Article

Substring Position Search over Encrypted Cloud Data Supporting Efficient Multi-User Setup †

Mikhail Strizhov *, Zachary Osman and Indrajit Ray

Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA; zosman@cs.colostate.edu (Z.O.); indrajit@cs.colostate.edu (I.R.)

* Correspondence: strizhov@cs.colostate.edu; Tel.: +1-970-491-5305

† This paper is an extended version of our paper published in the 2015 IEEE International Conference on Cloud Engineering (IC2E), Tempe, AZ, USA, 9–13 March 2015.

Academic Editors: Eduardo Fernández-Medina Patón and David G. Rosado
Received: 15 January 2016; Accepted: 23 June 2016; Published: 4 July 2016

Abstract: Existing Searchable Encryption (SE) solutions are able to handle simple Boolean search queries, such as single or multi-keyword queries, but cannot handle substring search queries over encrypted data that also involve identifying the position of the substring within the document. These types of queries are relevant in areas such as searching DNA data. In this paper, we propose a tree-based Substring Position Searchable Symmetric Encryption (SSP-SSE) to overcome the existing gap. Our solution efficiently finds occurrences of a given substring over encrypted cloud data. Specifically, our construction uses the position heap tree data structure and achieves asymptotic efficiency comparable to that of an unencrypted position heap tree. Our encryption takes $O(kn)$ time, and the resulting ciphertext is of size $O(kn)$, where k is a security parameter and n is the size of stored data. The search takes $O(m^2 + occ)$ time and three rounds of communication, where m is the length of the queried substring and occ is the number of occurrences of the substring in the document collection. We prove that the proposed scheme is secure against chosen-query attacks that involve an adaptive adversary. Finally, we extend SSP-SSE to the multi-user setting where an arbitrary group of cloud users can submit substring queries to search the encrypted data.

Keywords: substring position search; searchable symmetric encryption; cloud computing; position heap tree

PACS: J0101

1. Introduction

Owing to the wide adoption of cloud computing services, public, as well as private organizations now outsource their data to remote servers. Cloud computing services provide efficient and cost-effective solutions for data storage. Nevertheless, outsourced data may contain sensitive information that needs to be protected. Traditional encryption techniques protect the data from unauthorized access; however, they introduce new challenges to data utilization. Specifically, allowing users to efficiently search over encrypted data is one of the most pressing issues in cloud computing.

In order to enable search over encrypted data, many Searchable Encryption (SE) schemes have been proposed in recent years [1–20]. (Note, we use the term searchable encryption somewhat loosely to include schemes, such as private information retrieval, as well.) Generally, SE solutions involve building an encrypted searchable index that hides the sensitive information from the remote server, yet they allow a search on the encrypted data. SE solutions differ in the level of efficiency and security guarantees that they offer; however, most of them support only exact keyword search. As a result, there is no tolerance of format inconsistencies that are part of typical cloud user behavior; and

they happen frequently. It is quite common that the search queries do not exactly match the pre-set keywords due to lack of exact knowledge about the data. For example, a financial company stores its employees' income tax documents in encrypted form in the cloud. A tax accountant may issue a search query of "form1040", which describes multiple keywords, such as "form1040", "form1040A", "form1040-ez", "form1040es", and she wants to find a position of the first occurrence of the query in each encrypted document that contains the string of characters. The significant drawback of existing schemes underlines an important need for new techniques that support search flexibility over encrypted documents. In this work, we consider the problem of efficient substring position search over encrypted data. The users can query the remote untrusted server for a set of encrypted documents that contain a substring of characters. The cloud server retrieves the set of matching documents together with positions where the queried string begins.

An important application of this work is in the area of searching a genome sequence against genomic databases. Such a search can be used in the analysis of genetic diseases, genetic fingerprinting or genetic genealogy and requires a set of results that do not simply match the genome, but rather the position of the genome sequence within the genome database. The major contribution of our work is to initiate the study of a very important problem, namely substring position search over encryption data. Our solution should not be considered as a complete approach for the subject, which has very strong future directions of research. Nonetheless, our solution provides the preliminary foundation for the study of the subject, including formal definitions, building blocks, basic construction, as well as security proofs. In this work, we continue exploring the line of recent searchable encryption solutions, but from a slightly different standpoint.

We now give an overview of our contributions:

1. We present a Substring Position Searchable Symmetric Encryption (SSP-SSE) scheme that allows a substring search over an encrypted document collection. The scheme is based on a position heap tree data structure recently proposed by Ehrenfeucht et al. [21].
2. We formally define two leakage functions and security against the adaptive chosen-query attack of a tree-based SSP-SSE scheme. Apart from the traditional access and search patterns, we include the definition of the path pattern in the leakage functions of a tree-based searchable encryption. We show that SSP-SSE enjoys the strong notion of semantic security [6].
3. We present a construction that is very efficient and does not require large ciphertext space. Our encryption takes $O(kn)$ time, and the ciphertext is of size $O(kn)$, where k is the security parameter and n is the size of stored data. The search protocol takes $O(m^2 + occ)$ time and three rounds of communication, where m is the length of the queried substring and occ is the number of occurrences of the substring in the document collection. We perform a thorough experimental evaluation of our solution on a real-world genomic dataset.
4. We consider a natural extension of the SSP-SSE scheme, where an arbitrary group of data users can submit substring queries to search the encrypted collection. We design a scheme support distributed setup, where data users choose their own secret key rather than receive the key from a trusted authority. We formally define a Multi-User Substring Position Searchable Symmetric Encryption (MSSP-SSE) and present an efficient construction.

We organize the rest of the paper as follows: Section 2 gives an outline of the most recent related work. In Section 3, we give an overview of the system and threat models, notations and preliminaries. In Section 4, we present algorithms and data structures that allow a substring search on the plaintext data. We give a brief overview of each data structure and later present a discussion on choosing the right data structure to enable substring search in an untrusted cloud environment. In Section 5, we provide the details of the SSP-SSE scheme and define the security definitions and requirements. Section 6 is devoted to security and performance analysis. The extension of our solution towards an arbitrary group of users is presented in Section 7. Lastly, we conclude in Section 8.

2. Related Work

Efficient searchable encryption methods are extensively studied in the literature. Traditional searchable encryption schemes focus on the problem of searching for a keyword in the document collection. In this setting, each document is assumed to consist of a sequence of keywords. The cloud server must be able to determine which encrypted documents contain a particular queried keyword, which is also encrypted. Song et al. [2] presented the first searchable symmetric encryption scheme. Their scheme has provable security properties, linear-time search complexity in the length of the document collection. Goh et al. [3] introduced formal security definitions of searchable symmetric encryption and proposed a scheme that is based on the Bloom filters [22]. The scheme requires a linear search time and provides some false positive results. Many other schemes have been proposed to improve the efficiency of keyword search by implementing an inverted searchable index [4,6,13,23]. Chang et al. [13] showed an index construction that enables keyword search without false positive results. Curtmola et al. [6] gave the first solution that enables sublinear search time for the entire document collection. Here, the searchable index consists of a keyword trapdoor and encrypted document identifiers whose corresponding data files contain the keyword. Recently, Cao et al. [10] proposed the multi-keyword ranked search scheme. The solution ranks encrypted documents based on a similarity score. The score is calculated between the search query (that contains multiple keywords) and the set of encrypted documents. Moataz et al. [4] developed the Boolean Symmetric Searchable Encryption (BSSE) scheme. The scheme is based on the orthogonalization of the keywords according to the Gram–Schmidt process. Later, Moataz et al. [24] proposed the Conjunctive Symmetric Searchable Encryption scheme that allows conjunctive keyword search on encrypted documents with different privacy assurances. Orencik's solution [5] proposed the privacy-preserving multi-keyword search method that utilizes minhash functions.

In the public-key setting, Boneh et al. [8] were the first to propose a searchable encryption using asymmetric cryptography. The authors developed the construction where anyone with the public key can write to the data stored on the remote server, but only authorized users with the private key can search. The other asymmetric solution was provided by Di Crescenzo et al. in [25], where the authors designed a public-key encryption scheme with keyword search based on a variant of the quadratic residuosity problem. To support more complex queries, conjunctive keyword search, subset query and range query over encrypted data have also been proposed in [7,9,14,26,27].

All of the schemes above support only exact keyword search, i.e., there is no tolerance of format inconsistencies in the search. Li et al. [17] were the first to propose a fuzzy keyword search scheme over encrypted data. The authors developed the solution that constructs fuzzy keyword sets based on document collection and later uses the edit distance to measure the similarity between keyword query and the sets. Wang et al. [18] improved previous work and proposed a scheme that achieves constant search time complexity. Later, Boldyreva et al. [19] gave an efficient fuzzy-searchable encryption (EFSE) scheme to locate the similar records. The main drawback of fuzzy keyword search solutions is that they require a large ciphertext and computation overhead and, thus, may not be suitable for the real-world cloud storage systems.

The SSE solution proposed by Curtmola et al. [6] can be adopted to allow the substring search over encrypted data. To do this, we would have to generate all possible substrings of each keyword extracted from the document collection and consider these substrings as keywords. However, this solution induces a very large storage requirement since the data owner would have to generate and keep all possible combinations of substrings of any keyword. For example, for any keyword of length n, there are $\frac{n\times(n+1)}{2}$ possible substrings. For a document collection with m distinct keywords of length n, it would take $m \times \frac{n\times(n+1)}{2}$ entries in SSE to keep all substrings of all keywords at the cloud server.

3. Background and Building Blocks

3.1. System and Threat Models

Consider a cloud data hosting service shown in Figure 1 that involves three entities: the cloud provider, the data owner and the cloud user. The data owner has a collection of documents D that he wants to outsource to the cloud provider in a form C, encrypted with a secret key K. To enable search capability over C, the data owner constructs a searchable index I from D and then uploads both the index I and the encrypted document collection C to the cloud provider. When an authorized cloud user wants to perform a search on remote data, she first connects to the data owner to acquire the secret key K and the trapdoor information. The trapdoor serves to output secure search query Q without revealing its original input. Moreover, the trapdoor learning process is a one-time operation, and thus, the cloud user does not need to contact the data owner anymore. Finally, the cloud user submits the search query Q to the cloud provider. Upon receiving Q, the cloud provider is responsible for searching the index I and returning the matching set of encrypted documents $L \subseteq C$. Later, the cloud user uses the secret key K to decrypt L to its original view.

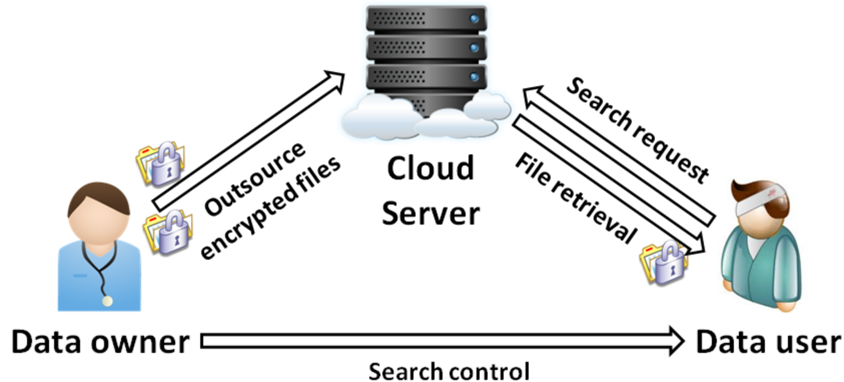


Figure 1. Cloud data hosting architecture.

As in previous works [6,11,12], the cloud provider is assumed to be an honest-but-curious entity. "Honest" means that the cloud provider can provide reliable data storage: it is always available to the users; it correctly follows the designated protocol specification; and it provides all services as expected. "Curious" means that the cloud provider may execute some background analysis to breach the confidentiality of the stored data. In the rest of the paper, we assume that the cloud provider and the adversary are the same entity. We do not consider the cloud provider being able to link the search query to a specific user.

