Pattern Matching on Encrypted Streams:
Applications to DPI and searches on genomic data

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

[1] Orange Labs, Applied Crypto Group, Cesson-Sévigné, France
[2] Université de Rennes 1, IRISA, Rennes, France
[3] Université de Rennes 1, IRISA & Institut Universitaire de France, Rennes, France

Abstract. Pattern matching is essential to applications such as filtering content in Data streams, searching on genomic data, and searching for correlation in medical data. However, increasing concerns of user and data privacy, exacerbated by threats of mass surveillance, have made the use of encryption practically standard for personal data. Hence, entities performing pattern-matching on data they do not own must now be able to provide the functionality of keyword-search on *encrypted* data.

Existent solutions in searchable encryption suffer from one of two main disadvantages: either an exhausted list of keywords needs to be hard-coded in the input ciphertexts, or the input must be *tokenized*, massively increasing the size of the ciphertext. In both cases, the symmetric-key approach provides faster encryption, but also induces a token-regeneration step at each instantiation (*i.e.*, essentially, for each user). Such approaches are not well-suited when either the data owner is unable to choose all the relevant keywords, or when a single searcher (*e.g.*, an IDS, a firewall, or an independent medical researcher) must screen ciphertexts from many different ownerships. Fast symmetric searchable encryption alternatives (SSE) also come with an extensive leakage, which is not well understood and has recently been under attack.

In this work, we introduce Searchable Encryption with Shiftable Trapdoors (SEST), a new primitive, which allows for pattern matching with universal tokens, *i.e.*, trapdoors which can function on ciphertexts produced by multiple entities, and which allow to match keywords of arbitrary lengths to arbitrary ciphertexts. Our approach relies on a public-key encryption scheme and on bilinear pairings. We essentially project the plaintext bit by bit on a multiplicative basis consisting of powers of a secret key. The keyword is also projected on the same basis, with the order of its bits encoded as a polynomial of degree equal to the keyword length. The searching entity receives unforgeable trapdoors for requested keywords, and can match these against the input ciphertexts, thus finding out whether the pattern matched, and at what position of the plaintext the keyword can be found. 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 (namely wildcards or interval/subset searches).

Our scheme is a variation of Rabin-Karp and has many potential applications in deep-packet inspection on encrypted streams, searching on

genomic data, as well as searching on encrypted structured data. Compared to other alternatives in the literature, our trapdoor size is only 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, namely the ability to distinguish based on different search results of a single trapdoor on two different plaintexts. 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

In many applications, such as searching on genomic data, deep-packet inspection, or delegatable searches in databases, it is desirable to be able to match a given pattern to a larger input, and learn whether that pattern occurs or not, and where in the input this happens. In such cases, the entity performing the search is called the *gateway* and it is only semi-trusted by the owner of the input data. This is the case, for instance, when a middlebox – such as a firewall, or an IDS – must inspect incoming and outgoing traffic between multiple sets of independent peers, or when a single piece of recovered DNA must be matched against large-scale genomic databases (in which each genome is owned by the physical person to which it corresponds).

In such cases, pattern matching on plaintext data can have potentially harmful consequences, and raises trust issues. In other words, a user may trust an anti-virus scanner to scan its full data for viruses, but they may not want that the full content of its data be disclosed to this entity. Such concerns have been lately exacerbated by threats of mass-surveillance, following the revelations of Edward Snowden. As a consequence, data *encryption* is slowly becoming the standard approach to protecting personal information. Hence, pattern matching must now be performed on encrypted data.

In cryptography, this research area is closely related to that of Symmetric Searchable Encryption [SWP00,BDOP04,CGKO06,CK10a,CS15]. Though we review such solutions in more detail later in this section, we mention that though SSE is quite efficient even for large input, it comes with a lot of unnecessary leakage that is easily exploitable, see *e.g.*, [CGPR15,ZKP16]. In addition, many of the proposed scenarios only allow for searches for pre-chosen keywords, which are hard-coded in the encrypted input. Thus, it is impossible to do pattern-matching with arbitrary keywords.

Some further symmetric-key solutions to this problem require so-called *tokenization* [SLPR15]. In particular, a sliding-window technique is used to encode key-words 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. In this paper, we focus on extending this solution to public-key settings, and in particular, we achieve secure pattern-matching on encrypted data with *universal tokens*.

We review further related work in Section 1.2.

1.1 Our contributions

For our construction, the encryption process does not depend on the searchable keywords (even if the latter are of different sizes) and the size of the trapdoors does not depend on the length of encrypted bitstrings. It is thus well-suited for the cases of DPI, of pattern-matching on genomic data, and on delegated searches on medical data. We also support regular expression such as the presence of wildcards or matching encrypted input to general data-subsets.

Intuitively, in our construction we use public-key searchable encryption and *project* each coordinate of the plaintext B (and then of the keyword W) on a geometric basis consisting of some values z^i, for $i = 1, \ldots, |B|$. 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 then also provide a 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 bitstring B beyond the presence of the keyword W_k in B, 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.* [BGM+10])). Encryption of plaintexts B only requires operations in the group $\mathbb{G}_1$, while detection of the keyword W is done by performing pairings. Both operations require only the public key; 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-compatbile 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 Rabin-Karp and consequently, we can use it to solve similar problems. In addition to the three presented use-cases, our technique can also be used to perform 2D pattern

matching in images, or searching subtrees in a 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.

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 [SLPR15], 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 $\ell+1$ pairing computations per each position in the ciphertext (with $n-\ell$ shifts, the middlebox needs to compute $(n-\ell)(\ell+1)$ 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 bitstring. We modify the GDH assumption 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 bitstrings 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 [Jar12,HREJ14,SLPR15].

1.2 Related work

Pattern matching. Symmetric Searchable Encryption (SSE) enables pattern matching on encrypted data [SWP00,BDOP04,CGKO06,CK10a,CS15]. A peculiarity of the latter is that the encryption is done with a symmetric key; it is required, upon encryption, to select key-words which can be later searched, *i.e.*, the space of possible patterns to match. However, such approaches are not always suitable to deep-packet inspection on data streams, nor on arbitrary matching of (relatively) short patterns to much larger genomic data, since in those cases

it is difficult to hard-code the exhaustive list of all possible keywords in the encryption of the input data. We review in more detail how (S)SE works in the following paragraph.

Searchable encryption is not limited to database searches – which is the pet example in SSE. In particular, Chase and Kamara [CK10b] showed how to generalize SSE to searching on encrypted structured data (such as web graphs and social networks). Unfortunately, this work inherits some of the disadvantages in SSE, such as more extensive leakage than that given by the queries themselves. In our case, we do not seek to protect the privacy of our keyword, just to make trapdoors for fresh keywords unforgeable without the secret key; as a result, the privacy we obtain is more straight-forward to quantify and understand.

How searchable encryption works. In searchable encryption (SE), any party that is given a trapdoor 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 or running an antivirus on encrypted traffic are typical applications which require such a functionality.

In general, 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 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".

The solution proposed in [SLPR15] to search keywords of length ℓ is to split the string $B = b_1 \dots b_n$ into $[b_1 \dots b_\ell]$, $[b_2 \dots b_{\ell+1}]$,...,$[b_{n-\ell+1} \dots b_n]$ and then to encrypt each of these substrings using a searchable encryption scheme (the substrings are thus the keywords associated with B). 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 B 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 B. Moreover, repeating this procedure for every keyword W_k will lead 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 B with significant probability.

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 [Gen09], functional encryption [AGVW13,ABDP15],...), supporting a wide class of functions, have been proposed but they suffer from a rather high complexity. 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 [BEM+12,MNSS12,HL10,GHS16,KM10,TPKC07,LLN14], 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. [LLN14], which presents an application to genomic data. The authors here go much farther 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.* [KSW13]) or hidden vector encryption [BW07] 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 $b_1, \ldots, b_n$ which together make up a word B, while the middlebox, knowing the suitable secret keys, could detect whether B contains a substring W. The encryption process would then not depend on the searchable keywords and the anonymity property of these schemes would ensure that the ciphertext does not leak more information on B.

