Malaysian Journal of Mathematical Sciences 11(S) August: 89 - 102 (2017)
Special Issue: The 5th International Cryptology and Information Security Conference (New Ideas in Cryptology)

MALAYSIAN JOURNAL OF MATHEMATICAL SCIENCES
Journal homepage: http://einspem.upm.edu.my/journal

Protection of Data on Multiple Storage Providers

Mohamad, M. S.[*1], **Poh, G. S.**[1] and **Chin, J. J.**[2]

[1]*Information Security Lab, MIMOS Berhad, Malaysia*
[2]*Faculty of Engineering, Multimedia University, Malaysia*

E-mail: soeheila.mohamad@mimos.my
[*]*Corresponding author*

ABSTRACT

In utilising cloud storage services, the multiplicity of storage can be used to achieve higher security for the data. This work proposes symmetric searchable encryption which utilizes multiple servers to reduce leakage to the storage servers. This is achieved by partitioning the documents and scattering the encrypted blocks onto independent storage providers. Security model for SSE is adapted to fit this environment and a concrete scheme is presented and analysed.

Keywords: Searchable encryption, secure storage, cloud storage.

1. Introduction

As cloud technology is being widely accepted, the cloud owners offer storage service to the public. Among well-known cloud storage providers are Google Drive, Dropbox, Amazon S3 and Box. Such storage services can be utilized by users to optimize their data storage cost. However, users are aware of information disclosure by putting their documents on third party repositories. There are cloud storage providers with encryption features such as Spider Oak. Using the client application or webpage, an encryption key is generated and users' files are encrypted before being uploaded to the storage server. This way the server is unable to read the files. However, the user can only identify files to use by the filename but are unable to perform any additional operations such as searching on the content of the files.

To search contents of encrypted files, one could use symmetric searchable encryption (SSE). Considering users who subscribe to multiple cloud storage services, existing SSE schemes are only applicable in the trivial manner. In particular, the user can organize his documents into disjoint sets which he then stores at different storage providers via an SSE scheme. This setup has higher security than a single provider setup in the sense that each provider only has some but not all of the user's documents.

Song et al. (2000) introduced SSE by proposing that every word in the documents be encrypted, and hence searching involves comparing the search token to every word. Next, Goh (2003) proposed index design with the IND-CKA security notion, and achieve search complexity linear to number of documents. The following proposals (Chang and Mitzenmacher, 2005, Curtmola et al., 2006) improved on index design to achieve index and search query security. More importantly, Curtmola et al. (2006) defined SSE security to encompass adaptive chosen keyword attacks. At this point, it is recognized that some information beyond document sizes and search result is leaked. Curtmola et al. (2006) defined a search pattern as record of user's search queries whether it is the same keyword or not, and defined an access pattern as the list of document identifiers or ciphertexts in the search results.

The concept of information leaked to a server defined by Curtmola et al. (2006) is then parameterized by Chase and Kamara (2010) because the type of information revealed may differ from scheme to scheme. The parameters are presented as leakage functions for two situations, setup leakage and search leakage. Besides the access pattern and search pattern (renamed as query pattern (QP)), Chase and Kamara (2010) introduced the intersection pattern (IP) into the search leakage.

For the schemes presented in (Chase and Kamara, 2010), the index hides the number of documents corresponding to each keyword by padding. As such, their structured schemes have less information in setup leakage compared to preceding schemes. On the other hand, Cash et al. (2013) designed their index to hide the number of documents per keyword without padding. The index contains keyword-document pairs with only one document identifier per entry. The number of documents resulting from the search is revealed after the first search of a particular keyword. The first scheme to partition documents into blocks before encryption was proposed by Naveed et al. (2014). Consequently, the number of documents per keyword and document size is hidden until the document is in a search result and downloaded by the user.

Practical attacks have been demonstrated in Cash et al. (2015), Islam et al. (2012), Zhang et al. (2016) against SSE schemes proven to be secure. The attacks have shown that the number of documents associated with a keyword cannot leak. The importance of such leakages motivated the definition of leakage profiles in Cash et al. (2015) for SSE schemes based on when the information is leaked. Four leakage profile levels are defined and the most secure is L1 which means the scheme does not reveal the number of documents associated to any keyword before any query is made. Schemes by Cash et al. (2013), Chase and Kamara (2010), Naveed et al. (2014) fall into the L1 profile. The document distribution can be obfuscated by distributing the data over many storage providers. A generic form of multiserver SSE has been proposed in Poh et al. (2016) in which any single server SSE scheme can be deployed to a multiserver environment. A block-based multiserver scheme is also presented but the index reveals the number of document blocks for every keyword on one server.

