

Private Query on Encrypted Data in Multi-User Settings

Feng Bao¹, Robert H. Deng², Xuhua Ding², and Yanjiang Yang^{1,2}

¹ Institute for Infocomm Research, Singapore
baofeng@i2r.a-star.edu.sg

² School of Information Systems, SMU
{robertdeng,xhding,yjyang}@smu.edu.sg

Abstract. Searchable encryption schemes allow users to perform keyword based searches on an encrypted database. Almost all existing such schemes only consider the scenario where a single user acts as both the data owner and the querier. However, most databases in practice do not just serve one user; instead, they support search and write operations by multiple users. In this paper, we systematically study searchable encryption in a practical multi-user setting. Our results include a set of security notions for multi-user searchable encryption as well as a construction which is provably secure under the newly introduced security notions.

1 Introduction

With the prevalence of network connectivity, a typical paradigm of many enterprise database applications is for multiple users to access a shared database via a local area network or the Internet. For business-critical or security-sensitive data, encryption is often used as the last line of defense to combat unsolicited data accesses. Consider the following application example. A federation of healthcare institutes plans to establish a medical database so that their medical practitioners and researchers can share clinic records and research results. To reduce the operational cost, management of the database is outsourced to a database service provider. The database is encrypted in order to comply with patient privacy related laws, such as HIPPA in the United States, and yet the encrypted database must be searchable by authorized users.

Searchable encryption is a cryptographic primitive that enables users to perform keyword-based searches on an encrypted database just as in normal database transactions [8, 10, 12, 19]. However, all the existing schemes are limited to the single-user setting where the database owner who generates the database is also the single user to perform searches on it. To support multi-user searches, Curtmola et. al. [8], by directly extending their single-user schemes, suggest to share the secret key for database searching among all users. Their scheme allows only one user to write to the database, though multiple users are able to search. Unfortunately, many practical applications (e.g., the aforementioned healthcare federation example) require a database to support both write and search operations by multiple users. Moreover, user revocation in their scheme is based on broadcast encryption, where a revocation affects all non-revoked users.

Extending a single-user scheme to a *full-fledged* multi-user scheme by sharing secret keys (or the private keys of public key based systems (see Section 2)) among all

users is a naïve approach with several serious shortcomings. First, there is no feasible means to determine the originator of a query in a provable manner, since all queries are generated from the same key. This becomes unacceptable when accountability of queries is desired by the database application. Secondly, user revocation can be prohibitively expensive. In a multi-user application, user revocation is a routine procedure. For a key-sharing based scheme, revocation often implies a new round of key distribution involving all non-revoked users. Obviously, this is not scalable for large and dynamic systems where user revocation may occur frequently. One may suggest using access control to complement key sharing in order to address the problem of user revocation (i.e., user revocation does not entail key renewal). However, deployment of access control in practice is prohibitively expensive as pointed out in [8], and worse yet, users have to maintain an additional set of secrets. Thirdly, many searchable encryption schemes follow the symmetric access paradigm, i.e., the same key is used for index generation and search. Therefore, once a revoked user breaks the security perimeter of a database system and gains illegal access to the encrypted database, she is still able to search it at her will. One remedy could be to update the indexes after every user revocation. However, it is obviously infeasible for large databases due to the immense cost it entails.

In this paper, we systematically study searchable encryption in the multi-user setting. We formulate a system model and define its security requirements. We also propose an efficient construction, which offers not only the conventional query privacy, but also the following new features.

- Our system allows a group of users, each possessing a distinct secret key, to insert their encrypted data records to the database while every user in the group is able to search *all* the records using her chosen keywords with the assistance from a semi-trusted database server.
- Our system allows the user management of the database owner organization to dynamically and efficiently revoke users. Our revocation does not require distribution of new keys, nor needs to update the encrypted database including the indexes. After a revocation, the revoked users are no longer able to search the database, while the revocation process is transparent to those non-revoked users. Our system also allows for dynamic user enrollment, since a user joining does not affect other user's settings.
- Our system offers *query unforgeability* in the sense that neither a dishonest user nor the database server is able to generate valid queries on behalf of another user unless her secret key is compromised.

The rest of the paper is organized as follows. In the next section, we discuss the related work and highlight the difference between our work and other searchable encryption schemes. Then, we define the system and formulate security requirements in Section 3. Our proposed construction, together with a rigorous security analysis and a performance evaluation, are presented in Section 4. Concluding remarks are in Section 5.

2 Related Work

Our work is under the umbrella of *searchable encryption*, which in general allows a user to search among encrypted data and find the data containing a chosen keyword. The first practical scheme of this kind is due to Song et al [19], who consider searches across encrypted keywords within a file with an overhead linear to the file size. Goh [12] and Chang and Mitzenmacher [10] propose to search encrypted indexes of a set of documents. Their approaches improve the search efficiency at the cost of a large storage for the constructed indexes (the bit-length of the index for each document is proportional to the total number of keywords). A formal security notion of searchable encryption is defined in [8] which also constructs schemes provably secure against non-adaptive and adaptive adversaries. Yang et al [20] apply the concept of searchable encryption to dynamic databases. The work of [4] considers the variation of simultaneous search of conjunctive keywords.

The first public-key based searchable encryption scheme is due to Boneh et al [2], where the private key holder can perform a search among messages encrypted under the corresponding public key. Park et al [18] and Hwang et al [15] propose variations of conjunctive keywords search in the public key setting. Abdalla et al [1] further analyze the consistency property of public key based searchable encryption, and demonstrate a generic construction by transforming an anonymous identity-based encryption scheme.

Note that while public key based schemes allow for multi-user writing, only the key holder who knows the private key can perform searches. As a result, applying public key based searchable encryption schemes to the multi-user setting would face the same problem as that of symmetric key based ones. Although Curtmola et al. suggest in [8] to employ broadcast encryption to allow multiple user search, it only allows a single user to write to the database. Moreover, their scheme is more suitable for a static collection of documents than a dynamic database. By contrast, our work in this paper studies searchable encryption in database applications where a group of users share data in a way that all users are able to write to and search an encrypted database without sharing their secrets.

3 Model and Definitions

3.1 System Model

We consider a database system $\{D, UM, Serv, \mathcal{U}\}$, where D is a database; UM is the user manager of the data owner organization that is responsible for the management of users, e.g., user enrolment and user revocation; $Serv$ is the database server providing the search service; \mathcal{U} is a group of users.

