Multi-User Searchable Encryption in the Cloud

Cédric Van Rompay, Refik Molva, and Melek Önen

EURECOM, Sophia Antipolis, France,
{vanrompa,molva,onen}@eurecom.fr

Abstract. While Searchable Encryption (SE) has been widely studied, adapting it to the multi-user setting whereby many users can upload secret files or documents and delegate search operations to multiple other users still remains an interesting problem. In this paper we show that the adversarial models used in existing multi-user searchable encryption solutions are not realistic as they implicitly require that the cloud service provider cannot collude with some users. We then propose a stronger adversarial model, and propose a construction which is both practical and provably secure in this new model. The new solution combines the use of bilinear pairings with private information retrieval and introduces a new, non trusted entity called "proxy" to transform each user's search query into one instance per targeted file or document.

1 Introduction

Cloud computing nowadays appears to be the most prominent approach for outsourcing storage and computation. Despite well known advantages in terms of cost reduction and efficiency, cloud computing also raises various security and privacy issues. Apart from classical exposures due to third party intruders one of the new requirements akin to outsourcing is the privacy of outsourced data in the face of a potentially malicious or careless Cloud Service Provider (CSP).

While data encryption seems to be the right countermeasure to prevent privacy violations, classical encryption mechanisms fall short of meeting the privacy requirements in the cloud setting. Typical cloud storage systems also provide basic operations on stored data such as statistical data analysis, logging and searching and these operations would not be feasible if the data were encrypted using classical encryption algorithms.

Among various solutions aiming at designing operations that would be compatible with data encryption, Searchable Encryption (SE) schemes allow a potentially curious party to perform searches on encrypted data without having to decrypt it. SE seems a suitable approach to solve the data privacy problem in the cloud setting.

A further challenge is raised by SE in the multi-user setting, whereby each user may have access to a set of encrypted data segments stored by a number of different users. Multi-user searchable encryption schemes allow a user to search through several data segments based on some search rights granted by the owners of those segments. Privacy requirements in this setting are manifold, not only the confidentiality of the data segments but also the privacy of the queries should be assured against intruders and potentially malicious CSP. Recently, few research efforts [5, 8, 12, 15] came up with multi-user keyword search schemes meeting these privacy requirements, either through some key sharing among users or based on a Trusted Third Party (TTP).

In this paper, we first investigate the new privacy challenges for keyword search raised by the multi-user setting beyond the basic privacy concerns about data, queries and responses by focusing on the relationship among multiple queries and responses. We realize that while as analyzed in [7], the protection of the *access pattern privacy* (privacy of the responses) is optional for single-user searchable encryption mechanisms, this requirement becomes mandatory in the multi-user setting. Unfortunately all existing Multi-User Searchable Encryption (MUSE) schemes [5,8,12,15] suffer from the lack of such protection. We further come up with a new adversary model for MUSE that takes into account new security exposures introduced by the possible collusion of some users with the CSP.

After showing that all existing MUSE schemes fail at meeting the privacy requirements in our new adversarial model, we suggest a new solution for MUSE for which it is not the case, i.e., all users who have not been explicitly authorized to search a document can collude with the adversary without threatening the privacy of that document. Our solution for MUSE inherently ensures access pattern privacy through the use of Private Information Retrieval (PIR). While the PIR protocol together with the multi-user setting may add a significant complexity overhead, this overhead is outsourced from the users to a third party our scheme introduces, the *proxy*, that is in charge of multiplexing a user query into several PIR queries. Moreover the overhead of PIR is further lowered by querying binary matrices representing the keyword indices instead of querying the bulky keyword lists themselves. As opposed to most existing solutions based on a TTP [3,5,8,15], the proxy in our scheme does not need to be trusted. With the sole assumptions that the CSP and the proxy are honest-but-curious and that they do not collude with one another, we prove that our solution meets the privacy requirements defined for MUSE.

Section 2 states the problem addressed by MUSE. Section 3 describes our solution for MUSE and Section 4 defines the security properties for MUSE. Section 5 proves that our solution achieves the security properties we defined and Section 6 studies the algorithmic complexity of our solution. Section 7 reviews the state of the art and, finally, Section 8 concludes the paper.

2 Multi-User Searchable Encryption (MUSE)

A MUSE mechanism extends existing keyword search solutions into a multi-writer multi-reader [6] architecture involving a large number of users, each of which having two roles:

- as a *writer*, the user uploads documents to the server and delegates keyword search rights to other users.

– as a *reader*, the user performs keyword search operations on the documents for which she received delegated rights.

As any SE solution, MUSE raises two privacy requirements:

– *index privacy*: unauthorized parties should not discover information about the content of uploaded documents.
– *query privacy*: no one should get information about the targeted word and the result of a search operation apart from the reader who sent the corresponding query.

In addition to the CSP, any user that has not been explicitly given search rights on an index should be considered as potentially colluding with the CSP in order to violate index or query privacy. This assumption leads to a model in which the adversary is composed of a coalition of the CSP and some non-delegated users. This new adversary model extends the one used in other existing MUSE schemes [5,7,12,15], which although secure in their own adversary model do not achieve index and query privacy any more if non-delegated users collude with the CSP.

Figure 1 illustrates one example of the impact of a collusion between a CSP and a user on privacy by taking advantage of the lack of access pattern privacy. Assuming that R_1 is authorized to query both indices I_1 and I_2, by observing the access pattern of R_1's queries, the CSP can discover similarities between I_a and I_b. In a second phase, the CSP corrupts reader R_2 who is authorized to query I_b only. By exploiting the similarities between I_a and I_b and discovering the content of I_b through R_2, the CSP can easily discover the content of I_a. The index privacy is thus violated for I_a since the CSP partially learns the content of I_a although R_1, the only reader having delegated search rights for I_a, was not corrupted. Furthermore, once the CSP obtains information about the content of an index, observing the access pattern of the queries targeting this index enables the CSP to violate the privacy of these queries. This attack allows to violate both query and index privacy in all existing MUSE schemes since they all let the CSP discover the access pattern of the queries. The new adversary model we introduce not only prevents such an attack but also prevents any attack that would require the corruption of a non-delegated user.