Our Contributions A multiserver SSE scheme is presented and analysed. By the design of the index, and distributing the data to multiple servers, the scheme delays the servers' discovery of the number of blocks and documents per keyword, and document sizes until after queries are made, achieving leakage profile L1.

2. Preliminaries

2.1 Notation

We denote $\{1, 2, \ldots, n\}$ as $[n]$, $x \leftarrow A$ to mean x is an output of an algorithm A, $x \xleftarrow{R} X$ to mean random selection of a value x from a set X and $\|$ as

concatenation. We use $negl(x)$ to indicate a negligible function, $poly(x)$ to indicate a polynomial function and $id(x) \in \{0,1\}^\lambda$ to denote an identifier that uniquely identify object x. The object may be a document, a block or a server.

This work assumes a user who owns n documents, $D = \{D_1, D_2, \ldots, D_n\}$ and utilizes s servers, $S = \{S_1, S_2, \ldots, S_s\}$ for storage. Every document D_f is divided into h_f equal-length blocks $d_{f,i} \in \{0,1\}^{len(\lambda)}$ and written as $D_f = \{d_{f,1}\|1, d_{f,2}\|2, \ldots, d_{f,h_f}\|h_f\}$. Denote $\max(h_f) = m$. Further, denote the set of all blocks by $\mathbf{M} = \{M_1, \ldots, M_n\}$ where $M_f = \langle id(D_f), D_f \rangle$. The user also indicates the association of a keyword set to every document in an index, $\texttt{DB} = \{(id(D_f), W_f) \mid f = 1, \ldots, n\}$, where W_f denotes the set of keywords for document D_f. Indexes (or dictionaries) in our discussion is the data structure of the form $I[key] = value$. Given a key, the corresponding value is returned. For example, for the index DB, the key is the document identifier and the value is the associated set of keyword. Hence $\texttt{DB}[id(D_f)] = W_f$.

2.2 Symmetric encryption

A randomised symmetric encryption scheme $\mathcal{E} = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ consists of three PPT algorithms. Gen takes λ and outputs a secret key K; Enc takes a key K and a message $d \in \{0,1\}^{l(\lambda)}$ and outputs a ciphertext c. For all K from Gen and $d \in \{0,1\}^{l(\lambda)}$ we have $\texttt{Dec}(K, \texttt{Enc}(K, d)) = d$ with probability 1. We say $\mathcal{E}$ is indistinguishable under chosen-plaintext attack (CPA) if all PPT adversary has advantage $\mathbf{Adv}^{ind-cpa}_{\mathcal{E},\mathcal{A}}(\lambda)$,

$$\left|\Pr\left[\mathcal{A}(c)=1 : c \leftarrow \texttt{Enc}(K,d), K \leftarrow \texttt{Gen}(1^\lambda)\right] - \Pr\left[\mathcal{A}(c)=1 : c \xleftarrow{R} \{0,1\}^{l(\lambda)}\right]\right|$$

that is negligible, $negl(\lambda)$. Considering only IND-CPA symmetric encryption schemes, the adversary advantage in the following text is written as $\mathbf{Adv}_{\mathcal{E},\mathcal{A}}(\lambda)$.

2.3 Pseudorandom function

A function $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ from $\mathcal{F}$ the set of all functions $\{0,1\}^* \to \{0,1\}^\lambda$ is pseudo-random if all PPT adversary $\mathcal{A}$ has the advantage $\mathbf{Adv}^{prf}_{F,\mathcal{A}}(\lambda)$,

$$\left|\Pr\left[\mathcal{A}(F_K) : K \xleftarrow{R} \{0,1\}^\lambda = 1\right] - \Pr\left[\mathcal{A}(g) : g \xleftarrow{R} \mathcal{F} = 1\right]\right|$$

that is negligible, $negl(\lambda)$. From this point, adversary advantage on pseudo-random function is written as $\mathbf{Adv}_{F,\mathcal{A}}(\lambda)$.

3. SSE with partitioned data

The main objective of SSE with partitioned data (SSEwPD) is to disperse the power of storage provider on user's data by separating the data onto several independent storage in such away that none of the storage provider holds a complete document. SSEwPD consists of the same algorithms as SSE with similar correctness condition. The difference is in the construction of the index within the algorithms and in the adversary models.