The database D consists of m records $\{d_1, \dots, d_m\}$ of multiple attributes. One of the attributes is the *keyword* used for search (note that it is straightforward to consider multiple keyword attributes). The domain of the keyword attribute is denoted by \mathcal{W} . The keyword of d_i is denoted by $d_i.w$. $Serv$ does not host the database D directly; instead, it hosts an encrypted version of D , denoted by $D' = \{d'_1, \dots, d'_m\}$, where $d'_i = \langle E(d_i), I(d_i.w) \rangle$: the first component is an encryption of d_i and the second is the

output of an index generation function $I(\cdot)$ on $d_i.w$. Let $E_D = \{E(d_1), \dots, E(d_m)\}$ and $I_D = \{I(d_1.w), \dots, I(d_m.w)\}$.

With the assistance of *Serv*, an authorized user $u \in \mathcal{U}$, is allowed to insert data records to D' and to search data records including those inserted by others based on her chosen keywords. We use $q_u(w)$ to denote a query from user u on keyword $w \in \mathcal{W}$. On receiving query $q = q_u(w)$, *Serv* is expected to return $a_q = \{E(d_i) \mid d_i \in D, d_i.w = w\}$. Whenever necessary, *UM* may revoke a user's privilege of searching the database. Therefore, the user set \mathcal{U} is divided into an authorized user set \mathcal{U}_A and a revoked user set \mathcal{U}_R . Only users in \mathcal{U}_A are allowed to successfully search and write to the database.

UM is an offline user manager of the data owner organization and is responsible for user enrollment and revocation; therefore we assume that *UM* is trusted and all interactions with *UM* are secure (Please do not confuse *UM* with the system administrator of the database server). We consider a semi-trusted *Serv* as in [13], in the sense that it does not deviate from the prescribed protocol execution while it may try to derive as much information as possible from user queries and database access patterns. In particular, we assume that it will not launch active attacks such as collusion with users. Our trust model for *Serv* is based on the following observation. In practice, most database hosting services are run by large and reputable IT service providers which clearly understand the paramount importance of corporate reputation for business success. Therefore, it is logical to assume that the database hosting server follows trusted-but-curious (semi-trusted) behavior. Active attacks are easy to detect/notice and therefore risk the server from being caught. Even a rumor of violation of rules will result in very bad publicity and damage a company's reputation.

Throughout the rest of the paper, we use the following notations. For a set S , we write $x \in_R S$ to denote that x is selected uniformly at random from S , and write $|S|$ to denote the size of S . For an algorithm A , $x \leftarrow A$ denotes that A outputs x . A function $\nu : \mathbb{N} \rightarrow [0, 1]$ is negligible if for any polynomial p , there exists $k_p \in \mathbb{N}$ such that for all $k \geq k_p$, $\nu(k) \leq 1/p(k)$. For convenience of reference, other notations used in the sequel are listed in Table 1.

Notation	Semantic
k_{UM}	the secret key of <i>UM</i>
e	the encryption key for record encryption
$qk_u, ComK_u$	user u 's query key and her complementary key, respectively
U-ComK	a list of 2-tuples $(u, ComK_u)$ maintained by <i>Serv</i>

Table 1. Notations

3.2 Definitions

We now define the multi-user encrypted database system and its security notions. A multi-user encrypted database system, denoted by Γ , consists of the following algorithms:

- **Setup**(1^κ). A probabilistic algorithm executed by UM to set up the system and to initialize system-wide parameters, where κ is the security parameter. The algorithm outputs a secret key k_{UM} for UM and the record encryption key e for a semantically secure symmetric key encryption scheme.
- **Enroll**(k_{UM}, u). Executed by UM to enroll user u to the system. Taking as input k_{UM} and user identity u , it outputs a pair of query key and complementary key $(qk_u, ComK_u)$ for u . qk_u and e are then securely transported to user u , and $ComK_u$ is securely passed to $Serv$ who then updates the U-ComK list by inserting a new entry $(u, ComK_u)$.
- **GenIndex**($qk_u, w; ComK_u$). An interactive algorithm run between user u and $Serv$ to generate an index for keyword w . User u sends an index request on w to $Serv$, who then computes a response using the corresponding $ComK_u$. Finally, u outputs $I(w)$ based on $Serv$'s response.
- **Write**($qk_u, e, d_i; ComK_u$). Run between user u and $Serv$ to write an encrypted record d'_i to D' . The user u first invokes **GenIndex**($qk_u, d_i.w; ComK_u$) to generate $I(d_i.w)$, then computes $E(d_i)$ using e , and finally passes $d'_i = \langle E(d_i), I(d_i.w) \rangle$ to $Serv$ which appends it to D' .
- **ConstructQ**(qk_u, w). Run by a user u to construct a query. It takes as input the secret query key qk_u and a chosen keyword w , and outputs a query $q_u(w)$.
- **Search**($q_u(w), ComK_u, D'$). Run by $Serv$ to search D' for records containing w . Namely, on a query $q_u(w)$, it outputs $a_q = \{E(d_i) \mid d_i \in D, d_i.w = w\}$.
- **Revoke**(u). Run by UM to evict a user from the system. On an input user identity u , it revokes u 's search capability. As a result, $\mathcal{U}_A = \mathcal{U}_A \setminus \{u\}$, $\mathcal{U}_R = \mathcal{U}_R \cup \{u\}$, and u is no longer able to search the database.

A multi-user encrypted database system Γ is *correct* if an authorized user can always get the correct query reply. More formally, $\forall u \in \mathcal{U}_A, \forall w \in \mathcal{W}$, $\text{Search}(\text{ConstructQ}(qk_u, w), ComK_u, D') = \{E(d_i) \mid d_i \in D, d_i.w = w\}$.

We also formalize several security requirements of the multi-user encrypted database system, including *query privacy*, *query unforgeability* and *revocability*.

Query Privacy A common security requirement for all searchable encryption schemes is *query privacy*, which is a security notion on the amount of information leakage to the *server* regarding user queries. As discussed in [8], any searchable encryption scheme inevitably reveals certain query traces (defined shortly) to the server, unless using the *private information retrieval* techniques, or PIR for short [9]. We refer interested readers to [3, 9, 16, 17] for various discussions on PIR. For searchable encryption, the server always observes the database access patterns (e.g. two queries have the same reply), albeit the server is unable to determine the keyword in a query. However, apart from the information that can be acquired via observation and the information derived from it, no other information should be exposed to the server.