However, another issue arises with this solution. Indeed, $W = w_1 \ldots w_\ell$ can be contained at any position in B. Therefore, the gateway should receive the secret keys for $(w_1, \ldots, w_\ell, *, \ldots, *)$, $(*, w_1, \ldots, w_\ell, *, \ldots, *)$,...,$(*, \ldots, *, w_1, \ldots, w_\ell)$, 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 bitstrings B.

DPI with multi-context key-distribution. Naylor et al. [NSV+15] 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 messaged 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.

2 Searchable Encryption with Shiftable Trapdoors

We begin by presenting the syntax of our searchable encryption scheme with shiftable trapdoors. Note that in addition to producing a bit indicating whether the keyword was found in the (encrypted) plaintext, this scheme also outputs the position(s) at which the keyword is found. Although not apparent in the syntax, this is one advantage of shiftable trapdoors, namely yielding the exact position, within the target plaintext, of the search word.

2.1 Syntax

A searchable encryption 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.

- $\texttt{Setup}(1^k, n)$: This probabilistic algorithm takes as input a security parameter k and an integer n defining the maximum size of the bitstrings 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 then be considered as an implicit input to all algorithms and so will be omitted.
- $\texttt{Keygen}(n)$: 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 a bitstring W of any size $0 < \ell \leq n$, along with the receiver's secret key, and returns a trapdoor td_W.
- $\texttt{Encrypt}(B, \mathsf{pk})$: This probabilistic algorithm takes as input the receiver's public key along with a bitstring B of size $0 < m \leq n$ and returns a ciphertext C.
- $\texttt{Test}(C, \mathsf{td}_W)$: This deterministic algorithm takes as input a ciphertext C encrypting a bitstring $B = b_1 \dots b_m$ of size m along with a trapdoor td_W for a bitstring $W = w_1 \dots w_\ell$ of size ℓ. If $m > n$ or $\ell > m$, then the algorithm returns $\perp$. Else, the algorithm returns a set (potentially empty) $\mathcal{J} \subset \{0, \dots, m - \ell\}$ of indexes j such that $b_{j+1} \dots b_{j+\ell} = w_1 \dots w_\ell$.

Remark 1. Notice that searchable encryption, *e.g.*, [ABC+05,BW07], usually does not consider a decryption algorithm which takes as input sk and a ciphertext C encrypting B and which returns B. Indeed, this functionality can easily be added by also encrypting B under a conventional encryption scheme. Nevertheless, one can note that decryption can be performed by issuing a trapdoor for the bit 0 and running the $\mathtt{Test}$ algorithm on C. From the returned set $\mathcal{J}$ one can easily recover B, since $b_j = 0 \iff j - 1 \in \mathcal{J}$.

2.2 Security Model

Correctness. As in [ABC+05], we divide correctness into two parts. The first one stipulates that the $\mathtt{Test}$ algorithm run on (B, td_W) will always return j if B contains the substring W at index j. More formally, this means that, for any bitstring B of size $m \leq n$ and any bitstring W of length $\ell \leq m$:

$$b_{j+1} \dots b_{j+\ell} = w_1 \dots w_\ell \Rightarrow \mathtt{Pr}[j \in \mathtt{Test}(\mathtt{Encrypt}(B, \mathsf{pk}), \mathtt{Issue}(W, \mathsf{sk}))] = 1,$$

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 $\mathtt{Test}$ algorithm returns j despite the fact $b_{j+1} \dots b_{j+\ell} \neq w_1 \dots w_\ell$) only occur with negligible probability. More formally, this means that, for any bitstring B of size $m \leq n$ and any bitstring W of length $\ell \leq m$:

$$\mathtt{Pr}[j \in \mathtt{Test}(\mathtt{Encrypt}(B, \mathsf{pk}), \mathtt{Issue}(W, \mathsf{sk})) \wedge b_{j+1} \dots b_{j+\ell} \neq w_1 \dots w_\ell] \leq \mu(k)$$

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 $\mathcal{O}\mathtt{Issue}$ which returns a trapdoor td_W for any queried bitstring W, can decide whether a ciphertext C encrypts B_0 or B_1 as long as the trapdoors issued by the oracle do not allow trivial distinction of these two bitstrings. This is formally defined by the experiment $\mathtt{Exp}_{\mathcal{A}}^{ind-cpa-\beta}(1^k, n)$, where $\beta \in \{0, 1\}$ as described in Figure 1. The set $\mathcal{W}$ is the set of all the bitstrings W submitted to $\mathcal{O}\mathtt{Issue}$.

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

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

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

1. $pp \leftarrow \mathtt{Setup}(1^k, n)$
2. $\mathsf{pk} \leftarrow \mathtt{Keygen}(n)$
3. $(B_0, B_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathtt{Issue}}}(\mathsf{pk})$, with $B_i = b_1^{(i)} \dots b_m^{(i)}$ for some $m \leq n$
4. $C \leftarrow \mathtt{Encrypt}(B_\beta, \mathsf{pk})$
5. $\beta' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathtt{Issue}}}(C, \mathsf{pk})$
6. If $\exists W = w_1 \dots w_{\ell_W} \in \mathcal{W}$ and j such that:

$$b_{j+1}^{(i)} \dots b_{j+\ell_W}^{(i)} = w_1 \dots w_{\ell_W} \neq b_{j+1}^{(1-i)} \dots b_{j+\ell_W}^{(1-i)}$$

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

Fig. 1. SEST-IND-CPA Security Game

Selective-Indistinguishability (SEST-sIND-CPA). We also need a weaker security notion in which the adversary commits to B_0 and B_1 at the beginning of the experiment, before seeing pp and pk. Such a restriction is quite classical and is usually referred to as *selective* security [CHK03].

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 bit-by-bit, using a value α_1 to encrypt any 1-bit and α_2 to encrypt bits that equal 0. The same two values are used to encrypt the bits of the keyword. By using a *bilinear mapping* we are able to shift into the ciphertext, which is encrypted bit-by-bit, and compare a given fragment of suitable length to the encrypted keyword.

Note that in order to achieve the security notion of SEST-(s)IND-CPA, we need to at least guarantee that, given some trapdoors 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 [GPS08] 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.

Our construction will make use of type 3 pairings and its security will strongly rely on the lack of efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We therefore stress that it *must not* be instantiated with another type of pairings.

