Privacy Preserving Multi-Proxy Based Encrypted Keyword Search

Özgür Öksüz

Department of Software Engineering, Konya Technical University, Konya, Turkey
Corresponding Author: ooksuz@ktun.edu.tr

Research Paper Received: 23.07.2022 Revised: 30.10.2022 Accepted: 05.12.2022

Abstract—This paper presents a multi-proxy (2 proxies) based encrypted keyword search scheme that enjoys the following properties: This scheme provides data confidentially that encrypted data does not leak any keywords and documents to the attackers (data server and a proxy). Moreover, the proposed scheme provides trapdoor privacy. In other words, the attackers do not learn any information about searched keywords. Furthermore, even if a proxy is controlled by an attacker, the attacker does not learn any information about the queries (keywords that the user searches over the database) and database results. Different from other studies, this scheme provides lightweight user-side query and data processing. In other words, most of the job (query processing) is done by the proxies on behalf of the user. Finally, the proposed scheme relaxes the trust assumption that eliminates a single point of failure by introducing multi-proxy architecture.

Keywords—data privacy, multi-proxies, keyword search.

1. Introduction

Disclosing private information is a very crucial problem [1]. Databases keep very critical information about users and are managed by the database servers. Since the database servers keep users' sensitive information, an untrusted or compromised database server can read/view users' private information. The adversary simply exploits software vulnerabilities to get into the server to retrieve the users' private information. What an adversary does is to exploit software vulnerabilities to get into the servers [2] to steal the users' sensitive information. To hide the users' sensitive/private information from an untrusted server data should be encrypted. In this case, even if the attacker compromises the database server, it can not learn any useful information about plaintexts of the data. Encrypting the data is good for hiding sensitive information. However, it might be very inefficient. If the user needs to retrieve a specific document from the database, the user needs to download all the encrypted data. Then, the user locally decrypts all the encrypted documents to figure out which document it looks for. This requires very high bandwidth and computation costs. To eliminate these problems above, the user encrypts its data and put the data into the database server. When the user needs specific documents, instead of downloading all its data, the user gives a token (encrypted keyword or trapdoor) to the database server, and the database server responds with the encrypted documents that contain the keyword. In this way, the database server does not know anything about the keyword since it is encrypted. Moreover, this solution lowers the computational and communication costs.

There are bunch of encryption schemes that can be used to encrypt the database and keywords to obtain privacy preserving keyword search system. Fully homomorphic encryption, [3], allows servers to compute any functions over encrypted database but this encryption scheme is prohibitively expensive. Order-preserving encryption, [4], [5], [6], [7] and [8], allows servers to compare ciphertexts based on their orders. In other words, if the relation between plaintext messages are $m_1 > m_2$, then the ciphertexts of these messages have the same relation, $c_1 > c_2$. The order-preserving encryption leaks the orders of the data. Public key encryption such as Elliptic curve cryptosystem, Pallier encryption [9] and RSA [10] require much more computational cost than symmetric key encryption such as AES. Thus, using a symmetric key encryption scheme is more computationally efficient than using a public key encryption scheme to encrypt data [11].

The user encrypts sensitive data before it sends to the database server. This solution provides data confidentiality. However, encrypting every keyword and decrypting all the results set for each query can be computationally expensive with resource-limited devices like mobile devices. To eliminate this problem, a fully trusted proxy based systems can be used [7], [12], [13], [14], [15]. These systems provide lightweight user side data processing (encryption and decryption). In this systems, the proxy has encryption and decryption keys that is located intermediary between user application (usually application server runs it) and the database server. However, this kind of architecture has a single point of failure. When a fully trusted proxy is compromised, the keys are leaked to the attacker. This results that the adversary uses decryption key to decrypt all encrypted data. Moreover, it obtains the plaintexts of all the data. Moreover, the adversary learns what the user wants to search.

In order to address the problems above, we introduce a multi-proxy (2 proxies) based encrypted keyword search system that has the following properties:

- Our scheme eliminates single point of failure. We relax the trust assumption by introducing multi-proxy (two) based system. Even if one of the proxies is compromised, the adversary can not learn any useful information about the queries and the query results.
- Our scheme provides lightweight user side query and data processing. Since the encryption/decryption keys are given to the proxies, the proxies do the encryption and decryption computation on behalf of the user.
- We use symmetric key encryption scheme to have very fast encryption and decryption computations.

2. Related Work

There have been several studies that allow a database server to search on encrypted database. The study in [7] developed CryptDB that executes SQL queries over encrypted database using collection of operations such as equality check, order computations. [12] proposed a bucketization techniques that divide the domain of a column into partitions, randomly map the partitions, and then store the partition number for each data item. The studies in [13], [14] introduce a secure and efficient range search schemes over encrypted database. The work in [15] introduces a privacy-preserving queries on encrypted database protocol. These studies have the same weakness that their protocols have a fully trusted proxy. Therefore, these schemes have a single point of failure.

There have been some studies about search and query over encrypted data. Some of these studies use symmetric key encryption to encrypt data and queries, [16], [17], [18], [19], [20]. The others use public key encryption to encrypt data and queries [21], [22], [23], and [24]. These studies require significant user side query and data processing.