For a record d_i , we use $id(d_i)$ to denote the identifying information that is uniquely associated with d_i , such as its database position or its memory location. For a query q and its reply (i.e., the outputs of **Search**) a_q , we define $\Omega(q) = \{u_q, id(a_q)\}$, where u_q is the issuer of q and $id(a_q)$ represents the identifying information of each record in a_q . Let $Q_t = (q_1, \dots, q_t)$ be a sequence of t queries from the user group, and let

$W_t = (w_1, w_2, \dots, w_t)$ be the corresponding queried keywords, and $A_t = (a_1, a_2, \dots, a_t)$ be the corresponding t replies, where $t \in \mathbb{N}$ and is polynomially bounded. We define V_t as the *view* of an adversary (i.e., $Serv$) over the t queries as the transcript of the interactions between $Serv$ and the involved query issuers, together with some common knowledge. Specifically, $V_t = (D' = (E_D, I_D), id(d'_1), \dots, id(d'_{|D'|}), \text{U-ComK list}, Q_t, A_t)$. Following the notation from [8], the *trace* of the t queries is defined to be: $T_t = (|D'|, id(d'_1), \dots, id(d'_{|D'|}), \Omega(q_1), \dots, \Omega(q_t), |\mathcal{U}_A|)$, which contains all the information that we allow the adversary to obtain. Note that $|\mathcal{U}_A|$ equals the number of entries in the U-ComK list. A simulation-based definition of query privacy is formally presented as follows.

Definition 1 (Query Privacy). *A multi-user encrypted database system Γ achieves query privacy if for all database D , for all $t \in \mathbb{N}$, for all PPT algorithm \mathcal{A} , there exists a PPT algorithm (the simulator) \mathcal{A}^* , such that for all V_t, T_t , for any function f :*

$$|\Pr[\mathcal{A}(V_t) = f(D, W_t)] - \Pr[\mathcal{A}^*(T_t) = f(D, W_t)]| < \nu(\kappa)$$

where the probability is taken over the internal coins of \mathcal{A} and \mathcal{A}^* .

Intuitively, the notion of query privacy requires that all information on the original database and the queried keywords that can be computed by $Serv$ from the transcript of interactions she obtains (i.e., V_t) can also be computed from what it is allowed to know (i.e. T_t). In other words, a system satisfying query privacy does not leak any information beyond the information we allow the adversary to have. Note that query privacy implies record secrecy, i.e. the encrypted database $D' = (E_D, I_D)$ does not reveal information on the original database.

REMARK We stress that in the definition of query privacy, user-server collusion is not included in our adversarial model. As we argued earlier, this is a practically rational assumption. On the other hand, from a technical perspective, user-server collusion is able to comprise any searchable encryption scheme, since the sever can always compare the access patterns between a target user and the colluding user.

Query Unforgeability In our system, queries issued by user u is generated by her individual secret query key, which is distinct to any other user's query key. It is thus a basic requirement that neither another user nor the server can generate a *legitimate* query on behalf of u . We refer to this property, which is only applicable to the multiple-user setting, as *query unforgeability*. Query unforgeability allows a query to be uniquely bound to its issuer in a provable way. Therefore, it is the security basis of other system features, e.g. accountability and non-repudiation.

To define query unforgeability, we first define the legitimacy of user queries. For a user $u \in \mathcal{U}$, we define u 's legitimate query set as $Q_u = \{q_u(w) | q_u(w) \leftarrow \text{ConstructQ}(qk_u, w), w \in \mathcal{W}\}$. Namely, a query is user u 's legitimate query if it is indeed constructed by running **ConstructQ** with qk_u . Therefore, an informal meaning of query unforgeability is that for any user u , no adversary is able to compute q satisfying $q \in Q_u$ without compromising qk_u .

Query unforgeability is defined based on a game between an adversary and a challenger. We consider two types of adversaries: malicious users (possibly in a collusion)

and $Serv$. They have different knowledge and attack capabilities. Let \mathcal{A}_U be the adversary representing malicious users and \mathcal{A}_S representing $Serv$. Let \hat{u} be the target user. In \mathcal{A}_U 's game, the challenger simulates the execution of Γ and offers an oracle \mathcal{O} which answers \mathcal{A}_U 's queries on the executions of **Enroll**, **GenIndex**, **Write**, **Search**, and **ConstructQ**($qk_{\hat{u}}, \cdot$) which allows \mathcal{A}_U to obtain queries on keywords of her choices with respect to user \hat{u} ³. In \mathcal{A}_S 's game, she has the knowledge of all users' complementary keys and a collection of \hat{u} 's queries (gathered when \hat{u} searches the database). Thus the challenger gives \mathcal{A}_S the oracle access to **ConstructQ**($qk_{\hat{u}}, \cdot$).

The game is the following: the adversary (either \mathcal{A}_U or \mathcal{A}_S) first picks her target user \hat{u} . Then for \mathcal{A}_U , she is given the query keys of the remaining users, and she queries \mathcal{O} at her will with the restriction that the number of queries is polynomial-bounded. For \mathcal{A}_S , she is given the complementary keys of all users including the target user, and the oracle access to **ConstructQ**($qk_{\hat{u}}, \cdot$). In the end, the adversary halts and returns a query q . Let $Q'_{\hat{u}}$ denote the set of \hat{u} 's queries obtained by the adversary from querying **ConstructQ**($qk_{\hat{u}}, \cdot$). The adversary wins the game if and only if $q \in Q_{\hat{u}} \setminus Q'_{\hat{u}}$. The advantage of the adversary against query unforgeability is defined as the probability of she winning the game. We summarize the notion of query unforgeability as follows.

Definition 2 (Query Unforgeability). *A multi-user encrypted database system Γ achieves query unforgeability if for any $\hat{u} \in \mathcal{U}$, for all PPT algorithms \mathcal{A}_U and \mathcal{A}_S :*

$$\begin{aligned} & \Pr[q \in Q_{\hat{u}} \setminus Q'_{\hat{u}} : (k_{UM}, e) \leftarrow \text{Setup}(1^\kappa); \\ & \quad \forall u \in \mathcal{U} \ (qk_u, ComK_u) \leftarrow \text{Enroll}(k_{UM}, u); \\ & \quad q \leftarrow \mathcal{A}_U^{\mathcal{O}}(\{qk_u | u \in \mathcal{U} \setminus \{\hat{u}\}\}) \\ & \quad \text{or } q \leftarrow \mathcal{A}_S^{\text{ConstructQ}(qk_{\hat{u}}, \cdot)}(\{ComK_u | u \in \mathcal{U}\}) \\ &] < \nu(\kappa) \end{aligned}$$

where the probability is taken over the internal coins of \mathcal{A}_U , \mathcal{A}_S , **Setup**, and **Enroll**.