3.2 The Protocol

- $\mathtt{Setup}(1^k, n)$: 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 $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, n)$.
- $\mathtt{Keygen}(n)$: This algorithm selects three random scalars $\alpha_1, \alpha_2, z \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i \leftarrow g^{z^i}$ along with $(g_i^{\alpha_1}, g_i^{\alpha_2})$ for $i = 1, \ldots, n$. The public key pk is then set as $\{(g_i, g_i^{\alpha_1}, g_i^{\alpha_2})\}_{i=1}^n$ whereas sk is set as (α_1, α_2, z). One can note that $g_0 = g$ is provided in the public parameters pp.
- $\mathtt{Issue}(W, \mathsf{sk})$: To issue a trapdoor td_W for a bitstring $W = w_1 \ldots w_\ell$ of length $\ell \leq n$, the algorithm selects ℓ random scalars $v_i \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\widetilde{g}^V$, where $V = \sum_{i=1}^{\ell} v_i \cdot \alpha_1^{w_i} \cdot \alpha_2^{1-w_i} \cdot z^i$, along with $\widetilde{g}^{v_i}$, for $i = 1, \ldots, \ell$. The trapdoor td_W is then set as $(\widetilde{g}^V, \{\widetilde{g}^{v_i}\}_{i=1}^{\ell})$.
- $\mathtt{Encrypt}(B, \mathsf{pk})$: To encrypt a bitstring $B = b_1 \ldots b_m$, where $m \leq n$ the user selects a random scalar a and returns $C = \{(C_i, C_i')\}_{i=1}^m$, where $C_i \leftarrow g_{i-1}^a$ and $C_i' \leftarrow g_i^{a \cdot \alpha_1^{b_i} \cdot \alpha_2^{1-b_i}}$ for $i = 1 \ldots m$. The element C_i' is thus computed as $(g_i^{\alpha_1})^a$ if $b_i = 1$ and as $(g_i^{\alpha_2})^a$ otherwise.
- $\mathtt{Test}(C, \mathsf{td}_W)$: To test whether the string B encrypted by C contains the substring W, the algorithm parses td_W as $(\widetilde{g}^V, \{\widetilde{g}^{v_i}\}_{i=1}^{\ell})$ and C as $\{(C_i, C_i')\}_{i=1}^m$ and checks, for $j = 0, \ldots, m - \ell$, if the following equation holds:

$$\prod_{i=1}^{\ell} e(C'_{j+i}, \widetilde{g}^{v_i}) \stackrel{?}{=} e(C_{j+1}, \widetilde{g}^V).$$

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

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

$$\begin{aligned}
\prod_{i=1}^{\ell} e(C'_{j+i}, \widetilde{g}^{v_i}) &= \prod_{i=1}^{\ell} e(g_{i+j}^{\alpha_1^{b_{j+i}} \cdot \alpha_2^{1-b_{j+i}}}, \widetilde{g}^{v_i}) \\
&= \prod_{i=1}^{\ell} e(g_{i+j}^{a \cdot \alpha_1^{w_i} \cdot \alpha_2^{1-w_i}}, \widetilde{g}^{v_i}) \\
&= e(g, \widetilde{g})^{\sum_{i=1}^{\ell} a \cdot z^{i+j} \cdot \alpha_1^{w_i} \cdot \alpha_2^{1-w_i} \cdot v_i} \\
&= e(g_j^a, \widetilde{g}^{\sum_{i=1}^{\ell} z^i \cdot \alpha_1^{w_i} \cdot \alpha_2^{1-w_i} \cdot v_i}) \\
&= e(C_{j+1}, \widetilde{g}^V).
\end{aligned}$$

Therefore, the set $\mathcal{J}$ returned by `Test` contains j.

Now, let us assume that $\mathcal{J}$ contains j but that $b_{j+1} \ldots b_{j+\ell} \neq w_1 \ldots w_\ell$, *i.e.*, the algorithm returns a false positive. This means that:

$$\sum_{i=1}^{\ell} z^i \cdot \alpha_1^{w_i} \cdot \alpha_2^{1-w_i} \cdot v_i = \sum_{i=1}^{\ell} z^i \cdot \alpha_1^{b_{j+i}} \cdot \alpha_2^{1-b_{j+i}} \cdot v_i.$$

Let $\mathcal{I}_{\neq}$ be the (non-empty) set of indexes i such that $b_{j+i} \neq w_i$. The previous equation becomes

$$\sum_{i \in \mathcal{I}_{\neq}} (-1)^{b_{j+i}} \cdot z^i \cdot (\alpha_1 - \alpha_2) \cdot v_i = 0.$$

For randomly generated scalars $\alpha_1, \alpha_2, z, \{v_i\}_{i=1}^{\ell}$, this is very unlikely (the probability is smaller than $\frac{\ell}{p}$).

Remark 2. Our construction achieves the goals that we define at the beginning of Section 1.1. Indeed, the `Encrypt` procedure does not depend on the searchable words W, even if the latter are of distinct lengths. In particular, the size of C only depends on the length of the message it encrypts. Moreover, the trapdoors $\mathtt{td}_W$ allows to search the word W in $B = b_1 \ldots b_m$ 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.*,[BGM+10]).

Remark 3. We chose to consider bitstrings in our protocol but we stress that it can easily be tailored to any string $s_1 \ldots s_m$ where $s_i \in \mathcal{S}$ for some finite set $\mathcal{S}$: one must simply generates $|\mathcal{S}|$ scalars $\alpha_1, \ldots, \alpha_{|\mathcal{S}|}$ and associates each of them to an element of $\mathcal{S}$.

3.3 Intuition

Before analyzing the security of our construction we provide the intuition behind it. Informally, the idea is to associate the value 1 (respectively 0) to a secret scalar

α_1 (respectively α_2) and then to encrypt each bit b_i of B by computing $C'_i = g_i^{a \cdot x_i}$, for a random scalar a, where $x_i = \alpha_1$ if $b_i = 1$ and α_2 otherwise. Therefore, if one combines ℓ elements C'_i by computing $\prod_{j=1}^{\ell} C'^{u_j}_{i_j}$ for some scalars u_j, one gets g^U with $U = \sum_{j=1}^{\ell} u_j \cdot x_{i_j} z^{i_j}$. This property and the bilinearity of the pairing are at the heart of our construction. Indeed, if we provide, for a keyword $W = w_1 \dots w_\ell$, the element $\widetilde{g}^V \in \mathbb{G}_2$ where V is a polynomial of the form $\sum_{j=1}^{\ell} v_j \cdot x'_j \cdot z^j$, with $x'_j = \alpha_1^{w_j} \cdot \alpha_2^{1-w_j}$, then one could use the pairing properties to check whether $U = V$.

However, there are still some issues to address. First, one must ensures that V will only be used to test ℓ successive bits. This can easily be done by requiring that V does not contain any zero coefficient. Second, the gateway must be able to search for W at any possible offset. It is thus necessary to provide a way to shift the trapdoors. This is the reason why we add the elements C_i in the ciphertext. Finally, we must ensure that no adversary will be able to derive new trapdoors from the ones that it received. This is done by generating ℓ random coefficients v_i for the trapdoor td_W. In particular, we stress that a solution where $v_i = v_j$ for some $i, j \in [1, \ell]$ is not secure. Indeed, let C be a ciphertext encrypting a bitstring $B = b_1 \dots b_m$ and let us assume that $w_r = w_s = 1$ for all $r, s \in [1, \ell]$ (*i.e.* W is a sequence of "1"). Then, $e(C_1 \cdot C_{i+1-j}^{-1}, \widetilde{g}^V) = e(g, \widetilde{g})^{a(1-z^{i-j})V} = e(g, \widetilde{g})^{aV'}$, with

$$V' = \sum_{k=1}^{i-j} v_k \cdot x'_k \cdot z^k + \sum_{i-j+1}^{\ell} (v_k \cdot x'_k - v_{k-i+j} \cdot x'_{k-i+j}) z^k - \sum_{k=\ell+1}^{\ell+i-j} v_{k-i+j} \cdot x'_{k-i+j} \cdot z^k.$$

The coefficient of z^i is then $v_i \cdot x'_i - v_j \cdot x'_j = (v_i - v_j)\alpha_1 = 0$. Moreover, since $w_r = w_s = 1$, we have $x'_k = \alpha_1 \ \forall k \in [1, \ell]$, so:

$$V' = \alpha_1 (\sum_{k=1}^{i-j} v_k \cdot z^k + \sum_{i-j+1}^{i-1} (v_k - v_{k-i+j}) z^k + \sum_{i+1}^{\ell} (v_k - v_{k-i+j}) z^k - \sum_{k=\ell+1}^{\ell+i-j} v_{k-i+j} \cdot z^k),$$

and can thus be used to check whether $b_{u+1} \dots b_{u+i-1} = \overbrace{1 \dots 1}^{(i-1)\text{times}} \wedge b_{u+i+1} \dots b_{u+\ell+i-j} = \overbrace{1 \dots 1}^{(\ell-j)\text{times}}$, for any $u \in [0, m - \ell - i + j]$. Using td_W, a gateway is then able to get more information on B than the presence of W as a substring, which breaks the security of the construction. Such an attack can also occur if the gateway knows one of the coefficients v_i. It is therefore essential to randomly generate the latter. The formal proof that we provide in the next section will partly rely on this fact.