3.2. Preliminaries and Notations

Let $D = \{D_1, D_2, \ldots, D_l\}$ be an original set of documents, and let $C = \{C_1, C_2, \ldots, C_l\}$ be an encrypted collection of documents from D. If D_i and D_j are two documents, we denote text t as their concatenation by $D_i||D_j$. If A is an algorithm, then $a \leftarrow A(\ldots)$ represents the result of applying the algorithm A.

In addition to the notations above, we also make use of cryptographic notations. We begin with definitions of Pseudorandom Functions (PRF) and Pseudorandom Permutations (PRP), which are polynomial-time computable functions that cannot be distinguished from random functions by any probabilistic polynomial-time (PPT) adversary, and random oracles to which all parties have black box access.

Definition 1. ***(Pseudorandom Function (PRF)).*** A function f: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^l$ is a (t,ϵ,q)-pseudorandom function if:

- Given a key $K \in \{0,1\}^k$ and an input $X \in \{0,1\}^n$, there is an algorithm to compute $F_K(X) = F(X,K)$.
- For any t-time oracle algorithm A, we have:

$$|Pr_{K \leftarrow \{0,1\}^k}[A^{f_K}] - Pr_{f \in \mathrm{F}}[A^f]| < \epsilon \tag{1}$$

where $\mathrm{F} = \{f : \{0,1\}^n \rightarrow \{0,1\}^l\}$ and A makes at most q queries to the oracle.

Definition 2. ***(Pseudorandom Permutation (PRP)).*** If pseudorandom function f in Definition 1 is bijective, then it is a pseudorandom permutation as follows: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$.

Definition 3. ***(Symmetric Key Encryption (SKE)).*** A symmetric key encryption scheme consists of the following PPT algorithms:

- $Gen(1^k)$: a key generation algorithm that inputs a security parameter k and outputs a secret key K.
- $Enc(K,m)$: a probabilistic algorithm that inputs a secret key K and message m, and outputs a ciphertext c.
- $Dec(K,c)$: a deterministic algorithm that inputs a secret key K and ciphertext c, and outputs a message m or special symbol $\perp$ (if decryption failed).

Definition 4. ***(SKE correctness).*** Given the symmetric encryption scheme SKE that consists of three algorithms (Gen, Enc, Dec), for all k and all m, such that $K \leftarrow Gen(1^k)$, we require:

$$Dec(K, Enc(K,m)) = m. \tag{2}$$

We also require SKE to be secure against Pseudorandom Chosen-Plaintext Attacks (PCPA). We now give the definition of the PCPA security of the SKE scheme.

Definition 5. ***(PCPA-security).*** Let SKE = (Gen, Enc, Dec) be a symmetric encryption scheme, A be an adversary, and there is a probabilistic experiment $PCPA_{SKE,A}(k)$ that is run as follows:

- Use secret parameter k to output the secret key $K \rightarrow Gen(1^k)$.
- The adversary A is given oracle access to $Enc_K()$.
- The adversary A outputs a message m.
- Let $c_0 \leftarrow Enc_K(m)$ and $c_1 \stackrel{R}{\leftarrow} C$. C denotes the set of all possible ciphertexts. A bit b is chosen at random, and c_b is given to the adversary A.
- The adversary A is again given to the oracle access to $Enc_K()$, and A runs the number of polynomial queries to output a bit b'.
- The experiment outputs one if $b = b'$, otherwise zero.

The symmetric encryption scheme SKE is PCPA-secure if for all polynomial-size adversaries A,

$$Pr[PCPA_{SKE,A}(k) = 1] \leq \frac{1}{2} + negl(k), \tag{3}$$

where the probability is over the choice of bit b and the coins of Gen and Enc.

4. Substring Search Algorithms

In this section, we present the most popular algorithms and data structures that allow a substring search on the plaintext data. Specifically, we focus on mature data structures, like suffix tree [28] and suffix array [29], that have been widely used in many substring search applications. We also, present the details of the recently-proposed position heap tree [21]. For each data structure, we give a short

overview with examples, and then, we present the computation and storage efficiency. Lastly, we present a discussion about choosing the right data structure to enable substring position search in an untrusted cloud environment.

4.1. Suffix Tree

Suffix tree [28,30,31] is a tree-like representation of text supporting a wide range of applications on strings. The suffix tree is pre-processed data structure that enables a substring search on the stored string. We now give the definition of the suffix tree:

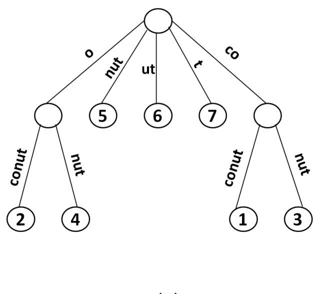
Definition 6. ***(Suffix tree).*** A suffix tree for string $t = t_1 \dots t_n$ is a rooted, directed tree with the following properties:

- Each edge is labeled with a non-empty substring of t, named the edge label.
- Every internal node has at least two children.
- No two edges out of a node have edge labels starting with the same character.
- The tree has n leaves, labeled from 1 to n.

Definition 7. ***(Path label).*** The path label of a node is the concatenation of the edge labels on the path from the root to that node.

Definition 8. ***(Suffix tree search).*** A string χ is a substring of t if and only if it is a prefix of some suffix of t.

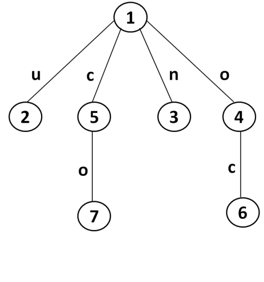
Figure 2a shows an example of a suffix tree constructed from the text "coconut". To check if a string χ is a substring, the algorithm searches for a path from the root whose labels match χ. For instance, searching for a string "coconut", we begin at the root node and start checking the neighbor edge labels, down to the matching node, i.e., Node 1 is the matching one, and it corresponds to the occurrence in the text. Similarly, the search of "co" leads us to the intermediate node whose leaf nodes $(1, 3)$ are the positions in the text.



(a)

Array Index	Suffix
1	coconut
3	conut
5	nut
2	oconut
4	onut
7	t
6	ut

(b)



(c)

Figure 2. An example of the data structure constructed from the text "coconut". (**a**) A suffix tree; (**b**) a suffix array; (**c**) a position heap tree.

The suffix tree can be constructed in $O(n)$ time for a string of length n [31]. Furthermore, it can be shown that a suffix tree has at most $2n$ nodes, and storing edge label for all edges would require $O(n^2)$ in the worst case. (Consider a suffix tree for the strings $t_1 t_2 \dots t_n$, where t_i is unique. The suffix tree would contain a distinct edge for each of the n suffixes $t_1 \dots t_n$, $t_2 \dots t_n$, $\dots$, t_n. These suffixes have a total length of $O(n^2)$.) Searching for a substring χ of length m takes $O(m + occ)$ time to find all occurrences of χ, where occ is the number of occurrences.

4.2. Suffix Array

A suffix array [29] is a sorted index array of all suffixes of a string. The suffix array data structure is used in full text indices, within the field of computational biology and others.

Definition 9. ***(Suffix array).*** Given a text t of length n, the suffix array for t is an array of integers ranging from one to n specifying the lexicographic ordering of the suffixes of the string t.

A suffix tree can be built in $O(n)$ time for a string of length n [29]. To search for a substring χ of length m, the search can be executed as simple binary search over the suffix array, i.e., for each element in the suffix array, we then compare the suffix of t at the element position with a substring χ. Thus, the search for any substring can be performed in $O(m \times log(n))$ time. This complexity can be improved by adding the longest common prefix information, so the search can be executed in $O(m + log(n))$ (see [29] for details).

Consider an example of a suffix array in Figure 2b constructed from the text "coconut". Searching for "coconut" gives the occurrence of (1), while searching for "co" results in $(1, 3)$ occurrences in the text.

4.3. Position Heap Tree

We now give an overview of the position heap tree data structure [21].

Definition 10. ***(Position heap).*** The position heap Λ of text t is a tree constructed by iteratively inserting the suffixes $(t_1, t_2, \ldots, t_n)$ of t in ascending order into Λ. That is, t_i is inserted by creating a new node in Λ that is the shortest prefix of t_i that is not already a node of the tree and labeling it with position i.

Figure 2c shows an example of position heap tree Λ constructed from the text "coconut". The first suffix "t" of text creates the root node in Λ with the position label of "1". Next, the second suffix "ut" of text creates a new node with Position 2 and connecting edge with label "u"; the third suffix "nut" creates a new node with Position 3 and connecting edge with label "n". Similarly, the seventh suffix "coconut" creates a new node with Position 7 and connecting edge with label "o" since there is already a node in Λ with edge label "c" created by the fifth suffix "conut". Following Definition 10, the position heap Λ is constructed. The construction can be executed for any text t, and since it is deterministic, the position heap Λ for a text is unique.

We now present the definition of the search in the position heap tree.

Definition 11. ***(Position heap search).*** The position heap search of all occurrences of a substring χ of text t in Λ consists of the following steps:

- Index into the position heap Λ to find the longest prefix p of χ that is a node of Λ. For each ancestor p' of p, lookup the position i stored in p'. Here, position i is an occurrence of p'. Determine if this occurrence is followed by $\chi - p'$. If yes, report i as an occurrence of χ.
- If $p = \chi$, also report all positions of the descendants of p.

Using the example tree in Figure 2c, the search for a substring "co" leads to the node (7). The set of traversed ancestor nodes $(5, 1)$ needs an inspection with text t. Indeed, only Position 5 matches the substring "co". Therefore, the positions of substring query "co" are $(5, 7)$. In the case of substring query "coconut", the search algorithm falls off the tree; thus, the search algorithm returns a set of traversed nodes $(7, 5, 1)$ for an inspection, where 7 is the only matching occurrence of "coconut" in the text.

The position heap tree for a text of length n can be constructed in $O(n)$ time [21]. All positions of substring χ of length m can be found in $O(m^2 + occ)$, where occ is the number of occurrences reported. We refer the reader to [21] for a detailed discussion on position heap properties.

4.4. Discussion

Recall that our goal is to construct a scheme that allows a substring search over encrypted data outsourced to the cloud. In our system model, the data owner has a set of documents with sensitive information that he wants to upload in encrypted form to the cloud provider. To enable the substring search, the data owner constructs a searchable index I from the set of plaintext documents D, and then, he places I at the cloud provider to allow the substring search. Our goal is to choose an optimal data structure that has low storage requirements and fast search execution. Later, we use the selected data structure to construct the searchable index I in our scheme.

As we have noted previously, many substring matching algorithms have been proposed, and they differ in terms of storage requirements and search execution. We outline the comparison of substring search data structures in Table 1. We use several comparison parameters: the construction time, the search execution time and the storage requirements. The construction time corresponds to the time it takes to create a data structure with the input of the text t of size n. The search execution time is the time it takes to find all occurrences of substring χ of length m in the text t. The cloud storage describes a storage of all combined textual labels in each data structure.

From Table 1, we can see that the suffix tree, suffix array and position heap tree have $O(n)$ preprocessing time of text t of size n. In search, the suffix tree has the best $O(m + occ)$ execution time for a substring χ of length m. However, the suffix tree has at most $2n$ nodes, and it would take $O(n^2)$ space to store the text at the cloud provider. On the other side, the suffix array can be constructed with n elements, but it has $O(n^2)$ storage. Only the position heap tree allows us to have the low storage $O(n)$ with n nodes; however, the substring search execution takes $O(m^2 + occ)$ time. In the rest of the paper, we use the position heap tree data structure as the main construction block for our scheme. In our choice between the data structures, we believe that the $O(n)$ storage requirement is the predominant criteria, since expanding any large dataset (e.g., the human genome with three billion letters) to a $O(n^2)$ storage would cause a substantial waste of cloud computing resources.

Table 1. Comparison of plaintext substring search data structures. n is the length of the text t, m is the length of the substring χ, occ is the number of occurrences of χ in t.