3 Our Solution

3.1 Idea

Our solution introduces a third party called the *proxy* that performs an algorithm called *QueryTransform* to transform a single reader query into one query per targeted document [1]. For each of these queries, the proxy sends to the CSP a specific

[1] Note that the set of targeted document can reveal the authorized set of documents for this particular user. However, such an additional information does not have a serious impact on index or query privacy as access pattern leakage has.

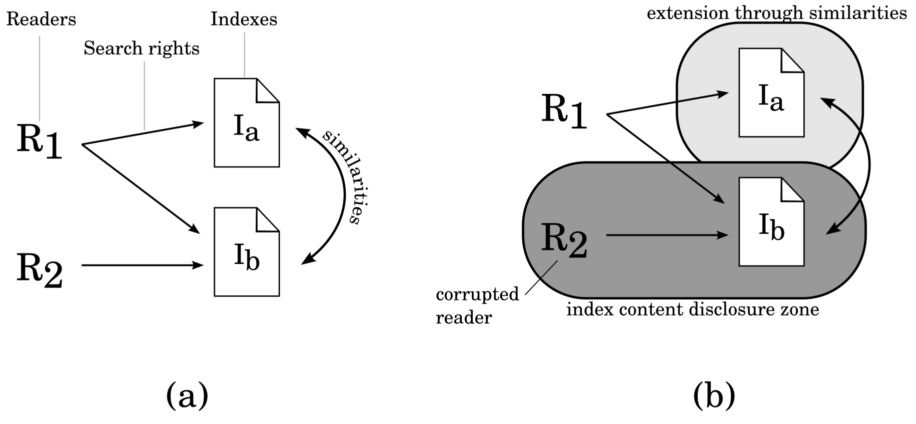

Fig. 1. In (a), discovery of similarities through the access pattern. In (b), use of the similarities to extend index privacy violation.

PIR request. Thanks to the PIR protocol the CSP does not have access neither to the content of the query nor to its result, which makes our scheme achieving query privacy (including access pattern privacy) against the CSP. While the use of PIR provides privacy against the CSP, a new privacy exposure raises with respect to the proxy. Indeed through the execution of *QueryTransform*, the proxy is able to discover the relationship between a query and the different ciphertexts in the targeted indices which are the encryption of the same keyword. However with the assumption that the proxy does not collude with the CSP, the proxy cannot realize whether these ciphertexts are present in their respective indices or not; thus, our scheme achieves index privacy against the proxy. Moreover thanks to some randomization of the queries and the encryption of the responses by the CSP with the reader's key, the proposed solution also ensures query privacy against the proxy. Consequently, while our solution does introduce a third party (the proxy), **this third party does not need to be trusted** and is considered as an adversary. Both the CSP and the proxy are then considered as potentially malicious in our scheme, and are only assumed *honest-but-curious* and non colluding with each other.

Another advantage of introducing the proxy into this new MUSE solution is scalability: Indeed, thanks to the $QueryTransform$ algorithm executed by the proxy a user does not need to generate several PIR queries (one per index) for the same keyword.

3.2 Preliminaries

Bilinear Pairings Let G_1, G_2 and G_T be three groups of prime order q and g_1, g_2 generators of G_1 and G_2 respectively. $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear map if e is:

- efficiently computable
- non-degenerate: if x_1 generates G_1 and x_2 generates G_2, then $e(x_1, x_2)$ generates G_T
- bilinear: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}\ \forall (a, b) \in \mathbb{Z}^2$

We assume that the widely used eXternal Diffie-Hellman (XDH) assumption [4] holds.

Definition 1 (External Diffie Hellman assumption). *Given three groups G_1, G_2 and G_T and a bilinear map $e : G_1 \times G_2 \to G_T$, the Decisional Diffie-Hellman (DDH) problem is hard in G_1, i.e., given $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta) \in G_1^4$, it is computationally hard to tell if $\delta = \alpha\beta$.*

Private Information Retrieval (PIR) A PIR protocol allows a user to retrieve data from a database without revealing any information about the retrieved data.

PIR consists of five algorithms:

- **PIR.Setup**$() \to PIRParams$
- **PIR.KeyGen**$() \to (PirKey)$: this algorithm outputs the keying material for PIR.
- **PIR.Query**$(PirKey, size, target) \to Query$: given PIR parameters, the size of the targeted database and a target position, this algorithm outputs a PIR query targeting the given position in a database of the given size.
- **PIR.Process**$(Query, DataBase) \to R$: this algorithm applies the query $Query$ on the database $DataBase$ and outputs a response R.
- **PIR.Retrieve**$(R, PirKey) \to Cell$: given a PIR response R and the PIR key used in corresponding query, this algorithm outputs the value of the retrieved database cell.

Single-database computational PIR has already been widely studied [1,2,10, 11], and the results presented in [1] show that solutions with practical performances already exist. Our solution uses the technique of recursive PIR which allows to reduce communication complexity as explained in [1]: The database is viewed as a matrix each row of which is considered as a sub-database. To query the whole database a single query is sent and this query is further applied on each row, resulting in the generation of many PIR responses.

