*, fully homomorphic encryption [22], functional encryption [3,4],etc.), supporting a wide class of functions, have been proposed; however, their very high complexity makes such solutions impractical. In practice, it is then better to use a scheme specifically designed for the function(s) that one wants to evaluate.

Several recent publications study secure substring search and text processing [5,21,23,26,28,29,33], specifically in two-party settings. Some of these papers provide applications to genomic data, specifically matching substrings of DNA to encrypted genomes. This was done by using secure multi-party computation or fully-homomorphic encryption. However, the former solution requires interaction between the searcher and the encrypter, whereas the use of FHE induces a relatively high complexity. Of particular interest here is the approach by Lauter et al. [28], which presents an application to genomic data. The authors here go much further than just matching patterns with some regular expressions, however, they require fully-homomorphic encryption (FHE) for their applications. We leave it as future work to investigate in how far we can modify our technique with universal tokens in order to provide some support to the algorithms presented by Lauter et al. for genomic matching.

At first sight, anonymous predicate encryption (*e.g.* [27]) or hidden vector encryption [11] provide an elegant solution to the problem of searching on encrypted streams. Indeed, the sender could use one of these schemes to produce a ciphertext for some attributes $s_0, \dots, s_{n-1}$ which together make up a word S, while the middlebox, knowing the suitable secret keys, could detect whether S contains a substring W. The encryption process would then not depend on the

Primitives	Issue	Public Parameters	Ciphertext	Trapdoors
SSE	$O(s \cdot q)$	$O(1)$	$O(n \cdot L)$	$O(s \cdot q)$
ASE	$O(q)$	$O(1)$	$O(n \cdot L)$	$O(q)$
PE/HVE	$O(n \cdot q)$	$O(n)$	$O(n)$	$O(n \cdot q)$
SEST (this work)	$O(q)$	$O(n)$	$O(n)$	$O(q)$

Fig. 1. Complexity comparison between related work and our primitive. The Issue process refers to the generation of trapdoors. The complexity indicated in the last three columns is the size complexity. The integers n, q, L, s denote respectively the length of the message to encrypt, the number of issued trapdoors, the number of different lengths among the q trapdoors and the number of users communicating with the receiver.

searchable keywords and the anonymity property of these schemes would ensure that the ciphertext does not leak more information on S.

However, another issue arises with this solution. Indeed, $W = w_1 \dots w_\ell$ can be contained at any position in S. Therefore, the gateway should receive the secret keys for $(w_1, \dots, w_\ell, *, \dots, *)$, $(*, w_1, \dots, w_\ell, *, \dots, *)$,...,$(*, \dots, *, w_1, \dots, w_\ell)$, where "$*$" plays the role of a wildcard, to take into account all the possible offsets. So, for each searchable keyword of size ℓ, the gateway would have to store $n - \ell + 1$ keys, which is obviously a problem for large strings S.

DPI with multi-context key-distribution. Naylor et al. [30] recently presented a multi-context key-exchange over the TLS protocol, which aims to allow middleboxes (read, write, or no) access to specific ciphertext fragments that they are entitled to see. This type of solution has some important merits, such as the fact that it is relatively easy to put into practice and allows the middlebox to perform its task with a very low overhead (the cost of a simple decryption). In addition, the parties sending and receiving messages need not deviate from the protocols they employ (such as TLS/SSL).

However, such solutions also have important disadvantages. The first of these is that the privacy they offer is not ideal. Instead of simply learning whether a specific content is contained within a given message or not, the middlebox learns entire chunks of messages. Moreover, the access-control scheme associated to the key-exchange scheme is relatively inflexible. The middlebox is given read or write access to a number of message fragments, and this is not easily modifiable (except by running the key-distribution algorithm once more). Finally, despite the efficiency of the search step (once the key-repartition is done), the finer-grained the access control is – thus offering more privacy – the more keys will have to be generated and stored by the various participating entities.

1.3 Benefits of SEST

Pattern matching on encrypted data is a very frequently-encountered problem, which can be addressed by many different primitives. In this context, the benefits of our new primitive (SEST) might not seem obvious. To better understand the intrinsic differences between all these approaches, we provide in Figure 1 a comparison of their asymptotic complexities. We choose to only consider the most relevant alternatives, namely Searchable Encryption (both Symmetric and Public-Key) and Predicate Encryption/Hidden Vector Encryption. Other solutions do exist, as explained above; however, they induce high complexity, interactivity or weaker privacy.

As we explained, searching substrings at any position using SSE or ASE requires a tokenization process which must be repeated for each possible length of keyword, hence the $O(n \cdot L)$ size of the ciphertext. ASE performance is an adaptation of the tokenization idea of BlindBox to the Public Key Encryption with Keyword Search of Boneh *et al* [9].

Conversely, PE and HVE offer a $O(n)$ complexity for the ciphertext but at the cost of generating and storing $n \cdot q$ trapdoors (to handle any possible offset).

We therefore argue that SEST is an interesting middle way which almost provides the best of the previous two types. Its only drawback compared to SSE and to ASE is the size of the public parameters but we believe this is a reasonable price to pay to achieve all the other features.

1.4 Pattern Matching and Privacy

At first sight, the ability to search patterns within a ciphertext may seem harmful to users' privacy, compared to standard end-to-end encryption. However, we stress that it is a lesser evil in many use-cases.

For example, in current solutions for DPI [25], the middlebox acts as a man in the middle to decrypt all traffic, which means that end-to-end encryption is gone anyway. Using SEST, the users can at least control which information can be leaked from their traffic since they are the only ones who can issue trapdoors. In particular, they can check that the keywords submitted by the middlebox are legitimate. For example, as we describe in Section 6.2, they could agree to issue trapdoors only for patterns associated to malwares, using public rules such as the ones provided by SNORT [1].

More generally, the incompatibility of standard encryption with any data processing often jeopardizes users' privacy since it gives no other choice than complete decryption of the traffic. We therefore argue that SEST is far from being a threat to privacy and can actually be used to improve it.

Outline. Our paper has the following structure. We begin in Section 2 by formally defining our new primitive, Searchable Encryption with Shiftable Trapdoors (SEST). Then, in Section 3, we describe an instantiation of this primitive, which relies on public-key encryption and bilinear pairings. In Section 4, we describe under which assumptions our scheme achieves provable security, and provide a security proof. We then describe how our construction can be used to handle regular expressions (wildcards and value intervals) in Section 5. Handling regular expressions is important in real-world applications, including DPI. In Section 6 we discuss the efficiency of our protocol and provide implementation results for pattern matching of all the SNORT rules on encrypted data of various sizes. Finally, we discuss our results and make some concluding remarks in Section 7.

2 Searchable Encryption with Shiftable Trapdoors

We begin by presenting the syntax of our SEST primitive. Note that in addition to indicating whether the keyword was found in the (encrypted) plaintext, this scheme also outputs the position(s) at which the keyword is found. This is one advantage of shiftable trapdoors[6], namely yielding the exact position, within the target plaintext, of the search word. Such a knowledge is indeed necessary for some use-cases (see Section 6.2).

To keep our model as general as possible we consider strings $S = s_0 \dots s_{m-1}$ whose characters s_i belong to a finite set $\mathcal{S}$. Since $\mathcal{S}$ is finite, we may assume that each of its elements s can be simply indexed by a unique integer $f(s)$ between 0 and $|\mathcal{S}| - 1$. For sake of simplicity, we will omit in the following the function f and will then directly use s as an index (for example $T[f(s)]$ will be denoted by $T[s]$).

2.1 Syntax

A searchable encryption scheme with shiftable trapdoors 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.