Data Structure	Construction	Search	Cloud Storage
Suffix Tree	$O(n)$	$O(m + occ)$	$O(n^2)$
Suffix Array	$O(n)$	$O(m + log(n))$	$O(n^2)$ [1]
Position Heap Tree	$O(n)$	$O(m^2 + occ)$	$O(n)$

[1] Note that the suffix array data structure stores only the array of integers (no need to store the suffixes of text), and the array can be accessed by running a binary search algorithm in $log(n)$ time, i.e., each time we access the element in the suffix array, we execute a lexicographical comparison of the strings of the suffix at the element position and the the given substring query. This can be executed locally (by the data owner); however, in our system model defined in Section 3.1, the data owner sends the data and constructed searchable index to the malicious cloud provider. Both the data and the searchable index are encrypted, so no plaintext (and no lexicographical order) is leaked to the cloud provider. If we were to encrypt the suffix array by encrypting each element of the suffix array, then the cloud provider would not be able to execute the search in $log(n)$ (in fact, it would observe the ciphertext at each element in the array, which gives no order in binary search execution). However, to keep the binary search $log(n)$ time, one solution is to store encrypted suffixes in each node of the binary search tree and to use an expensive homomorphic encryption (i.e., work by Gentry et al. [32]) that allows the search on the encrypted binary search tree. However, this would take $O(n^2)$ as the worst case storage for all suffixes in the tree.

5. Substring Position Searchable Symmetric Encryption

5.1. Algorithm Definitions

Definition 12. ***(Substring Position Searchable Symmetric Encryption (SSP-SSE)).*** A tree-based SSP-SSE scheme over a set of documents D is a tuple of six polynomial-time algorithms (KeyGen, BuildTree, Encrypt, ConstructQuery, Search, Decrypt), as follows:

1. $K \leftarrow KeyGen(1^k)$: a probabilistic key generation algorithm to setup the SSP-SSE scheme. The algorithm takes a secret parameter k and outputs a set of secret keys K.
2. $(\Lambda) \leftarrow BuildTree(D)$: a deterministic algorithm to build a position heap tree Λ. The algorithm takes a document collection $D = (D_1, \ldots, D_l)$ and outputs a position heap tree Λ.
3. $(I, C) \leftarrow Encrypt(K, \Lambda, D)$: a probabilistic algorithm to encrypt a position heap tree Λ and document corpus D. The algorithm inputs a set of secret keys K, a position heap tree Λ and a documents corpus D. The output of algorithm is a searchable index I and encrypted collection $C = (C_1, \ldots, C_l)$.
4. $[(Q) \leftarrow ConstructQuery(K, \chi)] \leftrightarrow [(L) \leftarrow Search(I, Q)]$: two deterministic algorithms that are executed interactively between the cloud user and the cloud provider. The ConstructQuery algorithm inputs a set of secret keys K, a substring χ, and it outputs a search query Q. The Search is an algorithm that inputs a searchable index I and a search query Q. The algorithm finds the set of matching encrypted document identifiers $L \in C$.
5. $(D_i, pos_{D_i}) \leftarrow Decrypt(K, C_i)$: a deterministic algorithm that takes a set of secret keys K and a ciphertext C_i as input and outputs an original document D_i, $\forall i \in [1; n]$ and a set of χ's positions pos_{D_i} in D_i.

Definition 13. ***(SSP-SSE correctness).*** We say that the tree-based SSP-SSE scheme is correct if $\forall k \in \mathbb{N}$, $\forall$ K produced by $KeyGen(1^k)$, $\forall$ D, $\forall$ Λ output by $BuildTree(D)$, $\forall$ χ, $\forall$ $i \in [1; n]$:

$$\begin{aligned} Search(Encrypt(K, \Lambda, D), ConstructQuery(K, \chi)) = \\ = C(\chi) \bigwedge Decrypt(K, C_i) = (D_i, pos_{D_i}) \end{aligned} \tag{4}$$

The SSP-SSE correctness ensures the proper output if all SSP-SSE algorithms are executed honestly by the cloud provider.

5.2. Security Model Definitions

The security goal of any searchable encryption scheme is to reveal as little information as possible to the adversary. Intuitively, in the SSP-SSE scheme, we want to provide the following security guarantees: given a searchable index I and a set of encrypted documents $C = \{C_1, \ldots, C_l\}$ to the adversary, no valuable information about the original documents $D = \{D_1, \ldots, D_l\}$ is leaked to the adversary; given a set of incoming search queries $Q = \{Q_1, \ldots, Q_t\}$, the adversary cannot learn any practical information about the content of the search query Q_i or the original document collection D. However, these security guarantees are difficult to achieve, and most known searchable encryption schemes [3,4,6,12,13] reveal some information, namely the access pattern and the search pattern. In SSP-SSE, we follow a similar approach to [6] to weaken the security guarantees and allow some limited information to the adversary.

Definition 14. ***(Access pattern).*** Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$, the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t, the access pattern $\kappa(C, Q)$ includes the set of document identifiers induced by a search query vector Q.

Definition 15. ***(Search pattern).*** Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$, the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t, the search pattern $\gamma(C, Q)$ is a $n \times t$ binary

matrix, such that $\forall i \in [1; n]$ and $\forall j \in [1; t]$, the cell element of i-th row and j-th column is one, if a document identifier id_i is returned by a search query Q_j. The search pattern reveals whether the same search was executed in the past or not.

Since our solution is based on the position heap tree data structure, we would like to capture the path pattern security notion. The path pattern of the position heap tree reveals the path traversed from the root node to the matching node for a given search query.

Definition 16. ***(Path pattern).*** Given the n encrypted documents C, where $C = \{C_1, \ldots, C_l\}$, and the searchable index I built from the document collection, the path pattern of (C, I) induced by the search query vector Q, where $Q = \{Q_1, \ldots, Q_t\}$ of size t, is a tuple $\delta(C, I, Q)$ that reveals the set of identifiers of nodes in the index I that are reached by query $Q_{i \in [1;t]}$.

Now, we define the leakage functions to capture all of the information leakage we have in this work:

- Leakage $\mathbb{L}_1(I, C)$. Given the encrypted collection $C = \{C_1, \ldots, C_l\}$ and the searchable index I, the leakage consists of the following information: the number of encrypted documents, the size of encrypted documents and the identifier of each encrypted document.
- Leakage $\mathbb{L}_2(Q, I, C)$. Given the encrypted collection $C = \{C_1, \ldots, C_l\}$, the searchable index I and the search query Q, the leakage function outputs the access pattern $\kappa(C, Q)$, search pattern $\gamma(C, Q)$ and path pattern $\delta(C, I, Q)$.

Definition 17. ***(Security against adaptive Chosen-Query Attack (CQA2)).*** Let SSP-SSE be tree-based SSE scheme that consists of six algorithms as described in Definition 12. Let $\mathbb{A}$ be a stateful adversary and $\mathbb{S}$ be a stateful simulator. We consider two probabilistic experiments $Real_{\mathbb{A}}$ and $Ideal_{\mathbb{A},\mathbb{S}}$ that involve $\mathbb{A}$, as well as $\mathbb{S}$, with two stateful leakage algorithms $\mathbb{L}_1$ and $\mathbb{L}_2$ and security parameter k:
$\underline{Real_{\mathbb{A}}(k)}$: The challenger runs the $KeyGen(1^k)$ to output the key set K. The adversary $\mathbb{A}$ sends constructed plaintext position heap tree Λ and collection D to the challenger and receives a tuple $(I, C) \leftarrow Encrypt(K, \Lambda, D)$ from the challenger. The adversary $\mathbb{A}$ makes a polynomial number of adaptive string searches $\chi = \chi_1, \ldots, \chi_t$ and sends them to the challenger. $\mathbb{A}$ then receives the search queries generated by the challenger, such that $Q_i \leftarrow ConstructQuery(K, \chi_i)$. The adversary returns one if his or her queries return the expected result, otherwise zero.
$\underline{Ideal_{\mathbb{A},\mathbb{S}}(k)}$: The adversary $\mathbb{A}$ outputs the tuple (D, Λ), where $\Lambda \leftarrow BuildTree(D)$, and sends it to the simulator. Given the leakage $\mathbb{L}_1$, simulator $\mathbb{S}$ generates the tuple (I, C) and sends it to the adversary. $\mathbb{A}$ makes a polynomial number of adaptive string searches $\chi = \chi_1, \ldots, \chi_t$ and sends them to the simulator. Given the leakage $\mathbb{L}_2$, the simulator $\mathbb{S}$ sends the appropriate search queries to the adversary. Finally, $\mathbb{A}$ returns one in the case of successful experiment, otherwise zero.

We say that SSP-SSE is adaptively secure against the chosen-query attack if for all probabilistic polynomial time adversaries $\mathbb{A}$, there exists a non-uniform probabilistic polynomial time simulator $\mathbb{S}$, such that:

$$|Pr[Real_{\mathbb{A}}(k)] = 1 - Pr[Ideal_{\mathbb{A},\mathbb{S}}(k) = 1]| \leq negl(k) \tag{5}$$

5.3. SSP-SSE Construction

We now present the details of the proposed SSP-SSE scheme. The scheme consists of two phases, namely the setup phase and search phase. The setup phase is done once by the data owner to upload the set of encrypted documents and the searchable index to the cloud provider. In this phase, the data owner uses the KeyGen, BuildTree and Encrypt algorithms to encrypt the document collection, as well as to construct the searchable index. The search phase is performed every time by the cloud user when a query is submitted. In this phase, the cloud user invokes the ConstructQuery algorithm to generate the search query. The cloud provider executes the Search algorithm to output matching results. Finally,

the cloud user invokes the Decrypt algorithm to decrypt the document collection to the original view. Our scheme is based on a set of important notations shown in Algorithm 1. We outline the details of setup phase in Section 5.3.1. We later show the search phase in Section 5.3.2.

Algorithm 1: Notations.

- $t = (t_1, t_2, \ldots, t_n)$ - the text constructed from document collection D. t_i is the letter in text t at position i.
- $v[i]$ - the node in Λ at index i ($i \in [1;n]$).
- $V(v[i])$ - the position value of node $v[i]$ in Λ.
- $pid(D_j) = id(D_j)||pos_{D_j}$ - concatenation of document identifier D_j ($j \in [1;l]$) with position i of character t_i in the document D_j.
- $L(v[i])$ - the path label of node $v[i]$ in position heap tree Λ.
- $L_{parent}(v[i])$ - the path label of $v[i]$'s parent node.
- $depth(v[i])$ - the depth of node $v[i]$ in Λ.
- $\overline{v[i]}$ - the encrypted node in $\bar{\Lambda}$ at index i.
- $\overline{V}(\overline{v[i]})$ - the encrypted value of node $\overline{v[i]}$ in $\bar{\Lambda}$.
- $\overline{L}(\overline{v[i]})$ - the encrypted path label of node $\overline{v[i]}$.
- $\overline{L_{parent}}(\overline{v[i]})$ - the encrypted path label $\overline{v[i]}$'s parent node.
- $descendants(\overline{v[i]})$ - the set of descendant (child) nodes in the subtree rooted at node $\overline{v[i]}$. If $\overline{v[i]}$ is the leaf node, then $descendants(\overline{v[i]}) = 0$.
- $ancestors(\overline{v[i]})$ - the set of ancestor (parent) nodes at node $\overline{v[i]}$. If $\overline{v[i]}$ is root node, then $ancestors(\overline{v[i]}) = 0$.

5.3.1. Setup Phase

The setup phase (Algorithm 2) includes the KeyGen, the BuildTree and the Encrypt algorithms. Let k be a security parameter, and let SKE = (Gen, Enc, Dec) be a PCPA-secure symmetric-key encryption scheme. The data owner begins with the KeyGen algorithm that inputs a secret parameter k and outputs a set of keys K_1, K_X, K_Y, K_V and set of random keys K_Q, K_L, K_2,$K_3 \stackrel{R}{\leftarrow} \{0,1\}^k$. He will use these keys to encrypt the document collection $D = (D_1, \ldots, D_l)$ and construct searchable index I.