3.1 Definition

The definitions of SSE in Curtmola et al. (2006) and Chase and Kamara (2010) have five algorithms, namely `KeyGen`, `Encryption`, `Trapdoor`, `Search` and `Decryption`. Here, a scheme is defined by three functions, `KeyGen`, `Setup` instead of `Encryption`, and `Search` which combines `Trapdoor`, `Search` and `Decryption`.

Definition 3.1. *An SSEwPD scheme* Π *consists of:*

$k \leftarrow \texttt{KGen}(1^\lambda)$*: A key generation algorithm, where on input the security parameter* λ*, it outputs a secret key* k*.*

$(\mathbf{K}, \mathbf{c}, I) \leftarrow \texttt{Setup}(k, \texttt{DB}, \mathbf{M}, S)$*: An index creation and encryption algorithm, where on input the secret key* k*, a database index* DB*, the set of data blocks* $\mathbf{M}$ *and the list of storage providers* S*, it outputs a set of provider's keys* $\mathbf{K}$ *which is stored securely by the user and, a set of encrypted data blocks* $\mathbf{c}$ *and a set of (encrypted) indexes* I *for every server.*

$D^w := \texttt{Search}(k, \mathbf{K}, w, I, S)$*: A search protocol, where on input the secret key* k*, the set of provider's keys* $\mathbf{K}$*, the set of (encrypted) indexes* I*, a query word* w *and the list of storage providers* S*, it outputs a set of decrypted documents* D^w *extracted from all the storage to the user.*

We say that Π *is correct if for any* λ, $\forall K \leftarrow \texttt{KGen}(1^\lambda)$, $\forall \texttt{DB}$, $\forall \mathbf{M}$, $\forall (K_S, \mathbf{c}, I) \leftarrow \texttt{Setup}(K, \texttt{DB}, \mathbf{M}, S)$, $\texttt{Search}(K, K_S, w, I, S)$ *returns a set of encrypted data blocks,* $\mathbf{c}_f = (c_{f,j})_{j=1}^{h_f}$ *such that when the set of decrypted blocks* $D^w = (\mathcal{E}.\texttt{Dec}(k_e, c_{f,j}))_{j=1}^{h_f}$*, where* $k_e \in K_S$*, are combined, the resulting documents have their identifiers in* $\texttt{DB}(w)$*.*

3.2 Leakages

Curtmola et al. (2006) states that efficient SSE inevitably leak information. Leakage is information allowed to be known by a server which is considered as the adversary. Leakage is declared in the scheme description and included in security argument according to the security model proposed by Chase and Kamara (2010), $\mathcal{L}$=($\mathcal{L}^{setup}$,$\mathcal{L}^{query}$).

Setup leakage $\mathcal{L}^{setup}$ is the information gained by the server from the stored indexes and encrypted documents or blocks. This usually includes the total number of documents and their sizes.

Search leakage $\mathcal{L}^{search}$ is the information gained by the server while performing a search. Information extracted from the query token and the search results includes the query pattern(QP), the access pattern(AP) and the intersection pattern (IP). The AP records the identifiers in the search results and the encrypted data accessed during particular search instance. The QP records whether the current search query is equal to a previous query. The IP records the identifiers in the search results and the encrypted data accessed during two or more different search queries.

3.3 Adversary Model

Besides the adversary's capability defined for single server SSE, multiserver SSE considers adversary power in terms of the number of servers the adversary controls:

- *Non-colluding server* is the adversary who controls only one of the storage servers.
- *Colluding servers* is the adversary who controls multiple, and possibly all of the storage servers.

Colluding servers is the stronger adversary as it holds more information, namely, leakages from all of the servers under its control.

3.4 Security Model

SSEwPD aims to achieve $\mathcal{L}$-security against adaptive chosen keyword attack by colluding storage providers which is defined by the following game:

Definition 3.2. *Given an SSE scheme,* $\Pi = (\texttt{KGen}, \texttt{Setup}, \texttt{Search})$ *with leakage function* $\mathcal{L} = (\mathcal{L}^{setup}, \mathcal{L}^{query})$, $\mathcal{L}$*-security is defined by two games.*