A comparison between our work and other symmetric searchable encryption schemes is shown in Table 1. The comparison is based on the data owner's computational cost when it generates a trapdoor and decrypts a file in a result set. Moreover, we also compare the storage size of the data owner. In the table, some studies do not include decryption phase in their original schemes [17], [19], [20]. We assume that they use AES encryption scheme for the decryption process. Thus, we use $*$ symbol for these studies' corresponding decryption columns. In study [20], the authors use a block cipher to generate trapdoor but they do not mention a specific block cipher. We assume that their scheme uses AES block cipher. Thus, we use $**$ symbol for study [20]'s corresponding trapdoor generation column. Moreover, in the table PRF is a pseudo random function operation. PRP is a pseudo random permutation operation. In [16], the data owner needs to do one PRF and one PRP operation to generate a trapdoor. In studies [17] and [19], the data owner needs to use PRF r times. In our work, the data owner needs to use one PRF operation. In studies [16], [17], [19], [20], the data owner needs to do one AES decryption. However, in our work the data owner needs to do only one XOR operation. In studies [16] and [19], the data owner needs to store $O(r)$ items (secret keys). In studies [16], [20] and this work, the data owner needs to store $O(1)$ elements. The work in [18] uses bucketization technique and maps each bucket to a random value. The authors encrypts the values and stores in the database server. In their scheme, they figure out optimal bucket size with preserving privacy. When the bucket size is large, their scheme results better privacy but introduces a large number of false positive results. Thus the database server needs to send a large number of result set. When the bucket size is small, they have weak privacy but small false positive result set. In our scheme we do not have such privacy-efficiency tradeoff. Moreover, our scheme does not introduce any false positive results.

A note that example of secure PRPs are $AES, 3DES$, which are block ciphers. Thus, PRP is also AES in the table. In a 10 round encryption/decryption algorithm of AES, there are 164 XOR, 120 $RIGHT/LEFT\ SHIFT$, and 136 AND operations [25]. Moreover, in practice, $HMAC-SHA1$ can be used to implement a secure pseudo random function (PRF). To implement a $HMAC-SHA1$, one needs $2XOR$ operations and to apply H (hash) 2 times.

Table 1.
Comparison between the Proposed Work and Other Studies.

Reference	Trapdoor Gen.	Decryption	Storage
[16]	PRF + PRP (AES)	AES	$O(1)$
[17]	rPRF	AES^*	$O(r)$
[19]	rPRF	AES^*	$O(r)$
[20]	AES^{**}	AES^*	$O(1)$
This work	PRF	XOR	$O(1)$

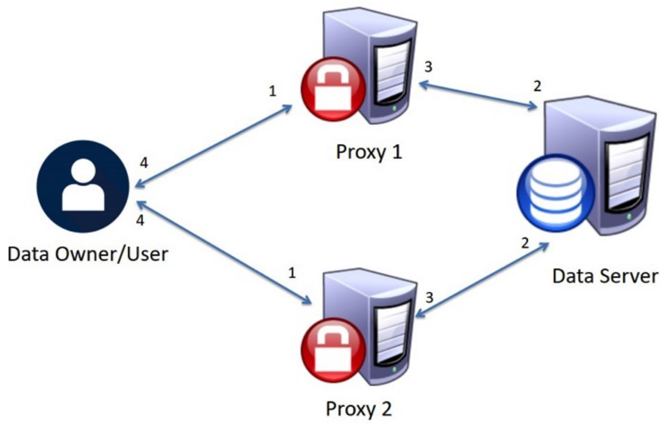

Figure 1. Sytem Architecture

3. System Architecture and Threat Model

3.1. System Architecture

Our proposed architecture is shown in Fig. 1. Our architecture has 4 entities: a data owner/user, (DO) two proxies ($Proxy_1$ and $Proxy_2$), and a database server (DB). The data owner encrypts its data and stores it in the database server, the proxies have encryption and decryption keys that are sent by the data owner via a secure channel. For simplicity, we assume the unencrypted data (T_{unenc}) is of the form depicted in Tab. 2.

The workflow of the system is as follows:

1. the user sends its keyword query to the proxies,
2. each proxy uses its encryption key to encrypt the keyword query and sends it to the database server,
3. the database server combines the received encrypted keyword queries from the proxies, retrieves the encrypted data (documents) from the database. Then, the database server sends the encrypted documents to the proxies,
4. each proxy decrypts the documents and sends the results to the user. At last, the user recovers the documents.

Table 2.
An Illustration of An Unencrypted Table (T_{unenc}) ($\|$ is the concatenation operation).

Keywords	Documents
$w_{1,1}\|w_{1,2}\|w_{1,3}\|\ldots\|w_{1,m}$	D_1
$w_{2,1}\|w_{2,2}\|w_{2,3}\|\ldots\|w_{2,m}$	D_2
$w_{3,1}\|w_{3,2}\|w_{3,3}\|\ldots\|w_{3,m}$	D_3
$\vdots \quad \vdots \quad \vdots \quad \ddots$	$\vdots$
$w_{n,1}\|w_{n,2}\|w_{n,3}\|\ldots\|w_{n,m}$	D_n

3.2. Threat Model

In our system, there are two honest-but-curious adversaries. One adversary is the database server that controls the database. The another adversary controls only one proxy. These two adversaries can not collude. The other proxy is trusted. Data owner/ user also trusted.

A honest but curious adversary follows the protocol specification: it does not change database values, query results but it can try to learn plaintext of the data and plaintext of the queries. We also assume that there are secure channels between user and the proxies. Furthermore, we assume that there are secure channels between the proxies and the database server.

Our security guarantees are two folds: Data privacy and Trapdoor privacy.

Data privacy: The untrusted database server can not see the plaintexts of the data and any useful information about the data (repeating elements in the data columns, documents in the clear). Moreover, the untrusted proxy is not able to learn query results (plaintexts of the documents).

Trapdoor privacy: The untrusted proxy and untrusted database server can not learn which keyword issued by the user.

We assume that the adversary (database server) observes polynomial number of queries t (a function of security parameter, λ) $(q_1, \ldots, q_t)$ queries. We say that once these queries are issued, the protocol leaks only access patterns (locations of the records that satisfy each query), search patterns (whether any two queries/trapdoors are generated from the same keyword or not) and encrypted table (size of each record, column and row sizes of the table). The adversary observes only these leakages. We are going to use simulation based proof system to have data privacy in our system. In simulation proof system, there are two worlds: real and ideal. In the real world, protocol is executed between the parties $Proxy_1$, $Proxy_2$, the data server. The adversary, $\mathbf{A}$, corrupts one entity ($Proxy_2$ or the data server) at a time. In the ideal world, there is a simulator (**SIM**). In the ideal world, each party receives the same input as that in the real world. The simulator simulates the all transactions (messages) between the entities using input and output (leakages) of the corrupted party in the real world so that the adversary can not distinguish real world from ideal world. In other words, the adversary can not know if it is interacted with ideal world or real world.