First, the data owner constructs a position heap tree Λ using the BuildTree algorithm outlined in Definition 10. The BuildTree algorithm inputs the text t, where t is constructed from the document collection $D = D_1||\$ \ldots \$ ||D_l$ padded with the unique terminator string \$ and outputs the single position heap tree Λ. In order to handle multiple documents in the collection, the data owner adds auxiliary information to each node that contains the document identifier D_i and the position of the letter in D_i. For example, if the character "a" appears in the document D_1 at Position 1, the node in Λ will have extra information of $pid(D_1) = id(D_1)||1$. Formally, we concatenate the identifier of D_j ($j \in [1;l]$) with position i of character t_i in the document D_j, i.e., $pid(D_j) = id(D_j)||pos_{D_j}$, and add this information in each node in the position heap tree. Figure 3a shows an example of position heap tree Λ of the text "ab\$aaa\$bb" constructed from three concatenated documents (D_1, D_2, D_3), where D_1 has text "bb", D_2 has text "aaa" and D_3 has text "ab". Note, a search of "ab" in the position heap tree returns a set of nodes $(9, 4, 1)$ where only 9 is the matching node, and it describes the document position of $D_3||2$. Thus, the search query "ab" appears only in the document D_3 at Position 2.

Algorithm 2: SSP-SSE setup phase.

Let $SKE = (Gen, Enc, Dec)$ be a PCPA-secure symmetric-key encryption scheme; let $F : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ be a PRF; and let $P : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a PRP.

SETUP PHASE.

$\underline{KeyGen(1^k)}$: given the security parameter k, generate $K_1, K_X, K_Y, K_V \leftarrow SKE.Gen(1^k)$ and K_Q, K_L, K_2,$K_3 \xleftarrow{R} \{0,1\}^k$. Output the key set $K = (K_1, K_X, K_Y, K_V, K_Q, K_L, K_2,K_3)$.

$\underline{BuildTree(D)}$: given the document collection $D = (D_1, \ldots, D_l)$:

1. construct text $t = t_1 t_2 \ldots t_n$ from document collection D, and and input t of size n to build the position heap tree Λ.
2. index into Λ, for each node $v[i]$ $(i \in [1, n])$:
   (a) set $V(v[i]) = pid(D_j)||V(v[i])$, where D_j $(j \in [1, l])$ is the document in collection D.
3. output the position heap tree Λ

$\underline{Encrypt(K, \Lambda, D)}$: given the secret key set K, position heap tree Λ and the set of documents $D = (D_1, \ldots, D_l)$.

Build encrypted tree:

1. index into Λ, traverse from the root node:
2. for each node $v[i]$ $(i \in [1, n])$:
   (a) set $\overline{L}(\overline{v[i]}) = F_{K_Q}(L(v[i])||depth(v[i])||\overline{L_{parent}}(\overline{v[i]})||K_L)$ (i.e., apply PRF F with key K_Q on the concatenation of the path label L of $v[i]$, depth of the node $v[i]$, encrypted parent label $\overline{L_{parent}}$ of $v[i]$ and the secret key K_L.
   (b) set $\overline{V}(\overline{v[i]}) = SKE.Enc_{K_V}(i)$.
3. output encrypted $\bar{\Lambda}$.

Build encrypted arrays:

1. for each character t_i of t indexed from right-to-left (i.e., $t_n t_{n-1} \ldots t_1$), set an array $X[P_{K_2}(i)] = SKE.Enc_{K_X}(t_i)$.
2. for each $i = [1, n]$: set an array $Y[P_{K_3}(i)] = SKE.Enc_{K_Y}(V(v[i]))$.

Encrypt document collection:

1. for each document D_i where $i \in [1, l]$, let $C_i \leftarrow SKE.Enc_{K_1}(D_i)$.
2. output $C = (C_1, C_2, \ldots, C_l)$.

Output: index $I = (\bar{\Lambda}, X, Y)$ and encrypted document collection $C = (C_1, C_2, \ldots, C_l)$.

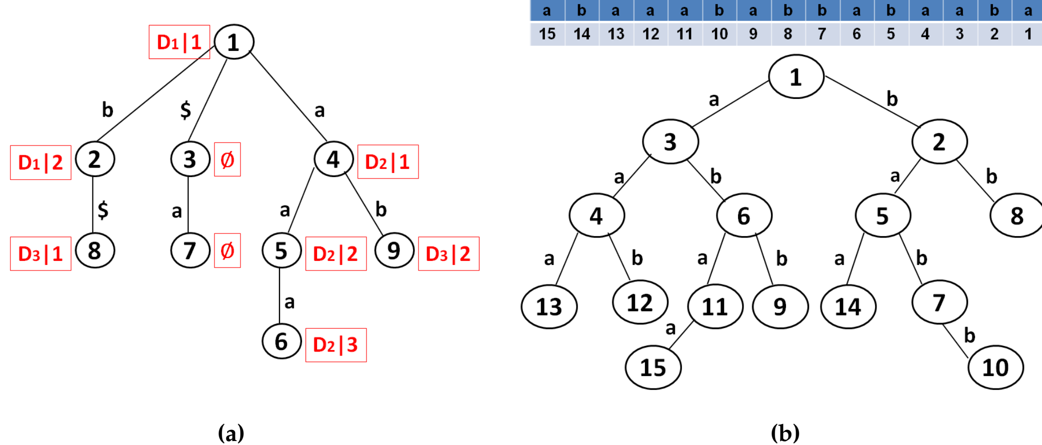
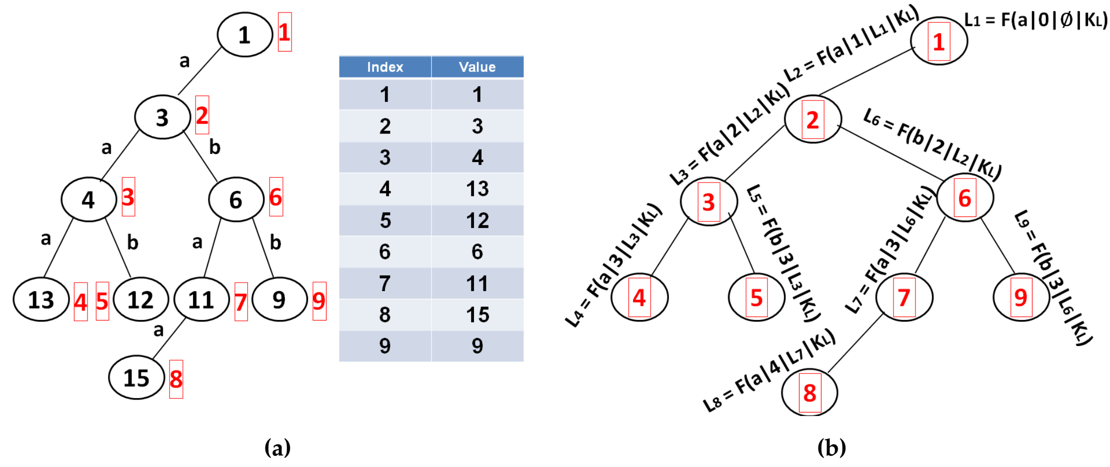


Figure 3. An example of the position heap tree. (**a**) Constructed from the text “abaaabb”’ extracted from documents (D_1, D_2, D_3); (**b**) constructed from the text “abaaababbabaaba”.

The data owner constructs the searchable index that is based on the position heap tree data structure. To present the details, we use an example of the position heap tree Λ shown in Figure 3b. The figure depicts constructed position heap tree Λ from text $t =$ "abaaababbabaaba" and text array X (shown at the top of the figure), where each array element has a single character of text t indexed from right-to-left.

The data owner begins by extracting position information from Λ as follows: index each node in tree Λ, and create a position array Y, such that each index in Y corresponds to the node value of Λ. Figure 4a shows an example of the left-side branch of position heap tree Λ and constructed position array Y. In this example, nodes in Λ are marked with the red color index, and their corresponding values (positions) are stored as elements in Y. (In Figure 4a we show an example of the position array Y for nine nodes of Λ for demonstration purposes only. The actual algorithm is executed on all nodes in Λ.) With this, the data owner is ready to encrypt the position heap tree Λ, text array X and position array Y data structures.



Index	Value
1	1
2	3
3	4
4	13
5	12
6	6
7	11
8	15
9	9

Figure 4. Construction of a searchable index. (**a**) An example of position array Y; (**b**) an example of the path label encryption of position heap tree.

First, to encrypt the position heap tree Λ, the data owner uses a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k$ and PCPA-secure symmetric-key encryption scheme SKE = (Gen, Enc, Dec). For each node i in Λ, the data owner applies PRF F with key K_Q on the concatenation of the path label of node i, depth of the node i, the encrypted path label of the i'-th parent node and the secret key K_L. Figure 4b shows an example of the path label encryption. For instance, the label of Node 4 is $L_4 = F_{K_Q}(a||3||L_3||K_L)$, where $L_3 = F_{K_Q}(a||2||L_2||K_L)$. The root path label is a special case, and its label is $L_1 = K_{K_Q}(a||0||\varnothing||K_L)$. In this way, the data owner encrypts all path labels in the tree. This hides the plaintext path labels of the same character at different levels of the tree Λ. Moreover, this makes the ciphertext unique for all path labels in the tree. To hide the index information of each node in Λ, the data owner uses SKE encryption with key K_V on the index of the node, i.e., $V_i = SKE.Enc_{K_V}(i)$, where $i \in [1, n]$. For instance, the value of Node 8 is $V_8 = SKE.Enc_{K_V}(8)$. With no plaintext left in Λ, the data owner outputs an encrypted position heap tree $\bar{\Lambda}$.

Second, the data owner utilizes a pseudorandom permutation $P : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ and PCPA-secure symmetric-key encryption SKE to hide plaintext elements of text array X and position array Y. For each i ($i \in [1, n]$) in X, the data owner applies PRP P with secret key K_2 on each i, i.e., $P_{K_2}(i)$. For each corresponding character t_i at index i in X, he applies SKE with secret key K_X on character t_i, i.e., $SKE.Enc_{K_X}(t_i)$. The data owner sets the encrypted array X as $X[P_{K_2}(i)] = SKE.Enc_{K_X}(t_i)$. Next, for each i ($i \in [1, n]$) in Y, he utilizes PRP P with secret key K_3 and SKE with secret key K_Y as follows: $Y[P_{K_3}(i)] = SKE.Enc_{K_Y}(V_i)$, where V_i is the i'-th element in Y.

Finally, the data owner encrypts each document D_i in the collection D using the PCPA-secure symmetric-key encryption scheme SKE with secret key K_1 to produce the encrypted document $C_i \leftarrow SKE.Enc_{K_1}(D_i)$. After all, the data owner uploads the encrypted collection C along the searchable index $I = (\bar{\Lambda}, X, Y)$ to the cloud provider. Now, the collection is available for selective cloud retrieval.

5.3.2. Search Phase

The search phase (Algorithm 3) includes both the ConstructQuery and the Search interactive algorithms that are executed between the cloud user and the cloud provider. The cloud user keeps the set of secret keys $K = (K_1, K_X, K_Y, K_V, K_Q, K_L, K_2, K_3)$ received from the data owner.

In order to search a substring χ of length m, the cloud user begins with creating a search query Q: for each character χ_i in χ; he applies PRF F with secret key K_Q on the concatenation of χ_i, i, the output of previous query Q_{i-1} and the secret parameter K_L. The cloud user forms a query $Q = (Q_1, \ldots, Q_l)$ and sends Q to the cloud provider. For instance, for a substring "aba", the cloud user creates $Q_1 = F_{K_Q}(a|1|L_{root}|K_L)$ (L_{root} is shared by the data owner to the cloud user), $Q_2 = F_{K_Q}(b|2|Q_1|K_L)$, $Q_3 = F_{K_Q}(a|3|Q_2|K_L)$ and sends $Q = (Q_1, Q_2, Q_3)$ to the cloud provider. The cloud server indexes into the encrypted position heap tree $\bar{\Lambda}$, and for each given Q_i, it matches encrypted label $\overline{L}$ of each node in $\bar{\Lambda}$ to Q_i and continues until the longest matching node $\overline{v_{match}}$ in $\bar{\Lambda}$ is found. The cloud server returns the set of ancestor and descendant nodes of $\overline{v_{match}}$ to the cloud user. Using the example in Figure 4b and search query $Q = (Q_1, Q_2, Q_3)$, the cloud provider returns the set of encrypted ancestor nodes ($SKE.Enc_{K_V}(1)$, $SKE.Enc_{K_V}(2)$, $SKE.Enc_{K_V}(6)$) and the set of encrypted descendant nodes ($SKE.Enc_{K_V}(7)$, $SKE.Enc_{K_V}(8)$).