Real$_{\Pi,\mathcal{A}}(\lambda)$**:** *Adversary* $\mathcal{A}$ *selects a list of storage providers* S*, generates a set of documents* D*, a database index* $\texttt{DB} = \{(id(D_f), W_f) \,|\, f = 1, \ldots, n\}$ *and processes* D *as* $\mathbf{M} = (id_{D_f}, D_f)_{f=1}^{n}$*.* $\mathcal{A}$ *gives the challenger* $(\texttt{DB}, \mathbf{M}, S)$*. The challenger executes* $\texttt{KGen}(1^{\lambda})$ *to generate a secret key* k *and runs* $\texttt{Setup}(k, \texttt{DB}, \mathbf{M}, S)$ *resulting in* $(\mathbf{K}, \mathbf{c}, I)$*, in which* $\mathbf{K}$ *is stored secretly by the challenger while* $\mathbf{c}$ *and* $\mathcal{I} \in \{I, \perp\}$*, are given to* $\mathcal{A}$*, who then queries the* Search *protocols.*

For Search*,* $\mathcal{A}$ *chooses a sequence of query words* $w_1, \ldots, w_{p(\lambda)}$ *where* $p(.)$ *is a polynomial. The challenger executes* $\texttt{Search}(k, \mathbf{K}, w_t, I, S)$ *and returns the corresponding set of encrypted blocks,* $E^{w_1}, \ldots, E^{w_{p(\lambda)}}$ *to* $\mathcal{A}$*.*

After receiving the challenger's replies, $\mathcal{A}$ *gives* $\mathcal{I}$*,* $(E^{w_1}, \ldots, E^{w_{p(\lambda)}})$ *and* $(w_1, \ldots, w_{p(\lambda)})$ *to distinguisher* $\mathcal{D}$ *and obtain reply a bit* b*. Finally* $\mathcal{A}$ *outputs* b*.*

Ideal$_{\Pi,\mathcal{A},\mathcal{S}}(\lambda)$**:** *Adversary* $\mathcal{A}$ *selects a list of storage providers* S*, generates a database* $\texttt{DB} = \{(id(D_f), W_f) \,|\, f = 1, \ldots, n\}$ *and processes a set of documents* D *to form the set* $\mathbf{M} = \{(id(D_f), D_f) \,|\, f = 1, \ldots, n\}$*. The simulator* $\mathcal{S}$ *simulates* $(\mathbf{K}^*, \mathbf{c}^*, I^*)$ *based on the leakage information from* $\mathcal{L}_{setup}$*, and gives* $\mathbf{c}^*$ *and* $\mathcal{I}^*$*, where* $\mathcal{I}^* \in \{I^*, \perp\}$*, to* $\mathcal{A}$*, who then queries the* Search *protocols.*

For Search*,* $\mathcal{A}$ *chooses a sequence of query tokens* w_t *for* $t = 1, \ldots, p(\lambda)$ *where* $p(.)$ *is a polynomial. For every query* w_t $\mathcal{S}$ *is given* $\mathcal{L}_{query}$ *and returns to* $\mathcal{A}$ *the corresponding simulated encrypted data blocks* $E^{w_t^*}$*.*

After getting the replies from the challenger, $\mathcal{A}$ *gives* $(E^{w_1^*}, \ldots, E^{w_{p(\lambda)}^*})$*,* $\mathcal{I}^*$*,* $(w_1, \ldots, w_{p(\lambda)})$ *to distinguisher* $\mathcal{D}$ *which returns a bit* b*. Finally* $\mathcal{A}$ *outputs* b*.*

Π *is* $\mathcal{L}$*-secure against non-adaptive attacks if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that*

$$\left| \Pr\left[\mathbf{Real}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}^{\mathcal{S}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \right| \leq \texttt{negl}(\lambda).$$

3.5 Colluding Server's Advantage

The most powerful adversary is one who controls all of the storage providers. Such adversary can view all indexes and ciphertexts, and gain all leakages. Nevertheless, a multiserver SSE is at least as secure as a single server SSE.

Proposition 3.1. *Let* Π *be a multiserver SSE with* n *servers* S*, and* $\widetilde{\Pi}$ *be a single server SSE with the security parameter* λ*. Let* $\mathcal{A}$ *be an adversary of* Π *which controls multiple servers* (colluding storage providers)*,* $S' \subseteq S$*, and let* $\mathcal{B}$ *be an adversary of* $\widetilde{\Pi}$*. Then,*

$$\mathbf{Adv}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}_{\widetilde{\Pi},\mathcal{B}}(\lambda). \tag{1}$$

Equality in 1 is achieved if $\mathcal{A}$ controls all servers, $S' = S$ and Π uses the same key to generate the data on all servers. This is because index from every server would have the same index key for the same keyword, and hence can be consolidated trivially to form one index as in the case of $\widetilde{\Pi}$. Specifically, for any index key τ, $I^{\widetilde{\Pi}}[\tau] = \bigcup_{i=1}^{n} I_i^{\Pi}[\tau]$. Similarly for the leakages, $\mathcal{L}^{\widetilde{\Pi}} = \bigcup_{i=1}^{n} \mathcal{L}_i^{\Pi}$.

