Shared and Searchable Encrypted Data for Untrusted Servers

Changyu Dong[1], Giovanni Russello[2], Naranker Dulay[1]

[1]Department of Computing, Imperial College London, 180 Queen's Gate, London, SW7 2AZ, UK {changyu.dong,n.dulay}@imperial.ac.uk

[2]Security Area, Create-Net, Via alla Cascata 56/D Povo, 38100 Trento, Italy giovanni.russello@create-net.org

December 9, 2009

Abstract

Current security mechanisms are not suitable for organisations that outsource their data management to untrusted servers. Encrypting and decrypting sensitive data at the client side is the normal approach in this situation but has high communication and computation overheads if only a subset of the data is required, for example, selecting records in a database table based on a keyword search. New cryptographic schemes have been proposed that support encrypted queries over encrypted data. But they all depend on a single set of secret keys, which implies single user access or sharing keys among multiple users, with key revocation requiring costly data re-encryption. In this paper, we propose an encryption scheme where each authorised user in the system has his own keys to encrypt and decrypt data. The scheme supports keyword search which enables the server to return only the encrypted data that satisfies an encrypted query without decrypting it. We provide a concrete construction of the scheme and give formal proofs of its security. We also report on the results of our implementation.

1 Introduction

The demand for outsourcing data storage and management has increased dramatically in the last decade. The foremost reason is that for nearly all organisations, data growth is inevitable. Data is at the heart of business operations and applications, driving the critical activities that help the organisations improve customer satisfaction and accelerate business growth. Huge amounts of data are collected or generated everyday and put into data storage for future processing and analysing. According to Forrester Research, enterprise storage needs grow at 52 percent per year [6]. To reduce the increasing costs of storage management, many organizations would like to outsource their data storage to third party service providers. Recent research from TheInfoPro shows that nearly 20% of Fortune 1000 organizations outsource at least some portion of their storage management activities [10]. Apart from business data, there is also an emerging trend in personal data outsourcing. People are demanding more storage space from service provider for various reasons: data backup [1], sharing photos and videos with family and friends [2] or even to manage their medical record [3].

One of the biggest challenges raised by data storage outsourcing is data confidentiality. Business data is vital to many companies, any security breaches will leave the companies with lost revenues, reduced shareholder value, lawsuits as well as damaged reputations. Exposing this valuable information to outsiders poses huge risks. While companies may trust a Storage Service Provider's (SSP) reliability, availability, fault-tolerance and performance, they cannot trust that the SSP is not going to use the data for other purposes. The same problem also exists in personal data outsourcing. For privacy reasons, individuals want to be sure that the data can only be accessed by particular people and certainly not by the SSP's employees. The negative impact of this distrust is two-fold. From the customers' point of view, it is hard to find a trusted service provider to host their data. From the SSPs' point of view, as long as they cannot dispel the concern, they will lose potential customers.

Traditional access controls which are used to provide confidentiality are mostly designed for in-house services and depend greatly on the system itself to enforce authorisation policies, effectively relying on a trusted infrastructure. In the absence of trust, traditional security models are no longer valid. Another common approach to provide data confidentiality is cryptography. Server side encryption is not appropriate when the server is not trusted. The client must encrypt the data before sending it to the SSP and later the encrypted data can be retrieved and decrypted by the client. This would ease a company's concern about data leakage, but introduces a new problem. Because the encrypted data is not meaningful to the SSP's servers, many useful data operations are not possible. For example, if a client wants to retrieve documents or records containing certain keywords, can we keep the data incomprehensible to servers and their administrators while efficiently retrieving the data? Consider the following scenarios:

Scenario 1 *Company A is considering outsourcing its data processing centre to a service provider B. This will cut its annual IT cost by up to 25%. But the company is concerned about data security. The company's databases contain valuable production data and customer information. It would be unacceptable if competitors got hold of the data. Administrative controls such as formal contracts, confidential agreements and continuous auditing provide a certain level of assurance, but the company would also like to encrypt the sensitive data and have fast searches over it.*

Scenario 2 *Bob subscribes to a Personal Health Record service. The service allows Bob to maintain his electronic medical records and share them with his doctors through a web interface. Bob wants to encrypt his records,* *ensuring that the employees of the service provider will not be able to know what is inside.*

A trivial solution is to download all the data to the client's computer and decrypt it locally. This does not scale to large datasets. Several schemes have been proposed to partially address the above problems. The basic idea is to divide the cryptographic component between the client and the server. The client performs the data encryption/decryption and manages keys. The server processes encrypted search queries by carrying out some computation on the encrypted data. The server learns nothing about the keys or the plaintexts of the data nor the queries, but is still able to return the correct results.

These schemes have an important limitation. The operations, e.g. encryption, decryption and query generation, more or less rely on some shared secret keys. This implies that the operations can only be executed by one user, or by a group of users who share the secret keys somehow. A single user is usually not an adequate assumption for data outsourcing. Perhaps the biggest problem for supporting multiple user access to encrypted data is key management. Sharing keys is generally not a good idea since it increases the risk of key exposure. In response to this, keys must be changed regularly. The keys must also be changed if a user is no longer qualified to access the data. However, changing keys may result in decrypting all the data with the old key and re-encrypting it using the new keys. For large data sets, this is not practical.

In this work, we propose a scheme for multi-user searchable data encryption based on proxy cryptography. We consider the application scenario where a group of users share data through an untrusted data storage server which is hosted by a third party. Unlike existing schemes for searchable data encryption in multi-user settings which have constraints such as asymmetric user permissions (multiple writers, single reader) or read-only shared data set, in our scheme the shared data set can be updated by the users and each user in the group can be both reader and writer. The server can search on the encrypted data using encrypted keywords. More importantly our scheme do not rely on shared keys. This significantly simplifies key revocation. Each authorised user in the system has his own unique key set and can insert encrypted data, decrypt the data inserted by other users and search encrypted data without knowing the other users' keys. The keys of one user can easily be revoked without affecting other users or the encrypted data at the server. After a user's keys have been revoked, the user will no longer be able to read and search the shared data.

2 Related Work

Many systems designed for securing untrusted storage rely on the clients to encrypt the data. For example, in cryptographic distributed file systems such as [23, 20, 16, 26], the untrusted file servers store encrypted files and have no knowledge of the keys. Authorised users can retrieve an encrypted file by its identifier and decrypt it using a key obtained from the owner of the file. The servers perform only the basic I/O function and cannot do advanced operations such as keyword search.

Several schemes have been developed to encrypt data on the client-side and enable server-side searches on encrypted data. Song et.al. [25] introduced the first practical scheme for searching on encrypted data. The scheme enables clients to perform searches on encrypted text without disclosing any information about the plaintext to the untrusted server. The untrusted server cannot learn the plaintext given only the ciphertext, it cannot search without the user's authorisation, and it learns nothing more than the encrypted search results. The basic idea is to generate a keyed hash for the keywords and store this information inside the ciphertext. The server can search the keywords by recalculating and matching the hash value. Yang et. al. [29] proposed an elegant scheme for performing queries on encrypted data and also provided a secure index to speed up queries by two-step mapping. Goh's scheme [15] enables searches on encrypted data that employed a secure index based on a bloom filter which has low storage overheads. Damiani et.al. [12] proposed an approach to indexing encrypted data which allows efficient data access. The indexed attribute values are encrypted directly with a key or hashed. This approach also support range queries by creating an encrypted B+-tree. Agrawal et.al. [4] proposed an order preserving encryption for numeric data which allows queries based on comparison conditions. The plaintext is encrypted so that the ciphertext follows a target distribution provided by the user while the order of the data is preserved.

In the *bucketization* approach for searching encrypted databases [17, 18], an attribute domain is partitioned into a set of buckets each of which is identified by a tag. These bucket tags are maintained as an index and are utilised by the server to process the queries. Bucketization has relatively small performance overhead and enables more complex queries such as range queries and comparison queries at the cost of revealing more information about the encrypted data.

All the encrypted search schemes above for searches on encrypted data rely on secret keys, which implies single user access or sharing keys among a group of users. Boneh et. al. [8] presented a scheme for searches on encrypted data using a public key system that allows mail gateways to handle email based on whether certain keywords exist in the encrypted message. The application scenario is similar to [25], but the scheme uses asymmetric encryption schemes instead of symmetric ones. Asymmetric keys allow multiple users to encrypt data using the public key, but only the user who has the private key can search and decrypt the data. Curtmola et. al. [11] partly solved the multi-user problem by using broadcast encryption. The set of authorised users share a secret key r (which is used in conjunction with a trapdoor function). Only people who know r will be able to access/query the data. A user can be revoked by changing r, and using broadcast encryption to send the new key r' to the set of authorised users. The revoked user does not know r', and hence cannot search. In this scheme, the database is searchable, but is read-only and cannot be updated. In our scheme, any authorised user can read, search and update the encrypted data.

Our scheme is dependent on proxy encryption. The notion of proxy encryption was first introduced in [7]. In a proxy encryption scheme, a ciphertext encrypted by one key can be transformed by a proxy function into the corresponding ciphertext for another key without revealing any information about the keys and the plaintext. Proxy encryption schemes can be built on top of different cryptosystems such as El Gamal [14] and RSA [24]. Applications of proxy encryption include: secure email lists [22], access control systems [5] and attribute based publishing of data [21]. A comprehensive study on proxy cryptography can be found in [19].

3 Threat Model

Before presenting our scheme, it is necessary to discuss the threat model. In this section, we first identify the entities involved and the assumptions underlying the system design. Then we identify the potential adversaries and the possible attacks.

3.1 Entities

There are three types of entities in our system:

- Users: Authorised users are able to read, write and search encrypted data residing on the remote server. Sometimes we may need to revoke an authorised user. After being revoked, the user is no longer able to access the data.
- Server: The main responsibility of the data storage server is to store and retrieve encrypted data according to authorised users' requests.
- Key management server (KMS): The KMS is a fully trusted server which is responsible for generating and revoking keys. It generates key sets for each authorised user and is also responsible for securely distributing generated key sets. When a user is no longer trusted to access the data, the KMS revokes the user's permission by revoking his keys.

Authorised users are fully trusted. They are given permissions to access the shared data stored on the remote server by the data owner. They are believed to behave properly and can protect their key sets properly. The data storage server is not trusted in the sense that we believe that the server will execute requests it received correctly, but we do not rely on them to maintain data confidentiality. In other words, the server is modelled as "honest but curious" in our trust model.The KMS is also fully trusted. Although requiring a trusted KMS seems at odds with using an untrusted data storage service, we argue that the KMS requires less resources and less management effort. Securing the KMS is much easier since a very limited amount of data needs to be protected and the KMS can be kept offline most of time.

3.2 Assumptions

We assume that there are mechanisms in place which ensure integrity and availability of the remotely stored data. Our scheme focuses only on confidentiality issues and does not provide protection against attacks such as data forgery and denial of service. Our scheme is a building block that can be integrated into larger more comprehensive frameworks for securing data on untrusted servers

We also assume that the server cannot collude with the users. In our scheme, if a user and the server collude and combine their keys, they can recover the encrypted data. Although this assumption is quite strong, we will show later in the paper that in practice the assumption can be weakened to an acceptable level by introducing certain auxiliary mechanisms.

3.3 Adversary and Attacks

We consider a Curious server[1] as the adversary. As we mentioned earlier, we model the server as honest but curious. The server has full access to the encrypted data stored on it and all the server side key sets. It can also observe the communication to and from users. We limit the attacks mounted by the server to passive attacks. That is, the server will not actively manipulate the stored data and communication in order to gain knowledge. This is because we assume that there are mechanisms taking care of integrity issues, which can detect active attacks.