REMARK Since each user possesses a distinct query key, it would be a natural requirement to maintain *secrecy of query keys*, i.e., a query key is only known to its owner. It is straightforward to observe that if a system is query unforgeable, it also preserves *secrecy of query keys*. Otherwise, the knowledge of the target's query key easily leads to generating a legitimate query.

Revocability User eviction is an indispensable part of a multi-user application. It is desirable to allow UM to revoke the search capabilities of users who are deemed no longer appropriate to search the database. Since the incapability of searching the database indexes is implied by the incapability of distinguishing them, we define revocability based on index indistinguishability.

An adversary's advantage in attacking revocability is defined as her winning probability in the following game. The adversary \mathcal{A} runs in two stages, \mathcal{A}_1 and \mathcal{A}_2 : In the first stage, \mathcal{A}_1 acts as an authorized user and is allowed to access the oracle \mathcal{O} as in Definition 2. At the end of the first stage, \mathcal{A}_1 chooses two new keywords w_1 and w_2 ,

³ An malicious user may observe \hat{u} 's queries by attacking her system or the communication channel.

which have not been queried thus far. Let $state$ represent the knowledge \mathcal{A}_1 gains during the first stage. In the second stage, \mathcal{A}_2 is revoked, and is given the index of one of the two keywords. \mathcal{A}_2 finally outputs a bit b' . \mathcal{A} wins the game if and only if $b' = b$. We summarize the notion of revocability as follows.

Definition 3 (Revocability). A multi-user encrypted database query system Γ achieves revocability if for all PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\Pr[b' = b : \begin{array}{l} (k_{UM}, e) \leftarrow \mathbf{Setup}(1^\kappa); \\ \forall u \in \mathcal{U}_A \ (qk_u, ComK_u) \leftarrow \mathbf{Enroll}(k_{UM}, u); \\ (qk_{\mathcal{A}}, ComK_{\mathcal{A}}) \leftarrow \mathbf{Enroll}(k_{UM}, \mathcal{A}); \\ (state, w_0, w_1) \leftarrow \mathcal{A}_1^{\mathcal{O}}(qk_{\mathcal{A}}); \\ \mathbf{Revoke}(\mathcal{A}); \\ b \in_R \{0, 1\}, I(w_b) \leftarrow \mathbf{GenIndex}(qk_u, w_b; ComK_u)_{u \in_R \mathcal{U}_A}; \\ b' \leftarrow \mathcal{A}_2(state, I(w_b), w_0, w_1), \end{array}] < 1/2 + \nu(\kappa)$$

where the probability is taken over the internal coins of \mathcal{A} , \mathbf{Setup} , \mathbf{Enroll} , and the instance of u .

Intuitively, the definition demands that all successful searches rely on the assistance from $Serv$ using the corresponding complementary keys. With this feature, UM is able to efficiently revoke a user by instructing $Serv$ to delete the relevant key.

REMARK The definition of revocability based on the index indistinguishability addresses the cryptographic strength of the searching protocol. A revoked user might mount attacks on the system or the communication channel in order to perform a search. For instance a replayed query may help a revoked user to search the database. We argue that this type of attacks can be neutralized by deploying secure communication channels or a user authentication mechanism, which are out of the scope of this paper.

4 Our Construction

4.1 Technical Preliminaries

Pseudorandom Function and Pseudorandom Permutation A pseudorandom function is a function whose outputs cannot be efficiently distinguished from the outputs of truly random functions. A keyed cryptographic hash function is often modeled as a pseudorandom function. The main difference between a pseudorandom function and the random oracle is that the former can be accessed only by the key holder, while the latter is publicly accessible. If a pseudorandom function is a permutation, then it is *pseudorandom permutation*. Symmetric key encryption schemes are often modeled as pseudorandom permutations.

Bilinear Map Let G_1 and G_2 be two groups of prime order p . A bilinear map is a function $\hat{e} : G_1 \times G_1 \rightarrow G_2$, satisfying the following properties:

1. Bilinear: For all $g_1, g_2 \in G_1$ and all $x_1, x_2 \in \mathbb{Z}_p^*$, $\hat{e}(g_1^{x_1}, g_2^{x_2}) = \hat{e}(g_1, g_2)^{x_1 x_2}$.
2. Non-degenerate: If g is a generator of G_1 , then $\hat{e}(g, g)$ is a generator of G_2 .

3. Computable: $\hat{e}(g_1, g_2)$ can be efficiently computed for any $g_1, g_2 \in G_1$.

Note that the G_1 is a Gap-Diffie-Hellman group (GDH group), where the Decisional DH problem (DDH) is easy while the Computational DH problem (CDH) is still hard. The CDH problem is to compute g^{ab} , given g, g^a, g^b ; and the DDH problem is to determine $c \stackrel{?}{=} g^{ab}$, given g, g^a, g^b, c , where G is a cyclic group generated by g of prime order p , $c \in_R G$, and $a, b \in_R Z_p^*$.

BLS Short Signature Boneh et al. proposed a short signature scheme in [6] based on bilinear maps. A brief recall of the scheme is as follows: Let G_1, G_2, \hat{e} be defined as the above, and g be a generator of G_1 ; $h : \{0, 1\}^* \rightarrow G_1$ be a collision resistant hash function. A user's key pair is $(x \in Z_p^*, y = g^x \in G_1)$, where x is the private signing key. Then, the signature on a message m is defined to be $\sigma = h(m)^x$. Signature verification is to check $\hat{e}(g, \sigma) \stackrel{?}{=} \hat{e}(y, h(m))$. The BLS short signature achieves existential unforgeability if h is modeled as a random oracle.

