# Access Pattern Hiding in Searchable Encryption

Fateh Boucenna
CERIST, research center, Algeria
Email: fboucenna@cerist.dz

Omar Nouali
CERIST, research center, Algeria
Email: onouali@cerist.dz

Kamel Adi
UQO, Qc, Canada
Email: kamel.adi@uqo.ca

Samir Kechid
USTHB university, Algeria
Email: skechid@usthb.dz

***Abstract*—Cloud computing is a technology that provides users with a large storage space and an enormous computing power. For privacy purpose, the sensitive data should be encrypted before being outsourced to the cloud. To search over the outsourced data, *searchable encryption* (SE) schemes have been proposed in the literature. An SE scheme should perform searches over encrypted data without causing any sensitive information leakage. To this end, a few security constraints were elaborated to guarantee the security of the SE schemes, namely, the keyword privacy, the trapdoor unlinkability, and the *access pattern*. The latter is very hard to be respected and most approaches fail to guarantee the access pattern constraint when performing a search. This constraint consists in hiding from the server the search result returned to the user. The non respect of this constraint may cause sensitive information leakage as demonstrated in the literature. To fix this security lack, we propose a method that allows to securely request and receive the needed documents from the server after performing a search. The proposed method that we call the *access pattern hiding* (APH) technique allows to respect the access pattern constraint. An experimental study is conducted to validate the APH technique.**

## I. Introduction

Nowadays, companies and individuals demand an increasing amount of storage space for their data and computing power for their applications. For this purpose, several new technologies have been designed and implemented such as the cloud computing. The latter provides its users with a storage space and computing power according to their needs in a flexible and personalized way. However, the outsourced data such as emails, electronic health records, and company reports are sensitive and confidential, hence, they must be encrypted before being sent to the cloud.

To perform a search over these data, it is no longer possible to exploit traditional search engines, given that the outsourced data are encrypted. Consequently, lots of searchable encryption (SE) schemes have been proposed in the literature [1], [2], [3], [4], [5]. The common point between these approaches is that the *data owner* starts by encrypting the data collection and the generated index before outsourcing them. The *cloud server* exploits the encrypted index after receiving a trapdoor (encrypted query) in order to retrieve and return a top-k document identifiers to the *data user*. Finally, the latter asks the server to return him the needed documents among the top-k ones. In addition, the search process should be performed without decrypting any data and without causing any sensitive information leakage.

Furthermore, when designing an SE scheme, it is important to take into consideration some security constraints [1], [2]. The first one is called the keyword privacy and consists in preventing the server from making any link between terms and documents. The second one is called trapdoor unlinkability and consists in preventing the server from making links between a set of trapdoors. Finally, the *access pattern* constraint consists in hiding the search results from the server. Both the keyword privacy and the trapdoor unlinkability constraints are respected by most recent approaches. Nevertheless, it is very difficult to design an SE scheme that hides the access pattern from the server.

Some approaches tried to guarantee the access pattern constraint by encrypting the *search result* [6], [7]. However, after receiving and decrypting the result, the authorized user might request some relevant documents from the server. Consequently, a part of the search result (the requested documents set) is revealed to the cloud and thus, the access pattern constraint is not fully respected even if the search result is encrypted. Unfortunately, the non-respect of the access pattern constraint may cause sensitive information leakage as demonstrated by Islam et al. [8] in their proposed attack.

To solve this problem, a few techniques have been proposed in the literature. One of these techniques is called the *blind storage* [9] which consists in splitting each document into several blocks in order to avoid the server from knowing the number of documents or distinguishing between them. However, the drawback of this technique is that a document is recognized by the server as soon as it is accessed by a user. Consequently, this technique does not guarantee the access pattern constraint. Another approach proposed by Kellaris et al. [10] allows preserving privacy while accessing data by combining two techniques which are the differential privacy [11] and the oblivious RAM [12]. Nevertheless, the oblivious RAM dramatically decreases the search performance which makes this approach impracticable.

The aim of this work is to secure the access to the outsourced data after performing a search in order to guarantee a full respect of the access pattern constraint. Our method is applicable in any approach that respects the access pattern constraint *during* the search and needs a secure access to the data collection. Our idea consists in breaking any link on the cloud side between the index and the data collection as well as between the returned result and the accessed documents[1]. This new method prevents the server from deducing the search

[1]The set of documents which are requested by the user after performing a search.

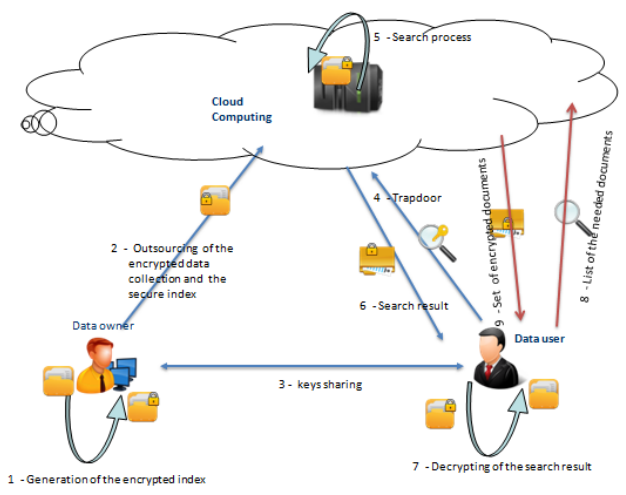

5 - Search process
Cloud Computing
4 - Trapdoor
9 - Set of encrypted documents
8 - List of the needed documents
2 - Outsourcing of the encrypted data collection and the secure index
6 - Search result
Data owner
Data user
3 - keys sharing
1 - Generation of the encrypted index
7 - Decrypting of the search result

Fig. 1: General architecture.

result from the accessed documents which provides a full respect of the access pattern constraint.