Now, the cloud user applies the SKE scheme with secret key K_V to decrypt the ancestor and descendant nodes, i.e., $(1, 2, 6)$ ancestor nodes and $(7, 8)$ descendant nodes. Next, he uses PRP P with secret key K_3 on each decrypted node, i.e., $y_{idx} = P_{K_3}(idx)$, where idx is $(1, 2, 6, 7, 8)$, and sends the resulting query y to the cloud provider.

The cloud provider uses array Y to fetch the elements at index y_i ($i \in [1; 5]$) as $Y[y_i]$ and sends back the results. Once received, the cloud user applies SKE with secret key K_Y to decrypt the positions in the ancestor and descendant nodes, i.e., $(1, 3, 6)$ positions in ancestor nodes and $(11, 15)$ positions in descendant nodes. According to Definition 11, descendant nodes $(11, 15)$ are the positions of query "aba" in the text, and ancestor nodes $(1, 3, 6)$ require an inspection, since some of them can point at "aba" in the text. Note, since the substring "aba" has a length of three, the substring may exist at positions $(6, 5, 4)$ and $(3, 2, 1)$ in the text. Therefore, to launch the inspection, the cloud user applies PRP P with secret key K_X at each position $(6, 5, 4, 3, 2, 1)$ as $x_{idx} = P_{K_2}(idx)$ and sends query x to the cloud provider.

Now, the cloud provider uses array X and sends back the elements of the array at index $X[x_i]$ ($i \in [1; 6]$). The cloud user uses SKE.Enc with secret key K_X to decrypt the characters t_j at positions $(6, 5, 4, 3, 2, 1)$ (i.e., received characters are (a, b, a, a, b, a)). Using this information, the cloud user verifies if substring characters χ_i match received characters t_j at each ancestor position. The inspection of ancestors shows that only $(6, 3)$ are the positions. Thus, the cloud user concludes that substring query "aba" is at position $(3, 6, 11, 15)$ in the text.

Note, if multiple documents are involved in the original text construction, ancestor and descendant nodes contain the document identifiers, which can be later used by the cloud user to download the matching encrypted documents and decrypt them locally using PCPA-secure symmetric-key encryption SKE with secret key K_1.

Algorithm 3: SSP-SSE search phase.

SEARCH PHASE.

$[(Q) \leftarrow ConstructQuery(K, \chi)] \leftrightarrow [(L) \leftarrow Search(I, Q)]$ is an interactive protocol between the cloud user and the cloud provider. The cloud user keeps the key set $K = (K_1, K_X, K_Y, K_V, K_Q, K_L, K_2, K_3)$ and queries cloud provider for a substring χ. The cloud provider executes search on searchable index $I = (\bar{\Lambda}, X, Y)$ and returns results back to the cloud user.

1. cloud user: given the secret key K_Q and the string of interest χ, output the search query Q as follows:
   (a) for each character χ_i, $i \in [1; m]$, where $m = |\chi|$
      i. set $Q_i = F_{K_Q}(\chi_i || i || Q_{i-1} || K_L)$ (i.e., apply PRF F with key K_Q on the concatenation of character χ_i of χ, i, output of query Q_{i-1} and secret key K_L.)
      ii. set $Q = (Q_1, Q_2, \ldots, Q_m)$.
   (b) send search query Q to the cloud provider.
2. cloud provider: index into $\bar{\Lambda}$, start at the root node
   (a) for each Q_i and each node $\bar{v}$ in $\bar{\Lambda}$, match the encrypted label $\overline{L}(\bar{v})$ to Q_i. Continue until the longest node $\overline{v[match]}$ is found.
   (b) If $\overline{v[match]} \neq \perp$, return $(descendants(\overline{v[match]}), ancestors(\overline{v[match]}))$, otherwise return $\perp$.
3. cloud user: let TMP-AN and TMP-DE be two arrays; let TMP-RES = TMP-AN + TMP-DE be an array that combines elements from TMP-AN and TMP-DE.
   (a) for each node $\bar{v}$ in $ancestors(\overline{v[match]})$:
      i. if $SKE.Dec_{K_V}(\overline{V}(\bar{v})) = \perp$, abort. Otherwise, output idx, and add to TMP-AN.
   (b) for each node $\bar{v}$ in $descendants(\overline{v[match]})$:
      i. if $SKE.Dec_{K_V}(\overline{V}(\bar{v})) = \perp$, abort. Otherwise, output idx, and add to TMP-DE.
   (c) set TMP-RES = TMP-AN + TMP-DE, for each idx in TMP-RES, set $y_{idx} = P_{K_3}(idx)$. Send $(y_1, \ldots, y_{num})$ to the cloud provider.
4. cloud provider: get $Y_i = Y[y_i]$ ($i \in [1, num]$), output $(Y_1, \ldots, Y_{num})$.
5. cloud user: let AN and DE be two arrays.
   (a) for $i = [1, m]$, if $SKE.Dec_{K_Y}(Y_i) = \perp$, abort; otherwise, add output to AN.
   (b) for $i = [m+1, num]$, if $SKE.Dec_{K_Y}(Y_i) = \perp$, abort; otherwise, add output to DE.
   (c) parse each element from AN as $pid(D)||pos$.
   (d) for each pos in AN, for $j = pos$, $j > (j - m)$ (where $(j - m) > 0$), $j - -$, let $x_j = P_{K_2}(j)$, send $(x_1, \ldots, x_h)$ to the cloud provider.
6. cloud provider: get $X_i = X[x_i]$ ($i \in [1, h]$), output $(X_1, \ldots, X_h)$.
7. cloud user: let REAL-AN be an array.
   (a) for $i = [1; h]$, if $SKE.Dec_{K_X}(X_i) = \perp$, abort. Otherwise, parse the output as t_j.
   (b) for each pos in AN, compare characters $\chi_u = t_j$, where $u = 0$, $u < m$, $u + +$ and $j = pos$, $j > (j - l)$ (where $(j - l) \not< 0$), $j - -$. If all $\chi_u = t_j$ match at given pos, add pos to REAL-AN; otherwise, ignore pos.
   (c) let RES = REAL-AN + DE. Parse each element of array RES as $id(D_h)||pos_{D_h}$, where pos_{D_h} is the position of substring χ in document D_h ($h \in [1, l]$).

$Decrypt(K_1, K_2, C_i)$:

1. retrieve set $C = (C_1, \ldots, C_k)$ from the cloud provider.
2. $D_i \leftarrow SKE.Dec_{K_1}(C_i)$, where $i \in [1; k]$.
3. output $((D_1, pos_{D_1}), \ldots, (D_k, pos_{D_k}))$.

6. Security and Performance Analysis

6.1. Security

In this section, we focus on the the security of the SSP-SSE scheme. First, we show that the SSP-SSE scheme is correct according to Definition 13. Second, we prove that the SSP-SSE scheme is secure against the Chosen-Query Attack (CQA-2) executed by the adaptive adversary according to Definition 17.

Theorem 18. (Correctness). *The Substring Positions Searchable Symmetric Encryption (SSP-SSE) scheme consisting of six polynomial-time algorithms (KeyGen, BuildTree, Encrypt, ConstructQuery, Search, Decrypt) is correct according to Definition 13.*

Proof. The index I in the Search algorithm consists of the encrypted position heap tree $\bar{\Lambda}$ and two arrays X, Y (both encrypted). Since the path labels in $\bar{\Lambda}$ and the search query Q are both encrypted with the same instance of pseudorandom function F with the same secret key K_Q, the correctness of the SSP-SSE scheme relies on the correctness of the pseudorandom function.

When the cloud provider receives the search query Q in the Search algorithm, it traverses the path labels in the encrypted position heap tree $\bar{\Lambda}$ according to Definition 11. Search query Q is constructed using the pseudorandom function F applied on the substring χ with key K_Q. Each encrypted path label in $\bar{\Lambda}$ is constructed using the pseudorandom function F with the key K_Q on the set of characters extracted from the plaintext document collection $D = \{D_1, \ldots, D_l\}$. The search algorithm outputs true if the document D_i contains the string of characters χ. Thus, the cloud provider outputs a set of documents that matches the search query Q. □

Theorem 19. (Security). *Let SKE be a symmetric PCPA-secure encryption scheme, F be a pseudorandom function and P be a pseudorandom permutation. Substring Position Searchable Symmetric Encryption (SSP-SSE) presented above is* $(\mathbb{L}_1, \mathbb{L}_2)$*-adaptively secure against chosen-query attacks defined in Definition 17 (CQA-2 security), where* $\mathbb{L}_1$ *and* $\mathbb{L}_2$ *are the possible leakages.*

In a nutshell, the proof of security of SSP-SSE scheme works as follows. The simulator $\mathbb{S}$ generates a simulated searchable index $\widetilde{I}$ that consists of simulated encrypted position heap tree $\widetilde{\Lambda}$, simulated position array $\widetilde{Y}$ and simulated text array $\widetilde{X}$, i.e., $\widetilde{I} = (\widetilde{\Lambda}, \widetilde{Y}, \widetilde{X})$; as well as the simulated set of ciphertexts $\widetilde{C} = \{\widetilde{C_1}, \ldots, \widetilde{C_l}\}$. Both $\widetilde{I}$ and $\widetilde{C}$ are constructed using the leakage $\mathbb{L}_1$ that discloses the number of encrypted documents, the size of the encrypted documents and the identifier of each encrypted document. The simulated encrypted position heap tree $\widetilde{\Lambda}$ is constructed using the pseudorandom function F and symmetric-key encryption SKE with random values $\{0, 1\}$. Both simulated $\widetilde{Y}$ and $\widetilde{X}$ are constructed using the pseudorandom permutation P and symmetric-key encryption SKE on random values $\{0, 1\}$. The security of the proposed scheme relies on the following assumptions. The pseudo-randomness of F guarantees that the simulated encrypted position heap tree $\widetilde{\Lambda}$ is indistinguishable from the real encrypted position heap tree $\bar{\Lambda}$. The pseudo-randomness of P will guarantee that simulated $\widetilde{Y}$ and $\widetilde{X}$ are indistinguishable from the real Y and X. Moreover, the simulated set of ciphertext $\widetilde{C}$ is indistinguishable from the real encrypted document collection C.

The search algorithm is simulated in a similar way that requires keeping track of different dependencies between the result output and the search query. However, since the real search query is constructed with pseudorandom function F and pseudorandom permutation P, the simulator is not able to distinguish it from the simulated query. Similarly, the simulated outcome of the search is indistinguishable from the real set of nodes. We outline the formal proof as follows.

Proof. Polynomial-size simulator $\mathbb{S}$ can be defined such that for any challenger and any polynomial-time adversary $\mathbb{A}$, the outputs of two experiments $Ideal_{\mathbb{A},\mathbb{S}}(k)$ and $Real_{\mathbb{A}}(k)$ with secret parameter k are computationally indistinguishable according to Definition 17. We now describe the details of experiment $Ideal_{\mathbb{A},\mathbb{S}}(k)$ that presents the simulator $\mathbb{S}$.

- $\mathbb{S}(1^k, \mathbb{L}_1)$: The simulator $\mathbb{S}$ has a leakage $\mathbb{L}_1$, which gives the simulator information about the number and size of documents, as well as identifier of each encrypted document. The simulator $\mathbb{S}$ randomly generates a set of simulated ciphertexts $\widetilde{C}$ and simulated searchable index $\widetilde{I}$ as follows:
  - Simulator $\mathbb{S}$ outputs the set of ciphertexts $\widetilde{C} = \{\widetilde{C_1}, \ldots, \widetilde{C_l}\}$, where $\widetilde{C}_i \xleftarrow{R} \{0,1\}^{|D_i|}$.
  - Simulator $\mathbb{S}$ sets the simulated encrypted position heap tree $\widetilde{\Lambda}$, where each node is set as $\widetilde{V}(\nu[i]) \xleftarrow{R} \{0,1\}^k$ and each path label of node $\nu[i]$ is set as $\widetilde{L}(\nu[i]) \xleftarrow{R} \{0,1\}^k$, where $i \in [1;n]$. The simulator outputs the encrypted position heap tree $\widetilde{\Lambda}$.
  - Simulator $\mathbb{S}$ then constructs simulated arrays $\widetilde{X}$ and $\widetilde{Y}$: $\widetilde{X}[i] = \{0,1\}^k$ and $\widetilde{Y}[i] = \{0,1\}^k$, where $i \in [1;n]$.
  - Simulator $\mathbb{S}$ outputs simulated searchable index $\widetilde{I} = (\widetilde{\Lambda}, \widetilde{Y}, \widetilde{X})$ and the set of simulated ciphertexts $\widetilde{C}$.