4.2 Protocol

We now present our construction. Let G_1, G_2 be two cyclic groups of a prime order p , and a bilinear map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ between them as defined above. Let g be the generator of G_1 . Let $[m]_k$ denote an encryption of a message $m \in \mathcal{M}$ under a secure symmetric encryption scheme with the secret key $k \in \mathcal{K}$, where \mathcal{M} is the message domain and \mathcal{K} is the domain of the secret key. We use $\langle c \rangle_k$ to denote its decryption. Let $h : G_2 \rightarrow \mathcal{K}$ be a collision-resistant hash function mapping an element in G_2 to an element in \mathcal{K} , and $h_s : \mathcal{S} \times \mathcal{W} \rightarrow G_1$ be a keyed hash function under a seed $s \in \mathcal{S}$ that maps a keyword to an element in G_1 , where \mathcal{S} is the domain of the secret seeds. The details of our protocol for the multi-user encrypted database system (MuED) are shown in Figure 1. In this construction, the encryption of records is performed using a semantically secure symmetric key encryption $E()$ with the key e . Note that $[.]_k$ and $E()$ can be different symmetric encryption schemes (in fact, $[.]_k$ is not necessarily semantically secure), so we distinguish them for clarity reason.

Note also that while the database server *Serv* maintains a Com_K list, the list is not intended to enforce access control, or to make any verification on the legitimacy of the user or the relationship between the user and the complementary key $ComK_u$. *Serv* simply uses the complementary key indicated by the querying user in the algorithm of Search.

4.3 Correctness

The correctness of the protocol is straightforward. Suppose that a record $\langle E(d_i), I(w) \rangle$ is generated by user u , where $d_i.w = w$ and $I(w) = \langle r, [r]_k \rangle$. Note that k essentially is equal to $h(\hat{e}(h_s(w), g)^x)$. Consider a user \bar{u} with a secret key $x_{\bar{u}}$ and a complementary key at *Serv* being $ComK_{\bar{u}}$. Her query on the keyword w is $q_{\bar{u}}(w) = h_s(w)^{x_{\bar{u}}}$; and the key used in Search is thus $k' = h(\hat{e}(q_{\bar{u}}(w), ComK_{\bar{u}})) = h(\hat{e}(h_s(w)^{x_{\bar{u}}}, g^{\frac{x_{\bar{u}}}{x}})) = h(\hat{e}(h_s(w), g)^x)$. Since $k = k'$, $E(d_i)$ is inserted into the reply set and returned to \bar{u} .

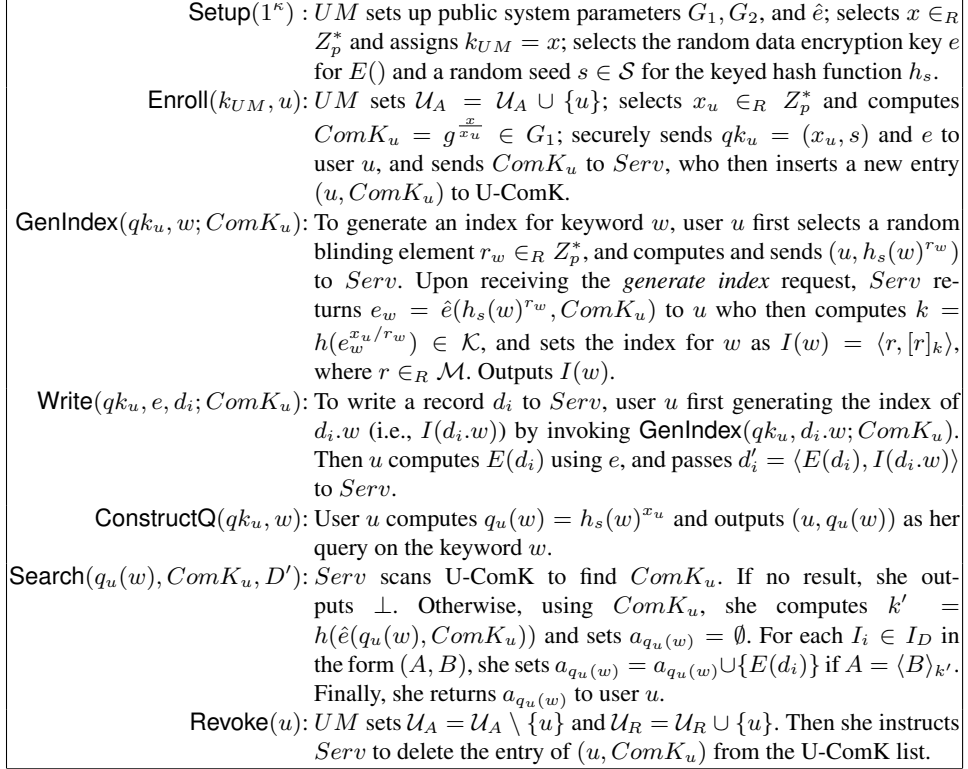


Fig. 1. The Construction of MuED.

according to the protocol. We remark that since $h(\cdot)$ and $h_s(\cdot)$ are collision resistant hash functions, the probability of computing the same key from two different keywords are negligible.

4.4 Security Analysis

In this section, we analyze the security of our protocol, and in particular show that the construction of MuED in Figure 1 satisfies the security requirements in Section 3.

Query Privacy. We first prove that our protocol achieves query privacy in the following theorem.

Theorem 1. *MuED achieves query privacy in Definition 1 if $E(\cdot)$ and $[\cdot]_k$ are pseudorandom permutations and $h_s(\cdot)$ is a pseudorandom function.*

Proof. It suffices for us to construct a PPT simulator \mathcal{A}^* such that for all $t \in \mathbb{N}$, for all PPT adversaries \mathcal{A} , all functions f , given the trace of t queries T_t , \mathcal{A}^* can simulate $\mathcal{A}(V_t)$ with non-negligible probability. More specifically, we show that $\mathcal{A}^*(T_t)$ can generate a view V_t^* which is computationally indistinguishable from V_t , the actual view of \mathcal{A} . Recall that $T_t = (|D'| = m, 1, \dots, m, \Omega(q_1), \dots, \Omega(q_t), |\mathcal{U}_A|)$ and $V_t = (E_D = \{E(d_1), \dots, E(d_m)\}, I_D = \{I_1, \dots, I_m\}, 1, \dots, m, \text{U-ComK}, Q_t, A_t)$.