## II. Problem formulation

### A. System Model

The search process in a cloud environment is different from the traditional search. This difference comes from the fact that the outsourced cloud data are encrypted and the search process should not cause any sensitive information leakage. The architecture of a searchable encryption scheme is composed of three main entities, namely, the data owner, the cloud server, and the authorized users (Figure [1]). The data owner is responsible for the creation of an index from the data collection and encrypting both of them before outsourcing them to the cloud. The index should be encrypted using an appropriate cryptosystem that keeps the ability to make calculations on the index such as homomorphic encryption [13] and the S$k$NN algorithm [14], whereas, the data collection should be encrypted using a robust cryptosystem. Then, the data owner shares the encryption/decryption keys with each authorized user using a secure communication protocol. To perform a search, an authorized user formulates a query, encrypts it using the private key, and sends the trapdoor to the cloud server. Upon receipt of the trapdoor, the server initiates the search process over the secure index and returns to the user an encrypted result. Finally, the user decrypts the result and requests a set of relevant documents from the server.

### B. Threat Model

Security is a crucial aspect in cloud computing given that the outsourced data are often personal or professional. The cloud server is exposed to all kinds of external attacks and hence, it is necessary to encrypt each data before sending it to the cloud. In addition, the cloud server itself is semi-honest[2] and may collect information about the content of documents by doing statistical analyses. Therefore, the search process should be secure and must protect the data privacy. When designing a SE scheme, it is important to take into account the threats discussed below. For this reason, security constraints were elaborated in the literature [2], [1], [15].

- *Keyword privacy.* The proposed scheme must be able to hide from the server the term distribution and the inter-distribution. The term distribution corresponds to the frequency of a given term in each document of the data collection, whereas, the inter-distribution indicates the distribution of scores of terms in a given document. Hiding these two features allows to prevent the server from making any link between terms and documents.
- *Trapdoor unlinkability.* The proposed scheme must be able to prevent the server from deducing the relationship between a given set of encrypted queries. For that, the cryptosystem should be non-deterministic.
- *Access pattern.* The proposed scheme must be able to hide from the server the sequence of results returned to a user during the search.

### C. Access Pattern Leakage

Most approaches that have been proposed in the literature try to secure the search process without taking into account the step performed when the search process is done. This step consists in requesting the needed documents from the server after receiving the search result by the user. Consequently, a part of the search result (the requested documents) is leaked to the server at this stage. That is why, the access pattern constraint is not fully respected in many approaches.

Islam et al. [8] demonstrated that the non-respect of this constraint may lead to the disclosure of the query and documents content. As a proof, they propose an attack that exploits the access pattern leakage. The attacker should have some background collected by doing statistical analyses on a few unencrypted data collections similar to the encrypted one. This background corresponds to a matrix $M$ containing the probabilities of term co-occurrence. Loosely speaking, the attack consists in calculating from the search results of two trapdoors, the probability $P$ that the two keywords corresponding to the trapdoors appear together in the same document. After that, the matrix $M$ is exploited to find two terms appearing together with the closest probability to $P$.

To guarantee the access pattern constraint, we propose a method that hides from the server the search result even after requesting the needed documents. Our technique enables the data user to securely access and get documents from the cloud without disclosing their identities to the server. To this end, we propose a new method which is an aggregation of several proposed techniques.

## III. The proposed approach

The proposed technique that we call the *access pattern hiding* (APH) technique is an aggregation of four methods that are presented in this section. The aim of this technique

[2]The server is honest when applying the protocol, but it may collect information about the queries and the data collection.

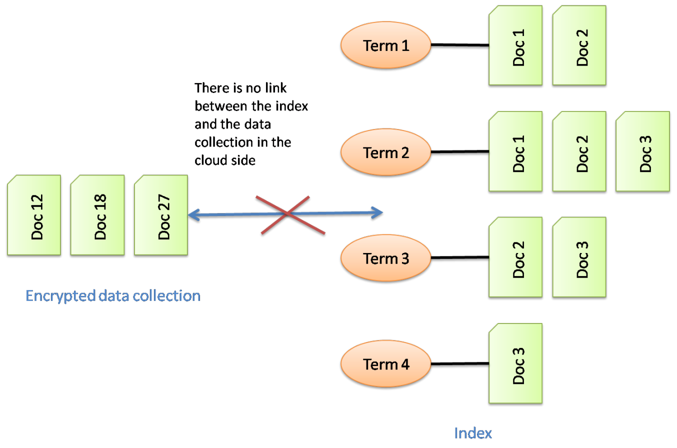

Term 1
Term 2
Term 3
Term 4
Doc 1
Doc 2
Doc 3
Doc 12
Doc 18
Doc 27
There is no link between the index and the data collection in the cloud side
Encrypted data collection
Index

Fig. 2: The separating technique: There is no link between the index and the data collection in the cloud side.

| Data ID | Document ID |
|---|---|
| $Doc_{12}$ | $D_3$ |
| $Doc_{18}$ | $D_1$ |
| $Doc_{27}$ | $D_2$ |

TABLE I: The correspondence table linking between data IDs and documents IDs (example of Figure 2).

is to avoid the leakage of the access pattern in a searchable encryption scheme.