3.3 Protocol Description

Figure 2 illustrates the structure and the various flows of our solution. We define two phases in the protocol, the *upload phase* and the *search phase*: During the upload phase, a writer A uploads a secure index to the CSP by encrypting each keyword with the *Index* algorithm. A then delegates search rights to reader B using the *Delegate* algorithm which computes an authorization token using B's public key and A's private key. The authorization token is sent to the proxy. During the search phase, B can further search all the indices for which she has

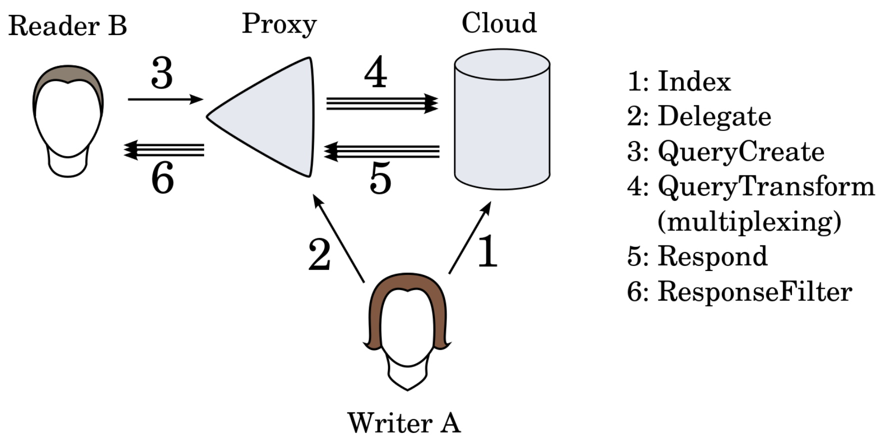

Fig. 2. Overview of our solution.

been given search rights, by creating a single query through the execution of $QueryCreate$. Whenever the proxy receives the B's query it uses the authorization tokens attributed to B to transform this query into one PIR query per authorized index through the execution of $QueryTransform$. Upon reception of a PIR query, the CSP through the execution of *Respond* builds a binary matrix using the corresponding encrypted index, applies the query to the matrix and encrypts the resulting PIR answers. The responses are then pre-processed by the proxy through the execution of $ResponseFilter$. Finally B obtains the result of her search query by executing $ResponseProcess$.

Revocation in our solution only consists in the deletion of the appropriate authorizations by the proxy upon a writer's request.

The set of users is denoted by $u_{i 1 \leq i \leq N}$. For the sake of clarity, each user u_i is assumed to own only one index I_i.

- **Setup**$(\kappa) \rightarrow$ *params*: given the security parameter κ, this algorithm outputs the parameters *param* consisting in:
  - a description of the bilinear map that will be used: the three groups G_1, G_2, G_T of prime order q, the two generators g_1 and g_2 and the map itself e.
  - a cryptographically secure hash function $h: \{0,1\}^* \rightarrow G_1$
  - the size n of the matrices for PIR, and a hash function $H : G_T \rightarrow [\![0, n-1]\!]$ to transform encrypted keywords into positions in the matrices. Without loss of generality, n is assumed to be a perfect square.
  - the PIR parameters $PIRParams$ from the execution of $PIR.Setup$
  - a symmetric cipher *Enc* and the corresponding decipher algorithm *Dec*.

  All these parameters are considered implicit for each further algorithm.
- **KeyGen**$(\kappa) \rightarrow (\gamma, \rho, P, K)$: given the security parameter κ, a user u_i generates the following keys:

  - a *secret writer key* $\gamma_i \xleftarrow{\$} \mathbb{Z}_q^*$
  - a *private reader key* $\rho_i \xleftarrow{\$} \mathbb{Z}_q^*$
  - a *public reader key* $P_i = g_2^{\frac{1}{\rho}}$
  - a *transmission key* K_i used for Enc/Dec. This key is shared with the CSP.
- **Index**$(w, \gamma_i) \rightarrow \tilde{w}$: Writer u_i executes this algorithm to encrypt keyword w with his key γ_i. The algorithm outputs $\tilde{w} = e(h(w)^{\gamma_i}, g_2)$.
- **Delegate**$(\gamma_i, P_j) \rightarrow \Delta_{i,j}$: Provided with the public key P_j of reader u_j, writer u_i executes this algorithm using its secret key γ_i to generate $\Delta_{i,j} = P_j^{\gamma_i}$ the authorization token that authorizes u_j to search the index I_i. The output $\Delta_{i,j}$ is sent to the proxy which adds it to the set D_j. Note that this token can only be created by the legitimate data owner and cannot be forged by any other party including the CSP and the proxy.
- **QueryCreate**$(w, \rho_j) \rightarrow Q_j$: This algorithm is run by an authorized reader to generate a query for keyword w using its private reader key ρ_j. The algorithm draws a randomization factor $\xi \xleftarrow{\$} \mathbb{Z}_q^*$ and outputs $Q_j = h(w)^{\xi \rho_j}$.
- **QueryTransform**$(Q_j, D_j) \rightarrow < Q'_{i,j} >$: Whenever the proxy receives a reader's query Q, it calls this algorithm together with the set D_j.
  For each authorization token $\Delta_{i,j}$ in D, the algorithm creates a PIR query Q'_j as follow:
  - compute $\tilde{Q}_{i,j} \leftarrow e(Q_j, \Delta_{i,j})$
  - compute $x'||y' \leftarrow H(\tilde{Q}_{i,j})$
  - some PIR keying material is generated: $PirKey \leftarrow PIR.KeyGen()$
  - a $\sqrt{n}$-size PIR query is created that targets position y': $Q'_{i,j} \leftarrow PIR.Query(PirKey, \sqrt{n}, y')$