A trapdoor is an issued keyword that the data owner wants to search it over database. In order to have trapdoor privacy in our protocol, the searched keyword (trapdoor) should be encrypted. Thus, we use encryption on searched keywords to achieve trapdoor privacy.

4. Definitions

In this section we introduce how encrypted keyword search works. In a symmetric encrypted keyword search scheme, the data owner/user has a collection of documents/files $\mathcal{D} = (D_1, \ldots, D_n)$, and each document D_i has set of unique keywords $W_i = \{w_{i,1}, \ldots, w_{i,m}\}$. The data owner encrypts these documents and the keywords with a symmetric key encryption scheme. Then, the data owner puts these encrypted values to honest-but-curious server.

The server will not learn any useful information about the plaintext documents and the keywords. In the system, the data owner gives an ability to the server to search the keywords. Then, the database server returns appropriate results (encrypted documents) to the user.

Definition 1 (Encrypted Keyword Search) *The definition of an encrypted keyword search scheme has the following algorithms:*

Setup(1^λ)**:** *It is a randomized key generation algorithm that is run by the data owner to generate an encryption key. It takes security parameter λ as an input, it outputs a secret key K.*

Enc$(K, \mathcal{D})$**:** *This algorithm is run by the data owner to encrypt the documents and the keywords. It takes a secret key K and a document collection $\mathcal{D} = (D_1, \ldots, D_n)$, and their keywords $\mathcal{W} = (W_1, W_2, \ldots, W_m)$ as inputs, it outputs a sequence of ciphertexts (for documents) $C_D = (C_{D_i}, \ldots, C_{D_n})$, and (for keywords) $C_{\mathcal{W}} = (C_{W_1}, \ldots, C_{W_n})$, where C_{D_i} is the ciphertext of document D_i, and C_{W_i} is the encrypted keyword set W_i for document D_i. We use the following notations: $C_{\mathcal{D}} \leftarrow Enc_K(\mathcal{D})$ and $C_{w_{i,j}} \leftarrow Enc_K(w_{i,j})$, where $C_{w_{i,j}} \in C_{W_i}$.*

Trpdr(K, w)**:** *It is a deterministic algorithm run by the data owner to generate a trapdoor for a given keyword. It takes a secret key K and a keyword w as inputs, and outputs a trapdoor $tr_{K,w}$. We can write this as $tr_{K,w} \leftarrow Trpdr(K, w)$.*

Search$(C_{\mathcal{W}}, tr_{K,w})$**:** *It is a deterministic algorithm run by the server to search for the documents in $\mathcal{D}$*

that contain keyword w. It takes encrypted keyword ciphertexts $C_{\mathcal{W}}$, and trapdoor $tr_{K,w}$ as inputs, it outputs a document identifier set id that contains indexes of the documents that contain w.

Dec$\big(K, C_{\mathcal{D}}^{id}\big)$: *It is a deterministic algorithm run by the data owner to recover the plaintexts of documents. It takes a secret key K, a set of indexes $C_{\mathcal{D}}^{id}$ and their corresponding ciphertexts as inputs, it outputs document D_i. We sometimes write this as $D_i \leftarrow Dec_K\left(C_{D_i}\right)$, where $i \in C_{\mathcal{D}}^{id}$.*

Definition 2 (Pseudo Random Function)
The property of a pseudo-random function is computationally indistinguishable from a totally random function. If given pairs $(x_1, H(k, x_1))$, $(x_2, H(k, x_2))$, $\ldots$, $(x_\mu, H(k, x_\mu))$, an adversary cannot predict $H(k, x_{\mu+1})$ for any $x_{\mu+1}$. A (t, ϵ, q)-pseudo random function $H : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ satisfies the following properties:

- *$H(k, x)$ can be computed efficiently from input $x \in \{0,1\}^n$ and key $k \in \{0,1\}^\lambda$.*
- *for any t time oracle algorithm $\mathbf{A}$ that makes at most q adaptive queries,*
  $|Pr[\ \mathbf{A}^{H(k,.)} = 0 | k \leftarrow \{0,1\}^\lambda]$ - $Pr[\ \mathbf{A}^{H'} = 0 | H' \leftarrow \{F : \{0,1\}^n \to \{0,1\}^n\}| < \epsilon$.

5. Multi-Proxy Based Encrypted Keyword Search

In this section, we introduce our encrypted keyword search scheme that has two proxies. Our scheme has distributed encryption and decryption. The data owner does not generate the trapdoor alone. The data owner encrypts all data using two different symmetric secret keys and hands them over to the proxies. The proxies generate the encrypted trapdoor on behalf of the user. Then each proxy sends the partially encrypted trapdoor to the database server. The database server than returns the result sets to the proxies. Moreover, the data owner does not decrypt the documents alone. The proxies decrypt the ciphertexts of the documents on behalf of the user/data owner. This process can be thought as secret sharing for encryption and decryption. An attacker can not learn any useful information when it corrupts one proxy. It needs to corrupt both proxies.

Moreover, in the proposed protocol, none of the proxies does not learn any useful information about searched keyword. Also, the proxies do not know any useful information about plaintext documents. Furthermore, the data owner only combines the results sent by the proxies. Thus, the data/user owner does not do much computation. Our scheme provides lightweight user side query and data processing.