4 Security Analysis

4.1 Complexity Assumptions

Let us consider an adversary $\mathcal{A}$ which, knowing q trapdoors td_{W_k}, would like to decide if a ciphertext C encrypts B_0 or B_1. The natural restrictions imposed by

the security model imply that there is at least one index i^* such that $b_{i^*}^{(0)} \neq b_{i^*}^{(1)}$ and that, for all $k \in [1,q]$ and all $j \in [1,\ell_k]$ (where ℓ_k is the length of W_k), $b_{i^*-\ell_k+j}^{(0)} \dots b_{i^*+j-1}^{(0)}$ and $b_{i^*-\ell_k+j}^{(1)} \dots b_{i^*+j-1}^{(1)}$ both differ from $w_{k,1}, \dots, w_{k,l_k}$. In other words, any substring of B_0 (or respectively B_1) of length l_k containing $b_{i^*}^{(0)}$ (resp. $b_{i^*}^{(1)}$) 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 C'_{i^*} is $g_{i^*}^{a\cdot\alpha_1}$ or $g_{i^*}^{a\cdot\alpha_2}$. 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 td_{W_k}). All of them are of the form $g^{P_u(a,\alpha_1,\alpha_2,z)}$ or $\widetilde{g}^{Q_v(a,\alpha_1,\alpha_2,z,v_{i,W_k})}$ 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 [BBG05], whose asymmetric version [Boy08] is recalled below.

Definition 4 (GDH assumption). *Let r, s, t, and c be four positive integers and $R \in \mathbb{F}_p[X_1,\dots,X_c]^r$, $S \in \mathbb{F}_p[X_1,\dots,X_c]^s$, and $T \in \mathbb{F}_p[X_1,\dots,X_c]^t$ 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 R, S, and T. For any polynomial $f \in \mathbb{F}_p[X_1,\dots,X_c]$, we say that f is dependent on $< R,S,T >$ if there are $\{a_j\}_{i=1}^s \in \mathbb{F}_p^s \backslash \{(0,\dots,0)\}$, $\{b_{i,j}\}_{i,j=1}^{i=r,j=s} \in \mathbb{F}_p^{r\cdot s}$ 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 $\{g^{R^{(i)}(x_1,\dots,x_c)}\}_{i=1}^r$, $\{\widetilde{g}^{S^{(i)}(x_1,\dots,x_c)}\}_{i=1}^s$ 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 $< R,S,T >$.

Unfortunately, we cannot directly make use of this assumption unless we severely restrict the size n of the bitstrings 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 forge a trapdoor for 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 $B_\beta = b_1^{(\beta)} \dots, b_m^{(\beta)}$, 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 B_β containing $b_{i^*}^{(\beta)}$, the elements $\{g_i^a\}_{i=1}^n$ and $\{g_i^{a\cdot\alpha_1^{b_i^{(\beta)}}\cdot\alpha_2^{1-b_i^{(\beta)}}}\}_{i\in[1,n]\backslash\{i^*\}}$ along with the challenge element $U \in \mathbb{G}_1$ associated with the polynomial $f = z^{i^*}(\alpha_1^{b_{i^*}^{(\beta)}}\alpha_2^{1-b_{i^*}^{(\beta)}})$.

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 $g^{R^{(i)}(x_1,\ldots,x_c)}$, $\widetilde{g}^{S^{(i)}(x_1,\ldots,x_c)}$ and $e(g,\widetilde{g})^{T^{(i)}(x_1,\ldots,x_c)}$ 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. To achieve 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 [BBG05]) relies on the same argument as its interactive variant: as long as the "challenge" polynomial f does not depend on $< R, S, T >$, $g^{f(x_1,\ldots,x_c)}$ 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 5 (i-GDH assumption). *Let r, s, t, c and k be five positive integers and $R \in \mathbb{F}_p[X_1,\ldots,X_c]^r$, $S \in \mathbb{F}_p[X_1,\ldots,X_c]^s$ and $T \in \mathbb{F}_p[X_1,\ldots,X_c]^t$ be three tuples of multivariate polynomials over $\mathbb{F}_p$. Let $\mathcal{O}^R$ (resp. $\mathcal{O}^S$ and $\mathcal{O}^T$) be oracles that, on input $\{\{a^{(k)}_{i_1,\ldots,i_c}\}_{i_j=0}^{d_k}\}_k$, add the polynomials $\{\sum_{i_1,\ldots,i_c} a^{(k)}_{i_1,\ldots,i_c} \prod_j X_j^{i_j}\}_k$ to R (resp. S and T).*

Let $(x_1,\ldots,x_c)$ be a secret vector and q_R (resp q_S) (resp. q_T) be the number of queries to $\mathcal{O}^R$ (resp. $\mathcal{O}^S$) (resp. $\mathcal{O}^T$). The $i-$GDH assumption states that, given $\{g^{R^{(i)}(x_1,\ldots,x_c)}\}_{i=1}^{r+k\cdot q_R}$, $\{\widetilde{g}^{S^{(i)}(x_1,\ldots,x_c)}\}_{i=1}^{s+k\cdot q_S}$ and $\{e(g,\widetilde{g})^{T^{(i)}(x_1,\ldots,x_c)}\}_{i=1}^{t+k\cdot q_T}$, it is hard to decide whether $U = g^{f(x_1,\ldots,x_c)}$ or U is random if f is independent of $< R, S, T >$.

4.2 Security Results

Theorem 6. *The scheme described in Section 3 is* SEST-sIND-CPA secure under the i-GDH assumption for R, S and T initially set as $R = \{(z^i, x\cdot z^i, y\cdot z^i, a\cdot z^i)\}_{i=1}^{2n}$, $S = T = \emptyset$ and $f = a\cdot x\cdot z^n$.

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 $B_0 = b_1^{(0)}\ldots b_m^{(0)}$ and $B_1 = b_1^{(1)}\ldots b_m^{(1)}$ 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

$j = 1, \ldots, n$, 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 $b_i^{(0)} \neq b_i^{(1)}$ 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 $j = 1, \ldots, n$, 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 $b_i^{(\beta)}$ $\forall i \in \mathcal{I}_{\neq}$, so the adversary cannot distinguish whether it plays $G_n^{(0)}$ or $G_n^{(1)}$.

In particular, we can write:

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

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

Lemma 7. *For all $j = 0, \ldots, n-1$ 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 R, S and T initially set as $R = \{(z^i, x \cdot z^i, y \cdot z^i, a \cdot z^i)\}_{i=1}^{2n}$, $S = T = \emptyset$ and $f = a \cdot x \cdot z^n$.*

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

5 Proof of Lemma 7

First, let us note that if $|\mathcal{I}_{\neq}| \leq j$, then $\mathcal{I}_{\neq}^{(j+1)} = \mathcal{I}_{\neq}^{(j)}$. The games $G_j^{(\beta)}$ and $G_{j+1}^{(\beta)}$ are therefore exactly the same and there is nothing to prove. We thus just have to consider the case $|\mathcal{I}_{\neq}| \geq j + 1$.