## A. Separating Technique

This technique aims to break any link between the index and the data collection in the cloud side. For this purpose, each document ID in the index should be different from that identifying the same document in the data collection (Figure 2). Consequently, the correspondence between the index and the data collection should be done in the user side. The *separating* technique is performed as follows.

1) Before the indexing process, two identifiers called *document ID* and *data ID* are attached to each document of the data collection. The data IDs are used to represent the encrypted documents of the data collection, whereas, the document IDs are used to represent the documents in the index. The relationship between a data ID and a document ID can be revealed using a table stored on the user side (Table I).
2) When receiving a search result, the *correspondence table* is exploited on the user side to link between the document IDs that have been returned by the cloud and the data IDs. The latter are used to request the necessary documents stored in the cloud server.

The separating technique is able to prevent the server from making links between the data collection and the index as long as no search has been done. However, after a few searches, the server can easily deduce the relationship between some document IDs and data IDs. To overcome this vulnerability, we propose a second technique called the *splitting* technique.

| Document ID | Block ID | Rank |
|---|---|---|
| $D_1$ | $B_5$ | 1 |
| | $B_3$ | 2 |
| | $B_{10}$ | 3 |
| $D_2$ | $B_1$ | 1 |
| | $B_4$ | 2 |
| | $B_8$ | 3 |
| $D_3$ | $B_7$ | 1 |
| | $B_2$ | 2 |
| | $B_9$ | 3 |
| | $B_6$ | 4 |

TABLE II: The table of blocks (TB) which is generated from the example of Figure (3).

| Block ID | Version ID | Block ID | Version ID |
|---|---|---|---|
| $B_5$ | $V_5, V_{11}, V_{27}$ | $B_3$ | $V_{12}, V_3, V_{19}$ |
| $B_{10}$ | $V_9, V_{15}, V_{25}$ | $B_1$ | $V_2, V_{30}, V_{24}$ |
| $B_4$ | $V_{20}, V_{18}, V_{13}$ | $B_8$ | $V_{21}, V_{14}, V_{23}$ |
| $B_7$ | $V_{22}, V_1, V_{16}$ | $B_2$ | $V_6, V_{29}, V_{28}$ |
| $B_9$ | $V_7, V_8, V_{10}$ | $B_6$ | $V_{17}, V_4, V_{26}$ |

TABLE III: The table of versions (TV) which is generated from the example of Figure (3).

## B. Splitting Technique

This technique consists in splitting each document into several blocks to avoid the server from distinguishing between the documents of the data collection. Then, in order to make the server task even more complicated, we propose to generate during the encryption process several versions of each block (Figure 3). To request a document, only one version of each block is randomly chosen in the user side. Consequently, different blocks are requested by the data users for different accesses to the same document. The number of versions should be chosen by the data owner before the encryption process. The *splitting* technique is done on the data owner side as follows.

1) The data owner starts by choosing the block size $S$ (for example, $S = 4kb$ for each block) as well as a parameter

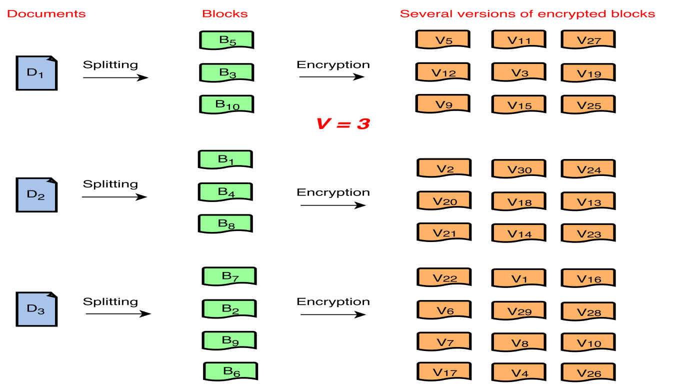


Fig. 3: Splitting of three documents into several blocks encrypted in different versions.

$V$ (for example, $V = 10$) that corresponds to the number of versions of each block.

2) Then, each document of the data collection is split into several blocks of size $S$. A rank $r_i$ is attached to each block $b_i$ to indicate its position when merging back the blocks.
3) After that, each block $b_i$ is encrypted several times using a randomized encryption technique to obtain different versions $\{v_{i1}, v_{i2}, v_{i3}, ...\}$. The number of versions is equal to the parameter $V$.
4) Then, the data owner generates two tables. The first one that we call the *table of blocks (TB)* (Table II) associates each document to its different blocks with their associated ranks, whereas, the second one that we call the *table of versions (TV)* (Table III) associates each block to its different encrypted versions. Note that the different versions of the same block have the same rank. Both tables are encrypted and stored in the user side.
5) To request a document, different set of blocks are selected for each new access to the document which prevents the server from knowing whether the same document has been accessed.