4 Definitions

4.1 Preliminaries

Throughout the paper, we always consider a scheme to be secure in the sense that the probability that an adversary can break the scheme is a negligible function of the security parameter. By negligible we mean an adversary's success probability is "too small to matter". Formally:

Definition 1 (Negligible Function). *A function f is negligible if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.*

Our proofs rely largely on the assumption that the Decisional Diffie-Hellman (DDH) problem is hard regarding certain groups, i.e. it is hard to tell the difference between group elements $g^{\alpha\beta}$ and g^{γ} given g^{α} and g^{β}.

Definition 2 (DDH Assumption). *We say that the DDH problem is hard regarding a group $\mathbb{G}$ if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$|Pr[\mathcal{A}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] - Pr[\mathcal{A}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1]| < negl(k)$$

[1]Here "server" actually means an administrator who manages the server or a hacker who compromises the server.

where $\mathbb{G}$ is a cyclic group of order q ($|q| = k$) and g is a generator of $\mathbb{G}$, $\alpha, \beta, \gamma \in \mathbb{Z}_q$ are uniformly randomly chosen.

The security of our scheme also depends on the existence of pseudorandom functions. Intuitively, pseudorandom functions are functions that cannot be efficiently distinguished from truly random functions when used as black boxes.

Definition 3 (Pseudorandom Function). *We say $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is a pseudorandom function if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$|Pr[\mathcal{A}^{f_k(\cdot)} = 1] - Pr[\mathcal{A}^{F(\cdot)} = 1]| < negl(n)$$

where $k \leftarrow \{0,1\}^n$ is chosen uniformly randomly and F is a function chosen uniformly randomly from the set of functions mapping n-bit strings to n-bit strings.

4.2 System Model

A Multi-user Searchable Data Encryption scheme is a mechanism such that a group of authorised users can share encrypted documents and perform keyword search on the encrypted documents without decrypting them. We first introduce some notations to be used in the definitions. Δ is the set of all possible data items (documents). $\mathcal{W} = \{w_1, ..., w_d\}$ is a dictionary which contains all the possible words that can be used in the queries. Each document D_i in Δ has an identifier $id(D_i)$ and can be associated with a list of keywords $kw(D_i)$ which is a subset of $\mathcal{W}$. The identifiers must be unique and must not contain information about the content of the documents. For example, the identifiers can be randomly generated strings. We use $\mathcal{D} = \{(D_1, kw(D_1)), ..., (D_n, kw(D_n))\}$ to denote an arbitrary set of documents with their keywords, i.e. $\mathcal{P}(\Delta \times \mathcal{P}(\mathcal{W}))$.

Definition 4 (Multi-user Searchable Data Encryption). *A multi-user searchable data encryption scheme is a tuple of probabilistic polynomial time algorithms* (Init, Keygen, Enc, Re-enc, Trapdoor, Search, Dec, Revoke) *such that:*

- *The* initialisation algorithm Init(1^k) *is run by the KMS which takes as input the security parameter 1^k and outputs master public parameters* Params *and a master key set* MSK.
- *The* user key sets generation algorithm Keygen(MSK,i) *is run by the KMS which takes as input the master keys* MSK *and a user's identity i, generates two key sets* $\mathsf{Ku_i}$ *and* $\mathsf{Ks_i}$. $\mathsf{Ku_i}$ *is the user side key set for user i and* $\mathsf{Ks_i}$ *is the server side key set for user i.*
- *The* data encryption algorithm Enc($\mathsf{Ku_i}$,D,kw(D)) *is run by a user who uses his key set* $\mathsf{Ku_i}$ *to encrypt a document* D *and a set of keywords associated* kw(D)*, then outputs ciphertext* $\mathsf{c_i^*}$(D,kw(D)).

- *The* data re-encryption algorithm Re-enc(i,$\mathsf{Ks_i}$,$\mathsf{c_i^*}$(D,kw(D))) *is run by the server to re-encrypt a ciphertext tuple* $\mathsf{c_i^*}$(D,kw(D)) *from a user* i. *The server finds the corresponding key set* $\mathsf{Ks_i}$ *and outputs re-encrypted ciphertext* c(D,kw(D)).
- *The* trapdoor generation algorithm Trapdoor($\mathsf{Ku_i}$,w) *is run by a user which takes the user's key set* $\mathsf{Ku_i}$ *and a keyword* w *and outputs a trapdoor* $\mathsf{T_i}$(w) *for the word.*
- *The* data pre-decryption algorithm Pre-dec(i,$\mathsf{Ks_i}$,c(D)) *is run by the server to partially decrypt an encrypted document for user i using i's server side key set* $\mathsf{Ks_i}$. *The result* $\mathsf{c_i'}$(D) *is returned to the user.*
- *The* search algorithm Search(i,$\mathsf{T_i}$(w),$E(\mathcal{D})$,$\mathsf{Ks_i}$) *is run by the server. For each* c(D,kw(D)) $\in E(\mathcal{D})$*, if the queried keyword* w $\in$ kw(D) *then the server runs the data pre-decryption algorithm and returns the result* $\mathsf{c_i'}$(D) *to the user.*
- *The* data decryption algorithm Dec($\mathsf{Ku_i}$,$\mathsf{c_i'}$(D))*is run by a user which decrypts* $\mathsf{c_i'}$(D) *by using the user's key set and outputs* D.
- *The* user revocation algorithm Revoke(i) *is run by the server. Given* i*, the server updates its user-key mapping set* $\mathcal{K}s = \mathcal{K}s \setminus$ (i,$\mathsf{Ks_i}$).

The system supports the following primitive data operations:

- Get: Given a document identifier $id(D_i)$, retrieves the document D_i. The user sends $id(D_i)$ to the server which locates the ciphertext from the data storage and runs the data pre-decryption algorithm. The partially decrypted document is returned to the user and the user runs the data decryption algorithm to recover the plaintext of the document.
- Search: Given a keyword w, retrieves all the documents associated with the keyword. The user runs the trapdoor generation algorithm and sends the trapdoor generated from w to the server which runs the search algorithm to find all matching documents and returns the result set to the user. The user runs the data decryption algorithm to decrypt all the documents.
- Insert: Inserts a new document and the keyword list associated with it into the data storage. The user runs the data encryption algorithm to encrypt the document and the keywords. The ciphertext is sent to the server which runs the data re-encryption algorithm and stores the re-encrypted ciphertext.
- Remove: Given a document identifier $id(D_i)$, removes the document D_i and the associated keyword list. The server locates the ciphertext by the identifier and removes it from the data storage.

The primitive operations can be combined to build more complicated ones. For example, update cannot be done by the system directly due to the fact that the documents are encrypted. However, it can be implemented by removing the old document followed by inserting the modified version.

4.3 Security Definition

The security definition for searchable encryption is difficult because searching leaks information about stored data inevitably. As long as the searching algorithm is correct, it always returns the same result set for the same query. Although the queries and the result sets are encrypted, the adversary can still build up search patterns. Therefore the security definition for searchable encryption should be modified to reflect the intuition that nothing should be leaked beyond the outcome and the pattern of a sequence of searches. Here we adapt the definition from [11] of non-adaptive indistinguishability security. Informally, non-adaptive indistinguishability security means that given two non-adaptively generated query histories with the same length and outcome, no PPT adversary can distinguish one from another based on what it can "see" in the interaction. Non-adaptive means the adversary cannot choose queries based on the previous queries and results. This is acceptable because in our setting only the authorised user can generate queries.

The major difference between our definition and the one in [11] is that the definition presented in [11] is defined for a static document set. The whole document set is encrypted before searching can take place. Afterwards no modification is allowed on the document set. While in our scheme, the document set can be continuously updated by authorised users. Therefore in our definition we have to consider the dynamic changes on the document set.

We formalise the interaction between the server and the authorised users as *history*. Generally speaking, a history is defined as a sequence of queries over a set of documents. Let $\mathcal{Q} = (\Delta \times \mathcal{P}(\mathcal{W})) \cup \mathcal{W}$ denotes all possible queries, we call a query $q \in \mathcal{Q}$ an insert query if $q \in \Delta \times \mathcal{P}(\mathcal{W})$ and a search query if $q \in \mathcal{W}$. Intuitively, an insert query is issued by a user to insert a document into the shared data storage and a search query is issued to search a keyword over currently stored data. We use q^u to denote a query issued by a user u.

Definition 5 (History). *A history $H_i \in \mathcal{P}(\Delta \times \mathcal{P}(\mathcal{W})) \times \mathcal{Q}^i$ is an interaction between a client and a server over i queries on a document set $\mathcal{D}$, i.e. $H_i = (\mathcal{D}, q_1^{u_1}, ..., q_i^{u_i})$.*

As we mentioned before, searching leaks information. The maximum information we have to leak is captured by a *trace*. In our settings, a trace contains information from three sources: the encrypted documents stored on the server, the queries and the search query pattern. The outcome of a search query $q = w$ can be represented as a set of identifiers of stored documents which are associated with the keyword being queried and the index of the keyword in the keyword list, i.e. $rs(w) = \{(id(D), i) | D \in \mathcal{D}^q \wedge w \textit{ is the ith element in } kw(D)\}$. The intuition of using document identifiers is that the adversaries can identify the documents but should not learn anything about the content of the documents. Here $\mathcal{D}^q$ means the document set stored on the server at the point of time the query q is issued. Because the users may insert new documents into the data storage, $\mathcal{D}^q$ can be different from the initial document set $\mathcal{D}$. We also define the search pattern Π_i over a history H_i to be a symmetric binary matrix where $\Pi_i[j,k] = 1$ if $q_j = q_k$, and $\Pi_i[j,k] = 0$ otherwise, for $1 \leq j,k \leq i$.

Definition 6 (Trace of a Query)**.** *Given a query q^u, the trace of q^u is defined as:*

$$Tr(q^u) = \begin{cases} (u, id(D), |D|, |kw(D)|) & if\ q^u = (D, kw(D))\ is\ an\ update\ query, \\ (u, rs(q^u)) & if\ q^u\ is\ a\ search\ query, \end{cases}$$

Definition 7 (Trace of a History)**.** *Given a history $H_i = (\mathcal{D}, q_1^{u_1}, ..., q_i^{u_i})$, let $m = |\mathcal{D}|$, the trace of H_i is defined as a list:*

$$Tr(H_i) = \begin{pmatrix} id(D_1), |D_1|, |kw(D_1)|, ..., id(D_m), |D_m|, |kw(D_m)|, \\ Tr(q_1^{u_1}), ..., Tr(q_i^{u_i}), \Pi_i \end{pmatrix}$$

The trace includes the following: the identifier, the size and the number of keywords of each document in the initial set, the trace of each query in the history and the search query pattern.

During an interaction, the adversary cannot directly see the plaintext of the documents or keywords. What the adversary can see is the ciphertext. The view of the adversary is then defined as:

Definition 8 (View of a Query)**.** *Given a query q^{u_i}, the view of q^{u_i} under a key set Ku_i is defined as:*

$$V_{Ku_i}(q^{u_i}) = \begin{cases} c^*_{u_i}(D, kw(D)) & if\ q^{u_i} = (D, kw(D))\ is\ an\ update\ query, \\ (T_{u_i}(q^{u_i}), c'_{u_i}(rs(q^{u_i}))) & if\ q^{u_i}\ is\ a\ search\ query, \end{cases}$$

In the above definition, $c'_{u_i}(rs(q^{u_i}))$ is shorthand for $\{c'_{u_i}(D) | id(D) \in rs(q^{u_i})\}$.