Since the data owner needs to combine the results, we introduce a new algorithm in our scheme. We call the new algorithm $Combine$. Combine algorithm takes two parameters as input: the first parameter comes from $Proxy_1$ (par_1) and the second parameter comes from $Proxy_2$ (par_2). The output of the combine algorithm is $(par_1 \oplus par_2)$ where $\oplus$ is the bitwise XOR operation. We write this as $(par_1 \oplus par_2) \leftarrow Combine_{par_1, par_2}$.

To make a secure and efficient encrypted keyword search scheme, we get the intuition from scheme in [15].

Our scheme is different than the scheme in [15] in the following ways:

- The work in [15] has single point of failure. When the proxy (front end) is compromised, all the keywords and documents in the clear are leaked to the adversary. Our scheme eliminates single point of failure that the proposed scheme does not depend on only one trusted party (proxy).

- The scheme in [15] does not have decryption and combine algorithms. In our scheme, once the database server sends the corresponding encrypted documents to the proxies, the proxies decrypt the ciphertexts separately. Then, the proxies send the results to the user. At the end, the user recovers the documents that contain the keyword by using combine algorithm.

Note that we assume that each document has the same number of unique keywords, m, in the construction. However, each file can have different number of unique keywords. The data owner needs to make equal each file's unique keyword size to m in the construction. This number is the maximum number of unique keywords that each file needs to have. If a file has $m' < m$ unique keywords, the data owner adds $m - m'$ random keywords to that file before the encryption process. The details of our scheme is as follows:

Setup(1^λ)**:** The data owner chooses 4 encryption keys $K = \{k_1, k_2, k_3, k_4\}$ randomly from set $\{0,1\}^\lambda$ and a pseudo random function $H : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^a$. Moreover, the data owner also chooses $p_{id_1} \leftarrow \{0,1\}^b, p_{id_2} \leftarrow \{0,1\}^b$, where p_{id_1}, p_{id_2} are identities of $Proxy_1$ and $Proxy_2$. The data owner sends (k_2, p_{id_1}) to $Proxy_1$ and (k_3, p_{id_2}) to $Proxy_2$.

Setup algorithm is given in Fig. 2.

Enc(K, D_i, W_i)**:** To encrypt document $D_i \in \mathcal{D}$ and keyword set $W_i \in \mathcal{W}$, the data owner does the followings for every $w_{i,j} \in W_i$, where $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$,

1. Computes $H(k_1, w_{i,j})$,
2. Computes $x_{i,j} = Enc_{k_2}(H(k_1, w_{i,j}), p_{id_1})$ and Computes $y_{i,j} = Enc_{k_3}(H(k_1, w_{i,j}), p_{id_2})$,
3. Computes $q_{i,j} = Enc_{k_2}(H(k_1, w_{i,j}), p_{id_1}) \oplus Enc_{k_3}(H(k_1, w_{i,j}), p_{id_2})$ $(x_{i,j} \oplus y_{i,j})$, where $\oplus$ is the bitwise XOR operation.

Figure 2. Algorithm $Setup$. (Done by Data Owner (DO))

```
**Input**: $(\lambda, a, b), (\lambda = a + b)$.
**Output**: $(K = \{k_1, k_2, k_3, k_4\}, H, p_{id_1}, p_{id_2})$.
- $k_1, k_2, k_3, k_4 \leftarrow \{0,1\}^\lambda$
- $p_{id_1}, p_{id_2} \leftarrow \{0,1\}^b$
- $H : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^a$
**DO** sends $(k_2, p_{id_1})$ to **Proxy₁**.
**DO** sends $(k_3, p_{id_2})$ to **Proxy₂**.
```

4. Chooses random (salt) $r_{i,j} \leftarrow \{0,1\}^b$ and computes $E_{i,j} = Enc_{k_4}(H(k_1, w_{i,j}), r_{i,j})$, where $b = \lambda - a$.
5. Computes $F_{i,j} = Enc_{q_{i,j}}(E_{i,j})$.
6. Computes $G_i = Enc_{k_2}(D_i \oplus r_i)$ and $H_i = Enc_{k_3}(r_i)$, where r_i is a new random (salt) element. The lengths of D_i and r_i are the same, $|D_i| = |r_i| = \lambda'$, where $\lambda < \lambda'$.
7. Sends $C_{w_{i,j}} = (E_{i,j}, F_{i,j})$ and $C_{D_i} = (G_i, H_i)$ to the database server.
8. The data owner does these steps for each document.

Enc algorithm is given in Fig. 3.

Trpdr(K, w)**:** To query keyword w, the data owner/user

9. Computes $H(k_1, w)$ and sends it to $Proxy_1$ and $Proxy_2$.
10. $Proxy_1$ computes $x' = Enc_{k_2}(H(k_1, w), p_{id_1})$ and $Proxy_2$ computes $y' = Enc_{k_3}(H(k_1, w), p_{id_2})$. They separately sends x' and y' to the database server. Here we can say $x' = tr_1$ (trapdoor one) and $y' = tr_2$ (trapdoor two).

Trpdr algorithm is given in Fig. 4.

Search$(C_\mathcal{W}, tr_{K,w})$**:** Once the server receives trap-

Figure 3. Algorithm Enc. (Done by Data Owner (DO))

```
Input: $(H, a, b, \lambda, \lambda', K, W_i, D_i)$
$i = [1:n], j = [1:m], |D_i| = \lambda'$.
Output: $(E_{i,j}, F_{i,j}, G_i, H_i)$.
- for $i = 1$ to $n$
  - $r_i \leftarrow \{0,1\}^{\lambda'}$
  - $G_i \leftarrow Enc_{k_2}(D_i \oplus r_i)$
  - $H_i \leftarrow Enc_{k_3}(r_i)$
  - for $j = 1$ to $m$
    - $x_{i,j} \leftarrow Enc_{k_2}(H(k_1, w_{i,j}), p_{id_1})$
    - $y_{i,j} \leftarrow Enc_{k_3}(H(k_1, w_{i,j}), p_{id_2})$
    - $q_{i,j} \leftarrow x_{i,j} \oplus y_{i,j}$
    - $r_{i,j} \leftarrow \{0,1\}^b$
    - $E_{i,j} \leftarrow Enc_{k_4}(H(k_1, w_{i,j}), r_{i,j})$
    - $F_{i,j} \leftarrow Enc_{q_{i,j}}(E_{i,j})$
  - end
- end
DO sends $G_i, H_i, E_{i,j}, F_{i,j}$ to
  Database Server (DB).
```