At this point, the simulator $\mathbb{S}$ generated the set of simulated encrypted documents $\widetilde{C}$ and simulated index $\widetilde{I}$. Next, the adversary $\mathbb{A}$ adaptively queries the polynomial-size simulator $\mathbb{S}$ as follows.

- $\mathbb{S}(1^k, \mathbb{L}_1, \mathbb{L}_2)$: The adversary $\mathbb{A}$ sends a new query Q to the simulator $\mathbb{S}$. The simulator then starts collecting various dependencies between the incoming search query and the resulting output.
  - With given search query Q, simulator $\mathbb{S}$ traverses the simulated encrypted position heap tree $\widetilde{\Lambda}$ starting from the root node, following the simulated path labels to find the set of matching encrypted nodes in $\widetilde{\Lambda}$. The simulator outputs the set of simulated matching nodes: $\widetilde{ancestors}$ and $\widetilde{descendants}$.
  - With given search requests $(\widetilde{y}_1, \ldots, \widetilde{y}_{num})$, the simulator performs a search in simulated array $\widetilde{Y}$ and returns matching elements $(\widetilde{Y}_1, \ldots, \widetilde{Y}_{num})$.
  - With given search requests $(\widetilde{x}_1, \ldots, \widetilde{x}_h)$, the simulator performs a search in simulated array $\widetilde{X}$ and returns matching elements $(\widetilde{X}_1, \ldots, \widetilde{X}_h)$.

We now need to show that the outputs of the two experiments $Ideal_{\mathbb{A},\mathbb{S}}(k)$ and $Real_{\mathbb{A}}(k)$ are indistinguishable. Since the simulator generates randomly the set of ciphertexts $\widetilde{C}$, the output of the simulator is truly indistinguishable from the real ciphertexts that are generated with the PCPA-secure symmetric encryption SKE scheme using secret key K_1. Otherwise, this would mean that the simulator could distinguish between the output of the PCPA-secure symmetric encryption scheme SKE and the random value. Next, the simulated encrypted position heap tree $\widetilde{\Lambda}$ is truly indistinguishable from the real encrypted position heap tree. Otherwise, this would mean that simulator could distinguish between the output of pseudorandom function F with secret key K_Q and the random values. Similarly, the simulated arrays $\widetilde{Y}$ and $\widetilde{X}$ are truly indistinguishable from the real arrays Y and X. Otherwise, this would mean that the simulator can distinguish between the output of pseudorandom permutation P with keys K_2, K_3, SKE scheme with keys K_Y, K_X and the random values. Thus, it is concluded that the outputs of the two experiments are indistinguishable. □

6.2. Performance

In this section, we outline the performance of the proposed solution. We assume that the encryption and decryption using the SKE scheme take $O(k)$ time, where k is the security parameter. We also assume that the element selection from the array takes $O(1)$ time.

We first focus on the encryption efficiency of the SSP-SSE scheme. Given plaintext position heap tree Λ with n nodes, we compute encrypted position heap tree Λ using SKE in $O(kn)$ time. The arrays

X and Y each have n elements and can be computed in $O(kn)$ time. Therefore, encryption takes $O(kn)$ time, and the total ciphertext is $O(kn)$ in size.

We now analyze the efficiency of proposed search algorithm. The cloud user inputs a substring χ of length m and outputs a search query in $O(m)$ time. The cloud provider uses $\bar{\Lambda}$, performs m matches in the tree and retrieves occ descendant nodes, in $O(m + occ)$ time. The cloud user then computes $y_1, \ldots, y_{m+occ}$ elements, and the cloud provider retrieves $Y[y_1], \ldots, Y[y_{m+occ}]$ in $O(m + occ)$ time. The cloud user then computes $x_1, \ldots, x_{m^2}$ elements (the cloud user wants to inspect m ancestor positions and the substring χ of m length that may appear at each ancestor position), and the cloud provider retrieves $X[x_1], \ldots, X[x_{l^2}]$ in $O(m^2)$ time. Now, the cloud user performs an inspection of m ancestors m times, making execution in $O(m^2)$ time. Thus, both the cloud user and the cloud provider take computation time $O(m^2 + occ)$ in the query protocol and three rounds of communication to complete the execution of the protocol.

We have developed and implemented a proof-of-concept prototype of the SSP-SSE scheme using C++ language. Our prototype leverages the libtomcrypt cryptographic library[33], which is a portable C cryptographic library that supports symmetric ciphers, one-way hashes, pseudo-random number generators and a plethora of support routines. We use libtomcrypt to build the searchable index I and encrypt the document collection. We utilize AES-CTR encryption for the SKE symmetric-key encryption scheme, HMAC-SHA1 for pseudorandom function F and DES encryption for pseudorandom permutation P.

We show a thorough experimental evaluation of the SSP-SSE scheme on a real-world dataset: the Genome database [34] (published by the National Center for Biotechnology Information, National Institutes of Health) that contains sequence data from the whole genomes of over 1000 species or strains. The database includes all three main domains of life (Bacteria, Archaea and Eukaryota), as well as many viruses, phages, viroids, plasmids and organelles. All experiments have been performed on a six-core Intel Xeon E5645 2.40-GHz processor and 98 GB memory running 64-bit Fedora 23. The cloud server, data owner and cloud user applications were run on the same machine, as the network communication overhead was assumed to be negligible.

For our experiments, we pick large mRNA transcript datasets of various insects. Table 2 shows the details of the experimental set. Figure 5a shows the overhead of constructing the encrypted position heap tree Λ. We compare the time of the construction of the plaintext position heap tree (original algorithm) and the encrypted position heap tree proposed in this work. Figure 5b shows the storage overhead of searchable index I that consists of encrypted position heap tree Λ, position array Y and text array X. In short, we notice that the proposed scheme adds insignificant overhead to the computation time; however, its storage overhead depends on the block cipher size of the underlying encryption schemes. We believe that the proposed solution can be easily deployed in a real-world cloud environment.

Table 2. Experimental database.

Organism Name	Description	mRNA Size (MB)	Organism Name	Description	mRNA Size (MB)
Dufourea novaeangliae		28	*Papilio Polytes*		41
Bactrocera dorsalis		49	*Fopius arisanus*		60
Halyomorpha halys		63	*Tribolium castaneum*		63
Stomoxys calcitrans		70	*Orussus abietinus*		72
Nasonia vitripennis		75	*Linepithema humile*		77

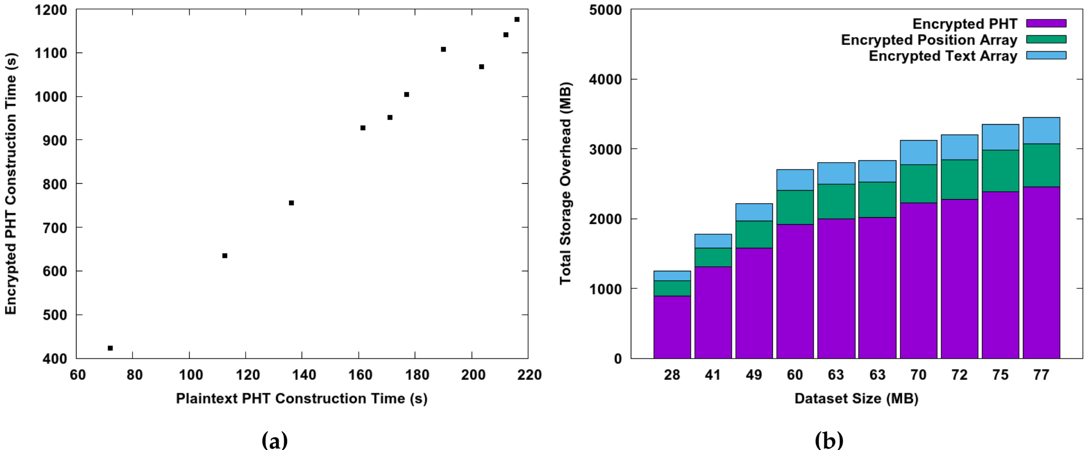


Figure 5. Experimental results. (**a**) The construction of the position heap tree; (**b**) the searchable index storage.

7. Multi-User Substring Position Searchable Symmetric Encryption

Our original system model shown in Figure 1 includes only three single entities. To make an important step towards widespread adoption of the searchable encryption techniques, there is a need to efficiently support hundreds, even thousands of users in the cloud. In this section, we consider a simple extension to our work, where a data owner has a document collection, and there is a group of data users that wants to query encrypted data in the cloud.

Curtmola et al.'s [6] solution extends the single-user searchable encryption framework with broadcast encryption [35], where the data owner sends the searchable index and encrypted document collection to the cloud, and a group of cloud users is allowed to invoke the search over encrypted cloud data. The framework describes the solution where the data owner distributes a single shared secret key among the group of cloud users. However, this solution may not work in the real-world cloud environment that involves a potentially large number of data users, since a single secret key is given to all participants. For instance, if the data owner decides to revoke the search access for one cloud user, he/she will have to generate a new key and distribute it to the remaining users. It is preferable that each cloud participant could keep its own secret key, thus making key management easier and more efficient.

We propose a new multi-user substring position searchable symmetric encryption (MSSP-SSE) scheme that solves the problem of managing access privileges and searching a substring over encrypted cloud data. Our solution is based on the distributed broadcast encryption scheme [36]. First, we present the definitions of a multi-user substring position searchable symmetric encryption scheme. Later, we give an efficient construction that combines the ideas of a single-user SSP-SSE scheme with the distributed broadcast encryption scheme.

7.1. Preliminaries

In this section, we present several definitions used in our work. We begin with the definition of the Witness Pseudo-Random Function (WPRF). Informally, a witness PRF for an NP language L is a PRF F, such that anyone with a valid witness that $x \in L$ can compute $F(x)$ without the secret key, but for all $x \notin L$, $F(x)$ is computationally hidden without knowledge of the secret key. Formally, a witness PRF is defined as follows.

Definition 20. (Witness Pseudo-Random Function (WPRF) [36]). A triple of algorithms $(Gen, F, Eval)$ is a witness PRF if:

- *Gen*: a probabilistic algorithm that inputs a security parameter λ and a circuit $R : \mathbb{X} \times \mathbb{W} \rightarrow \{0, 1\}$, and outputs a secret function key fk and a public evaluation key ek.
- *F*: a deterministic algorithm that inputs the function key fk and an input $x \in \mathbb{X}$ and outputs some output $y \in \mathbb{Y}$ for some set $\mathbb{Y}$.
- *Eval*: a deterministic algorithm that inputs the evaluation key ek, an input $x \in \mathbb{X}$ and a witness $w \in \mathbb{W}$ and that produces an output $y \in \mathbb{Y}$ or $\perp$.

A witness PRF is correct if the following holds:

$$Eval(ek, x, w) = \begin{cases} F(fk, x) & \text{if } R(x, w) = 1 \\ \perp & \text{if } R(x, w) = 0 \end{cases} \text{ for all } x \in \mathbb{X}, w \in \mathbb{W}. \quad (6)$$

A multiparty key exchange protocol allows a group of g users to simultaneously post a message to a public bulletin board, retaining some user-independent secrets. After reading off the contents of the bulletin board, all users establish the same shared secret key. The multiparty key exchange protocol consists of the following algorithms.

