Verifiable Symmetric Searchable Encryption for Multiple Groups of Users

Zachary A. Kissel and Jie Wang
Department of Computer Science, University of Massachusetts Lowell, Lowell, MA, USA

Abstract— *We present an efficient method for providing group level hierarchical access control over keywords in a multi-user searchable encryption scheme under the Semi-Honest-but-Curious model. We achieve this using a shared global index stored on the cloud and efficient key-regression techniques on the client side. Our method extends the multi-user searchable encryption model of Curtmola et. al. [1] to multiple groups of users. Moreover, our method provides verifiability of search results, and we show that our system is non-adaptively secure.*

Keywords: Clouds, Cryptography, Search methods

1. Introduction

Cloud storage provides a convenient platform for users to store data that can be accessed from anywhere at anytime without maintaining a storage infrastructure. However, cloud storage is inherently insecure. To use storage services provided by a cloud, users would need to trust, at least implicitly, the provider. There have been attempts to alleviate the need for this trust through cryptographic methods. Trivially, one may encrypt each file before uploading it to the cloud. This approach, however, would significantly hinder searching over the data, causing a loss in critical functionality.

This is where Symmetric Searchable Encryption (SSE) comes into play. SSE allows users to offload search queries to the cloud, which is responsible for returning the encrypted files that match the search queries (also encrypted). Most previous work was focused on keyword search [2], [3], [4], [5], [6], [7], while some more recent work has considered searching on phrases [8].

Searching over encrypted data involves a number of encrypted files stored on an untrusted cloud. The client issues an encrypted search query to the cloud, and wants the cloud to return the results of the search query without learning the query itself. Moreover, the client wants assurance that the cloud is faithfully executing the search. When multiple users are present, data owners want control over who may search for which piece of information. Any mechanism for searching over encrypted data with multiple users must provide these security guarantees.

Early SSE systems typically use an index mechanism under the Honest-but-Curious (HBC) cloud model. This approach constructs an index of keywords contained in a document, encrypts the index in a special way, and associates the index with a document or a set of documents. The documents themselves are encrypted with a standard symmetric encryption algorithm. The encryption over the index is constructed in such a way that the cloud can successfully apply queries over the documents when given a piece of trapdoor information. Specifically, with the trapdoor, the cloud can determine which files contain the queried keyword. Some systems return a copy of the encrypted file, while others return just a list of document identifiers. In the later case, the client must request each file individually.

Our main contribution is the construction of an efficient multi-user searchable encryption system with group level hierarchical access control. We further demonstrate how to make our group membership dynamic while still preserving security. We achieve the security guarantees under the Semi-Honest-but-Curious (SHBC) model [2] (which includes the HBC model). In the SHBC model, the cloud honestly stores encrypted files of the data providers, executes a fraction of the search operations (or the whole operations), and returns the search results. The cloud, however, may try to learn the underlying plaintext of the encrypted data. SHBC is more realistic than HBC, for it allows a cloud to limit the amount of bandwidth and computating power when handling queries. This is at odds with the desires of the clients to have complete and accurate results. Hence, we add to the model that an SHBC cloud will respond to all queries even if it may not do so honestly. Thus, the users need to verify that the search results are accurate and complete. To meet this requirement we demonstrate a verification method for our system. Moreover, our system is both space and time efficient for the client as well as the cloud provider, and we show that our system is non-adaptively secure. Our work improves on existing searchable encryption work for one group of users [1] to multiple groups of users.

The paper is organized as follows: We describe previous work on searchable symmetric encryption in Section 2. In Section 3 we define a system model for searchable encryptions for multiple groups of users. We provide background information in Section 4 needed for constructing our system, which is presented in Section 5. We conclude the paper in Section 6 and provide proof in the appendix that our system is non-adaptively secure.

2. Previous Work

Research on encrypted search is either based on symmetric encryptions or asymmetric encryptions, first studied by, respectively, Song et. al. [5] and Boneh et. al. [9]. We will only consider symmetric searchable encryption in this paper.

Early approaches to the encrypted search problem made use of the Bloom filter [10], a probabilistic data structure used as an index for the keywords in each encrypted file. These include, for example, Goh's indexed based system [7]. The closely related system of Chang and Mitzenmacher uses a deterministic indexed based system, similar to a Bloom filter, for keyword search secure under the HBC model [6]. Goh's system supports extra features not presented in Chang and Mitzenmacher's system, and does so under a slightly weaker model of security.