Let i^* be the $(j+1)$-st index of $\mathcal{I}_{\neq}$. From the i$-$GDH challenge containing $\{(g^{z^i}, g^{x \cdot z^i}, g^{y \cdot z^i}, g^{a \cdot z^i})\}_{i=1}^{2n}$ along with $U \in \mathbb{G}_1$, the simulator generates the public key pk by first defining $g_i = g^{z^{n+i-i^*}}$ (so $g_{i^*} = g^{z^n}$). Next, it sets $(g_i^{\alpha_1}, g_i^{\alpha_2})$

as $(g^{x \cdot z^{n+i-i^*}}, g^{y \cdot z^{n+i-i^*}})$ if $b_{i^*}^{(\beta)} = 1$ and as $(g^{y \cdot z^{n+i-i^*}}, g^{x \cdot z^{n+i-i^*}})$ otherwise. By setting $g = g^{z^{n-i^*}}$, one can note that pk is well-formed.

Upon receiving an $\mathcal{O}\texttt{Issue}$ query for a bitstring $W = w_1 \dots w_\ell$, the simulator checks that the latter fulfills the condition defined in step 6 of Figure 1 (namely the fact that W does not match only one of the bitstrings B_0 and B_1). The simulator then uses the $\mathcal{O}^S$ oracle to return a valid trapdoor. It is worth noting that this condition implies that $w_1 \dots w_\ell \neq b_{i^*-min(i^*,\ell)+j}^{(\beta)} \dots b_{i^*-min(i^*,\ell)+\ell-1+j}^{(\beta)}$, for all $j \in [1, min(\ell, n - i^* + 1)]$. Indeed, if this relation held for some j, then we would have:

$$\begin{aligned} b_{i^*-min(i^*,\ell)+j}^{(\beta)} \dots b_{i^*-min(i^*,\ell)+\ell-1+j}^{(\beta)} &= w_1 \dots w_\ell \\ &\neq b_{i^*-min(i^*,\ell)+j}^{(1-\beta)} \dots b_{i^*-min(i^*,\ell)+\ell-1+j}^{(1-\beta)} \end{aligned}$$

since $b_{i^*}^{(\beta)} \neq b_{i^*}^{(1-\beta)}$. Thus, the condition would not be satisfied.

Finally, the simulator creates the challenge ciphertext as follows. It sets C_i as $g^{a \cdot z^{n+i-i^*-1}}$ for $i = 1, \dots, m$ (all these elements are provided in the i-GDH challenge). It then generates $C'_i \xleftarrow{\$} \mathbb{G}_1$ for the first j-th indexes of $\mathcal{I}_{\neq}$, uses the $\mathcal{O}^R$ oracle to get valid C'_i for $i \notin \mathcal{I}_{\neq}^{(j+1)}$ and sets C'_{i^*} as U.

If $U = g^{a \cdot x \cdot z^n} = g_{i^*}^{a \cdot \alpha_1^{b_{i^*}} \cdot \alpha_2^{(1-b_{i^*})}}$, then C'_{i^*} is a valid element and the simulator is playing game $G_j^{(\beta)}$. Else, C'_{i^*} is a random element from $\mathbb{G}_1$ and the simulator is playing game $G_{j+1}^{(\beta)}$. An adversary able to distinguish $G_j^{(\beta)}$ from $G_{j+1}^{(\beta)}$ is thus able to break the i-GDH assumption if the polynomial $f = a \cdot x \cdot z^n$ is independent of the sets R, S, and T (after q queries to $\mathcal{O}^S$ and 1 query to $\mathcal{O}^R$), which remains to prove.

Before stating this result in the next lemma we first simplify the notations to make the proof easier to follow. First, we will omit the superscript (β) and so we will denote the challenge bitstring as $b_1 \dots b_m$ instead of $b_1^{(\beta)} \dots b_m^{(\beta)}$. Each query to $\mathcal{O}^S$ is associated with a bitstring $w_1^{(k)} \dots w_{\ell_k}^{(k)}$ (for $k \in [1, q]$) submitted to the $\mathcal{O}\texttt{Issue}$ oracle. In the following we will thus simply say that $w_1^{(k)} \dots w_{\ell_k}^{(k)}$ is submitted to $\mathcal{O}^S$. Such a query adds the polynomials $\sum_{i=1}^{\ell_k} v_i^{(k)} x_i^{(k)} z^i$ and $\{v_i^{(k)}\}_{i=1}^{\ell_k}$ to S, where $x_i^{(k)} = \alpha_1^{w_i^{(k)}} \alpha_2^{1-w_i^{(k)}}$. Similarly, a query to $\mathcal{O}^R$ adds $\{a \cdot x'_i \cdot z^{n-i^*+i}\}_{i \in [1,n] \setminus \{i^*\}}$ to R, where $x'_i = \alpha_1^{b_i} \alpha_2^{1-b_i}$.

We recall that the set R initially contains $\{(z^i, x \cdot z^i, y \cdot z^i, a \cdot z^i)\}_{i=1}^{2n}$, while S and T are initially empty.

Lemma 8. *Let R, S, and T be the sets defined above after q queries to $\mathcal{O}^S$ and one query to $\mathcal{O}^R$. If, for any $k \in [1, q]$, the string $w_1^{(k)} \dots w_{\ell_k}^{(k)}$ submitted to $\mathcal{O}^S$ differs from $b_{i^*-min(i^*,\ell_k)+j} \dots b_{i^*-min(i^*,\ell_k)+\ell_k-1+j}$ for all $j \in [1, min(\ell_k, n - i^* + 1)]$, then the polynomial $a \cdot x \cdot z^n$ is independent of $< R, S, T >$.*

Proof. We want to prove that one cannot find polynomials $P_1 \neq 0$, P_2, and P_3 with $P_1, P_3 \in S$ and $P_2 \in R$ such that:

$$(a \cdot x \cdot z^n) \cdot P_1(x, y, z, a, \{v_i^{(k)}\}) = (P_2 \cdot P_3)(x, y, z, a, \{v_i^{(k)}\}).$$

First note that the variable a is only involved in the polynomials included in R. The left hand side of the above equation is then necessarily of the form $a \cdot Q_1(x, y, z, \{v_i^{(k)}\})$ which implies that P_2 is also of the form $a \cdot Q_2(x, y, z, \{v_i^{(k)}\})$. Therefore, P_2 can only be a combination of elements of $\{a \cdot z^i\}_{i=1}^{2n}$ or $\{a \cdot x'_i \cdot z^{n-i^*+i}\}_{i \in [1,n] \setminus \{i^*\}}$. Let $\ell_{max} \leq n$ be the maximum length of the strings submitted to $\mathcal{O}^S$. Let $\{u_{i,k,j}\}_{i=0,k=1,j=0}^{i=2n,k=q,j=\ell_{max}}$ and $\{u'_{i,k,j}\}_{i=0,k=1,j=0}^{i=n,k=q,j=\ell_{max}}$ be such that:

$$\begin{aligned}
(a \cdot x \cdot z^n)(\sum_k [u'_{i^*,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) + \sum_{j=1}^{\ell_k} u'_{i^*,k,j} v_j^{(k)}]) \\
= \sum_{i=0}^{2n} a \cdot z^i (\sum_k [u_{i,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) + \sum_{j=1}^{\ell_k} u_{i,k,j} v_j^{(k)}]) \\
+ \sum_{i \in [1,n] \setminus \{i^*\}} a \cdot x'_i \cdot z^{n-i^*+i} (\sum_k [u'_{i,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) + \sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)}]).
\end{aligned}$$

Our goal is then to prove that $u'_{i^*,k,j} = 0$ for all $k \in [1, q]$ and $j \in [0, \ell_k]$.