  The algorithm outputs $< Q'_{i,j} >$ which are forwarded to the CSP together with the corresponding identifiers i of the indices. The proxy additionally stores each generated $PIRKey$ and x' in a table in order to use them upon reception of the corresponding response.
- **Respond**$(Q', I, \xi) \rightarrow R$: Whenever the CSP receives an individual PIR query Q', it executes this algorithm using the corresponding index I and the randomization factor ξ corresponding to this query.
  The algorithm initializes a $\sqrt{n} \times \sqrt{n}$ matrix M with "0". Then for each encrypted word $\tilde{w} \in I$, the cell $M_{x,y}$ is set to "1" where $x||y \leftarrow H(\tilde{w}^{\xi})$ (recall that $\tilde{w} \in G_T$). The response is the tuple of the outputs from the application of the PIR query Q' on each row of the binary matrix M: $\tilde{R} \leftarrow (PIR.Process(Q', M_x) \mid M_x \text{ a row of } M)$. Each component of $\tilde{R}$ is then encrypted with algorithm Enc using the transmission key K of the querying reader to obtain R which the algorithm outputs. This layer of encryption prevents the proxy from reading the result of the query.
- **ResponseFilter**$(R, x', PirKey) \rightarrow (R', PirKey)$: Whenever the proxy receives a response R it calls this algorithm together with the x' and $PirKey$ associated to the corresponding query. The purpose of this algorithm is to reduce the communication cost for the reader. Indeed the algorithm extracts

the x'-th component of R and outputs it together with the value for $PirKey$. This results in a filtered response which is much smaller than the original response.
- **ResponseProcess**$(R', PirKey, K) \rightarrow b \in \{0,1\}$: On receiving the filtered response R' with the corresponding $PirKey$, the reader executes this algorithm using her transmission key K. The algorithm further outputs the value of $PIR.Retrieve(Dec_K(R'), PirKey)$ which corresponds to the content of the retrieved matrix cell. An output of 1 means that the searched keyword is present in the index, and a 0 means that it is absent.

3.4 Correctness

We now show that a query correctly retrieves a particular cell which content corresponds to whether the queried keyword has been uploaded or not.

Let γ be the encryption key of a given index. If keyword w has been uploaded to that index, then the cell $M_{x,y}$ of the corresponding matrix is equal to 1 with $x||y = H(e(h(w), g_2^{\gamma}))$. Conversely if a given cell $M_{x,y}$ is equal to 1 then with high probability the corresponding keyword w where $x||y = H(e(h(w), g_2^{\gamma}))$ has been uploaded. A false positive implies a collision in either H or h. Thus the content of $M_{x,y}$ corresponds to the upload of w.

Secondly, a query for keyword w in that index will retrieve cell $M_{x',y'}$ with:

$$x'||y' = H(e(h(w)^{\rho}, g_2^{\frac{\gamma}{\rho}})) = H(e(h(w), g_2^{\gamma})) = x||y \ . \tag{1}$$

Thus a response to a query will decrypt to the content of the proper cell and our scheme is correct.

4 Security Model

Our security definitions are game-based definitions, where the games are represented by algorithms. Since the CSP and the proxy are considered as two non-colluding adversaries, security will be defined for each of them independently. The consequence of the non-collusion assumption is that each adversary will see the other one as an oracle. For each adversary type we define one game for index privacy and one game for query privacy. For each definition, the corresponding game consists of seven phases: a setup phase, a learning phase, a challenge phase, a restriction phase, second learning and restriction phases identical to the previous ones, and finally a response phase. The adversary is denoted by $\mathcal{A}$.

4.1 Security with the CSP as Adversary

We now formally define *Index Privacy* and *Query Privacy* considering the CSP as the adversary. In the following two definitions the setup and the learning phases are the same and are described in Alg. 1. The challenge and restriction phases for index privacy are further described in Alg. 2 and the ones for query privacy are described in Alg. 3. Finally during the *response phase*, $\mathcal{A}$ outputs a bit b^* representing its guess for the challenge bit b.

```
/* Setup phase                                                    */
$\mathcal{A} \leftarrow$ Setup();
for $i = 1$ to $N$ do
    $(\gamma_i, \rho_i, P_i, K_i) \leftarrow KeyGen(\kappa)$ ;
    $\mathcal{A} \leftarrow (i, P_i, K_i)$ ;
end
/* First learning phase                                           */
for $j = 1$ to a polynomial number $l_1$ do
    $\mathcal{A} \rightarrow query$ ;
    switch query do
        case Index for word $w$ and user $u_i$
            $\mathcal{A} \leftarrow Index(w, u_i)$;
        case Corrupt user $u_i$
            $\mathcal{A} \leftarrow (\rho_i, \gamma_i, K_i)$
        case Delegation of user $u_i$ by user $u_j$
            /* $\mathcal{A}$ does not receive any value, but the delegation will
               modify the set $D_i$ used in $QueryTransform$            */
        end
        case Queries for word $w$ from user $u_i$
            /* $D_i$ comes from the Delegations queried by $\mathcal{A}$          */
            $\mathcal{A} \leftarrow QueryTransform(QueryCreate(w, \rho_i), D_i)$;
            /* $\mathcal{A}$ also receives the randomization factor $\xi$         */
            $\mathcal{A} \leftarrow \xi$;
        case Queries for user query $Q$ from corrupted user $u_i$
            $\mathcal{A} \leftarrow QueryTransform(Q, D_i)$;
        case Filtered response for response $R$ from corrupted user $u_i$
            $\mathcal{A} \leftarrow ResponseFilter(R)$
    endsw
end
```