Consider a multiserver SSE scheme where the key set for each server is independent from the key sets for other servers. Due to the different keys, the index keys for one keyword on different servers are different. The probability that the adversary can match entries for the same keyword depends on the strength of pseudorandom function F used in the scheme, $\mathbf{Adv}_{F,\mathcal{A}}(\lambda)$. Consequently, $\mathbf{Adv}_{\Pi,\mathcal{A}}$ is strictly less than $\mathbf{Adv}_{\widetilde{\Pi},\mathcal{B}}$. In addition, this also implies that the leakages are independent, hence

$$\mathbf{Adv}_{\Pi,\mathcal{A}}(\lambda) \leq \sum_{i=1}^{t} \mathbf{Adv}_{\Pi,\mathcal{A}'_i}(\lambda)$$

where $t = |S'|$ and $\mathcal{A}'_i$ is the adversary controlling server S_i only.

4. A Concrete Scheme

SSEwPD is designed similar to SSE, with encrypted index and encrypted data. However, instead of indexing and storing documents, the scheme works with document blocks. The document blocks are separated into many sets to be sent to different storage providers. Figure 1 shows the definition of an SSEwPD scheme, Π^S.

The `KGen` function generates a master key k. The `Setup` function uses k to derive a secret key K_j for every storage provider S_j using the identifiers of the server $id(S_j)$ as input. Following that, in `Setup` algorithm the master key k is used to derive keyword-associated keys K_x and K_v. Then for each keyword w, and given every $id(D_f)$ in $\mathtt{DB}(w)$, `Setup` generates a pseudo-random label l_j based on K_x and encrypts $id(D_f)$ with storage provider identifier $id(S_j)$ using K_v, so that each storage provider stores a unique encrypted $id(D_f)$. It iterates

through all blocks of $id(D_f)$ by first selecting a storage provider S_j from all the storage providers. Temporary lists $\mathbf{t}$ are created to hold the encrypted $id(D_f)$ and the associated encrypted block identifiers for each storage provider. Then $\mathbf{t}$ is assigned to $\mathbf{QInx}_j$ for every key l_j. To search for documents with keyword

---

Preprocessing: Compile the list of storage providers, $S = \{S_1, S_2, \ldots, S_s\}$, database of document identifier/keyword pairs, $\mathtt{DB} = (id(D_f), W_f)_{f=1}^{n}$, and the document/data block pairs, $\mathbf{M} = (id(D_f), D_f)_{f=1}^{n}$, for $D_f = \{d_{f,1}\|1, d_{f,2}\|2, \ldots, d_{f,h_f}\|h_f\}$.

$k \leftarrow \mathtt{KGen}(1^\lambda)$:
Generate a random binary sequence, k, of length λ and output to user.

$(\mathbf{K}, \mathbf{c}, I) \leftarrow \mathtt{Setup}(k, \mathtt{DB}, \mathbf{M}, S)$:

1. For all $S_j \in S$, generate $K_j \leftarrow F(k, id(S_j))$ and set $\mathbf{K} = \{K_1, \ldots, K_s\}$.
2. For each $w \in \mathbf{W}$:
   $K_x \leftarrow F(k, 1||w)$ and $K_v \leftarrow F(k, 2||w)$.
   Initialize a counter z, $z = 0$.
   For each $id(D_f) \in \mathtt{DB}(w)$:
      For all $S_j \in S$,
         $l_j \leftarrow F(K_x, z\|id(S_j))$
         $e_{f,j} \leftarrow \mathcal{E}.\mathtt{Enc}(K_v, id(D_f)\|id(S_j))$.
      Increment z.
      Initialise temporary lists $\mathbf{t}_1 = e_{f,1}, \ldots, \mathbf{t}_s = e_{f,s}$.
      For $i = 1$ to h_f:
         $j \xleftarrow{R} [s]$
         $c_{f,i} \leftarrow \mathcal{E}.\mathtt{Enc}(K_j, d_{f,i}||i)$
         $p_{f,i} \leftarrow \mathcal{E}.\mathtt{Enc}(K_v, id(d_{f,i}))$.
         Add $c_{f,i}$ to $\mathbf{c}_j$.
         Append $p_{f,i}$ to $\mathbf{t}_j$.
   For every S_j, set $\mathbf{QInx}_j[l_j] = \mathbf{t}_j$.
3. Output to storage providers $(\mathbf{c}, I) = \{(\mathbf{c}_j, \mathbf{QInx}_j)$ for $S_j \mid j = 1, \ldots, s\}$, and output to user $\mathbf{K}$.