Figure 4. Algorithm $Trpdr$. (Done by Data Owner (DO), $Proxy_1$ and $Proxy_2$)

```
- Input: $(H, w, K, p_{id_1}, p_{id_2})$.
- Output: $(x', y')$.
- DO: computes $H(k_1, w)$ and sends it to
    $\mathbf{Proxy_1}$ and $\mathbf{Proxy_2}$.
- $\mathbf{Proxy_1}$: $x' \leftarrow Enc_{k_2}(H(k_1, w), p_{id_1})$
- $\mathbf{Proxy_2}$: $y' \leftarrow Enc_{k_3}(H(k_1, w), p_{id_2})$
$\mathbf{Proxy_1}$ sends $x'$ to Database Server (DB).
$\mathbf{Proxy_2}$ sends $y'$ to Database Server (DB).
```

doors (tr_1, tr_2), the server

11. Creates and initializes set $C_{\mathcal{D}}^{id} = \{\}$. Then, it computes $tr = q = tr_1 \oplus tr_2$
12. For document D_i, takes the keyword ciphertexts $C_{w_{i,j}} = (F_{i,j}, E_{i,j})$, then computes $Enc_q(E_{i,j})$ and checks if $F_{i,j} == Enc_q(E_{i,j})$, where $1 \leq i \leq n$, $1 \leq j \leq m$.
13. If the equality holds at $step - 12$, the server updates the result set as $C_{\mathcal{D}}^{id} = \{i\}$. If index $i \in C_{\mathcal{D}}^{id}$, the server finds $C_{D_i} = (G_i, H_i)$. Then, the server sends G_i to $Proxy_1$ and H_i to $Proxy_2$.
14. If the equality does not hold at $step - 12$, the server follows the $steps - 12, 13, 14$ for each document $D_{i'}$'s, where $1 \leq i' \leq n$ each keyword ciphertext $w_{i',j'}$, where $1 \leq j' \leq m$. If there is no match between the trapdoor and the keyword ciphertexts, the server outputs 0.

$Search$ algorithm is given in Fig. 5.

Figure 5. Algorithm $Search$. (Done by Database Server (DB))

```
- Input: $((E_{i,j}, F_{i,j}), x', y')$.
- Output: Index set $(C_{\mathcal{D}}^{id})$.
- $C_{\mathcal{D}}^{id} = \{\}$
- $q \leftarrow x' \oplus y'$
- for $i = 1$ to $n$
  - for $j = 1$ to $m$
  - $F'_{i,j} \leftarrow Enc_q(E_{i,j})$
    - if $F_{i,j} == F'_{i,j}$
      - $C_{\mathcal{D}}^{id} = C_{\mathcal{D}}^{id} \cup \{i\}$
      -break
    - end
  - end
- end
DB sends $G_i$ to $\mathbf{Proxy_1}$ and $H_i$ to $\mathbf{Proxy_2}$,
    where $i \in C_{\mathcal{D}}^{id}$.
```

Dec$(K, C_{\mathcal{D}}^{id})$**:** Once the server sends to corresponding document ciphertexts to the proxies, each proxy decrypts and sends the plaintexts to the user. In other words, assuming that G_i is sent to $Proxy_1$ and H_i is sent to $Proxy_2$ by the database server,

15. $Proxy_1$ computes $Dec_{k_2}(G_i) = D_i \oplus r_i$. Then, it sends $par_{1,i} = D_i \oplus r_i$ to the user.
16. $Proxy_2$ computes $Dec_{k_3}(H_i) = r_i$. Then, it sends $par_{2,i} = r_i$ to the user.

Dec algorithm is given in Fig. 6.

Figure 6. Algorithm Dec. (Done by $Proxy_1$ and $Proxy_2$)

```
- Input: $(G_i, H_i, C_{\mathcal{D}}^{id}, k_2, k_3)$.
- Output: $par_{1,i}, par_{2,i}$ $(i \in C_{\mathcal{D}}^{id})$.
- Proxy₁: $par_{1,i} \leftarrow Dec_{k_2}(G_i)$,
    $(par_{1,i} = D_i \oplus r_i)$.
- Proxy₂: $par_{2,i} \leftarrow Dec_{k_3}(H_i)$,
    $(par_{2,i} = r_i)$.
Proxy₁ sends $par_{1,i}$ to DO.
Proxy₂ sends $par_{2,i}$ to DO.
```

Combine$(par_{1,i}, par_{2,i})$**:** Once the user receives the results from $Proxy_1$ and $Proxy_2$, the user

17. Computes $(par_{1,i} \oplus par_{2,i})$, where $(par_{1,i} \oplus par_{2,i}) = D_i \oplus r_i \oplus r_i = D_i$.

$Combine$ algorithm is given in Fig. 7.