- `Setup`$(1^k, n)$: This probabilistic algorithm takes as input a security parameter k and an integer n defining the maximum size of the strings that one can encrypt. It returns the public parameters pp that will be taken in input by all the other algorithms. In the following, pp will be considered as an implicit input to all algorithms and so will be omitted.
- `Keygen`$(\mathcal{S})$: This probabilistic algorithm run by the receiver takes as input a finite set $\mathcal{S}$ and returns a key pair $(\mathsf{sk}, \mathsf{pk})$. The former value is secret and only known to the receiver, while the latter is public.
- `Issue`(W, sk): This probabilistic algorithm takes as input a string W of any size $0 < \ell \leq n$, along with the receiver's secret key, and returns a trapdoor $\mathtt{td}_W$.
- `Encrypt`(S, pk): This probabilistic algorithm takes as input the receiver's public key along with a string $S = s_0 \dots s_{m-1}$ of size $0 < m \leq n$ such that $s_i \in \mathcal{S}$ for all $i \in [0, m-1]$ and returns a ciphertext C.
- `Test`$(C, \mathtt{td}_W)$: This deterministic algorithm takes as input a ciphertext C encrypting a string $S = s_0 \dots s_{m-1}$ of size m along with a trapdoor $\mathtt{td}_W$ for a string $W = w_0 \dots w_{\ell-1}$ of size ℓ. If $m > n$ or $\ell > m$, then the algorithm returns $\perp$. Else, the algorithm returns a set (potentially empty) $\mathcal{J} \subset \{0, m-\ell\}$ of indexes j s.t. $s_j \dots s_{j+\ell-1} = w_0 \dots w_{\ell-1}$.

[6] Solutions using tokenization, such as Blindbox, also output the position. Here we compare with standard searchable encryption that usually does not reveal this information.

Remark 1: Notice that searchable encryption, *e.g.*, [2, 11], usually does not consider a decryption algorithm which takes as input sk and a ciphertext C encrypting S and which returns S. Indeed, this functionality can easily be added by also encrypting S under a conventional encryption scheme. Nevertheless, one can note that decryption can be performed by issuing a trapdoor for all characters $s \in \mathcal{S}$ and running the $\texttt{Test}$ algorithm on C for each of them.

2.2 Security Model

Correctness. As in [2], we divide correctness into two parts. The first one stipulates that the $\texttt{Test}$ algorithm run on (C, td_W) will always return j if S contains the substring W at index j (no false negatives). More formally, this means that, for any string S of size $m \leq n$ and any W of length $\ell \leq m$: whenever $s_j \dots s_{j+\ell-1} = w_0 \dots w_{\ell-1}$,

$$\texttt{Pr}[j \in \texttt{Test}(\texttt{Encrypt}(S, \mathsf{pk}), \texttt{Issue}(W, \mathsf{sk}))] = 1,$$

where the probability is taken over the choice of the pair $(\mathsf{sk}, \mathsf{pk})$.

The second part of the correctness property requires that false positives (*i.e.*, when the $\texttt{Test}$ algorithm returns j despite the fact $s_j \dots s_{j+\ell-1} \neq w_0 \dots w_{\ell-1}$) only occur with negligible probability. More formally, this means that, for any string S of size $m \leq n$ and any string W of length $\ell \leq m$:

$$\texttt{Pr}\left[\begin{array}{c} j \in \texttt{Test}(\texttt{Encrypt}(S, \mathsf{pk}), \texttt{Issue}(W, \mathsf{sk})) \\ \& \quad s_j \dots s_{j+\ell-1} \neq w_0 \dots w_{\ell-1} \end{array}\right] \leq \mu(k)$$

where μ is a negligible function.

Indistinguishability (SEST-IND-CPA). For the security requirement of Searchable Encryption with Shiftable Trapdoors (SEST), we adapt the standard notion of IND-CPA to this case (hence the name SEST-IND-CPA). Informally, this notion requires that no adversary $\mathcal{A}$, even with access to an oracle $\mathcal{O}\texttt{Issue}$ which returns a trapdoor td_W for any queried string W, can decide whether a ciphertext C encrypts S_0 or S_1 as long as the trapdoors issued by the oracle do not allow trivial distinction of these two strings. This is formally defined by the experiment $\texttt{Exp}_{\mathcal{A}}^{ind-cpa-\beta}(1^k, n)$, where $\beta \in \{0, 1\}$ as described in Figure 2. The set $\mathcal{W}$ is the set of all the strings W submitted to $\mathcal{O}\texttt{Issue}$.

We define the advantage of such an adversary as $\texttt{Adv}_{\mathcal{A}}^{ind-cpa}(1^k, n) = |\Pr[\texttt{Exp}_{\mathcal{A}}^{ind-cpa-1}(1^k, n)] - \Pr[\texttt{Exp}_{\mathcal{A}}^{ind-cpa-0}(1^k, n)]|$. A searchable encryption scheme with shiftable trapdoors is SEST-IND-CPA secure if this advantage is negligible for any polynomial-time adversary.

We note that this security notion is very similar to the *attribute hiding* property of predicate encryption [27]. However, we cannot directly use this latter property because of the differences between predicate encryption and our primitive (*e.g.*, the lack of decryption algorithm), hence the need for a new security game.

```
$\mathtt{Exp}_{\mathcal{A}}^{ind-cpa-\beta}(1^k, n)$

  1. $pp \leftarrow \mathtt{Setup}(1^k, n)$
  2. $\mathsf{pk} \leftarrow \mathtt{Keygen}(\mathcal{S})$
  3. $(S_0, S_1) \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Issue}}(\mathsf{pk})$, with $S_i = s_0^{(i)} \dots s_{m-1}^{(i)}$ for some $m \leq n$
  4. $C \leftarrow \mathtt{Encrypt}(S_\beta, \mathsf{pk})$
  5. $\beta' \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Issue}}(C, \mathsf{pk})$
  6. If $\exists W = w_0 \dots w_{\ell_W - 1} \in \mathcal{W}$ and $j$ such that:

        $s_j^{(i)} \dots s_{j+\ell_W-1}^{(i)} = w_0 \dots w_{\ell_W-1} \neq s_j^{(1-i)} \dots s_{j+\ell_W-1}^{(1-i)}$

     then return 0
  7. Return $(\beta = \beta')$
```

Fig. 2. SEST-IND-CPA Security Game

The restriction in step 6 simply ensures that if S_i contains $W \in \mathcal{W}$ at offset j, then this is also the case for S_{1-i}. Otherwise, running the `Test` algorithm on (C, td_W) would enable $\mathcal{A}$ to trivially win this experiment.

Although this kind of restriction is very common in predicate/functionnal encryption schemes (*e.g.* [27]), we stress that, in practice, one must take care that it does not lead to situations where security becomes meaningless. For example, if the adversary gets a trapdoor for every character $s \in \mathcal{S}$, then it will always fail the experiment (it will not be able to output two strings S_0 and S_1 complying with the requirement of step 6) while being able to decrypt any ciphertext (see Remark 1).

This example highlights the implicit restrictions placed on the set of trapdoors. This is obviously a limitation of the security model (that also applies to all predicate or searchable encryption schemes) but we believe that these restrictions are very hard to formalize and should rather be considered on a case-by-case basis. For example, in the context of DPI, the receiver could assess once and for all the set of rules to check that the leakage remains reasonable.

Selective-Indistinguishability (SEST-sIND-CPA). We also need a weaker security notion in which the adversary commits to S_0 and S_1 at the beginning of the experiment, before seeing pp and pk. Such a restriction is quite standard and is usually referred to as *selective* security [15].

Remark 2. We recall that in a public-key setting, it is always possible to recover W from td_W : one simply has to encrypt the $2^{|W|}$ strings of size $|W|$ and then run $\mathtt{Test}(., \mathsf{td}_W)$ on each resulting ciphertext. The correctness property ensures (with overwhelming probability) that one will always get an empty set, except for the encryption of W.

Therefore, unless we place restrictions on the set of keywords that one can query (in particular on its min-entropy, as in [10]), we cannot achieve relevant privacy notions for the trapdoor td_W itself. However, this is not a problem for, say, deep-packet inspection, in which many of the keywords can even be public [1].

Finally, we note that one can achieve interesting privacy notions for the trapdoors in the private-key setting (*e.g.* [13]).

3 Our Construction

We are able to construct our SEST scheme by "projecting" both the keyword and the plaintext onto a multiplicative basis of the type z^i for some secret integer z. We encrypt the plaintext character-by-character, using secret encodings α_s for each $s \in \mathcal{S}$. The latter are also used to generate the trapdoors associated with the keyword. By using a *bilinear mapping* we are able to shift into the ciphertext and compare a given fragment of suitable length to the trapdoor.

Note that in order to achieve the security notion of SEST-(s)IND-CPA, we need to at least guarantee that, given some trapdoors $\mathtt{td}_{W_i}$ for words W_i, the adversary is not able to forge a trapdoor for some fresh word W^*. By projecting keywords on a polynomial in a secret value z, we ensure that trapdoors on keywords W are essentially un-malleable.

We describe our construction in detail in what follows, prefacing our scheme by a brief introduction to bilinear groups and pairings.