This technique is able to prevent the server from making links between the data collection and the index as long as the number of accesses to a given document $d$ does not exceed a security threshold $\theta_1(d)$. The latter varies from one document to another and is calculated by the following formula:

$$\theta_1(d) = V^{\beta(d)} \tag{1}$$

Where $d$ is a document, $\theta_1(d)$ is the security threshold of the document $d$, $\beta(d)$ is the number of blocks[3] constituting the document $d$, and $V$ is a parameter indicating the number of versions of each block.

The document identity becomes vulnerable after being accessed a number of times higher than the security threshold. To overcome this problem, we propose an additional solution that we call the *scrambling* technique.

## C. Scrambling Technique

This technique consists in requesting the server other blocks that we call the *dummy blocks* in addition to those constituting the wanted document. The aim of these blocks is to add noise and confuse the issue in order to prevent the server from making a link between the true blocks and the corresponding document. The *scrambling* technique is performed on the user side as follows.

1) A parameter $\lambda$ is randomly chosen from an interval $(x, y]$ in order to determine the number of dummy blocks $N_{db}(d)$ which is calculated by the following formula.

$$N_{db}(d) = \lambda \times \beta(d) \tag{2}$$

Where $\beta(d)$ is the number of blocks composing the wanted document $d$, the interval $(x, y]$ is set by the data owner.

2) To obtain a document $d$, the authorized user has to request the encrypted blocks constituting this document. Furthermore, additional dummy blocks are also requested in order to add noise and prevent the server from knowing whether a requested block is a part of those composing the wanted document. The dummy blocks are randomly chosen from all blocks of the data collection and their number is calculated by applying Formula (2).
3) Finally, after receiving the requested blocks, the dummy ones are dropped, whereas, the true blocks are decrypted and merged in order to recover the wanted document that has been requested by the user.

The identity of documents is protected as long as the threshold $\theta_2(d)$ is not exceeded.

$$\theta_2(d) = A \times \theta_1(d) \tag{3}$$

Where $A \in \mathbb{N}_{>1}$ and it increases when $\beta(d)$ or/and $N_{db}(d)$ increase.

The scrambling technique is able to prevent the server from linking a set of blocks $E$ to a document $d$ even if the threshold $\theta_1(d)$ is exceeded given that $\theta_2(d)$ is many times greater than $\theta_1(d)$ . However, even though this technique complicates the server task to disclose the access pattern, it cannot permanently hide the identity of the documents frequently accessed. To achieve this, we propose an additional solution that we call the *grouping* technique.

## D. Grouping Technique

The dummy blocks are randomly selected from the whole blocks constituting the data collection, which explains the fact that there is almost no repeat when requesting them. Contrary to the true blocks that are repeated after a while. Therefore, after requesting a document several times, the server would be able to distinguish the true blocks from the dummy ones. To overcome this security breach, our idea is to divide the blocks constituting the data collection into several groups. When requesting the blocks of a document, the dummy ones are selected from the same groups of the true blocks. This technique allows to repeat a certain number of dummy blocks which avoids the attacker from distinguishing between the true blocks and the dummy ones. The *grouping* technique is performed as follows.

1) The blocks constituting the data collection are divided into several groups (see Table IV). Two blocks of the same document should not appear in the same group. The groups should have the same size $Sz$, where:

$$Sz > y + 1 \tag{4}$$

With $y$ is the parameter of the interval $(x, y]$ that has been used in the scrambling technique (Section III-C).

2) During the scrambling technique, after the number of dummy blocks $N_{db}(d)$ is calculated, the dummy blocks are selected from the groups to which the true blocks belong. For example, if the user wants to request three

[3] We consider only one version of each block.

| Group ID | Blocks | Group ID | Blocks |
|---|---|---|---|
| $G_1$ | $V_5$, $V_{21}$, $V_7$ | $G_2$ | $V_{23}$, $V_{10}$, $V_{35}$ |
| $G_3$ | $V_{11}$, $V_{20}$, $V_4$ | $G_4$ | $V_{24}$, $V_6$, $V_{26}$ |
| $G_5$ | $V_{16}$, $V_{12}$, $V_{36}$ | $G_6$ | $V_{19}$, $V_1$, $V_{33}$ |
| $G_7$ | $V_{15}$, $V_2$, $V_{17}$ | $G_8$ | $V_{27}$, $V_8$, $V_{34}$ |
| $G_9$ | $V_{18}$, $V_{22}$, $V_{31}$ | $G_{10}$ | $V_{25}$, $V_{30}$, $V_{29}$ |
| $G_{11}$ | $V_3$, $V_{14}$, $V_{28}$ | $G_{12}$ | $V_9$, $V_{13}$, $V_{32}$ |

TABLE IV: The grouping table (example of Figure 3).

true blocks belonging to the groups $A$, $B$, and $C$, respectively. If the number of dummy blocks calculated by Formula 2 is equal to six. Therefore, two dummy blocks should be randomly selected from the group $A$, two dummy blocks should be selected from the group $B$, and two dummy blocks should be selected from the group $C$.

Given that some dummy blocks are repeated with the true blocks, the server task becomes more difficult to disclose the document identity. Therefore, the identities of documents are protected as long as the threshold $\theta_3(d)$ is not exceeded, where, $\theta_3(d) \gg \theta_2(d) \gg \theta_1(d)$.

$$\theta_3(d) = B \times \theta_2(d) \tag{5}$$

Where $B \in \mathbb{Q}_{>1}$ and it increases when $\beta(d)$, $N_{db}(d)$ and/or $Sz$ increase[4].