Figure 7. Algorithm $Combine$. (Done by DO)

```
- Input: $(par_{1,i}, par_{2,i})$ $(i \in C_{\mathcal{D}}^{id})$
- Output: $D_i$ $(i \in C_{\mathcal{D}}^{id})$
- $D_i \leftarrow (par_{1,i} \oplus par_{2,i})$.
```

A privacy-preserving query process is shown in Fig. 8. For each query in the figure,

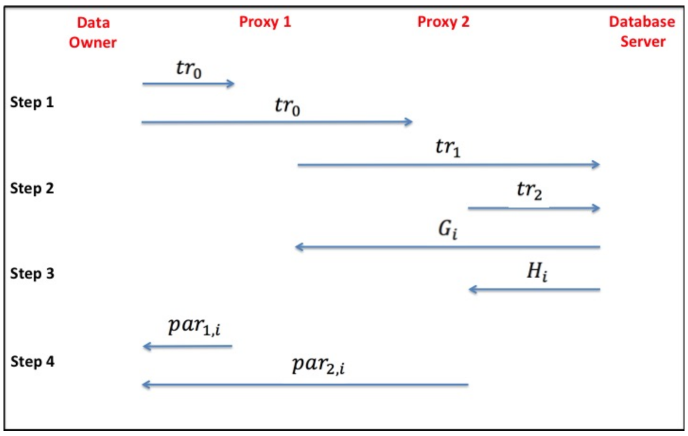

Figure 8. Data Flow in A Query Process

Step1. Data owner sends its initial trapdoor ($tr_0 = H(k_1, w)$) to both proxies.

Step2. Each proxy encrypts tr_0 with its own secret key (k_2 for $Proxy_1$ and k_3 for $Proxy_2$). Then, each proxy sends its encrypted trapdoor tr_1 ($Proxy_1$) and tr_2 ($Proxy_2$) to the database server.

Step3. The database server returns the ciphertext sets G_i and H_i that satisfy the query back to $Proxy_1$ and $Proxy_2$, where $i \in C_{\mathcal{D}}^{id}$.

Step4. Each proxy decrypts the received ciphertexts from the database using its own secret key (k_2 for $Proxy_1$ and k_3 for $Proxy_2$). Then, each proxy sends the resulting values $par_{1,i}$ and $par_{2,i}$ to the data owner, where $i \in C_{\mathcal{D}}^{id}$. At last, the data owner combines the results.

The encrypted table (T_{enc}) of Tab. 2 is shown in Tab. 3. In Table 2, the encryption of keyword $w_{1,1} \in D_1$ is represented as $\langle E_{1,1}, F_{1,1} \rangle$. Moreover, the encryption of $w_{3,3} \in D_3$ is represented as $\langle E_{3,3}, F_{3,3} \rangle$. In other words, the first index is for document number, the second index is for the keyword's location in the keyword set. Assuming that even two documents, D_1 and D_3, have the same keyword $w_{3,3} = w_{1,1} = w_1$, values $\langle E_{1,1}, F_{1,1} \rangle$ and $\langle E_{3,3}, F_{3,3} \rangle$ are different from each other. This is because the encryption algorithm uses different random value (salt) $r_{i,j}$ to encrypt keyword w_1 each time. In the encryption algorithm in $step-4$, $r_{i,j}$ is randomly chosen from set $\{0,1\}^b$ which is also know as salt. Then, $\langle E_{1,1}, F_{1,1}\rangle$ is computed in $step-4$ and $step-5$. The data owner uses random value $r_{1,1}$ to encrypt w_1 in D_1 while it uses random value $r_{3,3}$ to encrypt w_1 in D_3, where $r_{1,1} \neq r_{3,3}$. Thus, in Tab. 3, we have $E_{1,1} \neq E_{3,3}$ and $F_{1,1} \neq F_{3,3}$. As a result, the ciphertexts of the same keywords are not the same in Tab. 3. The database server is not able know if the ciphertexts contain the same keywords or not. Using different random (salt) $r_{i,j}$ values eliminates the relations between the ciphertexts. The adversary can not see the repeating ciphertexts in Tab. 3. Thus, the adversary (database server) can not do any statistical analysis for keywords.

However, some studies have this disadvantage, [7], [8], and [14]. In these studies, the adversary (database server) can do statistical attack to the database since the adversary can see the repeating ciphertexts on the database. Even the attacker does not know the content of the ciphertexts, it can do statistical analysis/frequency analysis. The server learns the frequency with which keywords appear.

Since the encryption and decryption algorithms are run by the proxies (this can be thought as secret sharing for encryption and decryption), the user does not do heavy computations. This results our scheme provides lightweight user side query and data processing different from other studies that provide heavy user side computations. Moreover, the proposed protocol eliminates single point of failure.

5.1. Complexity Analysis

In this subsection, we examine the complexity of our scheme.

Table 3.
An Illustration of An Encrypted Table (T_{enc}) ($\|$ is the concatenation operation).

C_{KW}	C_D
$\langle E_{1,1}, F_{1,1}\rangle \| \langle E_{1,2}, F_{1,2}\rangle \| \ldots \| \langle E_{1,m}, F_{1,m}\rangle$	$\langle G_1, H_1\rangle$
$\langle E_{2,1}, F_{2,1}\rangle \| \langle E_{2,2}, F_{2,2}\rangle \| \ldots \| \langle E_{2,m}, F_{2,m}\rangle$	$\langle G_2, H_2\rangle$
$\langle E_{3,1}, F_{3,1}\rangle \| \langle E_{3,2}, F_{3,2}\rangle \| \ldots \| \langle E_{3,m}, F_{3,m}\rangle$	$\langle G_3, H_3\rangle$
$\vdots \quad \vdots \quad \vdots \quad \ddots$	$\vdots$
$\langle E_{n,1}, F_{n,1}\rangle \| \langle E_{n,2}, F_{n,2}\rangle \| \ldots \| \langle E_{n,m}, F_{n,m}\rangle$	$\langle G_n, H_n\rangle$

The user (data owner) computes one hash function and $|id|$ XOR operations per query, where id is the identity set of the documents, $|id|$ is the number of identities in set id that satisfies the query. An XOR operation and computing a hash function are very fast. The encryption and decryption algorithms are done by the proxies.