3.1 Bilinear Groups

Bilinear groups are a set of three cyclic groups, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, of prime order p, along with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. for all $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{a \cdot b}$;
2. for any $g \neq 1_{\mathbb{G}_1}$ and $\widetilde{g} \neq 1_{\mathbb{G}_2}$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;
3. the map e is efficiently computable.

Galbraith, Paterson, and Smart [20] defined three types of pairings: in type 1, $\mathbb{G}_1 = \mathbb{G}_2$; in type 2, $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficient homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, while no efficient one exists in the other direction; in type 3, $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism exists between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction.

The security of our construction holds as long as no efficient homomorphism exists from $\mathbb{G}_1$ to $\mathbb{G}_2$. Our system must therefore be instantiated with pairings of type 2 or 3. However, in the following, we will only consider the latter type since it allows simpler security proofs thanks to the separation between the two groups $\mathbb{G}_1$ and $\mathbb{G}_2$. We stress that this is not a significant restriction since type 3 pairings offer the best performances among the three types.

3.2 Intuition

Intuitively, our scheme associates each element s of $\mathcal{S}$ with a secret encoding α_s. A trapdoor for a string $w_0 \dots w_{\ell-1}$ is associated with a polynomial $V =$

$\sum_{i=0}^{\ell-1} v_i \cdot \alpha_{w_i} \cdot z^i$ where v_i are random secret scalars whose purpose is to prevent forgeries of new trapdoors. The trapdoor then consists in the elements $\widetilde{g}^V$ and $\widetilde{g}^{v_i}$ for $i = 0, \ldots, \ell - 1$. In the meantime, a ciphertext encrypting a string $s_0 \ldots s_{n-1}$ is the sequence of "monomials" $C'_j = g^{a \cdot \alpha_{s_j} \cdot z^j}$ where a is a random factor (the Keygen algorithm will ensure that this can be done by only using elements from the public key). By using the bilinear map e, one can derive from the ciphertext and the trapdoor elements of the form $e(g, \widetilde{g})^U$ where U is a polynomial whose coefficients depends on the encodings α_{s_i} and on the scalars v_i.

In this encoding, if $s_0 \ldots s_{n-1}$ contains the pattern $w_0 \ldots w_{\ell-1}$ at offset j (*i.e.* if $s_{j+i} = w_i$ for $i = 0, \ldots, \ell - 1$) one can generate $e(g, \widetilde{g})^U = \prod_{i=0}^{\ell-1} e(C'_{j+i}, \widetilde{g}^{v_i})$ where $U = a \cdot z^j \cdot V$. Therefore, by extending the ciphertext with the elements $C_j = g^{a \cdot z^j}$, one can simply test the presence of W. By contrast, a difference $s_{j+i} \neq w_i$ or the combination of non-successive ciphertext elements will lead to a random-looking polynomial which would be useless to the adversary.

However, using this solution to search for a pattern of length ℓ within a string of length m requires $(\ell + 1)(m - \ell + 1)$ pairings, which quickly becomes prohibitive. While it seems natural that the complexity depends on the size m (since we have to search at every position), one could hope to reduce the factor $(\ell + 1)$.

A first attempt could be to set $v_i = v$ for all $i \in [0, \ell - 1]$ for some secret scalar v. Indeed, thanks to the bilinearity of e, the ℓ pairings $\prod_{i=0}^{\ell-1} e(C'_{j+i}, \widetilde{g}^{v_i})$ could be replaced by only one: $e(\prod_{i=0}^{\ell-1} C'_{j+i}, \widetilde{g}^v)$. Unfortunately, such a solution is insecure as proven by the following example.

Let C be a ciphertext encrypting a string $S = s_0 \ldots s_{m-1}$ and let us assume that W is a keyword such that $w_i = s$ for all $i \in [0, \ell - 1]$ (*i.e.* W is a sequence of identical values, equal to s). Then, for any $0 < j \leq \ell - 1$

$$e(C_0 \cdot C_j^{-1}, \widetilde{g}^{V_W}) = e(g, \widetilde{g})^{a(1-z^j)V_W} = e(g, \widetilde{g})^{aV'},$$

with

$$V' = \sum_{k=0}^{j-1} v \cdot \alpha_s \cdot z^k - \sum_{k=\ell}^{\ell+j-1} v \cdot \alpha_s \cdot z^k.$$

Therefore, $e(g, \widetilde{g})^{aV'}$ can be used to check whether

$$s_0 \ldots s_{j-1} = \overbrace{s \ldots s}^{j \text{ times}} \wedge s_\ell \ldots s_{\ell+j-1} = \overbrace{1 \ldots 1}^{j \text{ times}}.$$

Using td_W, a gateway is then able to get more information on S than the presence of W as a substring, which breaks the security of the construction.

However, this attack does not mean that we necessarily have to select different scalars v_i but simply that the generation process needs to be more subtle. We indeed prove that one can "recycle" the random elements v_i within the same trapdoor without jeopardizing security. More specifically, the issuing process that we describe in the next section is based on the observation that the secret encodings α_s already add some variability to the coefficients of the polynomial

V. This therefore means that this variability need not exclusively rely on the random scalars v_i. In particular when $w_i \neq w_j$, the coefficients $v_i \cdot \alpha_{w_i}$ and $v_j \cdot \alpha_{w_j}$ will be different even if $v_i = v_j$. In such a case, there is no need to chose distinct scalars, which allows us to batch the corresponding pairings for the test. Compared to the solution with random scalars v_i, this divides the whole number of pairings by up to $|\mathcal{S}|$ (*e.g.*, 256 if we consider bytestrings).

3.3 The Protocol

- $\texttt{Setup}(1^k, n)$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be the description of type 3 bilinear groups of prime order p, this algorithm selects $g \xleftarrow{\$} \mathbb{G}_1$ and $\widetilde{g} \xleftarrow{\$} \mathbb{G}_2$ and returns $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, n)$.
- $\texttt{Keygen}(\mathcal{S})$: On input a finite set $\mathcal{S}$, this algorithm selects $|\mathcal{S}| + 1$ random scalars $z, \{\alpha_s\}_{s\in\mathcal{S}}$ and computes $g_i \leftarrow g^{z^i}$ along with $\{g_i^{\alpha_s}\}_{s\in\mathcal{S}}$ for $i = 0, \dots, n-1$. The public key pk is set as $\{(g_i, \{g_i^{\alpha_s}\}_{s\in\mathcal{S}})\}_{i=0}^{n-1}$ whereas sk is set as $(z, \{\alpha_s\}_{s\in\mathcal{S}})$.
- $\texttt{Encrypt}(S, \mathsf{pk})$: To encrypt a string $S = s_0 \dots s_{m-1}$, where $m \leq n$ the user selects a random scalar a and returns $C = \{(C_i, C'_i)\}_{i=0}^{m-1}$, where $C_i \leftarrow g_i^a$ and $C'_i \leftarrow g_i^{a\cdot\alpha_{s_i}}$ for $i = 0 \dots m-1$.
- $\texttt{Issue}(W, \mathsf{sk})$: To issue a trapdoor td_W for a string $W = w_0 \dots w_{\ell-1}$ of length $\ell \leq n$, one uses the following algorithm.

```
$\texttt{Ind}[s] = 0$ for all $s \in \mathcal{S}$ ;
$L[i] = 0$ for all $i \in [0, \ell-1]$;
$V = 0$, $c = 0$;
for $i = 0, \dots, \ell-1$ do
    if $L[\texttt{Ind}[w_i]] = 0$ then
        $L[c] \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{I}_c \leftarrow \{i\}$;
        $c = c + 1$;
    else
        $\mathcal{I}_{\texttt{Ind}[w_i]} = \mathcal{I}_{\texttt{Ind}[w_i]} \cup \{i\}$;
    end
    $V = V + z^i \cdot \alpha_{w_i} \cdot L[\texttt{Ind}[w_i]]$;
    $\texttt{Ind}[w_i] = \texttt{Ind}[w_i] + 1$ ;
end
$\mathsf{td}_W \leftarrow (c, \{\mathcal{I}_j\}_{j=0}^{c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{c-1}, \widetilde{g}^V)$;
```
Algorithm 1: $\texttt{Issue}$

Our Issue algorithm formalizes the following principle: the random scalars (stored in L) can be re-used as long as the coefficients of the polynomial V are all distinct. In particular, if we write V as $\sum_{i=0}^{\ell-1} v_i \cdot \alpha_{w_i} \cdot z^i$, then $v_i \neq v_j$ if $w_i = w_j$.