Figure 1: The $\Pi^S.\mathtt{KeyGen}$ and $\Pi^S.\mathtt{Setup}$ algorithms.

w, the $\Pi^S.\mathtt{Search}$ protocol, as described in Figure 2, uses the master key k and w to derive K_x and K_v. K_x is broadcast to all storage providers and using this key each storage provider retrieves the encrypted document identifier $e_{f,j}$ and its associated encrypted block identifiers. These encrypted identifiers are

$D^w :=$ `Search`$(k,\ S,\ \mathbf{K},\ w)$:

1. $K_x \leftarrow F(k, 1||w)$ and $K_v \leftarrow F(k, 2||w)$.
2. Broadcast K_x to all storage providers.
3. For each storage provider S_j,

   Initialize a counter, $z = 0$.

   While matching entry on z exists,

   $(e_{f,j}, \mathbf{p}) \leftarrow \mathbf{QInx}_j[F(K_v, z\|id(S_j))]$.

   Send $id(S_j)$, $(e_{f,j},\ \mathbf{p})$ to the user.

   Increment z.

   Decrypt $e_{f,j}$: $id(D_f) \leftarrow \mathcal{E}.\mathtt{Dec}(K_v, e_{f,j})$.

   Decrypt every $p_{f,i} \in \mathbf{p}$: $id(d_{f,i}) \leftarrow \mathcal{E}.\mathtt{Dec}(K_v, p_{f,i})$

   Send every $id(d_{f,i})$ to S_j to retrieve $c_{f,i}$.

   Decrypt $c_{f,i}$: $d_{f,i} \leftarrow \mathcal{E}.\mathtt{Dec}(K_j, c_{f,i})$
4. Reconstruct D_f by concatenation: $d_{f,1}\|_{f,2}\| \ldots \|d_{f,h_f}$. Add D_f to D^w.
5. Output D^w to user.

Figure 2: The for Π^S.`Search` algorithm.

sent back to the user, whom decrypts and uses them to retrieve the encrypted blocks from the storage providers.

4.1 Leakages

After `Setup`, each $\mathbf{QInx}_j$ reveals its number of entries $N_j \leqslant N$, where $N = \Sigma_{w \in \mathbf{W}} |\mathtt{DB}(w)|$, and the number of encrypted block identifiers per entry $\left|\mathbf{p}_{l_j}\right|$. Each entry in the index is based on a document and a keyword. If a document has many keywords, then the document is related to many entries in the index. So, the number of entries N_j in $\mathbf{QInx}_j$ does not indicate number of documents on S_j. The number of documents associated to any keyword is also hidden before any query because the index keys in $\mathbf{QInx}_i$ and $\mathbf{QInx}_j$ for $i \neq j$ do not indicate they are from the same keyword, assuming the pseudorandom function is secure. Consequently, this scheme has leakage profile L1.

For search query w, let T_j^w be the set of resulting encrypted document identifiers from storage provider S_j. Then, the access pattern on storage provider S_j, AP_j consists of the number of encrypted document identifier $\left|T_j^w\right|$ and their number of blocks $t_j = \{|\mathbf{p}_{f,j}| \mid e_{f,j} \in T_{S_j}\}$, and the set of encrypted blocks

$R_j^w = \{c_{f,i} \mid id(d_{f,i}) \in \mathbf{p}_{f,j} \text{ and } e_{f,j} \in T_{S_j}\}$. From tokens queried, S_j can also build the query pattern $\mathrm{QP}_j(w)$ and $\mathrm{IP}_j(w)$. In summary, under scheme Π^S, leakage function for storage provider S_j is $\mathcal{L}_j = (\mathcal{L}_j^{setup}, \mathcal{L}_j^{query})$ where

$$\mathcal{L}_j^{setup} = (|\mathbf{c}_j|, |c|, N_j, |\mathbf{p}_{l_j}| \text{ for } l_j \text{ in } \mathbf{QInx}_j)$$
$$\mathcal{L}_j^{query} = (\mathrm{AP}_j = (|T_j|, t_j, R_j), \mathrm{QP}_j(w), \mathrm{IP}_j).$$

4.2 Security

Theorem 4.1. *The SSE scheme Π^S is $\mathcal{L}$-secure against non-adaptive chosen keyword attacks by colluding storage providers assuming F is a secure PRF and $\mathcal{E}$ is an IND-CPA symmetric encryption scheme.*

Proof. Based on the leakage functions defined above, we define the simulator $\mathcal{S}$ for non-colluding storage provider $\mathcal{A}_b$. Then we extend the simulator to be for colluding storage provider.