Definition 9 (View of a History)**.** *Given a history $H_i = (\mathcal{D}, q_1^{u_1}, ..., q_i^{u_i})$, let $m = |\mathcal{D}|$, the view generated under a set of user side key sets Ku is defined as*

$$V_{Ku}(H_i) = (c(D_1, kw(D_1)), ..., c(D_m, kw(D_m)), V_{Ku_1}(q_1^{u_1}), ...V_{Ku_i}(q_i^{u_i})\ )$$

where each $Ku_1, ..., Ku_i \in Ku$

Here the view of history is actually defined in a multi-user setting that combines encrypted queries from different users. We do not require each $Ku_1, ..., Ku_i$ in the history to be distinct: obviously some users may submit multiple queries during the interaction. A special case is when Ku contains only one user side key set, in this case the history becomes a history of a single user.

The definition of security against a server is then based on the idea that the scheme is secure if no more information is leaked beyond what the adversary can get from the traces. This intuition is formalised by defining a game where the adversary generates two histories which have the same trace. Then an interaction is started based on one of the two histories. After observing the interaction, the adversary must decide which history the interaction is based on. Since the traces are identical, the adversary cannot distinguish the two histories by the traces, i.e. the knowledge it already has. It must extract additional knowledge from what else it can observe during the interaction, i.e.

the view. The negligible probability of the adversary successfully distinguishing the two histories implies that it cannot get extra knowledge and in consequence the scheme is secure.

Definition 10 (Non-Adaptive Indistinguishability against a Curious Server). *A multi-user searchable data encryption scheme is secure in the sense of non-adaptive indistinguishable against a curious server if for all $i \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow Init(1^k), \\ (\mathcal{K}_u, \mathcal{K}_s) \leftarrow KeyGen(MSK, \mathcal{U}), \\ H_{i_0}, H_{i_1} \leftarrow \mathcal{A}(\mathcal{K}_s), \\ b \xleftarrow{R} \{0,1\}, \\ b' \leftarrow \mathcal{A}(\mathcal{K}_s, V_{\mathcal{K}_u}(H_{i_b})) \end{array} \right] < \frac{1}{2} + negl(k)$$

where $\mathcal{U}$ is a set of user IDs, $\mathcal{K}_u$ and $\mathcal{K}_s$ are the set of user side key sets and the set of server side keys generated for users in $\mathcal{U}$, H_{i_0} and H_{i_1} are two histories over i queries such that $Tr(H_{i_0}) = Tr(H_{i_1})$.

5 Concrete Construction

In this section, we introduce a concrete construction of the multi-user searchable data encryption scheme. The construction is based on a proxy encryption scheme built upon the El Gamal encryption scheme.

5.1 El Gamal-based Proxy Encryption Scheme

We first briefly review the El Gamal encryption scheme [14]. The El Gamal Encryption Scheme $\mathcal{E}$ consists of 3 algorithms:

- $\mathcal{E}$-Init(1^k): On input 1^k, output two prime numbers p, q such that q divides $p-1$, a cyclic group $\mathbb{G}$ with a generator g such that $\mathbb{G}$ is the unique order q subgroup of $\mathbb{Z}_p^*$. Choose $x \xleftarrow{R} \mathbb{Z}_q$ and compute $h = g^x$. The public key is $pk = (\mathbb{G}, g, h, q)$ and the private key is $sk = x$.
- $\mathcal{E}$-Enc(pk, m): Choose $r \xleftarrow{R} \mathbb{Z}_q$ and output the ciphertext $c(m) = (g^r, h^r m)$.
- $\mathcal{E}$-Dec$(sk, c(m))$: The ciphertext is decrypted as $h^r m \cdot (g^r)^{-x} = g^{rx-rx}m = m$.

The proxy encryption scheme is described below and the encryption/decryption process is shown in Figure 1:
The proxy encryption scheme $\mathcal{PE}$ consists of 6 algorithms:

- $\mathcal{PE}$-Init(1^k): On input 1^k, run $\mathcal{E}$-Init(1^k) to obtain $(\mathbb{G}, g, q, x)$. Publicise $(\mathbb{G}, g, q)$ as public parameters and keep x secret as the master key. This algorithm is run by a trusted key management server once when the system is set up .

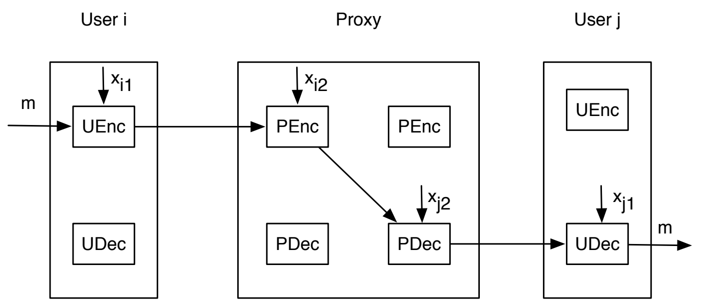


Figure 1: Encryption/Decryption in the El Gamal Proxy Encryption Scheme

- $\mathcal{PE}$-Keygen(MSK, i): For each user i, the key management server chooses $x_{i1} \xleftarrow{R} \mathbb{Z}_q$ and computes $x_{i2} = x - x_{i1}$. x_{i1} is securely transmitted to the user and (i, x_{i2}) is securely transmitted to the proxy.
- $\mathcal{PE}$-U-Enc(x_{i1}, m): The user side encryption algorithm. The user chooses $r \xleftarrow{R} \mathbb{Z}_q$ and outputs the ciphertext $c_i^*(m) = (g^r, g^{rx_{i1}}m)$. The ciphertext is sent to the proxy
- $\mathcal{PE}$-P-Enc(i,$x_{i2}, c_i^*(m)$): The proxy re-encryption algorithm. Given the ciphertext received from user i, the proxy finds the user's server side key x_{i2}, compute $(g^r)^{x_{i2}} \cdot g^{rx_{i1}}m = g^{r(x_{i1}+x_{i2})}m = g^{rx}m$. The ciphertext at the proxy side then becomes $c(m) = (g^r, g^{rx}m)$.
- $\mathcal{PE}$-P-Dec(j,$x_{j2}, c(m)$): The proxy side decryption algorithm. Before sending a ciphertext $(g^r, g^{rx}m)$ to a user j, the proxy finds j's server side key. Then the ciphertext is partially decrypted as $g^{rx}m \cdot (g^r)^{-x_{j2}} = g^{r(x-x_{j2})}m = g^{rx_{j1}}m$. $c'_j(m) = (g^r, g^{rx_{j1}}m)$ is sent to the user j.
- $\mathcal{PE}$-U-Dec$(x_{j1}, c'_j(m))$: After receiving $(g^r, g^{rx_{j1}}m)$, the user fully decrypts the ciphertext as $g^{rx_{j1}}m \cdot (g^r)^{-x_{j1}} = m$.

It is easy to see that the proxy encryption scheme is correct. We show that it is also secure under the DDH Assumption.

Theorem 1. *If the DDH problem is hard relative to $\mathbb{G}$, then the El Gamal based proxy encryption scheme is indistinguishable under chosen plaintext attack (IND-CPA) secure against the proxy. That is, for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$Succ^{\mathcal{A}}_{\mathcal{PE},P}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow \mathcal{PE}\text{-}Init(1^k), \\ (\mathcal{K}_u, \mathcal{K}_s) \leftarrow \mathcal{PE}\text{-}KeyGen(MSK, \mathcal{U}), \\ m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{PE}\text{-}U\text{-}Enc(\mathcal{K}_u,\cdot)}(\mathcal{K}_s) \\ b \xleftarrow{R} \{0,1\}, \\ c_i^*(m_b) = \mathcal{PE}\text{-}U\text{-}Enc(x_{i1}, m_b), \\ b' \leftarrow \mathcal{A}^{\mathcal{PE}\text{-}U\text{-}Enc(\mathcal{K}_u,\cdot)}(\mathcal{K}_s, c_i^*(m_b)) \end{array} \right] < \frac{1}{2} + negl(k)$$

Proof. Let's consider the following PPT adversary $\mathcal{A}'$ who attempts to solve the DDH problem using $\mathcal{A}$ as a sub-routine. Recall that $\mathcal{A}'$ is given $\mathbb{G}, q, g, g_1, g_2, g_3$ as input, where $g_1 = g^\alpha, g_2 = g^\beta$ and g_3 could be $g^{\alpha\beta}$ or g^γ for some random α, β, γ. $\mathcal{A}'$ does the following:

- $\mathcal{A}'$ sends $\mathbb{G}, q, g$ it received to $\mathcal{A}$ as the public parameters. It then chooses randomly $x_{i2} \overset{R}{\leftarrow} \mathbb{Z}_q$ for each $i \in \mathcal{U}$ and also computes $g^{x_{i1}} = g_1 \cdot g^{-x_{i2}}$. It then sends all (i, x_{i2}) to $\mathcal{A}$ and keeps all $(i, x_{i2}, g^{x_{i1}})$.
- Whenever $\mathcal{A}$ requires oracle access to the user encryption algorithm, it passes m to $\mathcal{A}'$. $\mathcal{A}'$ chooses randomly $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and replies with $(g^r, g^{rx_{i1}}m)$.
- At some point of time, $\mathcal{A}$ outputs m_0, m_1. $\mathcal{A}'$ chooses a random bit b and sends $g_2, g_2^{-x_{i2}}g_3 m_b$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs b'. If $b = b'$, $\mathcal{A}'$ outputs 1, otherwise outputs 0.

There are two cases to consider:
Case 1: If $g_3 = g^\gamma$, we know that g^γ is a random group element of $\mathbb{G}$ because γ is chosen at random. Then $g_2^{-x_{i2}}g_3 m_b$ is also a random group element of $\mathbb{G}$ and gives no information about m_b, i.e. the distribution of $g_2^{-x_{i2}}g_3 m_b$ is always uniform no matter what value m_b takes. g_2 also contains no information about m_b. So the adversary $\mathcal{A}$ must distinguish m_0, m_1 without additional information. The probability it can successfully output $b' = b$ is exactly $\frac{1}{2}$ when b is chosen uniformly randomly. $\mathcal{A}'$ outputs 1 iff $\mathcal{A}$ outputs $b' = b$, then we have:

$$Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1] = \frac{1}{2}$$

Case 2: If $g_3 = g^{\alpha\beta}$, because $g_2 = g^\beta$ and $g_2^{-x_{i2}}g_3 m_b = g^{-\beta x_{i2}}g^{\alpha\beta} = g^{\beta(\alpha - x_{i2})} = g^{\beta x_{i1}}$, then $g_2, g_2^{-x_{i2}}g_3 m_b$ is a proper ciphertext encrypted under $\mathcal{PE}$. In this case, we have:

$$Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1] = Succ^{\mathcal{A}}_{\mathcal{PE},P}(k)$$

If DDH problem is hard regarding $\mathbb{G}$, then the following is true

$$\begin{aligned} |Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1] - Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1]| &< negl(k) \\ Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1] &< \tfrac{1}{2} + negl(k) \end{aligned}$$

So we have $Succ^{\mathcal{A}}_{\mathcal{PE},P}(k) < \frac{1}{2} + negl(k)$. □

Theorem 1 basically says that without knowing user side keys or without the help from a user, the proxy cannot distinguish the ciphertexts in a chosen plaintext attack.