Definition 21. (Non-Interactive Multiparty Key Exchange protocol (NIKE-WPRF) [36]). Let $G : \mathbb{S} \rightarrow \mathbb{Z}$ be a pseudo-random generator with $|S|/|Z| \leq negl$. Let $WPRF = (Gen, F, Eval)$ be a witness PRF. Let $R_g : \mathbb{Z}^g \times (\mathbb{S} \times [g]) \rightarrow \{0, 1\}$ be a relation that outputs one on input $((z_1, \ldots, z_g), (s, i))$ if and only if $z_i = G(s)$. The non-interactive key exchange protocol consists of:

- $Publish(\lambda, g)$: a probabilistic algorithm to output public and secret keys. The algorithm inputs the security parameter λ and the group order g. It computes $(fk, ek) \stackrel{R}{\leftarrow} Gen(\lambda, R_g)$. Next, it picks a random seed $sk \stackrel{R}{\leftarrow} \mathbb{S}$ and computes $z \leftarrow G(sk)$. It outputs a secret key sk and public values (z, ek), where sk is kept secret and (z, ek) are published to the bulletin board.
- $KeyGen(\{z_i, ek_i\}_{i \in [g]}, sk)$: a deterministic algorithm that inputs group g and user's secret sk. It outputs a group key $k = Eval(ek_i, (z_1, \ldots, z_g), (sk, i))$.

Broadcast encryption [35] allows an encryptor (data owner) to broadcast a message to a subset of recipients (data users). The system is said to be collusion resistant if non-data users can learn information about the plaintext. The most recent work by Zhandry et al. [36] proposes a distributed broadcast encryption scheme that removes the burden of key management from the encryptor and lets group establishment run by participating recipients. Each user is allowed to pick the desired participants and to establish a shared key. The distributed broadcast encryption scheme is defined as follows.

Definition 22. (Distributed Broadcast Encryption over NIKE (BE-NIKE-WPRF) [36]). The distributed broadcast encryption scheme over multi-party non-interactive key exchange protocol consists of the four following algorithms:

1. *Setup*: a probabilistic algorithm to setup the BE-NIKE-WPRF scheme. The algorithm outputs a secret parameter λ and group order g.
2. $Join(\lambda, g)$: a probabilistic algorithm to join the scheme that is executed by each participant. The algorithm inputs a secret parameter λ and group order g. The algorithm invokes $NIKE\text{-}WPRF.Publish(\lambda, g)$ to output secret sk and public values (z, ek). The user makes (z, ek) publicly available to other participants.
3. $Enc(\{z_i, ek_i\}_{i \in [g]}, sk, m)$: a probabilistic algorithm to encrypt message m under the shared key. The algorithm inputs the set of public values $\{z_i, ek_i\}_{i \in [g]}$, secret key sk and plaintext message m. The algorithm runs $NIKE\text{-}WPRF.KeyGen(\{z_i, ek_i\}_{i \in [g]}, sk)$ to derive the shared key k. The algorithm outputs a ciphertext c, which is the encryption of message m using the shared key k.

4. $Dec(\{z_i, ek_i\}_{i\in[g]}, sk, c_m)$: a deterministic algorithm to decrypt c_m. The algorithm invokes *NIKE-WPRF.KeyGen*$(\{z_i, ek_i\}_{i\in[g]}, sk)$ to derive k. If $k \neq \perp$, then the algorithm decrypts c_m using k and outputs the original message m.

7.2. Algorithm Definitions

Definition 23. ***(Multi-User Substring Position Searchable Symmetric Encryption (MSSP-SSE)).*** A tree-based MSSP-SSE scheme over a set of documents D is a tuple of nine polynomial-time algorithms (KeyGen, BuildTree, Encrypt, Join, GroupSetup, Remove, ConstructQuery, Search, Decrypt), as follows:

1. $(K, \lambda, g) \leftarrow KeyGen(1^k)$: a probabilistic key generation algorithm to setup the SSP-SSE scheme. The algorithm takes a secret parameter k and outputs a set of secret keys K, secret parameter λ and group g.
2. $(\Lambda) \leftarrow BuildTree(D)$: a deterministic algorithm to build a position heap tree Λ. The algorithm takes a document collection $D = \{D_1, \ldots, D_l\}$ and constructs a position heap tree Λ.
3. $(I, C) \leftarrow Encrypt(K, \Lambda, D)$: a probabilistic algorithm to encrypt a position heap tree and document corpus. The algorithm inputs a set of secret keys K, a position heap tree Λ and a documents corpus D. The output of algorithm is a searchable index I and encrypted collection $C = \{C_1, \ldots, C_l\}$.
4. $(sk, (z, ek)) \leftarrow Join(\lambda, g)$: a probabilistic algorithm run by each data user to participate in the scheme. The algorithm invokes *BE-NIKE-WPRF.Join* with an input of secret parameter λ and group order g. It outputs a pair $(sk, (z, ek))$.
5. $c_r \leftarrow GroupSetup(\{z_i, ek_i\}_{i\in[h]}, sk)$: a probabilistic algorithm run by the group owner to establish the group $h \subseteq g$ of authorized data users. The algorithm runs *BE-NIKE-WPRF.Enc* with an input of public values $\{z_i, ek_i\}_{i\in[h]}$, group owner's secret key sk and a sampled secret r. The output is encrypted ciphertext c_r.
6. $c_r \leftarrow Remove(\{z_i, ek_i\}_{i\in[h\setminus o]}, sk)$: a probabilistic algorithm run by the group owner to remove a user o from the set of authorized users. The algorithm invokes *BE-NIKE-WPRF.Enc* that inputs the set of public values $\{z_i, ek_i\}_{i\in[h\setminus o]}$, group owner's secret key sk and a new secret r. The output is encrypted ciphertext c_r.
7. $[(Q) \leftarrow ConstructQuery(K, \chi, c_r)] \leftrightarrow [(L) \leftarrow Search(I, Q, c_r)]$: two deterministic algorithms that are executed interactively between the authorized cloud user and the cloud provider. The algorithm inputs a set of secret keys K, ciphertext c_r and a substring χ, and it outputs a search query Q. The algorithm uses a query Q, searchable index I and ciphertext c_r. It outputs a sequence of identifiers $L \in C$.
8. $(D_i, pos_{D_i}) \leftarrow Decrypt(K, C_i)$: a deterministic algorithm that takes a set of secret keys K and a ciphertext C_i as input, and it outputs an original document $D_i, \forall i \in [1; n]$, and a set of χ's positions pos_{D_i} in D_i.

We now present the security model for a Multi-user Substring Position Searchable Symmetric Encryption (MSSP-SSE) scheme. Intuitively, our security model requires the security of a single-user SSP-SSE scheme and the security of a distributed broadcast encryption scheme. We formalize the security requirements of MSSP-SSE scheme as follows:

- Given searchable index I and the set of encrypted documents $C = \{C_1, \ldots, C_l\}$, the adversary should learn nothing about the original document collection $D = \{D_1, \ldots, D_l\}$.
- Given the set of incoming search queries $Q = \{Q_1, \ldots, Q_m\}$, access pattern, search pattern and path pattern, the adversary should learn nothing about the content of each search query Q_i or the content of resulted documents.
- Once a user is removed from the set of authorized cloud users, he/she is no longer allowed to invoke a search over encrypted documents in the cloud. Thus, we require the revocation of the cloud users.

In MSSP-SSE, we use the adaptive semantic security notion of a single-user SSP-SSE scheme. It provides the security against an adaptive adversary: the cloud server does not learn anything about the document collection and search queries beyond the access, search and path patterns. However, with the addition of the access privilege property, we expand our security definitions towards the Remove functionality (Algorithm 4). We define the Rev algorithm as follows:

Definition 24. (Revocation). Let MSSP-SSE = (KeyGen, BuildTree, Encrypt, Join, GroupSetup, Remove, ConstructQuery, Search, Decrypt) be a group SSP-SSE scheme, k be a security parameter and $\mathbb{A} = (\mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3)$ be an adversary. We use the following probabilistic experiment $Rev_{MSSP\text{-}SSE,\mathbb{A}}$(k):

Algorithm 4: $Rev_{MSSP\text{-}SSE,\mathbb{A}}(k)$.

$(st_A, D) \leftarrow \mathbb{A}_1(1^k)$
$(sk_A, (z_A, ek_A)) \leftarrow Join(\lambda, g)$
$c_r \leftarrow SetupGroup((z_A, ek_A), sk)$
$(\Lambda) \leftarrow BuildTree(D)$
$(I, C) \leftarrow Encrypt(K, \Lambda, D)$
$st_A \leftarrow \mathbb{A}_2^{O(I,C,st_{\mathbb{S}},\cdot)}(st_A, sk_A, (z_A, ek_A), c_r)$
$c_r^{'} \leftarrow Remove((z_A, ek_A), sk)$
$Q \leftarrow \mathbb{A}_3(st_A)$
$L \leftarrow Search(st_S, I, Q, c_r^{'})$
if $L \neq \perp$, output one, otherwise output zero

where $O(I, C, st_{\mathbb{S}}, \cdot)$ is an oracle that inputs a search query Q and outputs ciphertexts C indexed by $L \leftarrow Search(I, \Omega, c_r^{'})$ if $L \neq \perp$ and $\perp$ otherwise. We say that the Remove algorithm achieves user revocation if for all polynomial-size adversaries $\mathbb{A} = (\mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3)$, the following is correct:

$$Pr[Rev_{MSSP\text{-}SSE,\mathbb{A}}(k) = 1] \leq negl(k), \quad (7)$$

where the probability is over the coins of KeyGen, Join, GroupSetup, Remove and Encrypt.

7.3. MSSP-SSE Construction

Algorithm 5 shows the details of our multi-user scheme MSSP-SSE = (KeyGen, BuildTree, Encrypt, Join, GroupSetup, Remove, ConstructQuery, Search, Decrypt). Let SSP-SSE = (KeyGen, BuildTree, Encrypt, ConstructQuery, Search, Decrypt) be a single-user substring position searchable symmetric encryption scheme. Let BE-NIKE-WPRF = (Setup, Join, Enc, Dec) be a distributed broadcast encryption scheme. We require standard security notions for broadcast encryption, i.e., in addition to providing PCPA-security, it provides revocation-scheme security against a group of revoked users. Let ρ be a pseudorandom permutation, such that ρ: $\{0,1\}^k \times \{0,1\}^t \rightarrow \{0,1\}^t$ (ρ can be constructed as a pseudorandom permutation over domains of arbitrary size[37]), where t is the size of search query Q in the SSP-SSE scheme. We assume that the cloud server does not collude with revoked users; otherwise, our construction cannot prevent a revoked user from invoking the search.

Algorithm 5: MSSP-SSE construction.

$\underline{KeyGen(1^k):}$
1. generate $K \leftarrow SSP\text{-}SSE.KeyGen(1^k)$.
2. generate $\lambda, g \leftarrow BE\text{-}NIKE\text{-}WPRF.Setup(1^k)$.

Output the key set K, secret parameter λ and group g.

$\underline{BuildTree(D):}$
Given a document collection $D = \{D_1, \ldots, D_l\}$, output $\Lambda \leftarrow SSP\text{-}SSE.BuildTree(D)$.

$\underline{Encrypt(K, \Lambda, D):}$
1. set $(I, C) \leftarrow SSP\text{-}SSE.Encrypt(K, \Lambda, D)$.

Output (I, C).

$\underline{Join(\lambda, g):}$
1. generate $(sk, (z, ek)) \leftarrow BE\text{-}NIKE\text{-}WPRF.Join(\lambda, g)$.

Keep sk private; output (z, ek) to the cloud server.

$\underline{GroupSetup(\{z_i, ek_i\}_{i\in[h]}, sk):}$
1. pick $h \subseteq g$ and get public values $\{z_i, ek_i\}_{i\in[h]}$ from the cloud server.
2. sample $r \leftarrow \{0,1\}^s$ and compute $c_r \leftarrow BE\text{-}NIKE\text{-}WPRF.Enc(\{z_i, ek_i\}_{i\in[h]}, sk, r)$.

Output c_r to the cloud server.