– $\texttt{Test}(C, \texttt{td}_W)$: To test whether the string S encrypted by C contains the substring W, the algorithm parses $\texttt{td}_W$ as $(c, \{\mathcal{I}_j\}_{j=0}^{c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{c-1}, \widetilde{g}^V)$ and C as $\{(C_i, C'_i)\}_{i=0}^{m-1}$ and checks, for $j = 0, \ldots, m-\ell$, if the following equation holds:

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

It then returns the (potentially empty) set $\mathcal{J}$ of indexes j for which there is a match.

Correctness. First note that, if S contains the substring W at index j (*i.e.*, $s_{j+i} = w_i \ \forall i = 0, \ldots, \ell - 1$), then:

$$\begin{aligned}\prod_{t=0}^{c-1} e(\prod_{i\in\mathcal{I}_t} C'_{j+i}, \widetilde{g}^{L[t]}) &= \prod_{t=0}^{c-1} e(\prod_{i\in\mathcal{I}_t} g^{a\cdot\alpha_{s_{j+i}}\cdot z^{j+i}}, \widetilde{g}^{L[t]}) \\ &= \prod_{t=0}^{c-1} e(g^a, \widetilde{g}^{L[t]\cdot\sum_{i\in\mathcal{I}_t}\alpha_{w_i}\cdot z^{j+i}}) \\ &= \prod_{t=0}^{c-1} e(g^a, \widetilde{g}^{\sum_{i\in\mathcal{I}_t} L[t]\cdot\alpha_{w_i}\cdot z^{j+i}}) \\ &= e(g,\widetilde{g})^{a\cdot z^j\cdot V} = e(C_j, \widetilde{g}^V)\end{aligned}$$

The set $\mathcal{J}$ returned by $\texttt{Test}$ contains j.

Now, let us assume that $\mathcal{J}$ contains j but that $s_j \ldots s_{j+\ell-1} \neq w_0 \ldots w_{\ell-1}$, *i.e.*, the algorithm returns a false positive. Let $\mathcal{I}_{\neq}$ be the (non-empty) set of indexes i such that $s_{j+i} \neq w_i$. For all $i \in [0, \ell-1]$, we define $v_i = L[t_i]$ where t_i is such that $i \in \mathcal{I}_{t_i}$. Since j has been returned by $\texttt{Test}$, we have,

$$
\begin{aligned}
& \prod_{t=0}^{c-1} e\Big(\prod_{i\in\mathcal{I}_t} C'_{j+i}, \widetilde{g}^{L[t]}\Big) = e(C_j, \widetilde{g}^{V}) \\
\Leftrightarrow\quad & \prod_{i=0}^{\ell-1} e(C'_{j+i}, \widetilde{g}^{v_i}) = e(C_j, \widetilde{g}^{V}) \\
\Leftrightarrow\quad & \prod_{i\in\mathcal{I}_{\neq}} e(C'_{j+i}, \widetilde{g}^{v_i}) = e(C_j, \widetilde{g}^{\sum_{i\in\mathcal{I}_{\neq}} v_i\cdot\alpha_{w_i}\cdot z^i}) \\
\Leftrightarrow\quad & \prod_{i\in\mathcal{I}_{\neq}} e(g,\widetilde{g})^{a\cdot v_i\cdot\alpha_{s_{j+i}} z^{i+j}} = e(g,\widetilde{g})^{a\cdot z^j \sum_{i\in\mathcal{I}_{\neq}} v_i\cdot\alpha_{w_i}\cdot z^i} \\
\Leftrightarrow\quad & \sum_{i\in\mathcal{I}_{\neq}} v_i\cdot\alpha_{s_{j+i}} z^i = \sum_{i\in\mathcal{I}_{\neq}} v_i\cdot\alpha_{w_i}\cdot z^i \\
\Leftrightarrow\quad & \sum_{i\in\mathcal{I}_{\neq}} v_i(\alpha_{s_{j+i}} - \alpha_{w_i})\cdot z^i = 0.
\end{aligned}
$$

Since $\alpha_{s_{j+i}} \neq \alpha_{w_i}$ for all $i \in \mathcal{I}_{\neq}$, this amounts to evaluating the probability that a random scalar z is a root of a non-zero polynomial of degree at most $\ell-1$. The probability that `Test` returns a false positive j is thus at most $\frac{\ell-1}{p}$, which is negligible.

Remark 3. Our construction achieves the goals that we define at the beginning of Section 1.1. Indeed, the `Encrypt` procedure does not depend on the keywords W, and the latter may have distinct lengths. In particular, the size of C only depends on the length of the message it encrypts. Moreover, the trapdoors $\mathtt{td}_W$ allow to search the word W in $S = s_0 \dots s_{m-1}$ at any possible offset, while being of size independent of m.

All these features are provided using only asymmetric prime order bilinear groups, which can be very efficiently implemented on a computer (*e.g.*, [7]). We refer to Section 6 for a more thorough analysis of the efficiency of our protocol.

Remark 4. As explained in Section 2.1, public-key searchable encryption schemes often assume that the sender will also encrypt the string S by using a conventional encryption scheme Π. Such a solution enables fast decryption but should be used cautiously in some contexts, such as DPI, where the sender is likely to be malicious. Indeed, nothing prevents the latter from encrypting an harmless string S using the searchable encryption scheme while encrypting a different S' using Π. The message (S) checked by the gateway would then be different from the one forwarded to the receiver (S'), which would make the inspection pointless.

It is therefore necessary to check that both ciphertexts decrypt to the same string S, which can easily be done by the receiver. Indeed, after decrypting the conventional ciphertext, the latter (who knows sk) can verify whether $\{(C_i, C'_i)\}_{i=0}^{m-1}$ encrypts $S = s_0 \dots s_{m-1}$ by testing if $C'_i = C_i^{\alpha_{s_i}}$ for $i \in [0, m-1]$. One can also perform such tests only for a limited number $N \leq m$ of indexes i, but the probability of detecting cheating sender will become $\frac{N}{m}$.

4 Security Analysis

4.1 Complexity Assumptions

Let us consider an adversary $\mathcal{A}$ which, knowing q trapdoors $\mathtt{td}_{W_k}$, would like to decide if a ciphertext C encrypts S_0 or S_1. The natural restrictions imposed by the security model imply that there is at least one index i^* such that $s_{i^*}^{(0)} \neq s_{i^*}^{(1)}$ and that, for all $k \in [1, q]$ and all $j \in [0, \ell_k - 1]$ (where ℓ_k is the length of W_k), $s_{i^*-\ell_k+1+j}^{(0)} \dots s_{i^*+j}^{(0)}$ and $s_{i^*-\ell_k+1+j}^{(1)} \dots s_{i^*+j}^{(1)}$ both differ from $w_{k,0}, \dots, w_{k,\ell_k-1}$. In other words, any substring of S_0 (or respectively S_1) of length ℓ_k containing $s_{i^*}^{(0)}$ (resp. $s_{i^*}^{(1)}$) must be different from W_k, for all $k \in [1, q]$.

If we focus on the index i^*, $\mathcal{A}$ must then distinguish whether the discrete logarithm of C'_{i^*} in base g_{i^*} is $a \cdot \alpha_{s_{i^*}^{(0)}}$ or $a \cdot \alpha_{s_{i^*}^{(1)}}$. To this end, the attacker has access to many elements of $\mathbb{G}_1$ (the public parameters and the other elements of the ciphertext) and of $\mathbb{G}_2$ (the trapdoors $\mathtt{td}_{W_k}$). All of them are of the form $g^{P_u(a,\alpha_s,z)}$ or $\widetilde{g}^{Q_v(\alpha_s,z,v_{i,k})}$ for a polynomial number of multivariate polynomials P_u and Q_v. The assumption underlying the security of our scheme is thus related to the General Diffie-Hellman GDH problem [8], whose asymmetric version [12] is recalled below.

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

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

Let $(x_1, \dots, x_c)$ be a secret vector. The GDH assumption states that, given the values $\{g^{R^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{r}$, $\{\widetilde{g}^{S^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{s}$ and $\{e(g, \widetilde{g})^{T^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{t}$, it is hard to decide whether $U = g^{f(x_1,\dots,x_c)}$ or U is random if f is independent of $< \mathtt{R}, \mathtt{S}, \mathtt{T} >$.