Since $x'_{i^*} = x$ we can merge the left hand side of the equation into the sum $\sum_{i \in [1,n] \setminus \{i^*\}}$ on the right hand side. Moreover, by dividing both sides by a, we get:

$$\begin{aligned}
-\sum_{i=0}^{2n} z^i (\sum_k [u_{i,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) + \sum_{j=1}^{\ell_k} u_{i,k,j} v_j^{(k)}]) = \\
\sum_{i \in [1,n]} x'_i \cdot z^{n-i^*+i} (\sum_k [u'_{i,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) + \sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)}]).
\end{aligned}$$

Since x'_i and x_m belong to $\{x, y\}$ we can deduce that:

1. The term $-\sum_{i=0}^{2n} z^i (\sum_k u_{i,k,0} \sum_m v_m^{(k)} x_m^{(k)} z^m)$ is of the form $x \cdot U_1(z, \{v_i^{(k)}\}_{i,k}) + y \cdot V_1(z, \{v_i^{(k)}\}_{i,k})$;
2. The term $-\sum_{i=0}^{2n} z^i (\sum_k \sum_{j=1}^{\ell_k} u_{i,k,j} v_j^{(k)})$ is of the form $U_2(z, \{v_i^{(k)}\}_{i,k})$;
3. The term $\sum_{i \in [1,n]} x'_i \cdot z^{n-i^*+i} (\sum_k u'_{i,k,0} (\sum_m v_m^{(k)} x_m^{(k)} z^m))$ is of the form $x^2 \cdot U_3(z, \{v_i^{(k)}\}_{i,k}) + y^2 \cdot V_3(z, \{v_i^{(k)}\}_{i,k}) + x \cdot y \cdot W_3(z, \{v_i^{(k)}\}_{i,k})$;
4. The term $\sum_{i \in [1,n]} x'_i \cdot z^{n-i^*+i} (\sum_k \sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)})$ is of the form $x \cdot U_4(z, \{v_i^{(k)}\}_{i,k}) + y \cdot V_4(z, \{v_i^{(k)}\}_{i,k})$;

for some polynomials $U_1, U_2, U_3, U_4, V_1, V_3, V_4$, and W_4. Therefore, the equation holds only if the second and the third terms are both equal to 0. The fact that the third term vanishes implies that $u'_{i,k,0}(\sum_m v_m^{(k)} x_m^{(k)} z^m) = 0, \forall k \in [1,q]$ (since each term of the sum $\sum_k$ involves a polynomial $\sum_m v_m^{(k)} x_m^{(k)} z^m$ whose variables $v_m^{(k)}$ differ for each k). Therefore, $u'_{i,k,0} = 0\ \forall i \in [1,n]$ (and in particular for $i = i^*$) and $k \in [1,q]$.

We thus obtain that:

$$-\sum_{i=0}^{2n} z^i (\sum_k u_{i,k,0} (\sum_m v_m^{(k)} x_m^{(k)} z^m)) = \sum_{i \in [1,n]} x'_i \cdot z^{n-i^*+i} (\sum_k \sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)})$$

The smallest power of z on the right-hand side is z^{n-i^*+1}. The left-hand side thus cannot contain smaller powers of z, which means that $u_{i,k,0} = 0$ if $i < n - i^*$. The same reasoning applies for the greatest power of z, which allows to conclude that $u_{i,k,j} = 0$ if $i > 2n - i^* - \ell_k$.

Now, one can note that the left (resp. right) hand side of this equation is of the form $\sum_k P_k(x, y, z, \{(v_1^{(k)}, \dots v_{\ell_k}^{(k)}\}_k)$ (resp. $\sum_k Q_k(x, y, z, \{(v_1^{(k)}, \dots v_{\ell_k}^{(k)}\}_k)$) where P_k (resp. Q_k) are polynomials whose monomials are all multiples of $v_j^{(k)}$ for some $j \in [1, \ell_k]$. Since the variables $\{v_j^{(k)}\}_k$ are independent, the equality holds only if $P_k = Q_k$ for all $k \in [1,q]$. We then get, for each $k = 1, \dots, q$:

$$-\sum_{i=n-i^*}^{2n-i^*-\ell_k} z^i (u_{i,k,0} (\sum_m v_m^{(k)} x_m^{(k)} z^m)) = \sum_{i \in [1,n]} x'_i \cdot z^{n-i^*+i} (\sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)}).$$

Now if we consider the coefficients of z^{n-i^*+i} for $i = 1, \dots, n$, we get:

$$\sum_{m=1}^{\ell_k} u_{n-i^*+i-m,k,0} \cdot v_m^{(k)} \cdot x_m^{(k)} = x'_i (\sum_{j=1}^{\ell_k} u'_{i,k,j} v_j^{(k)}). \quad (1)$$

For $i = i^*$, (since $\ell_k \leq n$) the equation becomes:

$$\sum_{m=1}^{\ell_k} u_{n-m,k,0} \cdot v_m^{(k)} \cdot x_m^{(k)} = x'_{i^*} (\sum_{j=1}^{\ell_k} u'_{i^*,k,j} v_j^{(k)}).$$

Proving that $u'_{i^*,k,j} = 0$ is then equivalent to proving that $u_{n-j,k,0} = 0$ for all $j \in [1, \ell_k]$.

Now recall the restriction placed on the string $w_1^{(k)} \dots w_{\ell_k}^{(k)}$ for every $k \in [1,q]$. The fact that, $\forall j \in [1, min(\ell_k, n - i^* + 1)]$, $b_{i^*-min(i^*,\ell_k)+j} \dots b_{i^*-min(i^*,\ell_k)+\ell_k-1+j}$ differs from $w_1^{(k)} \dots w_{\ell_k}^{(k)}$ implies that there is at least one integer $d_j \in [0, \ell_k - 1]$ such that $w_{1+d_j}^{(k)} \neq b_{i^*-min(i^*,\ell_k)+j+d_j}$. Therefore, $x_{1+d_j}^{(k)} \neq x'_{i^*-min(i^*,\ell_k)+j+d_j}$ and the equation (1) for $i = i^* - min(i^*, \ell_k) + j + d_j$, gives us $u_{n-min(i^*,\ell_k)+j-1,k,0} = 0$ for $j \in [1, min(\ell_k, n - i^* + 1)]$.

It thus remains to show that $u_{n-j,k,0} = 0$ for $j \in]min(i^*, \ell_k), \ell_k]$ and for $j \in [1, 1 + min(i^*, \ell_k) - min(n - i^* + 1, \ell_k)[$.

Let us consider the first interval. If $min(i^*, \ell_k) = \ell_k$, then the interval is empty we have nothing further to prove. We can then focus on the case $i^* < \ell_k$. However, in that case $j \in]i^*, \ell_k]$ implies that $n - j < n - i^*$ and so that $u_{n-j,k,0} = 0$ since we previously proved that $u_{i,k,0} = 0$ if $i < n - i^*$.

The second interval is not-empty only if $min(i^*, \ell_k) > min(n - i^* + 1, \ell_k)$. We distinguish the two following cases:

- $i^* \leq \ell_k$. Then, $min(n - i^* + 1, \ell_k) = n - i^* + 1$ otherwise the previous inequality is not verified. For all $j \in [1, 1 + min(i^*, \ell_k) - min(n - i^* + 1, \ell_k)[$ it holds that:

$$\begin{aligned} n - j &> n - 1 - min(i^*, \ell_k) + min(n - i^* + 1, \ell_k) \\ &> n - 1 - i^* + n - i^* + 1 \\ &> 2n - 2i^* \\ &> 2n - i^* - \ell_k. \end{aligned}$$

- $i^* > \ell_k$. Then, we again have that $min(n - i^* + 1, \ell_k) = n - i^* + 1$ and, for all $j \in [1, 1 + min(i^*, \ell_k) - min(n - i^* + 1, \ell_k)[$:

$$\begin{aligned} n - j &> n - 1 - min(i^*, \ell_k) + min(n - i^* + 1, \ell_k) \\ &> n - 1 - \ell_k + n - i^* + 1 \\ &> 2n - i^* - \ell_k. \end{aligned}$$

For both cases, we have $n - j > 2n - i^* - \ell_k$, which implies, as explained above, that $u_{n-j,k,0} = 0$.