For $t = 0$ ($Q_t = \emptyset, A_t = \emptyset$), \mathcal{A}^* builds $V_t^* = (E_D^* = \{E(d_1)^*, \dots, E(d_m)^*\}, I_D^* = \{I_1^*, \dots, I_m^*\}, 1, \dots, m, \text{U-ComK}^*)$ as follows. For $1 \leq i \leq m$, it selects $E(d_i)^* \in_R \{0, 1\}^{|E(d_i)|}$, and sets $I_i^* = \langle I_i^*[1], I_i^*[2] \rangle$, where $I_i^*[1] \in_R \mathcal{M}$ and $I_i^*[2] \in_R \mathcal{M}$. To construct U-ComK^* , for each entry \mathcal{A}^* selects a random user identity and sets the corresponding complementary key as a random element from G_1 (the total number of entries is $|\mathcal{U}_A|$, which is contained in T_t). It is easy to check that V_t^* and V_t are computationally indistinguishable if the symmetric encryption (i.e., used to instantiate E and $[\cdot]_k$) is pseudorandom permutation. Note that in this proof, we did not consider the process of generating D' , and assume that D' is already in place. In fact, the generation of D' in MuED does not provide \mathcal{A} any additional knowledge on the keywords, since the only extra information \mathcal{A} obtains by observing the generation of D' is $h_s(w)^{r_w}$ for each w ($r_w \in_R Z_p^*$), which clearly is computationally indistinguishable from a random element from G_1 .

For $t > 0$, \mathcal{A}^* builds $V_t^* = (E_D^*, I_D^*, 1, \dots, m, \text{U-ComK}^*, Q_t^*, A_t^*)$ as follows. To be general, we suppose that all queries in Q_t are from distinct users (recall that the querier of a query can be seen from $\Omega(q_i)$), but some of them may query the same keywords. For $1 \leq j \leq |\mathcal{U}_A|$, it selects $x_j^* \in_R Z_p^*$ and sets $x^* = x_1^* \times \dots \times x_{|\mathcal{U}_A|}^*$.

- Generating E_D^* : Generation of E_D^* is the same as in the case of $t = 0$, where each $E(d_i)^*$ is a random value.
- Generating Q_t^* : Recall that from $\Omega(q_1), \dots, \Omega(q_t)$ contained in V_t , \mathcal{A}^* can determine which queries ask the same keyword. For each $\Omega(q_i)$, it selects a random user identity u_i^* as the querier, and picks up an element from $\{x_1^*, \dots, x_{|\mathcal{U}_A|}^*\}$, say x_i^* , for u_i^* . If there does not exist $j < i$ such that $\Omega(q_j) = \Omega(q_i)$ (note that $\Omega(q_j) = \Omega(q_i)$ means that q_i and q_j ask the same keyword), selects a random element $r_g \in_R G_1$ and computes $q_i^* = r_g^{x_i^*}$. Otherwise, re-uses the same r_g for q_j^* to compute q_i^* .
- Generating U-ComK^* : We actually associate each x_i^* with a user in \mathcal{U}_A . Accordingly, the complementary key corresponding to x_i^* is computed as g^{x^*/x_i^*} . To organize the U-ComK^* , the users together with the corresponding complementary keys involved in generating Q_t^* should be placed to the appropriate positions according to $\Omega(q_1), \dots, \Omega(q_t)$, while the remaining users and complementary keys can be placed randomly to fill the remaining entries of the list.
- Generating I_D^* : From $\Omega(q_i), 1 \leq i \leq t$, \mathcal{A}^* knows which records are retrieved by query q_i . Recall that for $\Omega(q_i), q_i^*$ is computed as $r_g^{x_i^*}$. Computes $k^* = h(\hat{e}(r_g, g)^{x^*})$, and for each of the records retrieved by q_i , the index is set as $\langle r \in_R \mathcal{M}, [r]_{k^*} \rangle$. At last, the index of each of the remaining records (i.e., those are not retrieved by Q_t) is set as $\langle r_1 \in_R \mathcal{M}, r_2 \in_R \mathcal{M} \rangle$.
- Generating A_t^* : Generation of A_t^* is straightforward. A_t^* is simply the set of records from E_D^* whose id's are contained in $\Omega(q_1), \dots, \Omega(q_t)$.

We show that V_t^* is computationally indistinguishable from V_t by comparing them component by component. It is easy to see that if E is a pseudorandom permutation, E_D^* and E_D are computationally indistinguishable. For Q_t^* and Q_t , let us consider an actual query $q_u(w) = h_s(w)^{x_u}$ and a simulated query $q_u^* = r_g^{x_u^*}$, where $r_g \in_R G_1$: $q_u(w)$ and q_u^* are computationally indistinguishable if h_s is a pseudorandom function. For

U-ComK* and U-ComK, an actual complementary key g^{x/x_u} and a simulated complementary key g^{x^*/x_u^*} is indistinguishable, since x_u and x_u^* are random values. It is also not hard to see that I_D^* and I_D are computationally indistinguishable if $[\cdot]_k$ is pseudorandom permutation. Finally, given the above indistinguishability results, the indistinguishability between A_t^* and A_t is straightforward. \square

Query unforgeability. In the following, we prove that our protocol satisfies query unforgeability.

Theorem 2. *If there exists a PPT adversary (either \mathcal{A}_U or \mathcal{A}_S) that breaks the query unforgeability of MuED in Definition 2 with an advantage ϵ , then there exists a PPT adversary \mathcal{B} who can succeed in forging BLS short signatures with the same amount of advantage.*

Proof. We prove the theorem relative to \mathcal{A}_U and \mathcal{A}_S , respectively. Given a BLS short signature scheme as specified in Section 4.1, where x is the secret signing key and y is the public key. Let \mathcal{B} be a PPT adversary attempting to forge a short signature with respect to y .

CASE 1 (For \mathcal{A}_U): Intuitively, \mathcal{A}_U obtains a set of queries $Q = \{h_s(w_1)^{x_{\hat{u}}}, h_s(w_2)^{x_{\hat{u}}}, \dots\}$ of the target user \hat{u} through the $\text{ConstructQ}(qk_{\hat{u}}, \cdot)$ oracle. Note that since \mathcal{A}_U knows s , h_s cannot be modeled as a pseudorandom function; rather, it is modeled as a random oracle. As a result, these queries are essentially the BLS short signatures under the signing key $x_{\hat{u}}$. If \mathcal{A} computes $h_s(w)^{x_{\hat{u}}}$ from Q , it clearly forges a signature on w . The detail of the proof follows.