Unfortunately, we cannot directly make use of this assumption unless we severely restrict the size n of the strings that one can encrypt. In our proof, presented in Section 4.2, one of the main important steps is showing that, even given a number of keyword trapdoors (and in particular, the polynomials V associated with those keywords), the adversary is unable to detect the presence of a fresh keyword; consequently, we can bound the leakage on the input plaintexts by only considering the adversary's queries to the issuing oracle. This can be mapped to an instance of GDH, but we will need the adversary to choose which of those polynomials are input to the GDH instance.

If we did bound the size n of the plaintext, by making a guess on the string $S_\beta = s_1^{(\beta)} \dots s_m^{(\beta)}$, one could define a GDH instance providing all the elements of the public parameters, the trapdoors for every word W that does not match any of the substrings of S_β containing $s_{i^*}^{(\beta)}$, the elements $\{g_i^a\}_{i=0}^{n-1}$ and $\{g_i^{a\cdot\alpha_{s_i}}\}_{i\in[0,n-1]\setminus\{i^*\}}$ along with the challenge element $U \in \mathbb{G}_1$ associated with the polynomial $f = a \cdot z^{i^*} \cdot \alpha_{s_{i^*}}$.

With such a GDH instance, the security proof becomes straightforward and only requires a proof that f does not depend on the polynomials underlying the provided elements. However, the reduction does not abort only if the initial guess is valid, which occurs with probability $\frac{1}{2^n}$.

So either we require n to be small (say $n \leq 30$, for example) or we choose to rely on an interactive variant of the GDH assumption, in which the elements $g^{R^{(i)}(x_1,\dots,x_c)}$, $\widetilde{g}^{S^{(i)}(x_1,\dots,x_c)}$ and $e(g,\widetilde{g})^{T^{(i)}(x_1,\dots,x_c)}$ can be queried to specific oracles, to offer enough flexibility to the simulator.

The latter solution is less than ideal because it essentially makes the GDH instance interactive and consequently our construction will end up offering less security than a static assumption. Nevertheless, we argue that this solution remains of interest for two reasons. The first is that it allows to construct a quite efficient scheme with remarkable features: the size of the ciphertext is independent of the ones of the searchable strings, and the size of the trapdoors is independent of the size of the messages. Achieving this while being able to handle any trapdoor query is not obvious and may justify the use of an interactive assumption.

A second reason is that, intrinsically, the hardness of the GDH problem (proven in the generic group model [8]) relies on the same argument as its interactive variant: as long as the "challenge" polynomial f does not depend on $< \mathtt{R}, \mathtt{S}, \mathtt{T} >$, $g^{f(x_1,\dots,x_c)}$ is indistinguishable from a random element of $\mathbb{G}_1$. The fact that the sets R, S, and T are defined in the assumption or by the queries to oracles does not fundamentally impact the proof. We therefore define the interactive-GDH (i-GDH) assumption and show that our scheme can be proven secure under it.

Definition 2 (i-GDH assumption.). *Let r, s, t, c, and k be five positive integers and $\mathtt{R} \in \mathbb{F}_p[X_1,\dots,X_c]^r$, $\mathtt{S} \in \mathbb{F}_p[X_1,\dots,X_c]^s$ and $\mathtt{T} \in \mathbb{F}_p[X_1,\dots,X_c]^t$ be three tuples of multivariate polynomials over $\mathbb{F}_p$. Let $\mathcal{O}^{\mathtt{R}}$ (resp. $\mathcal{O}^{\mathtt{S}}$ and $\mathcal{O}^{\mathtt{T}}$) be oracles that, on input $\{\{a^{(k)}_{i_1,\dots,i_c}\}_{i_j=0}^{d_k}\}_k$, add the polynomials $\{\sum_{i_1,\dots,i_c} a^{(k)}_{i_1,\dots,i_c} \prod_j X_j^{i_j}\}_k$ to R (resp. S and T).*

Let $(x_1,\dots,x_c)$ be a secret vector and $q_{\mathtt{R}}$ (resp $q_{\mathtt{S}}$) (resp. $q_{\mathtt{T}}$) 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 $\{g^{R^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{r+k\cdot q_R}$, $\{\widetilde{g}^{S^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{s+k\cdot q_S}$ and $\{e(g,\widetilde{g})^{T^{(i)}(x_1,\dots,x_c)}\}_{i=1}^{t+k\cdot q_T}$, it is hard to decide whether $U = g^{f(x_1,\dots,x_c)}$ or U is random if f is independent of $< \mathtt{R}, \mathtt{S}, \mathtt{T} >$.

4.2 Security Results

Theorem 3. *The scheme described in Section 3 is* SEST-sIND-CPA secure under the i-GDH assumption for $\mathtt{R}$, $\mathtt{S}$, and $\mathtt{T}$ initially set as $\mathtt{R} = \{(z^i, x_j \cdot z^i, a \cdot z^i)\}_{i=0,j=0}^{i=2n-1,j=|\mathcal{S}|-1}$, $\mathtt{S} = \mathtt{T} = \emptyset$ and $f = a \cdot x_0 \cdot z^n$.

Proof. Let $G_0^{(\beta)}$ denote the $\mathtt{Exp}_{\mathcal{A}}^{sind-cpa-\beta}$ game, as described in Section 2.2 – recall that this is the *selective* version of the IND-CPA security notion. Moreover, let $S_0 = s_0^{(0)} \dots s_{m-1}^{(0)}$ and $S_1 = s_0^{(1)} \dots s_{m-1}^{(1)}$ be the two substrings returned by $\mathcal{A}$ at the beginning of the game. Our proof uses a sequence of games $G_j^{(\beta)}$, for $j = 1, \dots, n$, 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.

Let $\mathcal{I}_{\neq}$ be the set of indexes i such that $s_i^{(0)} \neq s_i^{(1)}$ and $\mathcal{I}_{\neq}^{(j)}$ be the subset containing the first j indexes of $\mathcal{I}_{\neq}$ (if $j > |\mathcal{I}_{\neq}|$, then $\mathcal{I}_{\neq}^{(j)} = \mathcal{I}_{\neq}$). For $j = 1, \dots, n$, game $G_j^{(\beta)}$ modifies $G_0^{(\beta)}$ by switching the elements C'_i of the challenge ciphertext to random elements of $\mathbb{G}_1$, for $i \in \mathcal{I}_{\neq}^{(j)}$. Ultimately, in the last game, $G_n^{(\beta)}$, the challenge ciphertext contains no meaningful information about $s_i^{(\beta)}$ $\forall i \in \mathcal{I}_{\neq}$, so the adversary cannot distinguish whether it plays $G_n^{(0)}$ or $G_n^{(1)}$.

In particular, we can write:

$$\begin{aligned}
&\mathtt{Adv}_{\mathcal{A}}^{sind-cpa}(1^k, n) \\
&= |\Pr[\mathtt{Exp}_{\mathcal{A}}^{sind-cpa-1}(1^k, n)] - \Pr[\mathtt{Exp}_{\mathcal{A}}^{sind-cpa-0}(1^k, n)]| \\
&= |G_0^{(1)}(1^k, n) - G_0^{(0)}(1^k, n)| \\
&\leq \textstyle\sum_{j=0}^{n-1} |G_j^{(1)}(1^k, n) - G_{j+1}^{(1)}(1^k, n)| \\
&\quad + |G_n^{(1)}(1^k, n) - G_n^{(0)}(1^k, n)| \\
&\quad + \textstyle\sum_{j=0}^{n-1} |G_{j+1}^{(0)}(1^k, n) - G_j^{(0)}(1^k, n)| \\
&\leq \textstyle\sum_{j=0}^{n-1} |G_j^{(1)}(1^k, n) - G_{j+1}^{(1)}(1^k, n)| \\
&\quad + \textstyle\sum_{j=0}^{n-1} |G_{j+1}^{(0)}(1^k, n) - G_j^{(0)}(1^k, n)|.
\end{aligned}$$

In order to bound this result, we must prove that $\mathcal{A}$ cannot distinguish $G_j^{(\beta)}$ from $G_{j+1}^{(\beta)}$, which is formally stated by the lemma below.

Assuming that this lemma were proved, each term above is negligible under the i-GDH assumption, which concludes the proof.