We have thus proved that $u_{n-j,k,0} = 0$ for all $j \in [1, \ell_q]$ which is equivalent to $u'_{i^*,k,j} = 0$ for all $j \in [1, \ell_q]$. The polynomial $a \cdot x \cdot z^n$ is therefore independent of $< R, S, T >$.

6 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 the input strings we encrypt $s_1, \ldots, s_m$ are no longer bitstrings, but rather strings of characters belonging to some finite set $\mathcal{S}$. As mentioned in Remark 3, our scheme can be trivially extended to handle this case, although the larger size of $\mathcal{S}$ (compared to $\{0, 1\}$) may lead to more complex queries. Moreover, one might want to search for substrings of the form $W = w_1 \ldots w_{t-1} w_t^{(\mathcal{S}_t)} w_{t+1} \ldots w_\ell$ 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.

6.1 Handling Wildcards

The first case we consider here is that in which $W = w_1 \dots w_{i_1}^{(\mathcal{S}_{i_1})} \dots, w_{i_r}^{(\mathcal{S}_{i_r})} \dots w_\ell$ with $\mathcal{S}_{i_1} = ... = \mathcal{S}_{i_r} = \mathcal{S}$, which means that $w_{i_1}^{(\mathcal{S}_{i_1})}, \dots, w_{i_r}^{(\mathcal{S}_{i_r})}$ 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 $C_{j+1} \dots C_{j+\ell} = W$. This leads to the following variant of our main protocol. For sake of simplicity, we assume that there is a public indexation of the set $\mathcal{S}$, which means that any element of S is associated with an integer between 1 and $S = |\mathcal{S}|$. We can therefore, without loss of generality, talk of the w-th element of $\mathcal{S}$ and use w as an index for any element $w \in \mathcal{S}$.

- $\texttt{Setup}(1^k, n)$: 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 $g \xleftarrow{\$} \mathbb{G}$ and $\widetilde{g} \xleftarrow{\$} \mathbb{G}$ and returns $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, n)$.
- $\texttt{Keygen}(n)$: This algorithm selects $S+1$ random scalars $\alpha_1, \dots, \alpha_S, z \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i \leftarrow g^{z^i}$ along with $g_i^{\alpha_j}$ for $i = 1, \dots, n$ and $j = 1, \dots, S$. The public key pk is then set as $\{(g_i, g_i^{\alpha_j})\}_{i,j=1}^{i=n,j=S}$ whereas sk is set as $(\alpha_1, \dots, \alpha_S, z)$. One can note that $g_0 = g$ is provided in the public parameters pp.
- $\texttt{Issue}(W, \mathsf{sk})$: To issue a trapdoor td_W for a string $W = w_1 \dots w_{i_1}^{(\mathcal{S})} \dots, w_{i_r}^{(\mathcal{S})} \dots w_\ell$ of length $\ell \leq n$, the algorithm selects $\ell - r$ random scalars $v_i \xleftarrow{\$} \mathbb{Z}_p^*$, for $i \in \mathcal{D} = \{1, \dots, n\} \setminus \{i_1, \dots, i_r\}$. It then computes $\widetilde{g}^V$, where $V = \sum_{i \in \mathcal{D}} v_i \cdot \alpha_{w_i} \cdot z^i$, along with $\widetilde{g}^{v_i}$, for $i \in \mathcal{D}$. The trapdoor td_W is then set as $(\widetilde{g}^V, \{\widetilde{g}^{v_i}\}_{i \in \mathcal{D}})$.
- $\texttt{Encrypt}(B, \mathsf{pk})$: To encrypt a string $B = b_1 \dots b_m$, where $m \leq n$ the user selects a random scalar a and returns $C = \{(C_i, C_i')\}_{i=1}^m$ where $C_i \leftarrow g_{i-1}^a$ and $C_i' \leftarrow g_i^{a \cdot \alpha_{b_i}}$ for $i = 1, \dots, m$.
- $\texttt{Test}(C, \mathsf{td}_W)$: To test whether the string B encrypted by C contains the substring W, the algorithm parses td_W as $(\widetilde{g}^V, \{\widetilde{g}^{v_i}\}_{i \in \mathcal{D}})$ and C as $\{(C_i, C_i')\}_{i=1}^m$ and checks, for $j = 0, \dots, m - \ell$, if the following equation holds:

$$\prod_{i \in \mathcal{D}} e(C'_{j+i}, \widetilde{g}^{v_i}) \stackrel{?}{=} e(C_{j+1}, \widetilde{g}^V).$$

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

One can note that the encryption process remains unchanged compared to the original protocol. The only difference with the latter is that the coefficients v_i of the polynomial V associated with the trapdoor W are now set as 0 if the i-th element of W is a wildcard. The product in the `Test` algorithm can thus be performed only for the indexes $i \in \mathcal{D}$. Correctness of this variant follows directly from the one of the previous protocol.

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.

6.2 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 usually the case for regular expressions.

- $\texttt{Setup}(1^k, n)$: 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 $g \xleftarrow{\$} \mathbb{G}$ and $\widetilde{g} \xleftarrow{\$} \mathbb{G}$, and returns $pp \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g}, n)$.
- $\texttt{Keygen}(n, \mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(k)})$: 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 $w \in \mathcal{S}$ to the index of the set $\mathcal{S}^{(j)}$ which contains it is thus perfectly defined. The algorithm then selects $S + k + 1$ random scalars $\alpha_1, \ldots, \alpha_S, \beta_1, \ldots, \beta_k, z \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i \leftarrow g^{z^i}$ for $i = 1, \ldots, n$ along with $(g_i^{\alpha_j}, g_i^{\beta_d})$ for $d = 1, \ldots, k$ and all $j \in \mathcal{S}^{(d)}$ (recall that we associate each element of $\mathcal{S}$ with an integer in the interval [1,S]). The public key is then set to $\{g_i\}_{i=1}^n \cup_{d=1}^k \{(g_i^{\alpha_j}, g_i^{\beta_d})\}_{i \in [1,n], j \in \mathcal{S}^{(d)}}$ and sk as $\alpha_1, \ldots, \alpha_S, \beta_1, \ldots, \beta_k, z$.
- $\texttt{Issue}(W, \mathsf{sk})$: 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 $\mathcal{D} = \{1, \ldots, n\} \backslash \{i_1, \ldots, i_r\}$. The algorithm selects ℓ random scalars $v_i \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\widetilde{g}^V$ where $V = \sum_{i \in \mathcal{D}} v_i \cdot \alpha_{w_i} \cdot z^i + \sum_{i \in \{i_1, \ldots, i_r\}} v_i \cdot \beta_{h(i)} \cdot z^i$. The trapdoor td_W is then set to the tuple consisting of $\widetilde{g}^V$ along with $\widetilde{g}^{v_i}$, for $i = 1, \ldots, \ell$.

– $\texttt{Encrypt}(B, \mathsf{pk})$: To encrypt a string $B = b_1 \dots b_m$, where $m \leq n$ the user selects a random scalar a and returns $C = \{(C_i, C_i^{(1)}, C_i^{(2)})\}_{i=1}^m$ where $C_i \leftarrow g_{i-1}^a$, $C_i^{(1)} \leftarrow (g_i^{\alpha_{b_i}})^a$ and $C_i^{(2)} \leftarrow (g_i^{\beta_{f(b_i)}})^a$, for $i = 1 \dots m$.
– $\texttt{Test}(C, \mathsf{td}_W)$: To test whether the string $B = b_1 \dots b_m$ encrypted by C contains the substring $W = w_1 \dots w_{i_1}^{(\mathcal{S}_{i_1})} \dots, w_{i_r}^{(\mathcal{S}_{i_r})} \dots w_\ell$, the algorithm parses td_W as $(\widetilde{g}^V, \{\widetilde{g}^{v_i}\}_{i=1}^n)$ and C as $\{(C_i, C_i^{(1)}, C_i^{(2)})\}_{i=1}^m$ and checks, for $j = 0, \dots, m - \ell$, if the following equation holds:

$$\prod_{i \in \mathcal{D}} e(C_{j+i}^{(1)}, \widetilde{g}^{v_i}) \prod_{i \in \{i_1, \dots, i_r\}} e(C_{j+i}^{(2)}, \widetilde{g}^{v_i}) \stackrel{?}{=} e(C_{j+1}, \widetilde{g}^V).$$

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 thus be seen as an encoding of the subset $\mathcal{S}^{(j)}$, in the same way as the scalars α_i encode the i-th character of $\mathcal{S}$. 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.