## E. Putting all together

The access pattern hiding (APH) technique is the aggregation of the four proposed methods previously introduced, namely, the separating technique, the splitting technique, the scrambling technique, and the grouping technique. The APH technique is a universal method that allows to keep any SE scheme respecting the access pattern constraint by securely accessing the needed documents stored in the remote server. In addition, it does not depend nor on the index structure neither on the encryption methods used. In the following we explain how to exploit the APH technique in a searchable encryption scheme.

1) Before the encryption of the data collection, the data owner applies the *splitting technique*. For that, each document is split into several blocs of size $S$. A rank $r_i$ is attributed to each block $b_i$ in order to keep the ability to reconstruct the document later. Finally, the data owner generates the table of blocks ($TB$) to link between a document and its different blocks with their associated ranks.
2) After that, each block is encrypted into several versions. The number of versions is equal to the parameter $V$ chosen by the data owner. The latter constructs the table of versions ($TV$) in order to link between a block and its different encrypted versions. Finally, the table of blocks and the table of versions are encrypted and stored in the user side, whereas, the encrypted blocks are outsourced to the cloud server.
3) Then, the data owner applies the *grouping technique*. For that, he chooses the interval $(x, y]$ and uses the parameter $y$ in order to calculate the size of the groups ($Sz$). After that, he divides the blocks into several groups of size $Sz$. The groups should have the same size, and the blocks of the same document should appear in different groups.
4) During the indexing process, the data owner applies the *separating technique* to obtain an index with document identifiers different from those of the data collection. For that, the document IDs are used in the index, whereas, the version IDs are used in the data collection. The correspondence between the index and the data collection should be made on the user side using both tables $TB$ and $TV$. Finally, the index should be encrypted and outsourced to the cloud server.
5) During the search, the data user formulates a query and encrypts it to construct a trapdoor that is sent to the cloud server. Upon receiving the trapdoor, the cloud server launches the search and returns a list of top-k document IDs to the user. The latter chooses the document IDs that he needs from the returned results and uses the table of blocks to obtain the block IDs corresponding to the document IDs. After that, the table of versions is also exploited to randomly obtain an encrypted version ID corresponding to each block ID. Then, the *scrambling technique* is applied to add some dummy blocks (version IDs). The dummy blocks should be selected from the same groups as the true blocks. Finally, the dummy and the true blocks are requested from the server.
6) After receiving the requested blocks from the server, the dummy ones are dropped, whereas, the true blocks are exploited to reconstruct the needed documents. Note that the ranks of blocks stored in the table of blocks are primary in the reconstruction process. Finally, the documents are decrypted and exploited by the user.

## IV. Evaluation

## A. Security Analysis

We developed a program (available on-line[5]) using *JAVA* programming language to simulate the different techniques that we have proposed in this work. The experiments were performed on a computer equipped with an Intel i5-2400 3.10 GHz processor. We have exploited a data collection that contains 1 000 documents, and we simulated 100 000 access to this data collection in each experiment.

In the first experiment, we simulated the different accesses without trying to hide the access pattern. Figure 4 shows that the first 250 documents are the most accessed, then, the 250 documents that follows, and so on. We notice from this experiment that the accessed documents are known by the server, and thus, the access pattern constraint is not respected.

[4]Note that $Sz$ (the groups size) is different from $S$ (the block size) mentioned in section III-B.

[5]https://github.com/boucennafateh/APH

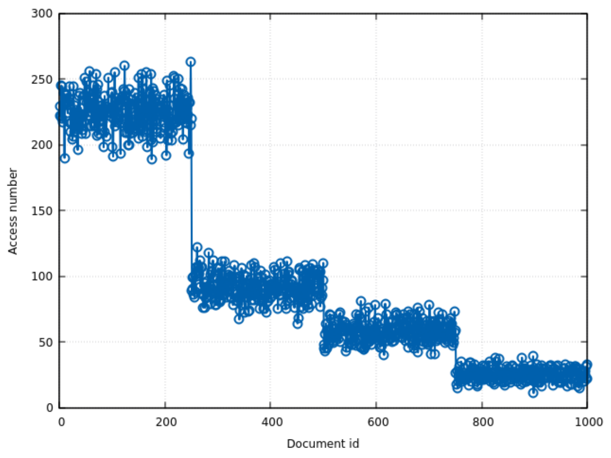


Fig. 4: Access pattern without applying any technique

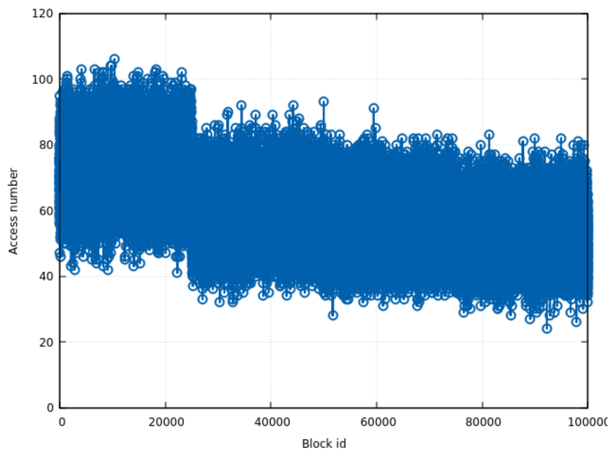


Fig. 6: Access pattern when applying separating, splitting, and scrambling techniques