$\underline{Remove(\{z_i, ek_i\}_{i\in[h\backslash o]}, sk)}$
1. set $(h\backslash o) \subseteq g$, and retrieve public values $\{z_i, ek_i\}_{i\in[h\backslash o]}$ from the cloud server.
2. sample new $r \leftarrow \{0,1\}^s$, and compute $c_r \leftarrow BE\text{-}NIKE\text{-}WPRF.Enc(\{z_i, ek_i\}_{i\in[h\backslash o]}, sk, r)$.

Output new c_r to the cloud server.

$\underline{[(Q) \leftarrow ConstructQuery(K, \chi, c_r)] \leftrightarrow [(L) \leftarrow Search(I, Q, c_r)]}$
1. cloud user:
   (a) get c_r from the cloud server.
   (b) compute $r \leftarrow BE\text{-}NIKE\text{-}WPRF.Dec(\{z_i, ek_i\}_{i\in[h]}, sk, c_r)$. If $r = \perp$, output $\perp$.
   (c) calculate $Q' \leftarrow SSP\text{-}SSE.ConstructQuery(K, \chi))$ and $Q \leftarrow \rho_r(Q')$.
2. cloud provider:
   (a) compute $r \leftarrow BE\text{-}NIKE\text{-}WPRF.Dec(\{z_i, ek_i\}_{i\in[h]}, sk, c_r)$.
   (b) calculate $Q' \leftarrow \rho_r^{-1}(Q)$.
   (c) get $L \leftarrow SSP\text{-}SSE.Search(I, Q')$, where $L \in C$.
   (d) output L.

$\underline{Decrypt(K, C_i):}$
Output $(D_i, pos_{D_i}) \leftarrow SSP\text{-}SSE.Decrypt(K, C_i)$.

We now describe the scheme using the following hospital example. Consider a doctor (data owner) that performed a set of early cancer screening tests on a patient and wishes to share the resulting documents with a group of hospital nurses (data users). To remove the burden of key management, the doctor enables a distributed setup, where each nurse generates his or her own secret key and establishes a group of authorized participants that includes a head nurse and his or her subordinate nurses. First, the doctor samples the secret parameter k and generates the set of encryption keys K, secret key λ and group g for the distributed broadcast encryption. Second, the doctor encrypts the resulted documents with PCPA-secure symmetric encryption scheme SKE and outputs the searchable index I to the cloud server. Next, each participating nurse invokes the Join algorithm with secret λ, group g (both distributed by the doctor) to generate $(sk, (z, ek))$, where secret sk is kept private and (z, ek) are published to the cloud server.

Now, the head nurse (group owner) creates a group of authorized users that are allowed to invoke a search over encrypted documents in the cloud. The head nurse launches the GroupSetup algorithm,

where she selects public values $\{z_i, ek_i\}_{i\in h}$ of authorized participants $h \in g$, samples random secret parameter r and invokes the distributed broadcast encryption to output c_r.

In order to search for a substring χ, the authorized nurse first contacts the cloud provider to receive the latest ciphertext c_r and invokes distributed broadcast encryption with his or her own secret sk, public values $\{z_i, ek_i\}_{i\in h}$ to recover secret r. If r is successfully recovered, the nurse then constructs a single-user search query Q', encrypts it with pseudorandom permutation ρ with r and outsources $\rho_r(Q')$ to the cloud provider. The cloud provider recovers the search query Q' by computing $\rho_r^{-1}(\rho_r(Q'))$. Here, the key r is only known by the data owner and the set of authorized users that includes the cloud provider. Next, the ConstructQuery and the Search interactive algorithms are are executed between the authorized nurse and the cloud server.

If a nurse o is no longer the authorized user in the system, the head nurse samples a new key r' and generates new ciphertext c_r. The new c_r' is sent to the cloud provider to replace the old c_r. Since revoked nurse o is not able to recover the new secret r', permuted search query Q will not yield a valid search query. This simple extra layer given by the pseudo-random permutation ρ prevents cloud users from performing a successful search once they are removed from the system.

MSSP-SSE utilizes the security and performance of a single-user SSP-SSE scheme. Our construction is very efficient, since the cloud provider needs only to execute a pseudorandom permutation to evaluate the access privileges, thus eliminating the need of more expensive authentication protocols.

8. Conclusions

In this work, we present a new Substring Position Searchable Symmetric Encryption scheme (SSP-SSE) that allows efficient substring search on encrypted documents outsourced to the cloud. Specifically, our solution efficiently finds the occurrences and positions of a substring over encrypted cloud data. We formally define the leakage functions and security notions of SSP-SSE. We show that our scheme is secure against chosen-query attacks executed by an adaptive adversary. We also present a multi-user SSP-SSE scheme that supports a distributed setup, where data users choose their own secret key rather than receive the key from a trusted authority. As future work, we plan to focus on enhancing query privacy in SSP-SSE, while keeping all of the good properties in the current design. Furthermore, we plan to expand the SSP-SSE scheme to support dynamic updates on the document collection that will allow query execution when the document corpus is modified.

Acknowledgments: This work was partially supported by the U.S. National Science Foundation under Grant No. 0905232. We are grateful to anonymous referees for their constructive comments and valuable suggestions that helped to improve this paper.

Author Contributions: M.S. contributed to the design of the proposed scheme, literature survey and manuscript preparation; Z.O. implemented the scheme and performed the experiments; I.R. supervised this research work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Strizhov, M.; Ray, I. Substring Position Search over Encrypted Cloud Data Using Tree-Based Index. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E), Tempe, AZ, USA, 9–13 March 2015.
2. Song, D.X.; Wagner, D.; Perrig, A. Practical Techniques for Searches on Encrypted Data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000.
3. Goh, E.J. Secure Indexes. Cryptology ePrint Archive, Report 2003/216, 2003. Available online: http://eprint.iacr.org/2003/216/ (accessed on 10 January 2016).
4. Moataz, T.; Shikfa, A. Boolean Symmetric Searchable Encryption. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China , 8–10 May 2013.
5. Orencik, C.; Kantarcioglu, M.; Savas, E. A Practical and Secure Multi-keyword Search Method over Encrypted Cloud Data. In Proceedings of the 6th IEE International Conference on Cloud Computing, Santa Clara, CA, USA, 28 June–3 July 2013.

6. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006.
7. Boneh, D.; Waters, B. Conjunctive, Subset, and Range Queries on Encrypted Data. In Proceedings of the 4th IACR Theory of Cryptography Conference, Amsterdam, The Netherlands, 21–24 February 2007.
8. Boneh, D.; Crescenzo, G.D.; Ostrovsky, R.; Persiano, G. Public Key Encryption with Keyword Search. In Proceedings of the EUROCRYPT 2004, Jeju Island, Korea, 5–9 December 2004.
9. Lai, J.; Zhou, X.; Deng, R.H.; Li, Y.; Chen, K. Expressive Search on Encrypted Data. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China, 8–10 May 2013.
10. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data. In Proceedings of the 30th IEEE International Conference on Computer Communications, Shanghai, China, 31 July–2 August 2011.
11. Cash, D.; Jarecki, S.; Jutla, C.; Krawczyk, H.; Rosu, M.C.; Steiner, M. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In Proceedings of the 33rd Annual International Cryptology Conference CRYPTO 2013, Santa Barbara, CA, USA, 18–22 August 2013.
12. Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic Searchable Symmetric Encryption. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012.
13. Chang, Y.C.; Mitzenmacher, M. Privacy Preserving Keyword Searches on Remote Encrypted Data. In Proceedings of the 3rd International Conference on Applied Cryptography and Network Security, New York, NY, USA, 7–10 June 2005.
14. Shi, E.; Bethencourt, J.; Chan, T.H.H.; Song, D.; Perrig, A. Multi-Dimensional Range Query over Encrypted Data. In Proceedings of the 2007 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 20–23 May 2007.
15. Agrawal, R.; Kiernan, J.; Srikant, R.; Xu, Y. Order-Preserving Encryption for Numeric Data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, 13–18 June 2004.
16. Blanton, M. Achieving Full Security in Privacy-Preserving Data Mining. In Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust, Boston, MA, USA, 9–11 October 2011.
17. Li, J.; Wang, Q.; Wang, C.; Cao, N.; Ren, K.; Lou, W. Fuzzy Keyword Search over Encrypted Data in Cloud Computing. In Proceedings of the 29th Conference on Information Communications, London, UK, 7–9 September 2010.
18. Wang, C.; Ren, K.; Yu, S.; Urs, K. Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data. In Proceedings of the 31th Conference on Information Communications, Hertfordshire, UK, 29–31 October 2012.
19. Boldyreva, A.; Chenette, N. Efficient Fuzzy Search on Encrypted Data. In Proceedings of the 21st International Workshop on Fast Software Encryption, London, UK, 3–5 March 2014.
20. Strizhov, M.; Ray, I. Multi-keyword Similarity Search over Encrypted Cloud Data. In Proceedings of the ICT Systems Security and Privacy Protection, Marrakech, Morocco, 2–4 June 2014.
21. Ehrenfeucht, A.; McConnell, R.M.; Osheim, N.; Woo, S.W. Position Heaps: A Simple and Dynamic Text Indexing Data Structure. *J. Discret. Algorithms* **2011**, *9*, 100–121.
22. Bloom, B.H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* **1970**, *13*, 422–426.
23. Wang, C.; Cao, N.; Li, J.; Ren, K.; Lou, W. Secure Ranked Keyword Search over Encrypted Cloud Data. In Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, Genoa, Italy, 21–25 June 2010.
24. Moataz, T.; Justus, B.; Ray, I.; Cuppens-Boulahia, N.; Cuppens, F.; Ray, I. Privacy-Preserving Multiple Keyword Search on Outsourced Data in the Clouds. In Proceedings of the Data and Applications Security and Privacy XXVIII, Vienna, Austria, 14–16 July 2014.
25. Crescenzo, G.D.; Saraswat, V. Public Key Encryption with Searchable Keywords Based on Jacobi Symbols. In Proceedings of the 8th International Conference on Cryptology in India, Chennai, India, 9–13 December 2007.
26. Golle, P.; Staddon, J.; Waters, B. Secure Conjunctive Keyword Search over Encrypted Data. In Proceedings of the Applied Cryptography and Network Security 2004, Yellow Mountain, China, 8–11 June 2004.

27. Hwang, Y.H.; Lee, P.J. Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. In Proceedings of the First International Conference on Pairing-Based Cryptography, Tokyo, Japan, 2–4 July 2007.
28. Weiner, P. Linear Pattern Matching Algorithms. In Proceedings of the 14th Annual Symposium on Switching and Automata Theory (Swat 1973), Washington, DC, USA, 15–17 October 1973; pp. 1–11.
29. Manber, U.; Myers, G. Suffix Arrays: A New Method for On-Line String Searches. *SIAM J. Comput.* **1993**, *22*, 935–948.
30. Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*; Cambridge University Press: New York, NY, USA 1997.
31. Ukkonen, E. On-line Construction of Suffix Trees. *Algorithmica* **1995**, *14*, 249–260.
32. Gentry, C.; Goldman, K.; Halevi, S.; Julta, C.; Raykova, M.; Wichs, D. Optimizing ORAM and Using It Efficiently for Secure Computation. In Proceedings of the 13th Privacy Enhancing Technologies Symposium, Bloomington, IN, USA, 10–12 July 2013.
33. LibTomCrypt. Cryptographic Toolkit, 2016. Available online: https://github.com/libtom/libtomcrypt (accessed on 10 May 2016).
34. NCBI. Genome Database, 2016. Available online: http://www.ncbi.nlm.nih.gov/genome (accessed on 10 May 2016).
35. Fiat, A.; Naor, M. Broadcast Encryption. In Proceedings of the 13th Annual International Cryptology Conference CRYPTO '93, Santa Barbara, CA, USA, 22–26 August 1993.
36. Zhandry, M. How to Avoid Obfuscation Using Witness PRFs. In Proceedings of the 13th International Conference on Theory of Cryptography TCC 2016, Tel Aviv, Israel, 10–13 January 2016.
37. Morris, B.; Rogaway, P.; Stegers, T. How to Encipher Messages on a Small Domain. In Proceedings of the CRYPTO 2009, Santa Barbara, CA, USA, 16–20 August 2009.

© 2016 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).