Algorithm 1: Setup and learning phases of both index privacy and query privacy games, whereby $\mathcal{A}$ is the CSP

```
/* Challenge phase                                                */
$\mathcal{A} \rightarrow (u_{chall}, w_0^*, w_1^*)$;
$b \xleftarrow{\$} \{0, 1\}$;
$\mathcal{A} \leftarrow Index(w_b^*, u_{chall})$;
/* Restriction phase                                              */
if $u_{chall}$ is corrupted OR Index for $w_0^*$ or $w_1^*$ for user $u_{chall}$ has been previously
queried OR a corrupted user has been delegated by $u_{chall}$ then
    HALT;
end
```

Algorithm 2: Challenge and restriction phases of the index privacy game whereby $\mathcal{A}$ is the CSP

```
/* Challenge phase                                                    */
$\mathcal{A} \rightarrow (u_{chall}, w_0^*, w_1^*)$;
$b \overset{\$}{\leftarrow} \{0,1\}$;
$\mathcal{A} \leftarrow QueryTransform(QueryCreate(w_b^*, \rho_{chall}), D_{chall})$;
/* Restriction phase                                                  */
if $u_{chall}$ is corrupted then
    HALT;
end
```

Algorithm 3: Challenge and restriction phases of the query privacy game whereby $\mathcal{A}$ is the CSP

Definition 2 (Index Privacy Against the CSP). *We say that a MUSE scheme achieves index privacy against the CSP when the following holds for the index privacy game (Alg. 1 and Alg. 2): $|Pr[b = b^*] - \frac{1}{2}| \leq \epsilon$, with ϵ a negligible function in the security parameter κ.*

Definition 3 (Query Privacy Against the CSP). *We say that a MUSE scheme achieves query privacy against the CSP when the following holds for the query privacy game (Alg. 1 and Alg. 3): $|Pr[b = b^*] - \frac{1}{2}| \leq \epsilon$, with ϵ a negligible function in the security parameter κ.*

4.2 Security with the Proxy as Adversary

Due to space limitations we do not provide the detailed description of index and query privacy games whereby the proxy is considered as the adversary. In a nutshell, the main differences with the previous games are the following:

– during the learning phase the proxy can query for the *Respond* algorithm executed by the CSP, but does not query for the $QueryTransform$ and $ResponseFilter$ algorithms. Moreover the proxy receives the output of the *Delegate* algorithm, but does not get the transmission key and the randomization factors of the users.
– during the challenge phase, the proxy does not receive the output of the $Index$ algorithm for index privacy, and receives the output of $QueryCreate$ for query privacy.

5 Security Analysis

Inspired by the methodology in [14], in order to prove each security property we define a sequence of games $(game_i)_{i=0..n}$, the first game being the original security definition. For each game $game_i$ a "success event" S_i is defined as the event when the adversary $\mathcal{A}_i$ correctly guesses the challenge bit b used as part of the challenge. For every two consecutive games $game_i$ and $game_{i+1}$, it is shown that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible. Then it is shown that the probability of success $Pr[S_n]$ of the last game is the target probability, namely 0.5. Hence the probability of success of the first game is negligibly close to the target probability, which ends the proof.

Due to space limitations we provide a detailed proof for index privacy against the CSP only.

5.1 Index Privacy with the CSP as the Adversary

Theorem 1. *Our construction achieves index privacy against the CSP.*

***game*$_0$** Let $game_0$ be the game of Definition 2 (Alg. 1 and Alg. 2). The success event S_0 is "$b = b^*$".

***game*$_1$** The only difference between $game_0$ and $game_1$ is that in $game_1$, the adversary $\mathcal{A}_1$ can no longer send queries requesting the corruption of a user. Consequently $\mathcal{A}_1$ can neither send queries related to corrupted users, namely queries for $QueryTransform$ and $ResponseFilter$.

Lemma 1. *If* $\Pr[S_1]$ *is negligibly close to* 0.5*, then* $\Pr[S_1]$ *and* $\Pr[S_0]$ *are negligibly close.*

Proof. This Lemma is proved by introducing an adversary $\mathcal{A}_1$ executing the algorithm depicted in Alg. 4.

$\mathcal{A}_1$ plays $game_1$ using adversary $\mathcal{A}_0$ playing $game_0$. To that effect, $\mathcal{A}_1$ simulates an instance of $game_0$ with respect to $\mathcal{A}_0$ and responds at $game_1$ using the response of $\mathcal{A}_0$. Since, as opposed to $\mathcal{A}_0$, $\mathcal{A}_1$ cannot corrupt any user, $\mathcal{A}_1$ has to fabricate responses to $\mathcal{A}_0$'s corruption queries as part of the simulated instance of $game_0$. To do so, $\mathcal{A}_1$ simulates corrupted users by locally generating keys which are sent to $\mathcal{A}_0$ as a response to the corruption query. These same generated keys must be used in all responses related to this corrupted user in order for $\mathcal{A}_1$ to simulate a consistent instance of $game_0$. However $\mathcal{A}_0$ may have sent queries related to this user before the corruption query. A way for $\mathcal{A}_1$ to ensure the required consistency is to choose a set of users that will be simulated from the beginning. If $\mathcal{A}_0$ sends a request to corrupt a user that $\mathcal{A}_1$ chose not to simulate, $\mathcal{A}_1$ cannot simulate a proper instance of $game_0$ any more. Simulation also fails if a user that was simulated by $\mathcal{A}_1$ is chosen by $\mathcal{A}_0$ to be the challenge user or a delegate of the challenge user. We define the event C as when none of the previous cases occur, i.e., C is "$\mathcal{A}_0$ does not corrupt any non-simulated user and $\mathcal{A}_0$ does not chose any simulated user as either the challenge user or a delegate of the challenge user". We also define the event C' as "all users but the challenge user and her delegates are simulated". Since C' implies C we have $Pr[C] \geq Pr[C']$, and actually $Pr[C]$ is expected to be much greater than $Pr[C']$. Whenever the event C occurs, $\mathcal{A}_0$ received a valid instance of $game_0$ with the challenge value from the instance of $game_1$, and thus the probability for $\mathcal{A}_1$ to succeed at $game_1$ is the probability of $\mathcal{A}_0$ to succeed at $game_0$:

$$Pr[S_1|C] = Pr[S_0] \ . \tag{2}$$

```
$\mathcal{A}_1$ receives data from $game_1$ setup phase;
/* $\mathcal{A}_1$ simulates some users to $\mathcal{A}_0$                         */
$Sim \overset{\$}{\leftarrow} \mathcal{P}([1..N])$;
for $i \in Sim$ do
    $(\gamma'_i, \rho'_i, P'_i, K'_i) \leftarrow KeyGen(\kappa)$
end
for $i$ from 1 to $N$ do
    if $i \in Sim$ then
        $\mathcal{A}_0 \leftarrow (i, P'_i, K'_i)$;
    else
        $\mathcal{A}_0 \leftarrow (i, P_i, K_i)$
    end
end
/* Learning phase 1                                                 */
for a polynomial number $l_1$ of times do
    $\mathcal{A}_0 \rightarrow$ query;
    if $\mathcal{A}_1$ knows all the input values for the corresponding algorithm then
        $\mathcal{A}_1$ runs the algorithm locally and sends back the answer;
    else
        if query was for corruption then
            /* exit with random guess                               */
            $b^* \overset{\$}{\leftarrow} 0, 1$;
            $\mathcal{A}_1 \rightarrow b^*$;
            HALT;
        else
            $\mathcal{A}_1$ forwards the call to $game_1$ and forwards the answer to $\mathcal{A}_0$;
        end
    end
end
/* Challenge phase                                                  */
$\mathcal{A}_1$ forwards everything from $\mathcal{A}_0$ to $game_1$ and back.
/* Learning phase 2                                                 */
Same as learning phase 1;
/* Response phase                                                   */
$\mathcal{A}_1$ forwards the bit $b^*$ outputted by $\mathcal{A}_0$;
```

Algorithm 4: Algorithm run by $\mathcal{A}_1$ the transition adversary from $game_0$ to $game_1$. Restrictions phases are omitted.

If the simulation of $game_0$ fails, $\mathcal{A}_1$ can still give a random answer to $game_1$ which implies:

$$Pr[S_1|\neg C] = 0.5\ . \tag{3}$$

Finally we define the event C'_i as "user u_i is either simulated or challenge-or-delegate, but not both". We have $Pr[C'_i] = 0.5$ and $Pr[C'] = \prod_{i=1..N} Pr[C'_i]$ thus $Pr[C'] = 2^{-N}$ and it follows that $Pr[C] \geq 2^{-N}$. It seems reasonable to assume that the number N of users grows at most polylogarithmically with the security parameter κ, which implies that $Pr[C]$ is non-negligible:

$$\exists p \text{ polynomial in } \kappa,\ \frac{1}{Pr[C]} \leq p\ . \tag{4}$$

Then the following holds:

$$\begin{aligned}
Pr[S_1] &= Pr[S_1|C].Pr[C] + Pr[S_1|\neg C].Pr[\neg C] \\
Pr[S_1] &= Pr[S_0].Pr[C] + 0.5(1 - Pr[C]) \\
Pr[S_1] &= Pr[C].\,(Pr[S_0] - 0.5) + 0.5 \\
Pr[S_0] &= 0.5 + \frac{1}{Pr[C]}\,(Pr[S_1] - 0.5) \\
Pr[S_0] - Pr[S_1] &= (0.5 - Pr[S_1])\left(1 - \frac{1}{Pr[C]}\right)\ .
\end{aligned}$$

Then from (4) we have that if $(0.5 - Pr[S_1])$ is negligible then $|Pr[S_0] - Pr[S_1]|$ is negligible also. This conclude the proof of Lemma 1.

***game*$_2$** In $game_2$, calls to $QueryCreate$ are replaced by the generation of random bits.

Lemma 2. $\Pr[S_2]$ *is negligibly close to* $\Pr[S_1]$.

Proof. Distinguishing between $game_1$ and $game_2$ is equivalent to breaking the security of the encryption scheme used in the PIR construction. This holds because corruption is not allowed in $game_1$, and hence the adversary cannot obtain the required PIR parameters to open the PIR query. It follows that Lemma 2 is true.

***game*$_3$** In $game_3$, the call to $Index$ in the challenge phase is replaced by picking a random element in G_T.

Lemma 3. $\Pr[S_3]$ *is negligibly close to* $\Pr[S_2]$.

Proof. To prove this Lemma we build a distinguishing algorithm $\mathcal{D}_{DDH}$, described in Alg. 5, which uses a $game_2$ adversary $\mathcal{A}_2$ and which advantage at the DDH game is :