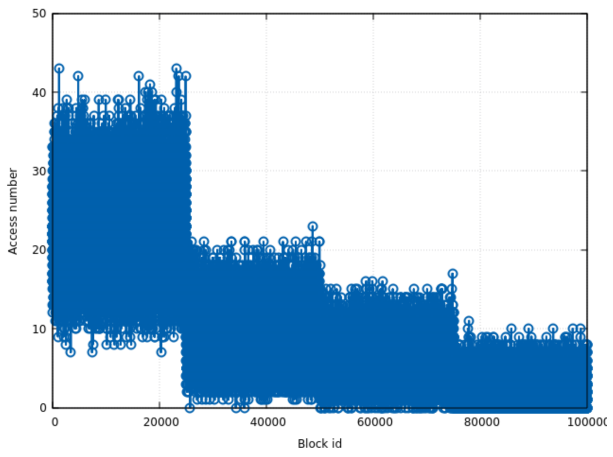


Fig. 5: Access pattern when applying separating and splitting techniques

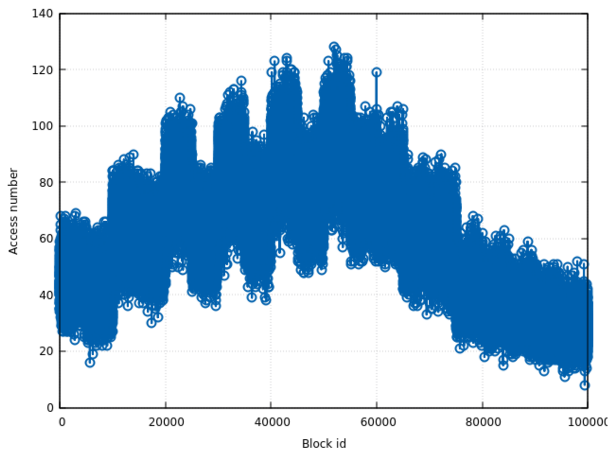


Fig. 7: Access pattern when applying APH technique (separating, splitting, scrambling, and grouping techniques)

In the second experiment, we applied both the separating and the splitting techniques. Each document is split into 10 blocks, and each block is encrypted into 10 versions. Figure 5 shows that the curve begins to become compact, but the access pattern is not completely hidden, and the server can recognize the documents (the set of blocks) that are the most accessed, and those which are the less accessed.

In the third experiment, we applied the separating, the splitting, and the scrambling techniques. For each requested document (10 blocks), 50 additional dummy blocks randomly chosen are also requested. Figure 6 shows that the curve is more compact than the previous experiment, but the access pattern is still not completely hidden, and the server can recognize that the first set of blocks is the most accessed by the users.

In the fourth experiment, we applied the APH technique (the aggregation of the separating, splitting, scrambling, and grouping techniques). We constructed 100 groups of 1 000 blocks. Figure 7 shows that the aggregation of the four techniques allows to obtain a compact curve, which means that the access pattern is hidden from the server. The latter is no longer able to recognize which blocks are the most accessed by the users.

## B. Performance Analysis

We performed our experiments on a collection of 100 documents using a computer equipped with an *Intel Xeon 2.13 GHz*. In this experimental study, we set $V = 5$, $S = 500$ bytes and $(x, y] = (0, 1]$.

First, we encrypted a collection of 100 documents using both the *APH* and the *AES* encryption. Figure 8a shows that the size of the data collection increases only slightly when using the *AES* method. However, it increases by a factor of $V$ when using the *APH* technique. After that, we compared the encryption time using the proposed APH method with the AES method. Figure 8b shows that the encryption process takes more than $5s$ to encrypt a collection of 100 documents using the APH method. On the other hand, less than $30ms$ are needed to encrypt the same data collection using the AES