5.2 Keyword Encryption

From the proxy encryption scheme, we also obtain a keyword encryption scheme. The aim of this keyword encryption scheme is to securely encrypt keywords and also allow users to generate trapdoors which can be tested over the encrypted keywords in search queries. The keyword encryption scheme $\mathcal{KE}$ is defined as follows:

- $\mathcal{KE}$-$\mathsf{Init}(1^k)$: On input 1^k, run $\mathcal{PE}$-$\mathsf{Init}(1^k)$ to obtain $(\mathbb{G}, g, q, x)$. Compute $h = g^x$. Choose a collision resistant hash function H, a pseudorandom function f and a random key s for f. Publicise $(\mathbb{G}, g, q, h, H, f)$ and keep x, s securely as the master key.

- $\mathcal{KE}$-$\mathsf{Keygen}(MSK, i)$: Run $\mathcal{PE}$-$\mathsf{Keygen}(MSK, i)$ to obtain x_{i1}, x_{i2}. Transmit (x_{i1}, s) securely to user i and (i, x_{i2}) securely to the server.

- $\mathcal{KE}$-$\mathsf{U\text{-}Enc}(x_{i1}, kw)$: The user chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$, the keyword kw is encrypted as: $c_i^*(kw) = (\hat{c}_1, \hat{c}_2, \hat{c}_3)$ where $\hat{c}_1 = g^{r+\sigma}$, $\sigma = f_s(kw)$, $\hat{c}_2 = \hat{c}_1^{x_i1}$, $\hat{c}_3 = H(h^r)$.

- $\mathcal{KE}$-$\mathsf{P\text{-}Enc}(i, x_{i2}, c_i^*(kw))$. The proxy computes $c(kw) = (c_1, c_2)$ such that $c_1 = (\hat{c}_1)^{x_i2} \cdot \hat{c}_2 = \hat{c}_1^{x_{i2}+x_{i1}} = (g^{r+\sigma})^x = h^{r+\sigma}$ and $c_2 = \hat{c}_3 = H(h^r)$.

Note that there is no decryption algorithm for this keyword encryption scheme. This is because the ciphertexts of the keywords are only used for testing whether there is a match and do not need to be decrypted.

Next we will prove that this keyword encryption scheme is secure, i.e. that the proxy learns nothing about the encrypted keyword in a chosen plaintext attack.

Theorem 2. *If the DDH problem is hard relative to $\mathbb{G}$, then the keyword encryption scheme is IND-CPA secure against the proxy. That is, for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$Succ_{\mathcal{KE},P}^{\mathcal{A}}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow \mathcal{KE}\text{-}Init(1^k), \\ (\mathcal{K}_u, \mathcal{K}_s) \leftarrow \mathcal{KE}\text{-}KeyGen(MSK, \mathcal{U}), \\ kw_0, kw_1 \leftarrow \mathcal{A}^{\mathcal{KE}\text{-}U\text{-}Enc(\mathcal{K}_u, \cdot)}(\mathcal{K}_s) \\ b \overset{R}{\leftarrow} \{0,1\}, \\ c_i^*(kw_b) = \mathcal{KE}\text{-}U\text{-}Enc(x_{i1}, kw_b), \\ b' \leftarrow \mathcal{A}^{\mathcal{KE}\text{-}U\text{-}Enc(\mathcal{K}_u, \cdot)}(\mathcal{K}_s, c_i^*(kw_b)) \end{array} \right] < \frac{1}{2} + negl(k)$$

Proof. This proof again starts from a PPT adversary $\mathcal{A}'$ who attempts to challenge the DDH problem using $\mathcal{A}$ as a sub-routine. $\mathcal{A}'$ does the following:

- $\mathcal{A}'$ sets $h = g_1$ and also chooses H, f, s. It then passes $(\mathbb{G}, g, q, h, H, f)$ as the public parameters to $\mathcal{A}$. $\mathcal{A}'$ chooses randomly $x_{i2} \overset{R}{\leftarrow} \mathbb{Z}_q$ for each $i \in \mathcal{U}$ and computes $g^{x_{i1}} = hg^{-x_{i2}} = g^{\alpha - x_{i2}}$. It keeps $(i, g^{x_{i1}}, x_{i2})$ and passes each (i, x_{i2}) to $\mathcal{A}$.

- Whenever $\mathcal{A}$ requires oracle access to the user encryption algorithm, it passes kw to $\mathcal{A}'$. $\mathcal{A}'$ chooses $r \xleftarrow{R} \mathbb{Z}_q$ and replies with $c_i^*(kw) = (\hat{c}_1, \hat{c}_2, \hat{c}_3)$ where $\hat{c}_1 = g^{r+\sigma}$, $\sigma = f_s(kw)$, $\hat{c}_2 = (g^{x_{i1}})^{r+\sigma} = \hat{c}_1^{x_i 1}$ and $\hat{c}_3 = H(h^r)$.
- $\mathcal{A}$ generates two keywords kw_0, kw_1. $\mathcal{A}'$ chooses a random bit b and generates: $c_1 = g_2 \cdot g^{\sigma_b}$, $c_2 = g_3 g_2^{-x_{i2}} g^{x_{i1}\sigma_b}$ and $c_3 = H(g_3)$. $c_i^*(kw_b) = (c_1, c_2, c_3)$ is given to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b', if $b = b'$, $\mathcal{A}'$ outputs 1, otherwise outputs 0.

Case 1: $g_3 = g^\gamma$, in this case since both β and γ are random we have:

$$Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^\gamma) = 1] = \frac{1}{2}$$

Case 2: $g_3 = g^{\alpha\beta}$, in this case $c_2 = g_2^{x_{i1}} g^{\sigma_b x_{i1}}$ which is a valid ciphertext, then we have:

$$Pr[\mathcal{A}'(\mathbb{G}, q, g, g^\alpha, g^\beta, g^{\alpha\beta}) = 1] = Succ^{\mathcal{A}}_{\mathcal{KE},P}(k)$$

If DDH problem is hard regarding $\mathbb{G}$, we have $Succ^{\mathcal{A}}_{\mathcal{KE},P}(k) < \frac{1}{2} + negl(k)$. □

In Theorem 1 and 2 we proved that no PPT adversary can learn anything about the plaintext given a single ciphertext. The following corollary shows that it follows that no PPT adversary can learn anything given multiple ciphertexts.

Corollary 1. *Under the same assumptions, the proxy encryption scheme $\mathcal{PE}$ and the keyword encryption scheme $\mathcal{KE}$ have indistinguishable multiple encryption under a chosen-plaintext attack against the proxy.*

Proof. The difference is that now the adversary $\mathcal{A}$ is allowed to challenge the game with two vectors of messages $\vec{M}_0$ and $\vec{M}_1$ where $|\vec{M}_0| = |\vec{M}_1| = t$. Let $\vec{C}^{(i)} = C(m_0^1), ..., C(m_0^i), C(m_1^{i+1}), ..., C(m_1^t))$. It is clear that

$$Succ_{\mathcal{A}}(k) = \frac{1}{2} Pr[A(\vec{C}^0) = 0] + \frac{1}{2} Pr[A(\vec{C}^t) = 1]$$

Consider an adversary $\mathcal{A}'$ that tries to challenge the single encryption IND-CPA game using the corresponding multiple encryption adversary $\mathcal{A}$ as a sub-routine. Since the settings in the two games are the same, $\mathcal{A}'$ passes all the parameters it received to $\mathcal{A}$ and answers $\mathcal{A}$'s oracle queries by querying the oracle in its own game. Then $\mathcal{A}'$ obtains two vectors $\vec{M}_0$ and $\vec{M}_1$ from $\mathcal{A}$. $\mathcal{A}'$ chooses a random $i \xleftarrow{R} [1, t]$, uses m_0^i, m_1^i as its challenge and gets c_b^i as the result. It then constructs a hybrid vector $(c_0^1, ...c_0^{i-1}, c_b^i, c_1^{i+1}, ...c_1^t)$ and sends it to $\mathcal{A}$. It outputs the bit b' which is output by $\mathcal{A}$. It is easy to see that in this case $\mathcal{A}$ is required to distinguish $\vec{C}^{(i)}$ and $\vec{C}^{(i-1)}$ and the probability that $\mathcal{A}$ can distinguish the two is:

$$Succ^i_{\mathcal{A}}(k) = \frac{1}{2} Pr[\mathcal{A}(\vec{C}^{(i)}) = 0] + \frac{1}{2} Pr[\mathcal{A}(\vec{C}^{(i-1)}) = 1]$$

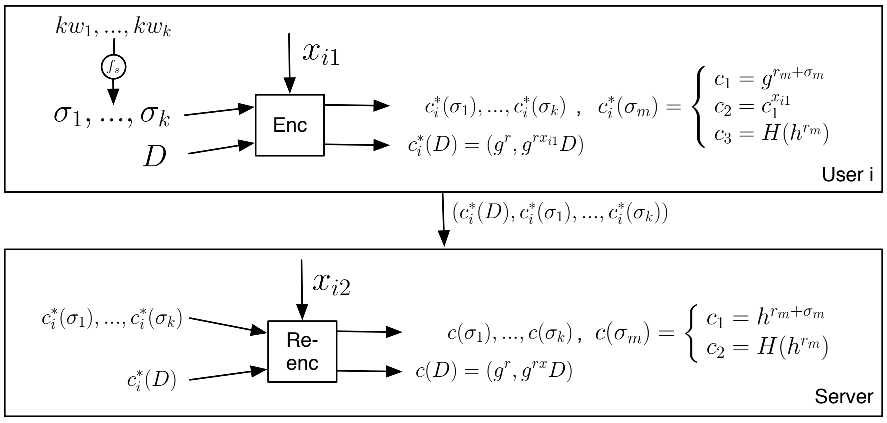
since i is randomly chosen, $Succ_{\mathcal{A}'}$ is

$$\begin{array}{rcl} Succ_{\mathcal{A}'}(k) & = & \sum_{i=1}^{t} Succ_{\mathcal{A}}^{i}(k) \cdot \frac{1}{t} \\ & = & \frac{1}{2t}(Pr[A(\vec{C}^0) = 0] + \sum_{i=1}^{t-1}(Pr(A(\vec{C}^i) = 0) + Pr[A(\vec{C}^i) = 1]) \\ & & +Pr[A(\vec{C}^t) = 1]) \\ & = & \frac{1}{t}(\frac{1}{2}Pr[A(\vec{C}^0) = 0] + \frac{1}{2}Pr[A(\vec{C}^t) = 1]) + \frac{t-1}{2t} \\ & = & \frac{1}{t}Succ_{\mathcal{A}}(k) + \frac{t-1}{2t} \end{array}$$

Since $Succ_{\mathcal{A}'}(k) < \frac{1}{2} + negl(k)$, it follows that $Succ_{\mathcal{A}}(k) < \frac{1}{2} + negl(k)$.