7 Some application scenarios

Far from being an esoteric cryptographic functionality, pattern matching on encrypted data has many interesting applications. We review just a few of those here, in particular focusing on scenarios in which it is desirable to be able to perform the search with universal tokens (arbitrarily-generated patterns, which are moreover universal over input generated by multiple users).

DPI. The basic use-case scenario of the BlindBox scheme [SLPR15] is that of deep-packet inspection (DPI). The latter refers to the ability of a gateway (usually a middlebox such as a firewall, an IDS, etc.) to verify whether some specific data does, or does not, contain given prohibited content (adult content, malware, etc.). An easy way to ensure that such middleboxes can provide the functionality of DPI with best results is to simply give them access to all data stored, sent, or received by a given party. However, this also allows the middlebox access to a great deal of data that is irrelevant to its goals. This raises privacy concerns that are only fuelled by the recent revelations of Edward Snowden, which warn of mass surveillance by massive collection of data. As a consequence, it is desirable to restrict the access of middleboxes such as IDS, virus scanners, or firewalls, so that they only access data on a need to know basis.

One approach to solving this problem is to encrypt ciphertexts with many different keys, each key used for a different fragment of the plaintext (the header, or a subfragment of the body for either the request or the response). Then, one can simply give the middlebox the keys it requires to perform its task. This

approach allows for efficient decryption; however, it also comes with a tradeoff between the privacy of the user data and the overhead in terms of the associated keys. In particular, the more finegrained the access control scheme is, the more privacy is offered to the user, with less information leaking to the middlebox. However, a finegrained access control mechanism also induces a very large number of keys, which must be computed and stored by the two communicating parties and the middlebox.

The approach we take in this paper is different. We will specifically only allow the middlebox to perform pattern matching on particular keywords, learning – as an output – both whether the pattern occurs in the encrypted data, and where exactly it occurs in the plaintext. This will not require any key-escrow; in fact the searching will be done without requiring any secret keys (though we do require trapdoors for the search patterns).

Matching genomic data. Genomic data is infamously large, and can be seen as very large strings of *private* data. Arguably, one's own genome is an individual least-replicable, most personal identity, and hence it should always be stored in encrypted data and accessed only upon the specific permission of its owner.

However, for both law-enforcement purposes and for the establishment of possible family ties, it is imperative to allow the matching of substrings of DNA to a target (encrypted) genome. We can see this process as pattern matching on streams, since – for efficiency purposes – it is better to be able to find patterns without backtracking. In law-enforcement, the DNA recovered from a crime scene is sensitive evidence material, which must only be handled locally; thus, the parties encrypting their genomes are not privvy to it. Moreover, if the search process is delegated to a third party, the latter must learn no information apart from the search results: notably, nothing about the keyword and nothing about the input plaintext.

For pattern matching on genomic data, we thus need a mechanism that uses universal tokens for searching. Furthermore, no data must be leaked about the input data than the exact results of the search process. Finally, since DNA evidence could also only provide partial data, it is better to have a pattern-matching mechanism that can allow to match some regular expressions on patterns, such as, *e.g.*, the concatenation of two patterns, or patterns with missing characters. The solution we present in this paper can effectively be used for applications with genomic data, including by using regular expressions on the keywords.

Searching on medical data. In medical research, data from many research centres (or hospitals) is collected in massive databases, which can then be used by researchers in detecting correlations between various factors, such as the diameter of a blood-vessel and known symptoms of diabetes. However, since patient data is a sensitive piece of information, most databases must be sanitized of identifying information first. In addition, medical researchers must make a formal request to be allowed to search for specific data in such databases. However, mass-surveillance techniques can be used to correlate even sanitized data to spe-

cific individuals, thus violating patient-data confidentiality. On the other hand, medical research would severely suffer if access to patient data were limited.

Instead, it would be desirable to detect matches of specific patterns (such as ranges of blood-vessel diameters) to encrypted patient records. In this way, such records could be used to the fullest extent of their possibilities, without leaking correlating data, on which no searches are authorized. In this application scenario, we thus have encrypted data from many different sources. The entity performing the search receives this encrypted data, and must ask permission to search from a different entity, which enables the search by means of a trapdoor. Our solution allows such searches, including with regular expressions, with universal trapdoors, *i.e.*, a single trapdoor can be used to search on the medical data from various searches; in addition, we support keywords of arbitrary lenghts.

In this paper we present – to our knowledge – the first searchable encryption scheme that allows for the generation of universal trapdoors (*i.e.*, trapdoors of any length, applicable to any ciphertext). In our scheme, the party that encrypts the ciphertext (the *encrypter*) will do so by using a public-key encryption scheme, for which the entity generating the trapdoor (the *issuer*) holds the secret keys. Finally, the *searcher* or *gateway* will be able to search for specific keywords (for which it received trapdoors from the issuer) in the encrypter's ciphertext, learning whether there is a match and also at which position in the ciphertext the match occurs.

8 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 immediate applications in deep-packet inspection, pattern-matching also has extensive uses in matching of small DNA strings to much larger full-genomic data. In this context we consider both very simplistic pattern matching (notably of a small, but fully-known string of DNA), as well as some basic regular expressions on such patterns. The advantage of our approach is that a single trapdoor for a given pattern can be matched against potentially very many encrypted genomes, even when those genomes were encrypted by many distinct entities (using the same public-key information). Moreover, the size of the ciphertext is independent of the size of the keywords.

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 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 a quadratic – in the size of the keyword – complexity (in terms of the number of pairings): consequently the efficiency is better for either relatively small keywords or for large ones (but not minimum ones).

We prove the security of our scheme under an interactive version of the GDH assumption. This is less than ideal; however note that 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

Pierre-Alain Fouque and Cristina Onete are grateful for the support of the ANR through project 16 CE39 0012 (SafeTLS).

References

[ABC+05] 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.

[ABDP15] 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.

[AGVW13] 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.

[BBG05] 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.

[BDOP04] 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.

[BEM+12] 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.

[BGM+10] 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.

[Boy08] 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.

[BW07] 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.

[CGKO06] 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.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of ACM CCS 2015*. ACM, 2015.

[CHK03] 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.

[CK10a] 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.

[CK10b] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology – ASIACRPYT*, volume 6477 of *LNCS*, pages 577–594. Springer-Verlag, 2010.

[CS15] Melissa Chase and Emily Shen. Substring-searchable symmetric encryption. *PoPETs*, 2015(2):263–281, 2015.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GHS16] 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.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[HL10] 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.

[HREJ14] 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.

[Jar12] Jeff Jarmoc. SSL/TLS interception proxies and transitive trust. *Presentation at Black Hat Europe*, 2012.

[KM10] 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.

[KSW13] 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.

[LLN14] 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.

[MNSS12] 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.

[NSV+15] 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.

[SLPR15] 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.

[SWP00] 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.

[TPKC07] 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.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of(USENIX Security 16)*, pages 707–720. USENIX Association, 2016.