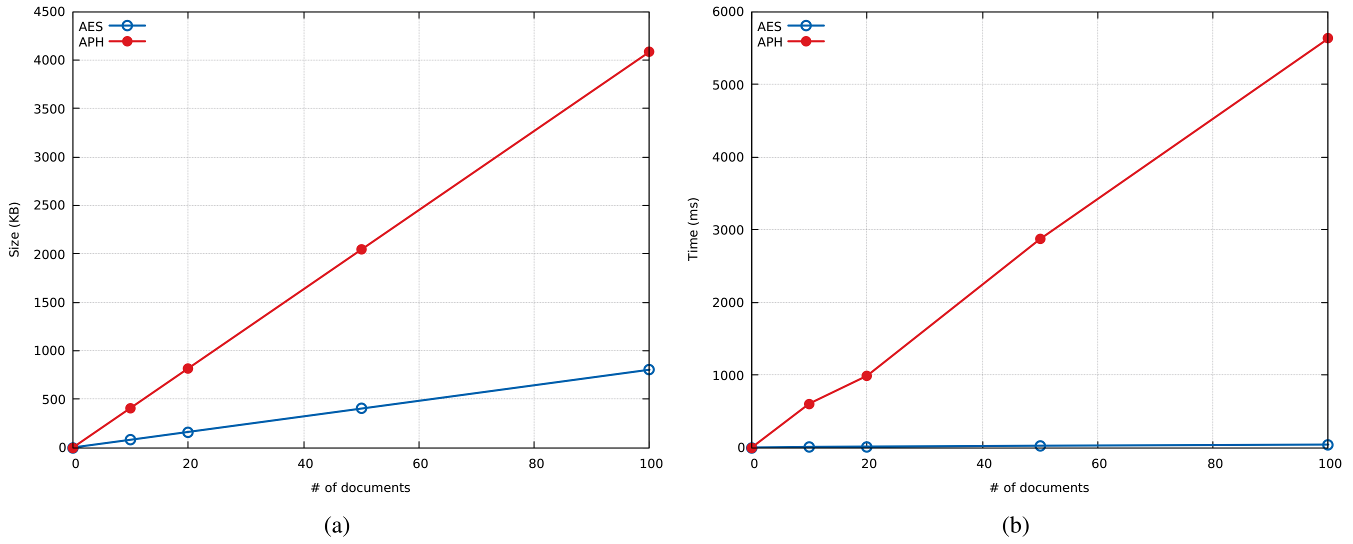


Fig. 8: The size of the encrypted data collection (a) and the encryption time (b) when using the *AES* and the *APH* techniques.

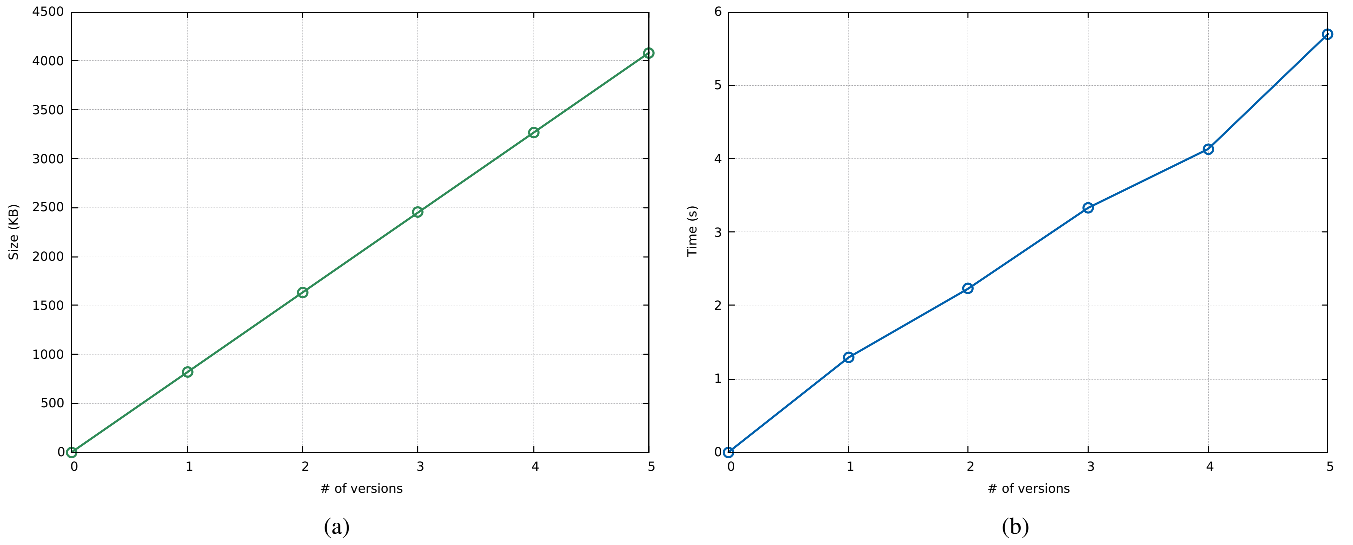


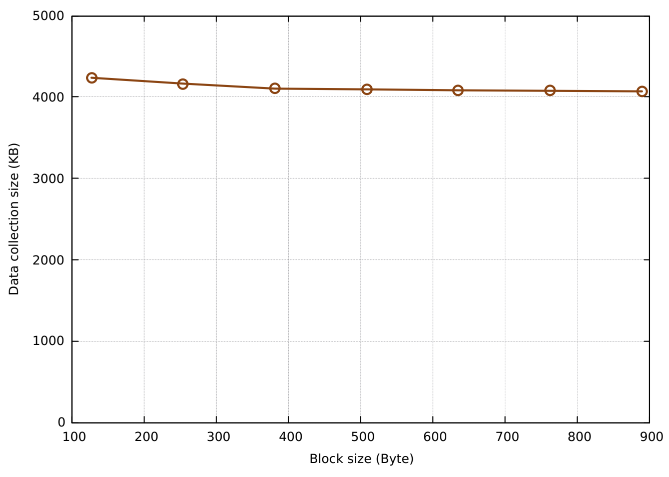
Fig. 9: The size of the encrypted data collection (a) and the encryption time (b) based on the number of block versions.

method. However, given that the data collection is split into several blocks when using the APH technique, the use of parallelism during the encryption process may significantly reduce the encryption time.