The communication cost between the user and the each proxy is $a + \lambda'|id|$ bits, where $\lambda' > \lambda$ (document size is bigger than keyword size). Thus, the total communication cost is $2a + 2\lambda'|id|$ bits per query. AES block cipher can be used to implement encryption and decryption algorithms. Thus, our scheme has lightweight user side query and data computations.

Each proxy does one encryption for trapdoor generation (λ bits) and $|id|$ decryptions ($\lambda'|id|$ bits) sent from the database server. Moreover, the communication cost between each proxy and the database server is $\lambda + \lambda'|id|$ bits per query. Thus, the total communication cost between the database server and each proxy is $\lambda + \lambda'|id|$ bits.

The computation cost of the database server is one XOR operation (λ bits), mn encryptions (λmn bits total) and mn equality checks (λmn bits total), where m is the number of unique keywords in a document and n is the number of documents.

5.2. Security Analysis

The proposed scheme's security is based on security of pseudorandom permutations [26] since a block cipher (AES) is modelled as a pseudorandom permutation. Moreover, the proposed scheme's security also depends on the security of pseudorandom function since HMAC-SHA1 is modelled as a pseudo random function.

Theorem 1 *If the block cipher (encryption algorithm) is a pseudorandom permutation and H is a pseudo random function, our scheme can not leak any useful information (documents and keywords) to the adversary 1 (database server) and adversary 2 ($Proxy_2$) other than what the adversaries observe.*

Proof: Since there are two adversaries in the proposed system. Thus, we examine two cases where each party is corrupted at a time.

When the database server is corrupted: We assume that the adversary observes t encrypted queries $q_1, q_2, \ldots, q_t$, where $q_\ell = tr_{1,\ell} \oplus tr_{2,\ell}$ and $1 \leq \ell \leq t$ in the real protocol. $tr_{1,\ell}$ is $Proxy_1$'s ℓth partial trapdoor for ℓth query and $tr_{2,\ell}$ is $Proxy_2$'s ℓth partial trapdoor for ℓth query in the real protocol. During these queries, the adversary observes two kinds of leakage:

1. The encrypted database table T_{enc}, including the number of rows and columns, and their sizes (bits).
2. The positions of the responses to encrypted queries in T_{enc} (C_{KW}), $P(q_1), \ldots, P(q_t)$.

SIM takes these leakages and simulates all the transcripts in the real protocol. **SIM** does the followings for query ℓ, where $1 \leq \ell \leq t$:

- For each $E'_{i,j}, G'_i, H'_i$, where $1 \leq i \leq n, 1 \leq j \leq m$ chooses random values. $|E'_{i,j}| = \lambda$, $|G'_i| = |H'_i| = \lambda'$.
- Chooses two random $tr'_{1,\ell}$ and $tr'_{2,\ell}$, where $|tr'_{1,\ell}| = |tr'_{2,\ell}| = \lambda$.
- If $P(q_\ell) \neq \emptyset$, for each position $(i, j) \in P(q_\ell)$ in C_{KW}.
  - Computes $F'_{i,j} = Enc_{q'_\ell}(E'_{i,j})$, where $q'_\ell = tr'_{1,\ell} \oplus tr'_{2,\ell}$.
- For other positions in the table that do not satisfy any of the queries, **SIM** chooses random values for $F'_{i,j}$, where $1 \leq i \leq n$, $1 \leq j \leq m$, and $|F'_{i,j}| = \lambda$.
- Outputs $(T'_{enc}, (tr'_{1,1}, tr'_{2,1}), \ldots, (tr'_{1,t}, tr'_{2,t}))$.

The indistinguishability is preserved in $Enc()$ (pseudo random permutation) and H (pseudo random function). Since H is a pseudo random function (PRF) which is indistinguishable from a totally random function, the adversary's advantage is negligible ϵ to distinguish $H(k_1, w)$ from H', where H' is a totally random function.

When $Proxy_2$ is corrupted: This case is straightforward that $Proxy_2$ only receives random values (pseudo random values that are not distinguishable from random values) $H(k_1, w_1), H(k_1, w_2), \ldots, H(k_1, w_t)$ from the data owner, and uniformly random values r_i that satisfy the queries. $Proxy_2$ can obtain r_i values after decrypting H_i values from the database server. Since the adversary observes random values from the data owner and the database server, it is straightforward to construct a simulator.

□

Theorem 2 *If H is a pseudo random function, the proposed scheme has trapdoor privacy.*

Proof: The proof is straightforward in that it follows from the property of a pseudo-random function. Adversary $Proxy_2$ gets pseudo random values/trapdoors $H(k_1, w_i)$ in the protocol. Since the adversary can not distinguish these values from random values with non-negligible probability, the adversary is not able to extract any useful (keyword) information from the trapdoor. Moreover, a pseudo random function is also a one-way function. If an output $H(k_1, w_i)$ is given, it is infeasible to compute w_i from it.

□

6. Discussion

The previous studies, [7], [12], [13], [14], and [15], have a fully trusted proxy, the user sends plaintext queries to the proxy. The proxy converts the plaintext query to encrypted query. Then, the proxy sends the encrypted query to database. In these studies, since the proxy is fully trusted, knowing of the plaintext query does not break user query privacy. However, in our protocol, we assume that one of the proxy is compromised by the adversary. This results that the adversary controls the proxy. Since one of the proxies is controlled by the adversary, the user can not send plain queries to the proxies. Otherwise, the corrupted proxy can learn what the user wants to search over the database. This breaks the trapdoor privacy. The trapdoor privacy says that the issued keyword by the user should be hidden from the adversary. To have trapdoor privacy in our scheme, we use keyed hash function to hide the keyword that user wants to search over encrypted database.