Lemma 4. *For all $j = 0, \dots, n-1$ and $\beta \in \{0, 1\}$, the difference $|\mathtt{Pr}[G_j^{\beta}(1^k, n) = 1] - \mathtt{Pr}[G_{j+1}^{\beta}(1^k, n) = 1]|$ is negligible under the i-*GDH *assumption for* $\mathtt{R}$, $\mathtt{S}$, *and* $\mathtt{T}$ *initially set as follows:* $\mathtt{R} = \{(z^i, x_j \cdot z^i, a \cdot z^i)\}_{i=0,j=0}^{i=2n-1,j=|\mathcal{S}|-1}$, $\mathtt{S} = \mathtt{T} = \emptyset$ *and* $f = a \cdot x_0 \cdot z^n$.

The proof is provided in the full version [19].

5 Handling Regular Expressions

Our solution, introduced in Section 3, allows for pattern matching of keywords of arbitrary lengths, for ciphertexts emitted from arbitrary sources (we call this having universal tokens). In this section, we extend our notion of keyword-search to a more generic case, in which some of the keyword characters are fully-unknown (wildcards) and some are only partially-unknown (in an interval of size greater than 1).

Consider the general case in which one wants to search for substrings of the form $W = w_0 \ldots w_{t-1} w_t^{(\mathcal{S}_t)} w_{t+1} \ldots w_{\ell-1}$ where $w_t^{(\mathcal{S}_t)}$ denotes any element from the set $\mathcal{S}_t \subset \mathcal{S}$. For example, $\mathcal{S}_t$ can be the set [0-9] of all integers between 0 and 9.

A trivial solution could be to issue a trapdoor for every possible value of w_t but this would imply, for the gateway, to store the $|\mathcal{S}_t|$ resulting trapdoors and to test each of them separately. This not only raises a question of efficiency, but it also gives the gateway much more information on the input string. Intuitively, at the end of the search, the gateway will not only be able to tell that a given character is within a certain subset, but also which particular element of the subset it corresponds to.

In the following, we show how to modify our construction to allow for two notable regular expressions: wildcards and interval searches, without leaking any additional information, and with a minimal efficiency loss.

5.1 Handling Wildcards

The first case we consider assumes $W = w_0 \ldots w_{i_1}^{(\mathcal{S}_{i_1})} \ldots w_{i_r}^{(\mathcal{S}_{i_r})} \ldots w_{\ell-1}$ with $\mathcal{S}_{i_1} = \ldots = \mathcal{S}_{i_r} = \mathcal{S}$, which means that $w_{i_1}^{(\mathcal{S}_{i_1})}, \ldots, w_{i_r}^{(\mathcal{S}_{i_r})}$ can take any value from the set $\mathcal{S}$ and can consequently be seen as “wildcards”.

Informally, this implies that the $(j+i_1)$-th,...,$(j+i_r)$-th ciphertext elements must not be taken into account when testing if $C_j \ldots C_{j+\ell-1}$ encrypts W. This leads to the following variant of our main protocol where only the `Issue` and the `Test` algorithms differ (slightly) from the original ones.

- `Issue`(W, sk): Let $\mathcal{D} = \{i_1, \ldots, i_r\}$. The issuance process of a trapdoor $\mathtt{td}_W$ for $W = w_0 \ldots w_{i_1}^{(\mathcal{S}_{i_1})} \ldots w_{i_r}^{(\mathcal{S}_{i_r})} \ldots w_{\ell-1}$ is described by Algorithm 2.
  The only difference with the original `Issue` algorithm is the additional condition $i \notin \mathcal{D}$ which ensures that V will have no monomial of degree i for $i \in \mathcal{D}$.
- `Test`$(C, \mathtt{td}_W)$: this algorithm remains unchanged except that the trapdoor now contains the set $\mathcal{D}$. The process still consists of checking if the equality

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

  holds for $j = 0, \ldots, m - \ell$.

```
$\texttt{Ind}[s] = 0$ for all $s \in \mathcal{S}$ ;
$L[i] = 0$ for all $i \in [0, \ell - 1]$;
$V = 0,\ c = 0$;
for $i = 0, \ldots, \ell - 1$ do
    if $i \notin \mathcal{D}$ then
        if $L[\texttt{Ind}[w_i]] = 0$ then
            $L[c] \xleftarrow{\$} \mathbb{Z}_p,\ \mathcal{I}_c \leftarrow \{i\}$;
            $c = c + 1$;
        else
            $\mathcal{I}_{\texttt{Ind}[w_i]} = \mathcal{I}_{\texttt{Ind}[w_i]} \cup \{i\}$;
        end
        $V = V + z^i \cdot \alpha_{w_i} \cdot L[\texttt{Ind}[w_i]]$;
        $\texttt{Ind}[w_i] = \texttt{Ind}[w_i] + 1$ ;
    end
end
$\texttt{td}_W \leftarrow (c, \mathcal{D}, \{\mathcal{I}_j\}_{j=0}^{c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{c-1}, \widetilde{g}^V)$;
```

Algorithm 2: `Issue` supporting wildcards

One can note that this variant does not increase the complexity of our scheme. Actually, this is the opposite: all the indexes in $\mathcal{D}$ are discarded in the product of (1). Regarding security, one can note that the proof of Section 4 still applies here, since the latter does not require the coefficients v_i to be different from 0.

5.2 Handling General Subsets

Now let us consider the general case where the substring W one wants to search contains $w_i^{(\mathcal{S}_i)}$ for a subset $\mathcal{S}_i \subsetneq \mathcal{S}$. For example, $\mathcal{S}_i$ can be the set [0,9] of all the integers $x \in [0, 9]$ or the set $\{a, \ldots, z\}$ of the letters of the Latin alphabet. Our construction can actually be modified to handle this kind of searches provided that: 1) the searchable sets $\mathcal{S}_i$ are known in advance, and can be used during the `Keygen` process; and 2) all these subsets are disjoint. We argue that both conditions are reasonable since this is often the case for regular expressions.

5.3 The Protocol

- `Setup`$(1^k, n)$: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be the description of type 3 bilinear groups of prime order p, this algorithm selects $g \xleftarrow{\$} \mathbb{G}_1$ and $\widetilde{g} \xleftarrow{\$} \mathbb{G}_2$ and returns $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, n)$.
- `Keygen`$(\mathcal{S}, \mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(k)})$: This algorithm now takes as input k disjoint subsets of $\mathcal{S}$. We can assume, without loss of generality, that $\mathcal{S} = \mathcal{S}^{(1)} \cup \ldots \cup \mathcal{S}^{(k)}$ since we can simply add the complement of all previous sets if this is not the case. The function $f : \mathcal{S} \to \{1, \ldots, k\}$ which maps any element $s \in \mathcal{S}$ to the index of the set $\mathcal{S}^{(j)}$ which contains it is thus perfectly defined. The algorithm then selects $|\mathcal{S}| + k + 1$ random scalars $\{\alpha_s\}_{s \in \mathcal{S}}, \beta_1, \ldots, \beta_k, z \xleftarrow{\$}$

$\mathbb{Z}_p$ and computes $g_i \leftarrow g^{z^i}$ for $i = 0, \ldots, n-1$ along with $(g_i^{\alpha_s}, g_i^{\beta_d})$ for $d = 1, \ldots, k$ and all $s \in \mathcal{S}^{(d)}$. The public key is then set to $\{g_i\}_{i=0}^{n-1} \cup_{d=1}^{k} \{(g_i^{\alpha_s}, g_i^{\beta_d})\}_{i \in [0,n-1], s \in \mathcal{S}^{(d)}}$ and sk as $\{\alpha_s\}_{s \in \mathcal{S}}, \beta_1, \ldots, \beta_k, z$.

```
$\texttt{Ind}[s] = 0$ for all $s \in \mathcal{S}$ ;
$\texttt{Ind}'[k] = 0$ for all $k \in [0, d-1]$ ;
$L[i] = 0$ for all $i \in [0, \ell-1]$;
$V = 0$, $c = 0$;
for $i = 0, \ldots, \ell - 1$ do
    if $i \notin \mathcal{D}$ then
        if $L[\texttt{Ind}[w_i]] = 0$ then
            $L[c] \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{I}_c \leftarrow \{i\}$;
            $c = c + 1$;
        else
            $\mathcal{I}_{\texttt{Ind}[w_i]} = \mathcal{I}_{\texttt{Ind}[w_i]} \cup \{i\}$;
        end
        $V = V + z^i \cdot \alpha_{w_i} \cdot L[\texttt{Ind}[w_i]]$;
        $\texttt{Ind}[w_i] = \texttt{Ind}[w_i] + 1$ ;
    else
        if $L[\texttt{Ind}'[h(i) - 1]] = 0$ then
            $L[c] \xleftarrow{\$} \mathbb{Z}_p$, $\mathcal{I}_c \leftarrow \{i\}$;
            $c = c + 1$;
        else
            $\mathcal{I}_{\texttt{Ind}'[h(i)-1]} = \mathcal{I}_{\texttt{Ind}'[h(i)-1]} \cup \{i\}$;
        end
        $V = V + z^i \cdot \beta_{h(i)} \cdot L[\texttt{Ind}'[h(i) - 1]]$;
        $\texttt{Ind}'[h(i) - 1] = \texttt{Ind}'[h(i) - 1] + 1$ ;
    end
end
$\mathsf{td}_W \leftarrow (c, \mathcal{D}, \{\mathcal{I}_j\}_{j=0}^{c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{c-1}, \widetilde{g}^V)$;
```