The proof involves constructing $\mathcal{B}^{\mathcal{O}(x)}(\text{Desc}(\mathcal{O}(x)))$, on input of $\text{Desc}(\mathcal{O}(x))$ which is a description of an instance of the BLS short signature scheme, and is provided the signing oracle $\mathcal{O}(x)$. Note that $\text{Desc}(\mathcal{O}(x))$ includes $g \in G_1$, the public key $y = g^x$, hash function h used in the signature scheme, $\hat{e} : G_1 \times G_1 \rightarrow G_2$, and possibly other parameters describing the scheme. The main idea is to set the signing key x to be the secret query key of the target user. As such, the most challenging part of the simulation is that if k_{UM} is selected randomly as in the original protocol construction, we have difficulty in computing the complementary key of the target user, which should be $g^{k_{UM}/x}$, since we only know g^x . The trick we have is to let \mathcal{B} choose k_{UM} in a ‘‘controlled’’ way, but \mathcal{A}_U does not detect the difference. Specifically, the details of the simulation are as follows.

1. **Setup:** In setting up the system, \mathcal{B} uses G_1, G_2, \hat{e} of the BLS short signature scheme as system parameters. \mathcal{B} then selects a random seed s and a random data encryption key e . To generate k_{UM} , \mathcal{B} chooses a set $X = \{x_1, \dots, x_j, \dots, x_{max}\}$, where $x_j \in_R Z_p^*$ and max is the maximum number of user in the system which is

polynomially bounded by the security parameter κ . Then \mathcal{B} sets $\tilde{k}_{UM} = y^{\prod_{j=1}^{|X|} x_j} = g^{x \cdot \prod_{j=1}^{|X|} x_j} \in G_1$. Note that \mathcal{B} actually does not need to know the discrete logarithm of k_{UM} .

2. **Enroll:** To enroll users in \mathcal{U} to the system, \mathcal{B} assigns a random element from X to a user as the query key. In particular, for each $u_i \in \mathcal{U} \setminus \{\hat{u}\}$, chooses a distinct $x_i \in X$, and sets $(qk_{u_i}, ComK_{u_i}) = ((x_i, s), y^{x_1 \dots x_{i-1} x_{i+1} \dots x_{|X|}})$. for the target

- user \hat{u} , it sets $ComK_{\hat{u}} = g^{\prod_{i=1}^{|X|} x_i}$ (as a result, $x = x_{\hat{u}}$ is a component of $qk_{\hat{u}}$, which \mathcal{B} tries to compute); \mathcal{B} then gives qk_{u_i} , $u_i \in \mathcal{U} \setminus \{\hat{u}\}$, together with e to \mathcal{A}_U .
3. Answer \mathcal{O} queries: \mathcal{B} needs to answer the following types of queries from \mathcal{A}_U : **Enroll**, **GenIndex**, **Write**, **Search**, and **ConstructQ**($qk_{\hat{u}}, \cdot$). It should be clear that \mathcal{B} can trivially answer the queries of **Enroll**, **GenIndex**, **Write** **Search**, because she has the correct complementary keys of all users. We thus focus on how \mathcal{B} answers **ConstructQ**($qk_{\hat{u}}, \cdot$) queries. \mathcal{A}_U can ask for queries on keywords of her choice constructed using the target user's query key. To answer a **ConstructQ**($qk_{\hat{u}}, \cdot$) query, \mathcal{B} resorts to its oracle $\mathcal{O}(x)$: on receiving a keyword w , \mathcal{B} submits the word to $\mathcal{O}(x)$; on getting the reply from $\mathcal{O}(x)$, \mathcal{B} tests the validity of the reply using the verification algorithm of the signature scheme. If it is not valid, \mathcal{B} continues to query $\mathcal{O}(x)$ until gets a valid reply. \mathcal{B} then returns to \mathcal{A}_U the reply it gets from $\mathcal{O}(x)$. Note that implicitly, $\mathcal{O}(x)$ simulates the random oracle $h(\cdot)$, and provides the oracle access to \mathcal{B} . Moreover, since \mathcal{A}_U knows s , \mathcal{B} needs to simulate $h_s(\cdot)$ to \mathcal{A}_U . To simulate $h_s(\cdot)$, \mathcal{B} uses the oracle $h(\cdot)$ from $\mathcal{O}(x)$: in particular, whenever getting a message from \mathcal{A}_U querying $h_s(\cdot)$, \mathcal{B} asks the same message to $h(\cdot)$, and returns to \mathcal{A}_U what is returned from $\mathcal{O}(x)$. As a result, the set of $\{h_s(w_1)^{x_{\hat{u}}}, h_s(w_2)^{x_{\hat{u}}}, \dots\}$ obtained by \mathcal{A}_U is actually $\{h(w_1)^{x_{\hat{u}}}, h(w_2)^{x_{\hat{u}}}, \dots\}$, which is a set of the BLS signatures.
 4. \mathcal{B} finally outputs what \mathcal{A}_U outputs.

CASE 2 (For \mathcal{A}_S): The proof is similar to Case 1. To avoid redundancy, we only highlight the differences between two proofs. \mathcal{A}_S also obtains a set of queries $Q = \{h_s(w_1)^{x_{\hat{u}}}, h_s(w_2)^{x_{\hat{u}}}, \dots\}$ of the target user \hat{u} through the **ConstructQ**($qk_{\hat{u}}, \cdot$) oracle, but \mathcal{A}_S does not know the seed s . This intuitively suggests that forging a query is much harder for \mathcal{A}_S than for \mathcal{A}_U . In the actual proof, \mathcal{B} does not choose s at all in **Setup** (the trick is actually that \mathcal{B} does not use any key for the hash function). Moreover, \mathcal{B} does not provide \mathcal{A}_S the oracle access to $h_s(\cdot)$, since \mathcal{A}_S does not know s and thus does not have access to $h_s(\cdot)$.

The simulation by \mathcal{B} is perfect in both cases. It is obvious that if q on w output by \mathcal{A}_U or \mathcal{A}_S is a legitimate query of the target user \hat{u} , then q must be equal to $h(w)^{x_{\hat{u}}}$, which is a valid BLS short signature on w . This completes the proof. \square

Revocability. The following theorem establishes that the construction of MuED satisfies revocability.