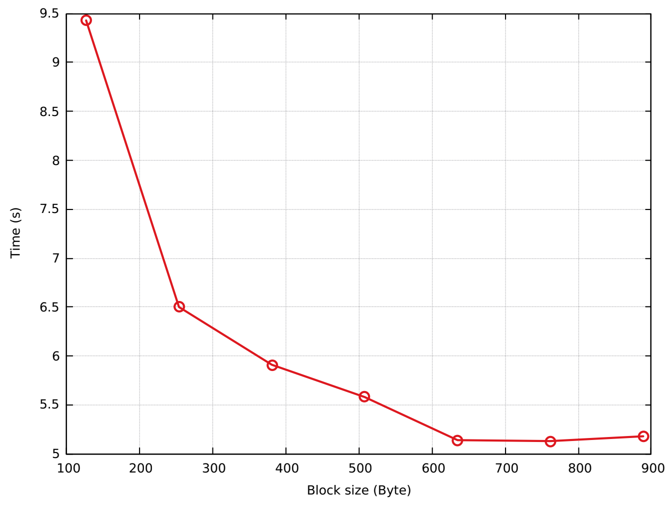
Then, we tested the APH technique using different values of $V$ (i.e. the number of encrypted versions of each document block). The experiment was performed on a collection of 100 documents split into several blocks of 500 bytes. Unsurprisingly, Figure 9a shows that the size of the encrypted data collection increases linearly with the parameter $V$ and the same goes for the encryption time (Figure 9b). Nevertheless, even if there is some decrease in the scheme performance, increasing the parameter $V$ allows to reinforce the security by obtaining a higher threshold $\theta_1$.

Finally, we tested the APH technique with different values of $S$ (i.e. the size of an encrypted block). Affecting a small value to the parameter $S$ (e.g. $S = 127$ bytes) generates a great number of small blocks which allows to enhance the security of the scheme by increasing the threshold $\theta_1$ (see Formula 1). Nevertheless, Figure 10b shows that the encryption time increases when $S$ is small. Indeed, the encryption process takes $9.5s$ to encrypt 100 documents split into blocks of size 127 bytes rather than $5.2s$ when $S$ is equal to 700 bytes. On the other hand, Figure 10a shows that changing the parameter $S$ does not affect the size of the encrypted data collection.

This experimental study is performed to test the performance of the APH technique. To obtain a better security, it is important to increase as much as possible the security threshold $\theta_1$. This amounts to maximize the number of block versions $V$ and the number of document blocks $\beta(d)$ (by



(a)



(b)

Fig. 10: The size of the encrypted data collection (a) and the encryption time (b) based on the size of blocks.

minimizing the block size $S$). On one hand, minimizing $S$ leads to increasing the encryption time. On the other hand, maximizing $V$ leads to increasing the ciphertext size and the encryption time. The encryption process is a one time operation and could be sped up using a kind of *HPCs* such as a *GPU*. Thus, it is preferable to set the parameters $V$ and $S$ so that the ciphertext size will be optimized. For this purpose, our recommendation consists in choosing a small value of $S$ rather than choosing a big value of $V$. For example, if we have a document of $100KB$ and we want to obtain a security threshold $\theta_1$ equal to 16. It is better (in term of storage space) to set $S = 25KB$ ($\beta(d) = 4$) and $V = 2$ ($\theta_1(d) = V^{\beta(d)} = 2^4 = 16$) rather than choosing $S = 50KB$ ($\beta(d) = 2$) and $V = 4$ ($\theta_1(d) = 4^2 = 16$).

## V. Conclusion

To design a secure SE scheme, it is primary to respect the security constraints elaborated in the literature. In spite of the keyword privacy and the trapdoor unlinkability constraints are respected by most recent approaches proposed in the literature, the access pattern constraint is still not fully respected. The problem comes from the fact that the approaches proposed in the literature do not take into consideration the step performed when the search is done which consists in requesting the needed document by the user. During this step, a part of the search result is disclosed to the server and thus, the access pattern is not correctly hidden. The APH method that we have proposed in this work allows to fix this security breach and thus, guarantee a full respect of the access pattern constraint.

## References

[1] K. Li, W. Zhang, K. Tian, R. Liu, and N. Yu, "An efficient multi-keyword ranked retrieval scheme with johnson-lindenstrauss transform over encrypted cloud data," in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*. IEEE, 2013, pp. 320–327.

[2] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 222–233, 2014.

[3] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2112–2120.

[4] F. Boucenna, O. Nouali, S. Kechid, and M. T. Kechadi, "Secure inverted index based search over encrypted cloud data with user access rights management," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 133–154, 2019.

[5] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[6] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, "Toward secure multikeyword top-k retrieval over encrypted cloud data," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 239–250, 2013.

[7] B. Wang, W. Song, W. Lou, and Y. T. Hou, "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 2092–2100.

[8] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." in *Ndss*, vol. 20, 2012, p. 12.

[9] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 639–654.

[10] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Accessing data while preserving privacy," *arXiv preprint arXiv:1706.01552*, 2017.

[11] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, vol. 3876. Springer, 2006, pp. 265–284.

[12] O. Goldreich, "Towards a theory of software protection and simulation by oblivious rams," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 182–194.

[13] C. Gentry, *A fully homomorphic encryption scheme*. Stanford University, 2009.

[14] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 139–152.

[15] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 18, 2015.