$$\epsilon_{DDH} = O(\frac{1}{Nl})|\Pr[S_3] - \Pr[S_2]|\ . \tag{5}$$

Given the DDH problem instance $(g_1, g_1^\alpha, g_1^\beta, g_1^\delta) \in G_1^4$, the intuition behind algorithm $\mathcal{D}_{DDH}$ is to "put" β in the challenge word, α in the challenge user key, and δ in the value given to $\mathcal{A}_2$ during the challenge phase. $\mathcal{D}_{DDH}$ makes some predictions on the queries of $\mathcal{A}_2$, namely on the user $\mathcal{A}_2$ will choose as the challenge user and on the moment $\mathcal{A}_2$ will call the hash function h on the

```
$\mathcal{D}_{DDH} \leftarrow (g_1, g_1^{\alpha}, g_1^{\beta}, g_1^{\delta})$;
$\mathcal{A} \leftarrow Setup()$;
$predict \xleftarrow{\$} [1..N]$;
$I \xleftarrow{\$} [0,..,l]$;
for $i$ from 1 to $N$ do
    $(\gamma_i, \rho_i, P_i, K_i) \leftarrow KeyGen(\kappa)$;
    $\mathcal{A} \leftarrow (i, P_i, K_i)$
end
for a polynomial number $l$ of times do
    $\mathcal{A} \rightarrow$ query;
    switch query do
        case hash of word $w$ through $h$
            if this is the $I$-th call to $\mathcal{O}$ then
                $\mathcal{A} \leftarrow g_1^{\beta}$
            else
                $\mathcal{A} \leftarrow g_1^{\mathcal{O}[w]}$
            end
        case Index for word $w$ and user $u_{predict}$
            $\mathcal{A} \leftarrow e((g_1^{\alpha})^{\mathcal{O}[w]}, g_2)$;
        otherwise
            normal handling of the query;
        end
    endsw
end
$\mathcal{A} \rightarrow (u_{chall}, w_0^*, w_1^*)$;
$b \xleftarrow{\$} \{0,1\}$;
if $chall \neq predict$ OR $I = 0$ and $\mathcal{O}$ has been called with input $w_b^*$ OR $I \neq 0$ and
$w_b^*$ does not correspond to the $I$-th call to $\mathcal{O}$ then
    $b_{DDH} \xleftarrow{\$} 0,1$;
    $\mathcal{D}_{DDH} \rightarrow b_{DDH}$;
    HALT;
end
$\mathcal{A} \leftarrow e(g_1^{\delta}, g_2)$;
$\mathcal{A} \rightarrow b^*$;
if $b^* = b$ then
    $\mathcal{D}_{DDH} \rightarrow 1$;
else
    $\mathcal{D}_{DDH} \rightarrow 0$;
end
```

Algorithm 5: Listing for the distinguishing algorithm $\mathcal{D}_{DDH}$ from $game_2$ to $game_3$.

challenge word. If these predictions prove false $\mathcal{D}_{DDH}$ sends a random answer to the DDH problem. Otherwise if the predictions prove true $\mathcal{D}_{DDH}$ outputs 1 if $\mathcal{A}_2$ wins the game and 0 if not. If the correct answer to the DDH game was 1 then

$\mathcal{A}_2$ was playing $game_2$ and $\mathcal{D}_{DDH}$ outputs 1 with probability $Pr[S_2]$. Similarly if the answer to DDH was 0 $\mathcal{D}_{DDH}$ outputs 1 with probability $Pr[S_3]$ During the whole game $\mathcal{D}_{DDH}$ simulates the hash function h as a random oracle, using $\mathcal{O}$ which behaves the following way: if $\mathcal{O}$ has stored a value for keyword w, $\mathcal{O}[w]$ returns this value; else it returns a random value and stores it for future queries.

The following variable change shows that if the predictions prove right, the adversary received a proper game instance:

$$\alpha \leftrightarrow \gamma_{chall}, \; g_1^{\beta} \leftrightarrow h(w_b^*) \; . \tag{6}$$

The probability that the predictions were correct is clearly non-negligible: $\Pr[u_{predict} = u_{chall}] = 1/N$ and the probability that predicted I is correct is $O(1/l)$, N and l being at most polynomial in the security parameter κ.

Finally from the XDH assumption, DDH is a hard problem in G_1. Thus ϵ_{DDH} is negligible in κ. Given that N and l are at most polynomial in κ and from (5), we have that $|\Pr[S_3] - \Pr[S_2]|$ is negligible which concludes the proof of Lemma 3.

Proof of Theorem 1. In $game_3$ the adversary does not receive any value which depends on the challenge bit, so $\Pr[S_3] = 0.5$. Then Lemma 3 implies that $Pr[S_2]$ is negligibly close to 0.5, Lemma 2 implies that $Pr[S_1]$ is negligibly close to 0.5 and finally Lemma 1 implies that $Pr[S_0]$ is negligibly close to 0.5. This concludes the proof of Theorem 1.

6 Performance Analysis

During the upload phase, the cost for a user of running the *Index* algorithm over the entire index is naturally linear towards the number of keywords in the index. The most costly operation within the *Index* algorithm is one pairing computation; however since inside a same index the second argument of the pairing remains the same between two executions of *Index*, pairing becomes much more efficient than in the case with independent pairings [13].

Furthermore, the *Delegate* algorithm only consists in one exponentiation.

As the search phase involves three parties, namely the reader, the proxy and the CSP, we evaluate the computational cost for each of them.

The *QueryCreate* algorithm executed by the reader is not costly since it only consists of one hashing and one exponentiation. This algorithm outputs a unique query for a given keyword to be searched in several indices. Therefore, the cost of this algorithm does not depend on the number of searched indices. On the other hand, the reader will receive one response per targeted index and will have to execute *ResponseProcess* over each received response. The cost for one response consists in one decryption through *Dec* and one *PIR.Retrieve* operation. Note that the retrieved value for each index is a single bit, and based on [1] the computational overhead can be considered as reasonable for a lightweight user.

The cost for the proxy of multiplexing the queries with *QueryTransform* and filtering the responses with *ResponseFilter* is linear towards the number of indices the querying reader is authorized to search, and for each queried index the proxy performs a pairing and one execution of $PIR.Query$. The $ResponseFilter$ algorithm can be considered negligible as it only extracts the relevant part of the response.