□

5.3 Multi-user Searchable Data Encryption

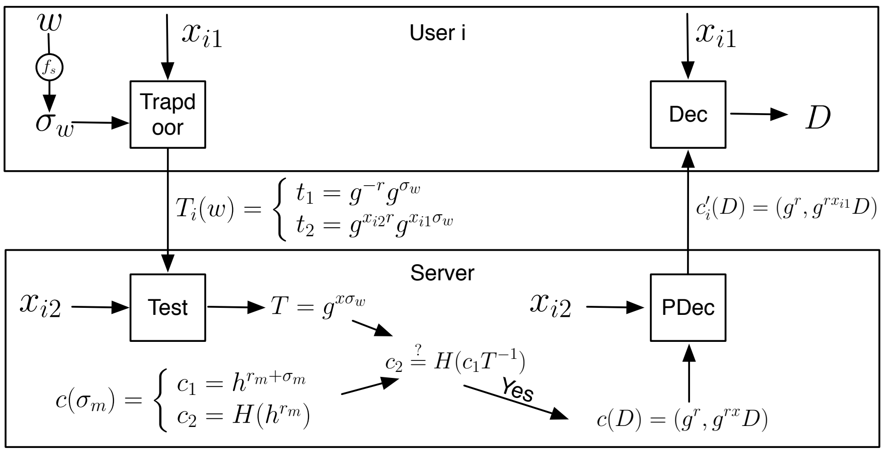


$kw_1, ..., kw_k$
f_s
x_{i1}
$\sigma_1, ..., \sigma_k$
D
Enc
$c_i^*(\sigma_1), ..., c_i^*(\sigma_k)$, $c_i^*(\sigma_m) =$
$c_1 = g^{r_m+\sigma_m}$
$c_2 = c_1^{x_{i1}}$
$c_3 = H(h^{r_m})$
$c_i^*(D) = (g^r, g^{rx_{i1}}D)$
User i
$(c_i^*(D), c_i^*(\sigma_1), ..., c_i^*(\sigma_k))$
x_{i2}
$c_i^*(\sigma_1), ..., c_i^*(\sigma_k)$
$c_i^*(D)$
Re-enc
$c(\sigma_1), ..., c(\sigma_k)$, $c(\sigma_m) =$
$c_1 = h^{r_m+\sigma_m}$
$c_2 = H(h^{r_m})$
$c(D) = (g^r, g^{rx}D)$
Server

Figure 2: Data Encryption Process

Now we are ready to present our multi-user searchable data encryption scheme $\mathcal{SE}$. It is constructed as follows:

- Init(1^k): the KMS takes as input the security parameter 1^k and invokes $\mathcal{KE}$-Init(1^k), outputs Params=$(\mathbb{G}, g, q, h, H, f)$ and a master key set MSK= (x, s).
- Keygen(MSK,i): the KMS takes as input the master key set and a user identifier i, generates two keys Ku_i, Ks_i by invoking $\mathcal{KE}$-Keygen(MSK, i). Each Ku_i is sent securely to the user i and each Ks_i is sent securely to the server along with the user ID. The server updates its user-key mapping set $\mathcal{K}s = \mathcal{K}s \cup (i, Ks_i)$.
- Enc(Ku$_i$,D,kw(D)): as shown in Figure 2, run by a user with his key Ku_i to compute $c_i^*(D) = \mathcal{PE}$-U-Enc(Ku_i, D) and for each keyword in $kw(D)$



w
f_s
σ_w
x_{i1}
Trapdoor
User i
x_{i1}
Dec
D
$T_i(w) = \begin{cases} t_1 = g^{-r} g^{\sigma_w} \\ t_2 = g^{x_{i2} r} g^{x_{i1}\sigma_w} \end{cases}$
$c_i'(D) = (g^r, g^{r x_{i1}} D)$
Server
x_{i2}
Test
$T = g^{x\sigma_w}$
x_{i2}
PDec
$c_2 \overset{?}{=} H(c_1 T^{-1})$
$c(\sigma_m) = \begin{cases} c_1 = h^{r_m + \sigma_m} \\ c_2 = H(h^{r_m}) \end{cases}$
Yes
$c(D) = (g^r, g^{rx} D)$

Figure 3: Search and Decryption Process

$c_i^*(kw) = \mathcal{KE}\text{-U-Enc}(Ku_i, kw)$, then outputs ciphertext $c_i^*(D, kw(D)) = (c_i^*(D), c_i^*(kw_1), ..., c_i^*(kw_k))$, where $k = |kw(D)|$.[2]

- Re-enc(i,Ks$_i$,c$_i^*$(D,kw(D))): as shown in Figure 2, run by the server to re-encrypt a ciphertext tuple from a user i. The server finds the corresponding key $Ks_i = x_{i2}$ and outputs re-encrypted ciphertext $c(D, kw(D)) = (c(D), c(kw_1), ..., c(kw_k))$, where $c(D) = \mathcal{PE}\text{-P-Enc}(i, Ks_i, c_i^*(D))$ and $c(kw_k) = \mathcal{KE}\text{-P-Enc}(i, Ks_i, c_i^*(kw_k))$ for each $c_i^*(kw_k)$. $c(D, kw(D))$ is then inserted into the encrypted data storage $E(\mathcal{D}) = E(\mathcal{D}) \cup c(D, kw(D))$.
- Trapdoor(Ku$_i$,w): as shown in Figure 3, run by a user which takes the user's key $Ku_i = (x_{i1}, s)$ and a keyword w and outputs a trapdoor. The user chooses a random number $r \overset{R}{\leftarrow} \mathbb{Z}_q$, computes $T_i(w) = (t_1, t_2)$ for the word such that $t_1 = g^{-r} g^{\sigma_w}$ and $t_2 = h^r g^{-x_{i1} r} g^{x_{i1}\sigma_w} = g^{x_{i2} r} g^{x_{i1}\sigma_w}$ where $\sigma_w = f_s(w)$.
- Pre-dec(i,Ks$_i$,c(D)): as shown in Figure 3, run by the server. $c(D)$ is partially decrypted as $c_i'(D) = \mathcal{PE}\text{-P-Dec}(i, Ks_i, c(D))$. $c_i'(D)$ is returned to the user i.
- Search(i,T$_i$(w), $E(\mathcal{D})$,Ks$_i$): as shown in Figure 3, run by the server. On receiving $T_i(w) = (t_1, t_2)$, find the corresponding $Ks_i = x_{i2}$ and compute $T = t_1^{x_{i2}} \cdot t_2 = g^{x\sigma_w}$. For each $c(D, kw(D)) \in E(\mathcal{D})$, test each $c(kw) =$

[2]For the sake of simplicity, in the description of the algorithm, the document is encrypted directly under $\mathcal{PE}$-U-Enc. However in practice, the document can be encrypted by a more efficient hybrid encryption scheme, where a secure symmetric cipher is chosen to encrypt the document under a random key and the random key is then encrypted under $\mathcal{PE}$-U-Enc. Using such a hybrid scheme will not decrease the security of the scheme as long as the symmetric cipher is IND-CPA secure.

$(c_1, c_2) = (h^r g^{x\sigma_{kw}}, H(h^r))$, if $c_2 = H(c_1 \cdot T^{-1})$ then $w = kw$ with a overwhelming probability, i.e. a match is found. Then $c(D)$ is partially decrypted as $c'_i(D) = \mathcal{PE}\text{-}\mathsf{P\text{-}Dec}(i, Ks_i, c(D))$. $c'_i(D)$ is returned to the user i.

- $\mathsf{Dec(Ku_i,c'_i(D))}$: as shown in Figure 3, a user decrypts all $c'_i(D)$ in the result set by invoking $\mathcal{PE}\text{-}\mathsf{U\text{-}Dec}(Ku_i, c'_i(D))$.

- $\mathsf{Revoke(i)}$ run by the server. Given i, the server updates its user-key mapping set $\mathcal{K}s = \mathcal{K}s \setminus (i, Ks_i)$.

It is easy to see that the correctness of the searching algorithm follows directly from the collision resistant property of H. If $w = kw$, then $c_1 \cdot T^{-1} = h^r g^{x\sigma_{kw}} \cdot g^{-x\sigma_w} = h^r$ for sure and therefore $H(h^r) = H(c_1 \cdot T^{-1})$. This means the search algorithm will not produce false negative results. In the case that a false positive occurs, we have $w \neq kw$ which means $c_1 \cdot T^{-1} = h^r g^{x\sigma_{kw}} \cdot g^{-x\sigma_w} \neq h^r$ and we also have $H(h^r) = H(c_1 \cdot T^{-1})$. This equally means we find a collision of H. Since the probability of finding a collision is negligible for a collision resistant hash function, the false positive rate is also negligible.

Note that the re-encryption step here is only used for controlling the write operation. That is, only an authorised user should be able to insert a document into the storage. If another mechanism for the same purpose is available, this step can be omitted. Because $h = g^x$ is available as a public parameter in the system, the user can encrypt a document and the keyword set as $c(D, kw(D))$ directly using g^x. Taking out this step will also decrease the server side computation load.

We now analyse the security of the multi-user searchable encryption scheme according to Definition 10.

Theorem 3. *If the DDH problem is hard relative to $\mathbb{G}$, then $\mathcal{SE}$ is a non-adaptive indistinguishable secure multi-user searchable encryption scheme.*

Proof. First let's consider an adversary $\mathcal{A}'$ who wants to challenge the proxy encryption IND-CPA game using $\mathcal{A}$ as a sub-routine. $\mathcal{A}'$ does the following:

- $\mathcal{A}'$ receives public parameters $\mathbb{G}, q, g$ and the server side keys. It then picks a random user ID u and queries its oracle with $m = 1$ to obtain the ciphertext $g^r, g^{rx_{u1}}$. $\mathcal{A}'$ then computes $g^{rx_{u1}} g^{rx_{u2}} = g^{rx}$. It can then compute for every user i, $g^{rx_{i1}} = g^{rx} g^{-rx_{i2}}$ since it knows x_{i2}.

- $\mathcal{A}'$ sets $g_0 = g^r$ and $h = g^{rx} = g_0^x$. $\mathcal{A}'$ also chooses H, f, s. $\mathcal{A}'$ sends $(\mathbb{G}, q, g_0, h, H, f)$ to $\mathcal{A}$ and also the server side keys.

- Recall that H_i is a sequence $(\mathcal{D}, q_1^{u_1}, ..., q_i^{u_i})$ $\mathcal{A}'$ can generate a view of the history by doing the following steps:

  1. For each document and its associated keyword set $(D, kw(D))$ in $\mathcal{D}$, do the following: for each keyword kw, choose a random number z, compute $\hat{c}_1 = g_0^{z+\sigma}$, $\hat{c}_2 = (g^{rx_{i1}})^{z+\sigma} = (g_0^{z+\sigma})^{x_{i1}}$, $\hat{c}_3 = H(h^z)$ where

$\sigma = f_s(kw)$. Then compute (c_1, c_2) as $c_1 = \hat{c}_1^{x_{i2}} \cdot \hat{c}_2, c_2 = \hat{c}_3$. (c_1, c_2) is the ciphertext for this keyword. It then chooses a random number $\tilde{r}$ and computes $(g_0^{\tilde{r}}, g_0^{\tilde{r}x} D)$.

2. For each insert query $q^{u_i} = (D, kw(D))$, compute $\hat{c}_1, \hat{c}_2, \hat{c}_3$ for each keyword as above and $(g_0^{\tilde{r}}, g_0^{\tilde{r}x_{i1}} D)$. $(\hat{c}_1, \hat{c}_2, \hat{c}_3)$ and $(g_0^{\tilde{r}}, g_0^{\tilde{r}x_{i1}} D)$ are valid ciphertext for the insert query.