Algorithm 3: `Issue` supporting general subsets

– `Encrypt`(S, pk): To encrypt a string $S = s_0 \ldots s_{m-1}$, where $m \leq n$ the user selects a random scalar a and returns $C = \{(C_i, C_i^{(1)}, C_i^{(2)})\}_{i=0}^{m-1}$, where $C_i \leftarrow g_i^a$, $C_i^{(1)} \leftarrow (g_i^{\alpha_{s_i}})^a$ and $C_i^{(2)} \leftarrow (g_i^{\beta_{f(s_i)}})^a$, for $i = 1 \ldots m$.

– To issue a trapdoor td_W for a string $W = w_1 \ldots w_{i_1}^{(\mathcal{S}_{i_1})} \ldots w_{i_r}^{(\mathcal{S}_{i_r})} \ldots w_\ell$ of length $\ell \leq n$, the algorithm first checks that all the involved subsets have been taken as input by the `Keygen` algorithm, *i.e.* $\mathcal{S}_{i_j} \in \{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(k)}\}$ for $j = 1, \ldots, r$, and returns $\perp$ otherwise. The function h which maps every index i_j to the integer $d \in \{1, \ldots, k\}$ such that $\mathcal{S}_{i_j} = \mathcal{S}^{(d)}$ is thus correctly defined. Let $\mathcal{D} = \{i_1, \ldots, i_r\}$, we modify the original `Issue` procedure as described in Algorithm 3.

– $\texttt{Test}(C, \texttt{td}_W)$: To test whether the string S encrypted by C contains the substring W, the algorithm parses $\texttt{td}_W$ as $(c, \mathcal{D}, \{\mathcal{I}_j\}_{j=0}^{c-1}, \{\widetilde{g}^{L[j]}\}_{j=0}^{c-1}, \widetilde{g}^V)$ and C as $\{(C_i, C_i^{(1)}, C_i^{(2)})\}_{i=0}^{m-1}$ and checks, for $j = 0, \ldots, m - \ell$, if the following equation holds:

$$\prod_{t=0}^{c-1} e\Big(\big(\prod_{i \in \mathcal{I}_t \wedge i \notin \mathcal{D}} C_{j+i}^{(1)}\big)\big(\prod_{i \in \mathcal{I}_t \wedge i \in \mathcal{D}} C_{j+i}^{(2)}\big), \widetilde{g}^{L[t]}\Big) = e(C_j, \widetilde{g}^V).$$

It then returns the set (potentially empty) $\mathcal{J}$ of indexes j for which there is a match.

The values β_j defined in this protocol can be seen as an encoding of the subset $\mathcal{S}^{(j)}$, in the same way as the scalars α_s encode the characters $s \in \mathcal{S}$. Actually, it is as if we worked with a larger set $\mathcal{S}'$ containing $\mathcal{S}$ but also the "characters" $\mathcal{S}^{(j)}$. The fact that one encrypts using both encodings makes the ciphertext compatible with any kind of trapdoors: if the i-th element of W is of the form w_j, we use $C_j^{(1)}$, whereas we use $C_j^{(2)}$ for an element of the form $w_j^{(\mathcal{S}_j)}$. Correctness and security follow directly from the original construction.

Regarding efficiency, encrypting for both encodings adds an element of $\mathbb{G}_1$ by character to the ciphertext. Nevertheless, as we explain in the next section, working with a larger set $\mathcal{S}'$ allows to reduce the number of random scalars that we need to generate the trapdoors, which leads to a faster $\texttt{Test}$ procedure.

6 The Complexity of our Scheme

We describe in this section the timings one can get for different parameters. But first we discuss the different strategies for choosing the set $\mathcal{S}$.

6.1 Generic Complexity

When considering data streams, the most relevant sets are the one of bits (*i.e.* $\mathcal{S} = \{0, 1\}$) or the one of bytes (*i.e.* $\mathcal{S} = \{0, \ldots, 255\}$). Larger sets (for example the one containing all sequences of r bytes for some $r > 1$) would improve the efficiency of the $\texttt{Test}$ procedure but would harm our ability to detect all patterns. We focus on four specific points: the sizes of (1) the public key, of (2) the ciphertext and of (3) the trapdoor along with (4) the number of pairings required to detect the presence of a pattern of size ℓ.

1. **The size of $\texttt{pk}$.** Let n be the maximum number of bytes one can encrypt with the protocol of section 3.3. If $\mathcal{S} = \{0, 1\}$, then the public key contains $(1+2)8n$ elements of $\mathbb{G}_1$ which amounts to $768n$ bytes using Barreto-Naehrig (BN) [6] curves. If we now consider bytestrings (*i.e.* $\mathcal{S} = \{0, \ldots, 255\}$), then $\texttt{pk}$ contains $(1 + 256)n$ elements of $\mathbb{G}_1$ which amounts to $8224n$ bytes using the same curves.

2. **The length of the ciphertext**. Each character is encrypted by 2 elements of $\mathbb{G}_1$ that represent 64 bytes. Therefore, encrypting m bytes requires $512m$ bytes if $\mathcal{S} = \{0, 1\}$ and $64m$ bytes if $\mathcal{S} = \{0, \ldots, 255\}$.
3. **The size of $\mathtt{td}_W$**. Our algorithm makes this evaluation much more difficult to perform. Indeed, the fact that we can reuse the same random scalar for two different characters $w_i \neq w_j$ implies that the size of $\mathtt{td}_W$ strongly depends on the keyword W itself. For example, a "constant" keyword $W = s \ldots s$ of size ℓ would entail a trapdoor containing $\ell + 1$ elements of $\mathbb{G}_2$. Conversely, a keyword $W = w_0 \ldots w_{\ell-1}$ with $w_i \neq w_j$ for $i \neq j$ would only require to store 2 elements of $\mathbb{G}_2$. Nevertheless, we notice that larger sets decrease the probability of having equal characters. More specifically, assuming uniform distribution of the characters within a keyword, a trapdoor contains, on average, $(1 + \lceil \ell/2 \rceil)$ elements of $\mathbb{G}_2$ if $\mathcal{S} = \{0, 1\}$ and only $(1 + \lceil \ell/256 \rceil)$ if $\mathcal{S} = \{0, \ldots, 255\}$. We can then hope to gain a factor 128 in the latter case.
4. **The number of pairings**. The number of pairings one must compute to test the presence of a keyword W of length ℓ within an encrypted string is related to the size of the corresponding trapdoor $\mathtt{td}_W$. More specifically, if $\mathtt{td}_W$ contains N elements of $\mathbb{G}_2$, then one must perform $N(m - \ell + 1)$ pairings, where m is the length of the encrypted string. Therefore, a shorter trapdoor implies a more efficient `Test` procedure, which means that it is better to work with $\mathcal{S} = \{0, \ldots, 255\}$ than with $\mathcal{S} = \{0, 1\}$.

Public key aside, we note that working on bytes instead of bits allows to significantly decrease complexity. Our timings then correspond to the case where $\mathcal{S} = \{0, \ldots, 255\}$.

6.2 Implementation of SEST for DPI

As we explain, evaluating the size of the trapdoors, and therefore the number of pairings requires to make assumptions about the distribution of the keywords. Previous estimations assumed a uniform distribution of the latter, which is unlikely in practice. We therefore evaluate our protocol on the SNORT public rules set [1] to provide a more concrete estimation[7].