Let S_b be the storage provider being controlled by adversary $\mathcal{A}_b$. Simulator $\mathcal{S}$ prepares a simulated index $\mathbf{QInx}_b^*$ and block ciphertexts $\mathbf{c}_b^*$ from $\mathcal{L}_b^{setup} = (|\mathbf{c}_b|, |c|, N_{S_b}, |\mathbf{p}_{l_b}| \text{ for } l_b \text{ in } \mathbf{QInx}_b)$ as follows. Randomly generate N_{S_b} binary strings x_i of length λ as the keys of the index, and randomly generate N_{S_b} binary strings of length λ as the encrypted document identifiers a_i. For each a_i, generate $|\mathbf{p}_{l_b}|$ random binary strings $\mathbf{b}_i$ to form $a_i \| \mathbf{b}_i$. Set $\mathbf{QInx}_{S_b}^* = \{(x_i, a_i \| \mathbf{b}_i) | i = 1, \ldots, N_{S_b}\}$. Finally, generate $|\mathbf{c}_b|$ random binary strings, each with length $|c|$ to form $\mathbf{c}_{S_b}^*$. Return $\mathbf{c}_b^*$, $\mathbf{QInx}_b^*$ to adversary.

For search queries the simulator $\mathcal{S}$ produces the transcript of interaction (T_b^*, D_b^*) from $\mathcal{L}_b^{query} = (\mathrm{AP}_b(w) = (|T_b|, t_b, |R_b(w)|), \mathrm{QP}_b(w), \mathrm{IP}_b(w))$. For query w, $\mathcal{S}$ must choose $|T_b|$ entries from $\mathbf{QInx}_b^*$ such that their values part matches the numbers in t_b to form T_b^*. First, $\mathcal{S}$ checks $\mathrm{QP}_b(w)$ whether this query has been made before. If it has been queried, $\mathcal{S}$ returns the entries returned previously. Otherwise, $\mathcal{S}$ selects unused entries according to t_b. Next, $\mathcal{S}$ forms D_w^*. If $\mathrm{QP}_b(w)$ indicates this query has been made before, $\mathcal{S}$ returns the previously returned D_w^*. If $\mathrm{IP}_b(w)$ indicates there are block identifiers accessed by this query are the same as those accessed by previous queries, $\mathcal{S}$ chooses the previously returned encrypted blocks.Otherwise, choose $|R_b(w)|$ unused strings in $\mathbf{c}_b^*$.

Given that $\mathcal{E}$ is IND-CPA and F is a secure PRF, then the adversary advantage in distinguishing the random entries in $\mathbf{QInx}_b^*$ from the entries in the real $\mathbf{QInx}_b$ is at most the advantage of the adversaries for $\mathcal{E}$ and F. Similarly, the

real search results T_b and R_b contain ciphertexts which are indistinguishable from random strings in (T_b^*, D_b^*) for IND-CPA $\mathcal{E}$. Hence,

$$\left|\Pr\left[\mathbf{Real}_{\Pi,\mathcal{A}_b}(\lambda)=1\right] - \Pr\left[\mathbf{Ideal}^{\mathcal{S}}_{\Pi,\mathcal{A}_b}(\lambda)=1\right]\right| \leqslant \mathbf{Adv}_F(\lambda) + \mathbf{Adv}_{\mathcal{E}}(\lambda)$$

which is negligible.