Theorem 3. *MuED achieves revocability in Definition 3 if $[\cdot]_k$ is a secure encryption scheme.*

Proof. The proof is pretty straightforward, and we only state the intuition behind the proof. The indexes of the two keywords w_1 and w_2 are $I(w_1) = \langle r_1, [r_1]_{k(w_1)} \rangle$ and $I(w_2) = \langle r_2, [r_2]_{k(w_2)} \rangle$, where $r_1, r_2 \in_R \mathcal{M}$, and $k(w_1)$ and $k(w_2)$ denote the secret keys generated from w_1 and w_2 , respectively. Since the complementary key of a revoked user is deleted from the U-ComK list, the revoked user can never get $k(w_1)$ and $k(w_2)$ from the keywords and the query key it has; moreover, it cannot get the keys either if $[\cdot]_k$ is a secure encryption scheme that does not expose the encryption key. As a result, $I(w_1)$ and $I(w_2)$ are independent of w_1 and w_2 , respectively, from the perspective of

the revoked user. So the advantage of the adversary guessing the correct bit cannot be significantly more than $1/2$. \square

4.5 Performance

We focus on the online query process, as other procedures (algorithms) have constant computational overhead. For query issuance, the main computation at the user side is simply an exponentiation operation. Thus its computational complexity is $\mathcal{O}(1)$. For a query process, the main computation at the server side includes a pairing operation, and n symmetric key decryption, where n is the number of records. Thus the computational complexity is asymptotically $\mathcal{O}(n)$. Note that all existing single-user searchable encryption schemes except those in [8] require $\mathcal{O}(n)$ server computation. This suggests that the searching efficiency of our protocol does not downgrade due to supporting multiple users.

The computation cost of [8] is linear to the size of the keyword set, instead of the document set. This performance gain is due to a preprocessing of all documents so that the index of a keyword links together all relevant documents. However, it introduces more cost in document deletion or insertion. An optimization approach for our scheme is for the server to group the records retrieved by a reply together by sharing a common index (since they contain the same keyword). It saves the server from repetitively searching the same keyword. As the system proceeds, this can greatly reduce the server computation overhead.

5 Conclusion

Existing efforts on searchable encryption have focused on single-user settings. Directly extending a searchable encryption scheme for the single-user setting to the multi-user setting has several downsides, e.g., the considerable costs associated with re-distributing query keys and re-generating the encrypted database. To solve these problems, in this paper we presented a systematic study on searchable encryption under a practical multi-user database setting. We first formulated a system model as well as a set of security requirements, then presented a concrete construction which provably satisfies those requirements. In our construction, each user employs a distinct key, and consequently, user revocation does not entail updating of query keys and re-encryption of the database, and is transparent to the non-revoked users. Moreover, each authorized user can also insert and search the database, an important feature to data sharing in the multi-user setting. Our construction is efficient, achieving similar performance as most of the existing single-user schemes.

Acknowledgement

This research is partly supported by the Office of Research, Singapore Management University.

References

1. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Lee, G. Neven, P. Paillier, and H. Shi. *Searchable Encryption Revisited: Consistency Properties, Ration to Anonymous IBE, and Extensions*, Proc. Crypto'05, LNCS 3621, pp. 205-222, 2005.
2. D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. *Public key encryption with keyword search*, Proc. Eurocrypt'04, pp. 506-522, 2004.
3. A. Beimel, Y. Ishai, E. Kushilevitz, and Jean-François Raymond. *Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval*, Proc. FOCS'02, pp. 261-270, 2002.
4. L. Ballard, S. Kamara, and F. Monrose. *Achieving Efficient Conjunctive Keyword Searches over Encrypted Data*, Proc. International Conference on Information and Communications Security, ICICS'05, pp. 414-426, 2005.
5. M. Bellare and P. Rogaway. *Random oracles are practical: A paradigm for designing efficient protocols*, Proc. ACM Conference on Computer and Communications Security, CCS'93, pp. 62-73, 1993.
6. D. Boneh, B. Lynn, and H. Shacham. *Short Signatures from the Weil Pairing*, Proc. Asiacrypt'01, LNCS 2248, pp. 514-532, 2001.
7. R. Canetti, O. Goldreich, and S. Halevi. *The random oracle methodology, revisited*, Proc. STOC'98, pp. 209-218, 1998.
8. R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky. *Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions*, Proc. ACM Conference on Computer and Communications Security, CCS'06, pp. 79-88, 2006.
9. B. Chor, E. Kushilevitz, O. Goldreich, and Madhu Sudan. *Private information retrieval*, Journal of the ACM, 1995.
10. Y. Chang, and M. Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*, Proc. Applied Cryptography and Network Security, ACNS'05, LNCS 3531, pp. 442-455, 2005.
11. A. Fiat, and A. Shamir. *How to prove yourself: Practical solutions to identification and signature problems*, Proc. Crypto'86, LNCS 263, pp. 186-194, 1986.
12. E. Goh. *Secure Indexes*, <http://crypto.stanford.edu/eujin/papers/secureindex/secureindex.pdf>, 2003.
13. O. Goldreich. *Foundations of Cryptography*, Volume 2, Cambridge University Press, 2004.
14. O. Goldreich, and R. Ostrovsky. *Software Protection and Simulation on oblivious RAMs*, Journal of ACM, Vol 43(3), pp. 431-473, 1996.
15. Y. H. Hwang, P. J. Lee. *Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-User System*, Proc. International Conference on Pairing-Based Cryptography, Pairing'07, 2007.
16. E. Kushilevitz, and R. Ostrovsky. *Replication is not needed: single database, computationally private information retrieval*, Proc. FOCS'97, pp. 364-373, 1997.
17. H. Lipmaa. *An oblivious transfer protocol with log-squared communication*, Proc. Information Security Conference, ISC'05, LNCS 3650, pp. 314-328, 2005.
18. D. Park, K. Kim, and P. Lee. *Public Key Encryption with Conjunctive Field Keyword Search*, Proc. International Workshop on Information Security Applications, WISA'04, pp. 73-86, 2004.
19. D. Song, D. Wagner, and A. Perrig. *Practical Techniques for Searches on Encrypted Data*, Proc. IEEE Symposium on Security and Privacy, S&P'00, pp. 44-55, 2000.
20. Z. Yang, S. Zhong, and R. N. Wright. *Privacy-Preserving Queries on Encrypted Data*, Proc.