The SNORT rules set contains thousands of rules which mostly consist in searching some specific patterns in a stream. We parsed all these rules and got 6048 different patterns. Figure 3 describes the sizes of the corresponding trapdoors.

This table highlights the advantage of our issuing protocol: even for large patterns we manage to keep most of the time short trapdoors thanks to the re-use (when possible) of the random scalars. The whole trapdoors set thus only amounts to 1.35 MB.

[7] We stress that the only goal of this section is to provide timings on a concrete and non-artificial set of patterns. We chose the DPI use-case for which searching on encrypted streams is particularly relevant. But we obviously do not claim that our solution is practical enough to handle all Internet traffic worldwide.

Trapdoors of size	2	3	4	5	6	7	8	9	10	11	12	13	14	15	18	23	27
Number	2067	1879	705	745	361	140	69	32	20	19	3	2	1	2	1	1	1

Fig. 3. Number of trapdoors of size N, where N is the number of elements of $\mathbb{G}_2$. In other words, among the 6048 trapdoors generated for the SNORT rules set, 2076 contain 2 elements of $\mathbb{G}_2$, 1879 contain 3 elements of $\mathbb{G}_2$, and so on.

Since the number of pairings is related to the size of the trapdoors, one could try to deduce from this table the total number of pairings required to test all SNORT patterns. However, we stress that this would only be a quite inaccurate upper bound. First, because many of these patterns are part of the same rule which enables to avoid unnecessary tests: if there is no match for a pattern defined by a rule, then it is pointless to test the other ones within the same rule. Second, because many rules include parameters called "depth", "offset", "distance" or "within" which allow to reduce the search to a smaller part of the stream.

The number of pairings for the whole SNORT rules set is thus significantly smaller than the one we could expect from the complexity evaluation we provide in Section 6.1. Moreover, we recall that the optimal Ate pairing [34] that we use to instantiate the map e can be split into two parts that are usually called the *Miller loop* and the *final exponentiation*. The latter, which roughly represents half of the computational cost of a pairing, can be performed once for all the pairings involved in the same equality test, which allows to further reduce the complexity of the `Test` procedure.

We ran an experiment on a stream of 1500 bytes using a computer running Linux 4.13 and equipped with an Intel E5-1620 3.70GHz processor. Testing all Snort rules took 28 minutes. This is obviously too much for online analysis but we stress that alternatives (*e.g.* FHE) offering the same features would be even more complex. Moreover, this corresponds to testing thousands of patterns on a single computer: by using parallelization and more powerful hardware, one could hope to dramatically reduce these timings.

Finally, we provide in Figure 4 the timings of the `Encrypt` and the `Test` algorithms for larger strings (up to 30 KB). It shows that encryption remains quite efficient even for large strings. The `Test` algorithm is obviously slower since it implies pairings computations but it takes (approximatively) only one second for strings of few kilobytes.

String length (B)	1500	3000	5000	10000	30000
`Encrypt` (s)	0.08	0.15	0.27	0.5	1.5
`Test` (s)	0.6	1.2	1.6	3.3	11.1

Fig. 4. Timings for encrypting a string of m bytes and searching a pattern of 100 bytes within it.

7 Conclusion

In this work, we introduced the concept of searchable encryption with shiftable trapdoors (SEST). This type of construction provides a practical solution to the generic problem of pattern matching with universal tokens. Notably, we are the first to provide a searchable encryption alternative that allows for arbitrarily-chosen keywords of arbitrary length, which can be applied to any ciphertext encrypted with the generated public key in this system. In particular, since we do not rely on symmetric keys, multiple entities can use the same public key to encrypt. Moreover, our construction is also highly usable for encrypted *streams* of data (we need no backtracking), and it returns the exact position at which the pattern occurs. Our instantiation of the SEST primitive uses bilinear pairings, and we allow for some regular expressions such as wildcards, or partial keywords in which we know some entries to be within a given interval.

Beyond applications in deep-packet inspection, the fact that our algorithm essentially follows the approach of Rabin-Karp allows us to also use that same algorithm for application scenarios such as searching on structured data, matching subtrees to labelled trees, delegated searches on medical data (compiled from multiple institutions), or 2D searches.

We propose a main construction, which we adapt to accounting for wildcards and for interval searches. The former adaptation is relatively simple, since the issued trapdoor just contains zero coefficients for the wildcards. For the interval searches we need to modify our key generation algorithm, providing special elements that we map interval characters to; however, this only works for intervals which are known in advance.

Our scheme provides trapdoors for the keywords which are at most linear in the size of the keywords *only*, and the size of the ciphertexts is linear in the size of the plaintext size. Although our public keys are large (linear in the size of the maximal plaintext size), we do achieve a complete decorrelation between the plaintext encryption and the trapdoor generation for the keywords. Our scheme provides in practice an almost linear – in the size of the plaintext – complexity (in terms of the number of pairings). Our implementation results for the publicly-given SNORT rules show that while the encryption algorithm scales well with the plaintext size, the testing algorithm – which is slower – will benefit from the fact that it is fully parallelizable.

We prove the security of our scheme under an interactive version of the GDH assumption. Our modification of this assumption is relatively minor, allowing the adversary to choose on which input to play the GDH instance. We also argue that our construction offers an interesting tradeoff between the secure, but quite cumbersome, systems based on existing cryptographic primitives and the fast, but unsecure, current solutions where the gateway decrypts the traffic. Moreover, we hope that the practical applications of this primitive will incite new work on this subject, in particular to construct new schemes which would rely on standard assumptions.

Acknowledgments

Nicolas Desmoulins and Olivier Sanders were supported in part by the French ANR Project ANR-16-CE39-0014 PERSOCLOUD. Pierre-Alain Fouque and Cristina Onete are grateful for the support of the ANR through project 16 CE39 0012 (SafeTLS).

References

1. https://www.snort.org/.
2. 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. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 205–222. Springer, August 2005.
3. Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015*, LNCS, pages 733–751. Springer, 2015.
4. Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 500–518. Springer, August 2013.
5. Joshua Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. 5PM: Secure pattern matching. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 222–240. Springer, September 2012.
6. 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, August 2006.
7. Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, pages 21–39. Springer, December 2010.
8. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, May 2005.
9. 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, May 2004.
10. 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, August 2013.
11. 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, February 2007.
12. Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56. Springer, September 2008.

13. Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. LNCS, pages 306–324. Springer, 2015.
14. 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, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 561–574. ACM, 2017.
15. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, May 2003.
16. Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, December 2010.
17. Melissa Chase and Emily Shen. Substring-searchable symmetric encryption. *PoPETs*, 2015(2):263–281, 2015.
18. 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 06*, pages 79–88. ACM Press, October / November 2006.
19. Nicolas Desmoulins, Pierre-Alain Fouque, Cristina Onete, and Olivier Sanders. Pattern matching on encrypted streams (full version). *IACR Cryptology ePrint Archive*, 2017:148, 2017.
20. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
21. Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Automata evaluation and text search protocols with simulation-based security. *J. Cryptology*, 29(2):243–282, 2016.
22. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
23. Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of Cryptology*, 23(3):422–456, July 2010.
24. Lin-Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. Analyzing forged SSL certificates in the wild. In *2014 IEEE Symposium on Security and Privacy*, pages 83–97. IEEE Computer Society Press, May 2014.
25. Jeff Jarmoc. SSL/TLS interception proxies and transitive trust. *Presentation at Black Hat Europe*, 2012.
26. Jonathan Katz and Lior Malka. Secure text processing with applications to private DNA matching. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10*, pages 485–492. ACM Press, October 2010.
27. 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.
28. Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In *Proceedings of Latincrypt*, volume 8895 of *LNCS*, pages 3–27. Springer-Verlag, 2014.
29. Payman Mohassel, Salman Niksefat, Seyed Saeed Sadeghian, and Babak Sadeghiyan. An efficient protocol for oblivious DFA evaluation and applications.

In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 398–415. Springer, February / March 2012.

30. David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of SIGCOMM 2015*, pages 199–212. ACM, 2015.
31. 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. ACM, August 2015.
32. 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.
33. Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 519–528. ACM Press, October 2007.
34. Frederik Vercauteren. Optimal pairings. *IEEE Trans. Information Theory*, 56(1):455–461, 2010.