3. For each search query $q^{u_i} = w$, construct the result set $rs(w)$ for the query. Choose a random number $\hat{r}$, compute $\sigma_w = f_s(w)$ and generate $t_1 = g_0^{-\hat{r}} g_0^{\sigma_w}$, $t_2 = g_0^{x_{i2}\hat{r}} g_0^{x_{i1}\sigma_w}$. (t_1, t_2) is the trapdoor. For each document whose identifier $id(D)$ is in $rs(w)$, find the encrypted version (the document must be encrypted in the above two steps) and partially decrypt it using the server side key.

- $\mathcal{A}$ outputs H_{i0}, H_{i1}. $\mathcal{A}'$ encrypts every keyword and trapdoor in H_{i1} by itself and challenges its game with $\vec{D}_0$, $\vec{D}_1$ which returns a result $\mathcal{PE}(\vec{D}_b)$ where $\vec{D}_b$ is the vector of all document in H_{ib}. It combines the results to form a view $(\mathcal{PE}(\vec{D}_b), \mathcal{KE}(\vec{kw}_1), Trapdoor(\vec{w}_1))$ (in the correct order as in the trace) and returns it to $\mathcal{A}$.

- $\mathcal{A}'$ outputs b' which is output by $\mathcal{A}$.

By Theorem 1 and Corollary 1, it is clear that:

$$\begin{array}{rcl} \frac{1}{2} + negl(k) & > & Succ_{\mathcal{PE}}^{\mathcal{A}'}(k) \\ & = & \frac{1}{2} Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw}_1), Trapdoor(\vec{w}_1))) = 0] \\ & & + \frac{1}{2} Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_1), \mathcal{KE}(\vec{kw}_1), Trapdoor(\vec{w}_1))) = 1] \end{array}$$

Now let us consider another adversary $\mathcal{A}''$ who wants to distinguish the pseudorandom function f using $\mathcal{A}$ as a sub-routine. It does the following:

- It generates $(\mathbb{G}, g, q, h, H)$ as public parameters, sends the parameters to $\mathcal{A}$ along with f. For each user i, it chooses randomly x_{i1}, x_{i2} such that $x_{i1} + x_{i2} = x$. Send all (i, x_{i2}) to $\mathcal{A}$ and keeps all (i, x_{i1}, x_{i2}).

- $\mathcal{A}$ outputs H_{i0}, H_{i1}. $\mathcal{A}''$ encrypts all the documents in H_{i0} as $\mathcal{PE}(\vec{D}_0)$. It also chooses a random b, then consults its oracle and encrypts all the keywords and trapdoors in H_{ib}. It combines the results to form a view $(\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw}_b), Trapdoor(\vec{w}_b))$ and returns it to $\mathcal{A}$.

- $\mathcal{A}$ outputs b'. $\mathcal{A}''$ outputs 1 if $b' = b$ and outputs 0 otherwise.

There are two cases to consider:
Case 1: the oracle in $\mathcal{A}''$'s game is the pseudorandom function f, then

$$\begin{array}{rcl} Pr[\mathcal{A}''^{f_s(\cdot)}(1^k) = 1] & = & \frac{1}{2} Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw}_0), Trapdoor(\vec{w}_0))) = 0] \\ & & + \frac{1}{2} Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw}_1), Trapdoor(\vec{w}_1))) = 1] \end{array}$$

Case 2: the oracle in $\mathcal{A}'$'s game is a random function F, then for each distinct keyword w_b, σ_{w_b} is completely random to $\mathcal{A}$. Moreover we know the traces are identical, so $\mathcal{KE}(\vec{kw_b})$ and $Trapdoor(\vec{w}_b)$ are completely random to $\mathcal{A}$. In this case

$$Pr[\mathcal{A}''^{F(\cdot)}(1^k) = 1] = \frac{1}{2}$$

Since f is a pseudorandom function, by definition

$$\begin{aligned} |Pr[\mathcal{A}''^{f_s(\cdot)}(1^k) = 1] - Pr[\mathcal{A}'^{F(\cdot)}(1^k) = 1]| &< negl(k) \\ Pr[\mathcal{A}''^{f_s(\cdot)}(1^k) = 1] &< \tfrac{1}{2} + negl(k) \end{aligned}$$

Sum $Succ^{\mathcal{A}'}_{\mathcal{PE}}(k)$ and $Pr[\mathcal{A}''^{f_s(\cdot)}(1^k) = 1]$ up, we have

$$\begin{aligned} 1 + negl(k) &> \tfrac{1}{2}Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw_0}), Trapdoor(\vec{w}_0))) = 0] \\ &\quad + \tfrac{1}{2}Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw_1}), Trapdoor(\vec{w}_1))) = 1] \\ &\quad + \tfrac{1}{2}Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_0), \mathcal{KE}(\vec{kw_1}), Trapdoor(\vec{w}_1))) = 0] \\ &\quad + \tfrac{1}{2}Pr[\mathcal{A}((\mathcal{PE}(\vec{D}_1), \mathcal{KE}(\vec{kw_1}), Trapdoor(\vec{w}_1))) = 1] \\ &= \tfrac{1}{2} + Succ^{\mathcal{A}}_{\mathcal{SE}}(k) \end{aligned}$$

Therefore we have $Succ^{\mathcal{A}}_{\mathcal{SE}}(k) < \frac{1}{2} + negl(k)$.

□

5.4 User Revocation

In our scheme, revoking a user's access is straightforward. The KMS sends a request to the server to remove the server side keys. After the keys have been removed, the user cannot access the data unless the KMS generates new keys for him.

Inserting data and generating trapdoors must be done by the server and must involve both the user side keys and the server side keys. If we assume the server is honest, it is trivial to show that after the server removes the revoked user's server side keys, the user cannot insert new data or search the data anymore.

The revocation of read permission, however, is not so trivial. Although a revoked user cannot directly access the documents stored on the server, he may be able to eavesdrop the communication between the server and the authorised users, namely the insert queries and the result sets of search queries. In the following theorem, we prove that a revoked user cannot distinguish insert queries generated by authorised users even if he has a revoked key set.

Theorem 4. *If the DDH problem is hard relative to $\mathbb{G}$, then the revoked users cannot distinguish the insert queries generated by authorised users. That is, for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$Succ^{\mathcal{A}}_{\mathcal{SE},U}(k) = Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow Init(1^k), \\ (\mathcal{K}_u, \mathcal{K}_s) \leftarrow KeyGen(MSK, \mathcal{U}), \\ (q_0, q_1) \leftarrow \mathcal{A}^{Enc(\mathcal{K}_u, \cdot)}(Ku_i), \\ j \leftarrow \mathcal{U}, j \neq i, b \xleftarrow{R} \{0,1\}, \\ b' \leftarrow \mathcal{A}^{Enc(\mathcal{K}_u, \cdot)}(Ku_i, V_{Ku_j}(q_b)) \end{array} \right] < \tfrac{1}{2} + negl(k)$$

where $q_b = (D_b, kw(D_b))$ *is an insert query such that* $Tr(q_0) = Tr(q_1)$.

Proof. Consider an adversary $\mathcal{A}'$ who attempts to challenge the DDH problem using $\mathcal{A}$ as a sub-routine.

- $\mathcal{A}'$ sets $h = g_1$ and also chooses H, f, s. It passes $(\mathbb{G}, g, q, h, H, f)$ as the public parameters to $\mathcal{A}$. $\mathcal{A}'$ also chooses for each authorised user u a random x_{u2} from $\mathbb{Z}_q$ and computes $g^{x_{u1}} = hg^{-x_{u2}}$. It keeps all $(u, g^{x_{u1}}, x_{u2})$. It then chooses a random x_{i1} from $\mathbb{Z}_q$ and sends (x_{i1}, s) to $\mathcal{A}$ as its user side key set.
- $\mathcal{A}'$ can answer $\mathcal{A}$'s oracle query for $q = (D, kw(D))$ as follows:
  1. For the document D, choose a random r from $\mathbb{Z}_q$ and compute $g^r, g^{rx_{u1}}D$.
  2. For each keyword $kw \in kw(D)$, choose a random r from $\mathbb{Z}_q$ and compute $\hat{c}_1 = g^{r+\sigma}$, $\sigma = f_s(kw)$, $\hat{c}_2 = (g^{x_{u1}})^{r+\sigma}$ and $c_3 = H(h^r)$.
- After a certain time, $\mathcal{A}$ generates q_0, q_1. q_0 and q_1 can be viewed as two vectors of equal size t where the first element in $\vec{q_b}$ is the document D_b and the rest $t-1$ elements are the keywords. $\mathcal{A}'$ chooses a random index $n \overset{R}{\leftarrow} [1, t]$ and a random bit b. It constructs a hybrid ciphertext vector $\vec{C}^n$ as follows:
  1. Encrypt the elements $q_0^1, ..., q_0^{n-1}$ in $\vec{q_0}$ and the elements $q_1^{n+1}, ..., q_1^t$ in $\vec{q_1}$ as above.
  2. Encrypt q_b^n as $(g_2, g_3 g_2^{-x_{u2}} D_b)$ when $n = 1$ and as (c_1, c_2, c_3) where $c_1 = g_2 g^{\sigma_b}$, $c_2 = g_3 g_2^{-x_{u2}} g^{x_{u1}\sigma_b}$ and $c_3 = H(g_3)$ when $n > 1$.
- $\vec{C}^n$ is returned to $\mathcal{A}$. $\mathcal{A}'$ then outputs the bit b' output by $\mathcal{A}$.

Use a hybrid argument similar to that used in the proof of Corollary 1, when $g_3 = g^c$, the probability of $\mathcal{A}$ outputs 1 is exactly $\frac{1}{2}$. When $g_3 = g^{ab}$, the probability of $\mathcal{A}$ outputs 1 is $\frac{1}{t} Succ^{\mathcal{A}}_{\mathcal{SE},U}(k) + \frac{t-1}{2t}$. Since $\mathcal{A}'$ outputs 1 when $\mathcal{A}$ outputs 1, if the DDH problem is hard, then it follows that $Succ^{\mathcal{A}}_{\mathcal{SE},U}(k) < \frac{1}{2} + negl(k)$. □

Theorem 5 states that a revoked user cannot distinguish the result sets of search queries. It follows directly from Theorem 4. Intuitively, each partially decrypted document in the result set returned to a user i is $c_i'(D) = (g^r, g^{rx_{i1}}D)$. And let us also consider an insert query submitted by the same user $q^i = (D, kw(D))$. The view of the query $V_{Ku_i}(q^i) = ((g^{\hat{r}}, g^{\hat{r}x_{i1}}D), c_i^*(kw(D)))$. It is clear that if the adversary can distinguish the search query result sets, it can also distinguish the insert queries, which contradicts the theorem we have proved.

Theorem 5. *If the DDH problem is hard relative to* $\mathbb{G}$, *then the the revoked users cannot distinguish the search result sets retrieved by authorised users. That is, for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function negl such that*

$$Succ^{\mathcal{A}}_{\mathcal{SE},U}(k) = Pr\left[b' = b \,\middle|\, \begin{array}{c} (Params, MSK) \leftarrow Init(1^k), \\ (\mathcal{K}_u, \mathcal{K}_s) \leftarrow KeyGen(MSK,\mathcal{U}), \\ (R_0, R_1) \leftarrow \mathcal{A}^{Enc(\mathcal{K}_u,\cdot)}(Ku_i), \\ j \leftarrow \mathcal{U}, j \neq i, b \xleftarrow{R} \{0,1\}, \\ b' \leftarrow \mathcal{A}^{Enc(\mathcal{K}_u,\cdot)}(Ku_i, c'_j(R_b)) \end{array} \right] < \frac{1}{2} + negl(k)$$

where $R_b = \{D_{b1}, ..., D_{bn}\}$ *is a search query result set such that* $|D_{0i}| = |D_{1i}|$ *for each* $D_{bi} \in R_b$*.*

Using the same techniques we used in proving Theorem 4, we can also prove Theorem 5. The proof is omitted.