In our scheme, even the two adversaries can collude which means that a new powerful adversary arises. This new adversary controls one proxy ($Proxy_2$) and the data server. The new adversary sees encrypted database, knows $Proxy_2$'s secret key and its identity, (k_3, p_{id_2}), sees $H(k_1, w_{i,j})$. However, the new adversary still can not learn any useful information about user's database. The new adversary can decrypt the half of the document ciphertexts. These ciphertexts are $H_i = Enc_{k_3}(r_i)$ values for document D_i. The new adversary decrypts H_i using secret key k_3 and it gets r_i (r_j for document D_j) which is a random value to the new adversary. In order to recover the documents, the new adversary should also know $D_i \oplus r_i$ ($D_j \oplus r_j$ for G_j). However, this value is not known by $Proxy_2$. Furthermore, the new adversary also observes value $H(k_1, w_{i,j})$ that is issued by the user. However, the new adversary can not recover the keyword $w_{i,j}$ from $H(k_1, w_{i,j})$ since it does not know secret key k_1.

Our scheme can easily be extended to multi-proxy, α, case where α can be the number of proxies in the system. In this case, if at most $\alpha - 1$ (at least one proxy is honest) of the proxies are controlled by adversary 2, our scheme still has trapdoor privacy and data privacy. In other words, adversary 1 and and adversary 2 can not learn any useful information about users' searched keyword and data. In multi-proxy case, the computation and communication costs are going to be linear in the number of proxies.

A note that our protocol leaks the search pattern of the user/data owner to adversaries (the database server, and $Proxy_2$). The adversary knows if given two trapdoors are generated from the same keyword or not by the data owner over time. Moreover, the database server learns which locations on the database (encrypted keywords) satisfy each query. The proxy and the database server learn the location of the encrypted documents (identities/indexes) that satisfy the query. Moreover, over time, the corrupted proxy ($Proxy_2$) figures out if the data owner issues the same keyword or not since the trapdoor/query pattern ($H(k_1, w)$) is deterministic. Even tough the proxy does not know which keyword that the data owner is interested in, the proxy figures out if the data owner wants to search the same keyword over time.

Even though all the data in the database server and each trapdoor issued by the proxies are encrypted, the adversaries (database server and $Proxy_2$) can make accurate inference on keyword plaintexts and the documents over time. This is because our protocol leaks the search pattern (trapdoor) of the data owner to the adversaries. Moreover, during the query process, the adversaries learn which encrypted trapdoors are associated with which documents.

7. Conclusion

We introduce a secure and efficient two-proxy based encrypted keyword search scheme. Our scheme eliminates the single point of failure attack that the proxy based schemes have [7], [12], [13], [14], and [15]. Moreover, our scheme uses only symmetric key encryption algorithm that provides very fast encryption and decryption computations. Furthermore, the proxies do the heavy computations on behalf of the user since trapdoor generation (encryption) and decryption computations are delegated to proxies by the data owner. The proxies do trapdoor generation and decryption computations. Trapdoor generation process has encryption computation.

For future work, we would like to use other types of queries such as range and multi-keyword queries using multi proxies. Moreover, we would like to improve my system to prevent access pattern leakage.

Acknowledgments

The author thanks the anonymous reviewers for their valuable comments and suggestions for improving the paper's quality and presentation.

References

[1] P. R. Clearinghouse. Chronology of data breaches. [Online]. Available: http://www.privacyrights.org/data-breach. July19,2022

[2] N. V. D. C. statistics. [Online]. Available: http://web.nvd.nist.gov/view/vuln/statistics,July19,2022.

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, vol. 9, 2009, pp. 169–178.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. of ACM SIGMOD*, June 2004.

[5] A. Boldyreva, N. Chenette, and A. OŃeill, "Order preserving symmetric encryption revisited: improved security analysis and alternative solutions," in *Proc. of CRYPTO*, 2011.

[6] A. Boldyreva, N. Chenette, Y. Lee, and A. OŃeil, "Order preserving symmetric encryption," in *Proc. of EUROCRYPT*, April 2009.

[7] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. of SOSP*, 2011.

[8] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Proc. of Security and Prvacy*, 2013.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*, J. Stern, Ed. Prague, Czech Republic: Springer Berlin Heidelberg, 2-6 May 1999, pp. 223–238.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[11] L. Kocarev and Z. Tasev, "Public-key encryption based on chebyshev maps," *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 3, pp. III–III, 2003.

[12] H. Hacigumus, B. Lyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database service provider model," in *Proc. of ACM SIGMOD*, June 2002, pp. 216–227.

[13] O. Oksuz, "Time-specific encrypted range query with minimum leakage disclosure," *IET Information Security*, vol. 15, no. 1, pp. 117–130, 2021.

[14] L. Zhang, O. Oksuz, L. Nazaryan, C. Yue, B. Wang, A. Kiayias, and A. Bamis, "Encrypting wireless network traces to protect user privacy: A case study for smart campus," in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, October 2016, pp. 1–8.

[15] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving queries on encrypted data," in *ESORICS*, 2006.

[16] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on*

Computer and Communications Security, ser. CCS '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 79–88.

[17] E. Goh, "Secure indexes," in *Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.*, 2003.

[18] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proc. of VLDB*, 2004.

[19] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing," *Proc. VLDB Endow.*, 2014.

[20] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.

[21] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYPT*, 2004.

[22] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography*, S. P. Vadhan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 535–554.

[23] B. Qin, Y. Chen, Q. Huang, X. Liu, and D. Zheng, "Public-key authenticated encryption with keyword search revisited: Security model and constructions," *Information Sciences*, vol. 516, pp. 515–528, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519311752

[24] Y. Yang, X. Liu, and R. Deng, "Expressive query over outsourced encrypted data," *Information Sciences*, vol. 442-443, pp. 33–53, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025518300987

[25] N. S. Joshi, R. Raghuwanshi, and B. R. Chandavarkar, "Computational complexity analysis of block ciphers of transport layer security," in *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2021.

[26] S. Goldwasser and M. Bellare, "Lecture notes on cryptography. summer course lecture notes at mit," 1999.