For a given keyword search query, the CSP builds one matrix per queried index, and executes $PIR.Process$ on each matrix. The building of one matrix requires one exponentiation in G_T per keyword. The operations performed by the CSP being similar to the ones in [9], the workload of the CSP is considered affordable for a cloud server.

To conclude our scheme achieves a very low cost at the reader side, which usually is the main requirement for a cloud computing scenario, and a reasonable cost at the CSP and the proxy. Figure 3 summarizes the cost of each algorithm considering a scenario where one writer uploads several indices and one reader send one query.

Algorithm	**Cost**	**number of executions**
Index	$h + e + exp_{G_1}$	$\mathfrak{i}.\mathfrak{k}$
Delegate	exp_{G_2}	$\mathfrak{i}.\mathfrak{d}$
QueryCreate	$h + mult_{\mathbb{Z}_q} + exp_{G_1}$	
QueryTransform	$\mathfrak{a}(e + H + PIR.KeyGen + PIR.Query)$	
Respond	$\mathfrak{k}(exp_{G_T} + h) + \sqrt{n}(PIR.Process + Enc)$	$\mathfrak{a}$
ResponseFilter	negligible (data forwarding)	$\mathfrak{a}$
ResponseProcess	$Dec + PIR.Retrieve$	$\mathfrak{a}$

Key:

- exp_X: cost of an exponentiation in X
- $mult_X$: cost of a multiplication in X
- $\mathfrak{k}$: number of keyword per index
- $\mathfrak{i}$: number of index owned by a writer
- $\mathfrak{d}$: number of reader with delegated search rights per index
- $\mathfrak{a}$: number of indices the reader is authorized to search
- name of a function: execution cost of this function

Fig. 3. Computational cost of each algorithm.

7 Related Work

Our review of the related work focuses on fully multi-user SE schemes. For a detailed survey on SE in general, we refer the reader to [6].

While solutions in [7,9] seem very efficient in the case where there is a single writer authorizing multiple readers, they become unpractical for the multi writer-multi reader case. Indeed each reader should at least store one key per writer and send one query (even if the same) per writer.

Among the few existing MUSE solutions [3, 5, 8, 12, 15], all of them except the one described in [12] require the existence of a TTP, which is an unpractical assumption that our solution does not make. Finally, all the solutions share a common pitfall as they do not ensure access pattern privacy. As already discussed in this paper, this leads to a serious privacy exposure in the case where users collude with the CSP. Furthermore the execution of $PIR.Process$ in our solution is less costly compared to the search operation at the CSP in all existing MUSE schemes, since in these schemes the trapdoor must be tested with each encrypted keyword in the index either until the test shows that the keyword is present, or until all the keywords in the index have been tested.

8 Conclusion

We have presented a new multi-user searchable encryption scheme that is provably secure under the newly proposed adversarial model witch considers the case where some users can collude with the CSP. All existing schemes become insecure under this new model. The proposed solution is very efficient for the user as it introduces a new party, the proxy, which bears most of the overhead. At the same time this overhead remains reasonable for both the CSP and the proxy.

Future work on this scheme will include implementation and benchmark results of the presented scheme with realistic datasets.

Acknowledgements

The authors thank the anonymous reviewers for their suggestions for improving this paper.

This work was partially funded by the FP7-USERCENTRICNETWORKING european ICT project (grant 611001).

References

1. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: Xpir: Private information retrieval for everyone. Cryptology ePrint Archive, Report 2014/1025 (2014), `http://eprint.iacr.org/`
2. Aguilar-Melchor, C., Gaborit, P.: A lattice-based computationally-efficient private information retrieval protocol. In: In WEWORC 2007 (2007)
3. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop. pp. 77–88. CCSW '13, ACM, New York, NY, USA (2013)
4. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417 (2005), `http://eprint.iacr.org/`
5. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Information Security Practice and Experience, pp. 71–85. Springer (2008)

6. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A Survey of Provably Secure Searchable Encryption. ACM Computing Surveys 47(2), 1–51 (Aug 2014), http://dl.acm.org/citation.cfm?doid=2658850.2636328
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. Cryptology ePrint Archive, Report 2006/210 (2006), http://eprint.iacr.org/
8. Dong, C., Russello, G., Dulay, N.: Shared and Searchable Encrypted Data for Untrusted Servers. In: Atluri, V. (ed.) Data and Applications Security XXII, Lecture Notes in Computer Science, vol. 5094, pp. 127–143. Springer Berlin Heidelberg (2008)
9. Elkhiyaoui, K., Önen, M., Molva, R.: Privacy preserving delegated word search in the Cloud. In: SECRYPT 2014, 11th International conference on Security and Cryptography, 28-30 August, 2014, Vienna, Austria. Vienna, Austria (2014), http://www.eurecom.fr/publication/4345
10. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 3580, pp. 803–815. Springer Berlin Heidelberg (2005)
11. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Zhou, J., Lopez, J., Deng, R., Bao, F. (eds.) Information Security, Lecture Notes in Computer Science, vol. 3650, pp. 314–328. Springer Berlin Heidelberg (2005)
12. Popa, R.A., Zeldovich, N.: Multi-Key Searchable Encryption (2013), http://people.csail.mit.edu/nickolai/papers/popa-multikey-eprint.pdf
13. Scott, M.: On the efficient implementation of pairing-based protocols. In: Chen, L. (ed.) Cryptography and Coding, Lecture Notes in Computer Science, vol. 7089, pp. 296–308. Springer Berlin Heidelberg (2011)
14. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. IACR Cryptology ePrint Archive 2004, 332 (2004), http://www.shoup.net/papers/games.pdf
15. Yang, Y., Lu, H., Weng, J.: Multi-User Private Keyword Search for Cloud Computing. pp. 264–271. IEEE (Nov 2011)