It follows from the above theorems that as long as the server is honest and the authorised users can protect their keys, revocability is guaranteed.

6 Discussion

6.1 Authentication

In our scheme, each authorised user has a secret key x_{i1} and the server holds the corresponding part x_{i2} such that $x_{i1} + x_{i2} = x$ where x is the master key in the system. This key pair can be used for mutual authentication and establishing secure channels. An example protocol to demonstrate the idea is presented below:

1. A user i sends a request to the server which contains his id: $\{i\}$.

2. The server retrieves the user's server side key and generates a fresh nonce N, replies the user with $\{g^r, h^r g^{-rx_{i2}} N\}$.

3. The user decrypts the message $g^{-rx_{i1}} h^r g^{-rx_{i2}} N = N$, then generates a fresh session key K and replies with $\{g^{r'}, h^{r'} g^{-r' x_{i1}} m\}$ where $m = (N - 1, K)$.

4. The server decrypts the message, checks $N - 1$ is correct, then the authentication succeeds and both parties have the shared session key K.

The benefit is obvious: encryption prevents the server from learning what is stored, but only gives us very basic controls over the users. We can control which users can access the data by controlling the keys. But this all-or-nothing mechanism may not be sufficient if we want to assign permissions to the users based on their need to know. Authentication makes it possible to implement fine-grained user access control mechanisms on top of the encryption scheme. Since we believe the server is honest, we can let the server authenticate the users and enforce our authorisation policies. It is also necessary if we need auditing and accounting.

6.2 Collusion Attacks

The main concern with proxy encryption schemes comes from a collusion attack. If a user colludes with the server, they can easily recover the master keys by combining their keys. Although some work has been done in [5] using bilinear maps to prevent the colluded parties from recovering the master key, the colluding parties are still able to decrypt the ciphertext with a weak secret they can recover. Theoretically, the design of fully collusion-resistant proxy encryption schemes is still an open problem.

However in practice, we can lower the risk to an acceptable level by implementing other mechanisms. One possible solution is to store user side keys in smart cards (or other tamper resistant devices). Another possible solution is to split the master key into multiple shares and introduce multiple servers from several SSPs competing with each other. The assumption is that a collusion attack is only possible when all the SSPs involved collude and the SSPs are competitors thus are unlikely to co-operate in such a collusion. We explain this solution further by an example.

Consider now that we have two servers and they belong to different SSPs. One server, say α, will be used as the primary server which stores the encrypted data. The other server, say β, is used for countering collusion and needs to be involved in the computation but does not necessarily need to store the data. The first step is still to run $\mathsf{Init}(1^k)$ to generate parameters $(\mathbb{G}, g, q, h, H, f)$ and a master key set key $\mathsf{MSK}= (x, s)$. This is the same as in section 5.3. Then we need to generate the user side key set and the server side key sets. The algorithm $\mathsf{Keygen}(MSK, i)$ is modified a little bit because we have two servers. Remember when we have only one server, x in the master key is split into x_{i1} and x_{i2} such that $x_{i1} + x_{i2} = x$. x_{i1} is given to the user and x_{i2} is given to the server. If we have two servers α and β we split x_{i2} into $x_{i\alpha} + x_{i\beta} = x_{i2}$ and send $x_{i\alpha}$ to the server α and $x_{i\beta}$ to the server β. It is easy to see that $x_{i1} + x_{i\alpha} + x_{i\beta} = x$.

The primary server α runs the server side algorithms in section 5.3 without any change. Only the user side algorithms are changed in order to add collusion control.

To insert a document D and the associated keyword list $kw(D)$, the user first runs $\mathsf{Enc}(Ku_i, D, kw(D))$. This algorithm is unchanged. So the document is encrypted as $(g^r, g^{rx_{i1}}D)$ and each keyword kw_m is encrypted as (c_{m1}, c_{m2}, c_{m3}) where $c_{m1} = g^{r_m+\sigma_m}, c_{m2} = c_{m1}^{x_{i1}}, c_{m3} = H(h^{r_m})$. Now the user needs to run an additional interaction with the server β. The user sends g^r in the ciphertext of the document and c_{m1} in the ciphertext of each keyword to β. β uses its key $x_{i\beta}$ to compute $g^{rx_{i\beta}}$ and $c_{m1}^{x_{i\beta}}$ and returns them back to the user. The user then updates the ciphertexts by multiplying $g^{rx_{i\beta}}$ with $g^{rx_{i1}}D$ and $c_{m1}^{x_{i\beta}}$ with c_{m2}. The ciphertext of the document becomes $(g^r, g^{r(x_{i1}+x_{i\beta})}D)$ and the ciphertext of each keyword becomes $(c_{m1}, c_{m1}^{x_{i1}+x_{i\beta}}, c_{m3})$. The updated ciphertexts are sent to server α.

After receiving the ciphertexts, the server α runs the re-encryption algorithm. This algorithm is unchanged. α computes $g^{rx_{i\alpha}}$ and $c_{m1}^{x_{i\alpha}}$ using the corresponding key $x_{i\alpha}$. After re-encryption, the ciphertext of the document becomes $(g^r, g^{rx}D)$ because $g^{rx_{i\alpha}}g^{r(x_{i1}+x_{i\beta})}D = g^{r(x_{i1}+x_{i\beta}+x_{i\alpha})}D$ and $x_{i1} + x_{i\beta} + x_{i\alpha} = x$. The ciphertext of a keyword kw_m becomes $(h^{r_m+\sigma_m}, H(h^{r_m}))$ because $c_{m1}^{x_{i\alpha}}c_{m1}^{x_{i1}+x_{i\beta}} = c_{m1}^{x_{i1}+x_{i\beta}+x_{i\alpha}} = c_{m1}^{x} = g^{(r_m+\sigma_m)x} = h^{r_m+\sigma_m}$. The ciphertexts are inserted into the encrypted data storage.

To search, the user must generate a trapdoor of a keyword w with the help from β. The user first computes $t_1 = g^{-r}g^{\sigma_w}$ and sends it to β. β computes $t_1^{x_{i\beta}}$ and returns it back to the user. The user then computes $t_2 = h^r t_1^{x_{i1}} t_1^{x_{i\beta}} = h^r t_1^{x_{i1}+x_{i\beta}} = g^{xr}g^{-r(x_{i1}+x_{i\beta})}g^{\sigma_w(x_{i1}+x_{i\beta})} = g^{rx_{i\alpha}}g^{\sigma_w(x_{i1}+x_{i\beta})}$. The user queries with (t_1, t_2).

Given the trapdoor (t_1, t_2), α runs the search algorithm which is the same as in section 5.3. α computes $T = t_1^{x_{i\alpha}}t_2 = g^{-rx_{i\alpha}}g^{\sigma_w x_{i\alpha}}g^{rx_{i\alpha}}g^{\sigma_w(x_{i1}+x_{i\beta})} = g^{\sigma_w(x_{i1}+x_{i\beta}+x_{i\alpha})} = g^{\sigma_w x} = h^{\sigma_w}$. Then α tests for each encrypted keyword $c(kw) = (h^{r_m+\sigma_m}, H(h^{r_m}))$ to see whether $H(h^{r_m}) = H(h^{r_m+\sigma_m}T^{-1})$. If so, a match is found and the encrypted document will be returned to the user.

To decrypt a document, the document is first partially decrypted by α. α runs the pre-decryption algorithm in section 5.3. Given the ciphertext $(g^r, g^{rx}D)$, α computes $g^{-rx_{i\alpha}}$ and $g^{rx-rx_{i\alpha}}D = g^{r(x_{i1}+x_{i\beta})}D$. $(g^r, g^{r(x_{i1}+x_{i\beta})}D)$ is returned to the user. The user then sends g^r to β and gets $g^{-rx_{i\beta}}$ back. The user can then fully decrypt the document by first computing $g^{-rx_{i1}}$ and then multiplying $g^{r(x_{i1}+x_{i\beta})}D$ with $g^{-rx_{i1}}$ and $g^{-rx_{i\beta}}$.

This approach can be easily extended to n servers by splitting x_{i2} into n shares such that they add up to x_{i2} and give one share to each server. To encrypt, search and decrypt, the user must first interact with all $n-1$ collusion control servers and then with the primary server.

Introducing more servers will not weaken the security of the system. It can be proved by a simple reduction: if in an n-server setting an adversary $\mathcal{A}'$ who controls up to n servers can break the system with non-negligible probability, then in a single-server setting an adversary $\mathcal{A}$ can use $\mathcal{A}'$ to break the system with non-negligible probability. $\mathcal{A}$ just splits x_{i2} randomly into n shares and gives $\mathcal{A}'$ the number of shares it needs. That is, with multiple servers, the scheme is at least as secure as with one server. And obviously, splitting the master key into multiple shares makes collusion harder: it requires all the servers and at least one user to collude together to recover the master key.

7 Implementation and Performance

We implemented a prototype of our scheme in Java 1.6 using the big integer class provided by the standard API. We used a 1024-bit prime p, a 160-bit prime q and SHA-1 as the hash function to encrypt a single table database. To compare, we also measured the performance of the enhanced RSA-based scheme that we proposed in the earlier version of this paper [13]. The RSA-based scheme was implemented with a 1024-bit RSA key pair for the proxy encryption and a 1024-bit prime p, a 160-bit prime q and SHA-1 for the keyword encryption. The tests were executed on a Intel Core2 Duo 2.53 GHz MacBook Pro with 4 GB of RAM.

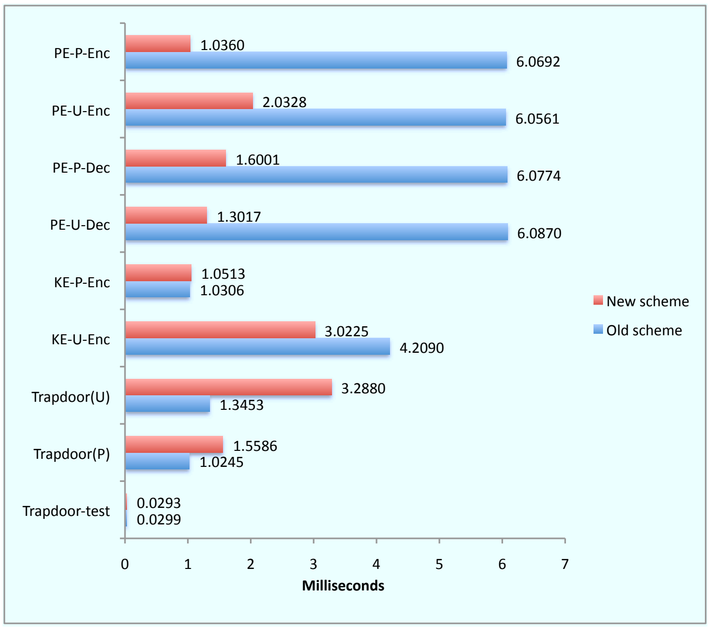


Figure 4: Performance Evaluation of Individual Operations

In the first experiment, we measured the execution time of each of the following operations: (1) **PE-P-Enc**: the server side proxy encryption; (2) **PE-U-Enc**: the user side proxy encryption; (3) **PE-P-Dec**: the server side proxy decryption; (4) **PE-U-Dec**: the server side proxy decryption; (5) **KE-P-Enc**: the server side keyword encryption; (6) **KE-U-Enc**: the user side keyword encryption; (7) **Trapdoor(U)**: the user side trapdoor generation; (8) **Trapdoor(P)**: the server side trapdoor generation; (9) **Trapdoor-test**: the trapdoor/keyword match test.