Now, consider colluding storage providers, $\mathcal{A}_B(\lambda)$ who controls $S_j \in B \subseteq S$. Notice that the every entry of index $\mathbf{QInx}_j$ consists of ciphertexts dependent on storage provider identifier. As a result, $\mathbf{QInx}_j$ for all j are disjoint. It follows that the output from each storage provider for `Search` are independent. So, the simulator for colluding storage providers, $\mathcal{S}'$, can prepare replies based on leakage of every storage provider $\mathcal{L}_j^{setup} = (|\mathbf{c}_j|, |c|, N_{S_j}, |\mathbf{p}_{l_j}| \text{ for } l_j \text{ in } \mathbf{QInx}_j)$ and $\mathcal{L}_j^{query} = (\mathrm{AP}_j = (|T_j|, t_j, R_j), \mathrm{QP}_j, \mathrm{IP}_j))$ independently for each j as for the non-colluding storage provider attack. In short, define $\mathcal{S}'$ to be the collection of single server simulators $\{\mathcal{S}_b \mid S_b \in B\}$ where $\mathcal{S}_b$ denotes simulator for server S_b as defined above. We conclude that the advantage of $\mathcal{A}_B(\lambda)$, is the sum of advantage of the adversaries controlling single storage provider $\mathcal{A}_b(\lambda)$ for each $S_j \in B$,

$$\left|\Pr\left[\mathbf{Real}_{\Pi,\mathcal{A}_B}(\lambda)=1\right] - \Pr\left[\mathbf{Ideal}^{\mathcal{S}'}_{\Pi,\mathcal{A}_B}(\lambda)=1\right]\right| \leqslant |B|\big(\mathbf{Adv}_F(\lambda) + \mathbf{Adv}_{\mathcal{E}}(\lambda)\big)$$

which is also negligible. □

4.3 Performance

The performance of Π^S can be measured in terms of indexes creation during `Setup` and communications during `Search`. The main setup cost is the storage of $\mathbf{QInx}_j$ for every storage provider S_j. The size of $\mathbf{QInx}_j$ is $O(Nm/s)$ given $N = \Sigma_{w \in \mathbf{W}} |\mathtt{DB}(w)|$ and since the overall index is partitioned into s storage providers. For a document, it stores maximally m encrypted block identifiers. The search cost is $O(rz)$ for $r = |\mathtt{DB}(w)|$ and $z = \Sigma_{j=1}^{s} |\mathbf{QInx}_j[l_j]|$ since for a query on w, the total number of matches is r with z the size of retrievals from all storage providers, whereby each $\mathbf{QInx}_j[l_j]$ returns the list of encrypted block identifiers associated to the matching documents. The communication cost is $O(s)$, linear to the number of participating storage providers since the query is broadcast to all storage providers.

5. Conclusion

This work proposes extending searchable symmetric encryption schemes to ensure a storage provider would not be able to obtain user's complete document.

The SSE security model is adapted to consider adversary who controls more than one service provider. In general, a multiserver SSE is at least as secure as a single server SSE. One concrete SSEwPD scheme is presented and proven to achieve $\mathcal{L}$-security under non-adaptive chosen keyword attack against colluding storage providers.

References

Cash, D., Grubbs, P., Perry, J., and Ristenpart, T. (2015). Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 668–679. ACM.

Cash, D., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M.-C., and Steiner, M. (2013). Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In Canetti, R. and Garay, J. A., editors, *Advances in Cryptology - CRYPTO 2013 and IACR eprint archive*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer.

Chang, Y. and Mitzenmacher, M. (2005). Privacy Preserving Keyword Searches on Remote Encrypted Data. In Ioannidis, J., Keromytis, A. D., and Yung, M., editors, *ACNS 2005*, volume 3531 of *LNCS*, pages 442–455. Springer.

Chase, M. and Kamara, S. (2010). Structured Encryption and Controlled Disclosure. In Abe, M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer.

Curtmola, R., Garay, J. A., Kamara, S., and Ostrovsky, R. (2006). Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In Juels, A., Wright, R. N., and di Vimercati, S. D. C., editors, *ACM Conference on Computer and Communications Security, CCS 2006*, pages 79–88. ACM.

Goh, E.-J. (2003). Secure indexes. Cryptology ePrint Archive, Report 2003/216. `http://eprint.iacr.org/2003/216/`.

Islam, M. S., Kuzu, M., and Kantarcioglu, M. (2012). Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012*. The Internet Society.

Naveed, M., Prabhakaran, M., and Gunter, C. A. (2014). Dynamic Searchable Encryption via Blind Storage. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 639–654. IEEE Computer Society.

Poh, G. S., Mohamad, M. S., and Chin, J. (2016). Searchable Symmetric Encryption Over Multiple Servers. In *Arctic Crypt 2016*.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical Techniques for Searches on Encrypted Data. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 44. IEEE Computer Society.

Zhang, Y., Katz, J., and Papamanthou, C. (2016). All Your Queries Are Belong To Us: The Power of File-Injection Attacks on Searchable Encryption. Cryptology ePrint Archive, Report 2016/172. `http://eprint.iacr.org/2016/172/`.