The chart in Figure 4 shows the results. The time on the X-axis is given in milliseconds. The figure provides the average time for 10,000 executions. We can see from the chart that the proxy encryption/decryption operations (PE-P-Enc, PE-U-Enc, PE-P-Dec and PE-U-Dec) are more efficient in the new scheme. This is because to achieve the same security level, the exponents used in the El Gamal based scheme are smaller than those used in the RSA based scheme (160-bit vs. 1024-bit). The test results of the server side keyword encryption operations in the two schemes are quite close. The performance of the user side keyword encryption operation is better in the new scheme. The difference comes from the fact that in the new scheme this operation requires two modular exponentiations while in the old scheme requires three. The trapdoor generation is slower in the new scheme because the algorithm is slightly more complex and requires more arithmetic operation. Overall, the El Gamal based scheme is

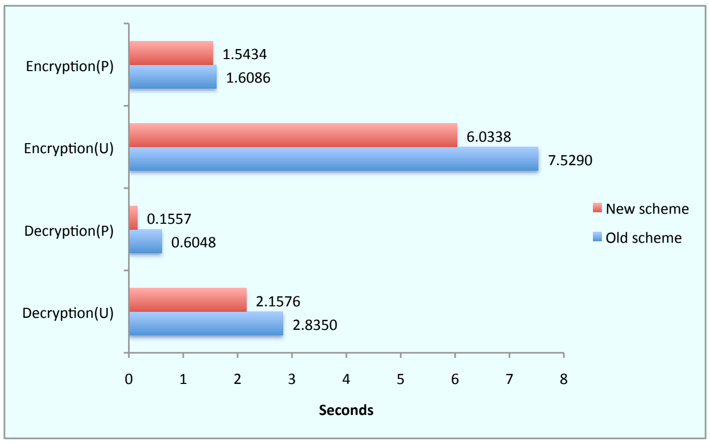


Figure 5: Performance Evaluation of Bulk Encryption/Decrytpion

better than the RSA based scheme in performance.

In the second experiment measured the performance of bulk encryption/decryption. The data set we used was 100 pdf files, the total size was 99.2 MB. Each document was associated with 10 keywords, i.e. 1000 keywords to encrypt in total. The files were encrypted with 128-bit AES and the AES keys were encrypted using proxy encryption. The results are shown in Figure 5. In the figure, **P** means server side operation and **U** means user side operation. We can see that the performance of the new scheme is better than the old one.

The third experiment was to measure the search times with various sized databases. We used databases containing 10,000 keywords, 100,000 keywords and 1,000,000 keywords. We used two strategies in searching the databases: the first one was to search the entire database with a single thread and the second one was to divide the database into several equal-sized parts and search all the parts in parallel using multiple threads. The results are shown in Figure 6. In the figure, **MT** means using the multi-threaded strategy. We can see that in both cases the performance of the new scheme is similar to or slightly better than the old one. We can also see that with multi-core computers, multi-threading does boost the performance. In our case, the computer used in the experiment has two cores and the search time was nearly halved with the largest database.

We also measured the performance with a different number of collusion control servers as described in 6.2. The number of collusion control servers ranges from 0 to 9. The collusion control servers ran on dedicated machines each with an Intel Core2 Duo 2.13 GHz CPU and 2 GB of RAM. We measured the time of the modified user-side operations which include the user-side processing time, the collusion control server side processing time and the communication time with the collusion control servers. Since the interactions with the collusion control servers are independent, they can be done in parallel. We can see from

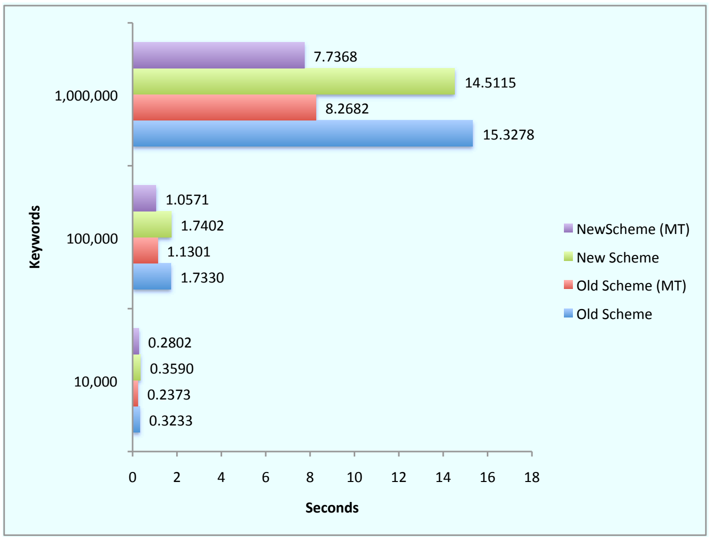


Figure 6: Performance Evaluation of Searches on Different Database Sizes

Figure 7 that there is an overhead when using collusion control servers. However, additional collusion control servers resulted in only very small increase in the overhead. Therefore, having more collusion control servers will not have a big impact on scalability of the system.

Note that we did not do any optimisation on the code. Also our implementation is based on group $\mathbb{Z}_p^*$ as described in the paper, however it is not hard to reformulate the idea based on elliptic curve groups, which might be more efficient.

8 Conclusion and Future Work

In this paper, we presented a new data encryption scheme that does not require a trusted data server. In the scheme the server can perform encrypted searches and updates on encrypted data without knowing the plaintext or the decryption keys. Unlike previous searchable data encryption schemes that require a shared key for multi-user access, each user in our system has a unique set of keys. The data encrypted by one user can be correctly decrypted by all the authorised users in the system. Moreover the keys can be easily revoked without any overhead, i.e. without having to re-encrypt the stored data. We provided a construction for the scheme built on top of proxy encryption schemes. We gave the formal definitions and proofs of security. We also implemented the scheme in Java and evaluated the performance.

One aspect of our future work is to achieve access pattern privacy. A weakness of our scheme and most of the other keyword-based search schemes is that

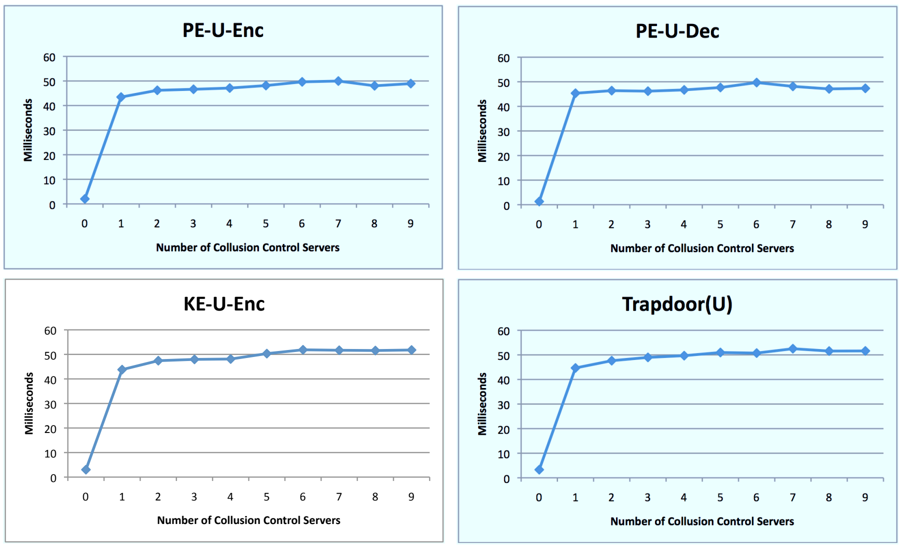


Figure 7: Performance Evaluation under Collusion control

the server knows the access pattern of the users which allows it infer some information about the queries. One possible direction is to combine our scheme with Private Information Retrieval (PIR) schemes. PIR schemes allow a user to retrieve data items from a database without revealing to the database which items were queried. By combining PIR, the search queries can be executed without revealing the access pattern to the server. This idea has been investigated in [9] and a scheme for single-reader/multiple-writers environment has been proposed in the paper. However, PIR schemes are computationally expensive. Using secure hardware assisted PIR schemes which are more efficient [27, 28] is a possible approach to address this.

Another possible extension may be to integrate bucketization [17, 18]. Bucketization means to partition an ordered attribute domain into a set of buckets and each of which is identified by a tag. Range queries can be translated into querying a set of bucket tags. A tag is equivalent to a keyword and can be easily encrypted and tested in our scheme. This would allow our scheme to support range queries. Another benefit of bucketization is the result set may contain some false positive results, which can obscure the access pattern. However, the impact on security needs to be carefully analysed.

Secure indexes [15, 29] is also a promising technique that is used to improve the performance and decrease the storage overhead of searchable encryption schemes. We will investigate the use of secure indexes in multiple users settings.

Ackowledgement

This work was supported by EPSRC research grant EP/C537181/1. We also thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] Amazon simple storage service. http://aws.amazon.com/s3.

[2] Megaupload. http://www.megaupload.com.

[3] Microsoft healthvault. http://healthvault.com/.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *SIGMOD Conference*, pages 563–574, 2004.

[5] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*. The Internet Society, 2005.

[6] J. Blackwood. Is storage outsourcing a viable alternative? http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2851289,00.html.

[7] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[8] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[9] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. S. III. Public key encryption that allows pir queries. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2007.

[10] D. Connor. Storage outsourcing on the rise. http://www.networkworld.com/news/2007/012207-storage-outsourcing-rises.html, 2007.

[11] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 79–88. ACM, 2006.

[12] E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *ACM Conference on Computer and Communications Security*, pages 93–102, 2003.

[13] C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In V. Atluri, editor, *DBSec*, volume 5094 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008.

[14] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[15] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216/.

[16] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *NDSS*. The Internet Society, 2003.

[17] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *SIGMOD Conference*, pages 216–227. ACM, 2002.

[18] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[19] A.-A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*. The Internet Society, 2003.

[20] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *FAST*. USENIX, 2003.

[21] A. Kapadia, P. P. Tsang, and S. W. Smith. Attribute-based publishing with hidden credentials and hidden policies. In *NDSS*. The Internet Society, 2007.

[22] H. Khurana, A. J. Slagell, and R. Bonilla. Sels: a secure e-mail list service. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 306–313. ACM, 2005.

[23] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. Reed. Strong security for network-attached storage. In D. D. E. Long, editor, *FAST*, pages 1–13. USENIX, 2002.

[24] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[25] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[26] M. W. Storer, K. M. Greenan, D. D. E. Long, and E. L. Miller. Secure data deduplication. In Y. Kim and W. Yurcik, editors, *StorageSS*, pages 1–10. ACM, 2008.

[27] S. Wang, X. Ding, R. H. Deng, and F. Bao. Private information retrieval using trusted hardware. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2006.

[28] P. Williams and R. Sion. Usable pir. In *NDSS*. The Internet Society, 2008.

[29] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 479–495. Springer, 2006.