These models, however, do not guarantee even a non-adaptive form of security. Non-adaptive security provides privacy when all search queries are issued at one time. It was not until the work of Curtmola, Garay, Kamara, and Ostrovsky [1] that a rigorous and exact security definition was given for SSE. They provided formal definitions for non-adaptive, respectively adaptive, indistiguishability notions and they showed a reduction to a form of non-adaptive, respectively adaptive, semantic security. Briefly, in non-adaptive security, privacy is only guaranteed when clients generate queries at one time. In the case of adaptive security, clients are guaranteed privacy even if they generate queries as a function of previous search outcomes. Until the work of Curtmola et. al. all systems were secure in the non-adaptive sense in the best case.

Tang et. al. [8] presented a two-phase protocol to handle phrase search over encrypted data. In the first phase, the cloud retrieves the document identifiers for documents that contain all the words in the phrase provided by the client, and returns the identifiers to the client. This phase relies on a global index shared among all documents in the cloud. In the second phase, the client sends a query and a list of document identifiers to the cloud. The cloud searches for an exact phrase match for each document in the per document index and returns to the client the actual encrypted documents that match the phrase. Their protocol, however, only provides security under the HBC adversarial model.

There were attempts to work on more realistic cloud models than the HBC model. In particular, Chai and Gong [2] studied verifiable symmetric searchable encryption under the SHBC model, which allows the client to verify that the cloud has returned the correct list of document identifiers. They achieved this through the use of tries [11].

3. Multi-User Searchable Encryption

Let $\mathcal{D} = \{D_1, D_2, \dots, D_n\}$ denote a collection of n encrypted documents in the cloud storage, Σ the alphabet over which characters from strings are drawn, and $\Delta = \{w_1, w_2, \dots, w_d\}$ a dictionary of d words drawn from Σ^*. We associate with each document in collection $\mathcal{D}$ an number we call the index. The function is denoted by $\mathsf{id} : \mathcal{D} \to \mathbb{Z}$. Let $\mathcal{D}(w_i)$ denote the set of document identifiers that contain the word $w_i \in \Delta$, and $\mathcal{G} = \{g_i \,|g_i \subset \mathcal{U}\}$ an indexable set of groups of users from the set of users $\mathcal{U}$.

For cryptographic primitives we denote a symmetric encryption algorithm by the tuple $(G, \mathcal{E}, D)$, where G takes a security parameter and generates a key of the correct size, $\mathcal{E}$ is an encryption algorithm that takes a key and a message as input and outputs a cipher text; and D is the decryption algorithm of $\mathcal{E}$. In addition we rely on both Pseudo-random Permutations and Pseudo-random Functions. We also need a keyed hash function $\mathcal{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^z$, where z is some security parameter.

3.1 Model

Curtmola et. al. [1] defined a multi-user SSE (M-SSE) system as follows:

Definition 1: (Multi-User Searchable Symmetric Encryption (M-SSE)) MSSE is a collection of six polynomial-time algorithms MKeygen, MBuildIndex, AddUser, RevokeUser, MTrapdoor, and MSearch, where

- MKeygen (1^k) is a probabilistic key generation algorithm run by the owner O. It takes a security parameter k as input and returns an owner secret key K_O .
- MBuildIndex $(K_O, \mathcal{D})$ is run by O to construct indexes. It takes K_O and $\mathcal{D}$ as inputs, and returns an index $\mathcal{I}$.
- AddUser (K_O, U) is run by O whenever O wishes to add a user to the group. It takes K_O and a user U as inputs, and returns U's secret key K_U.
- RevokeUser (K_O, U) is run by O whenever O wishes to revoke a user from G. It takes K_O and a user U as inputs, and revokes the user's searching privileges.
- MTrapdoor (K_U, w) is run by a user (including O) to generate a trapdoor for a given word w. It takes a user U's secret key K_U and a word w as inputs, and returns a trapdoor $T_{U,w}$.
- MSearch $(\mathcal{I}_\mathcal{D}, T_{U,w})$ is run by the server S to search for the documents in $\mathcal{D}$ that contain word w. It takes the index $\mathcal{I}_\mathcal{D}$ for collection $\mathcal{D}$ and the trapdoor $T_{U,w}$ for word w as inputs, and returns $\mathcal{D}(w)$ if user $U \in \mathsf{G}$ and $\perp$ if user $U \notin \mathsf{G}$.

We note that the group controlled searches presented in [1] only provides a single dynamic group. They showed that their system is non-adaptively secure, and that evicted users cannot perform a search, provided that they cannot collude with non-evicted users.

We extend this model to support multiple groups of users. Let $\mathcal{G} = \{g_i \,|g_i \subset \mathcal{U}\}$ be an indexable set of groups of users from the set of users $\mathcal{U}$. We associate with each group of users, g_i, a dictionary, Δ_i, of keywords allowed to be searched for. We further require that Δ_i contain all words in Δ_j for $j < i$. We define our extended model as follows:

Definition 2: (Hierarchical Access Controlled SSE (HAC-SSE)) Let $\mathcal{O}$ be the owner of a document collection $\mathcal{D}$ and $\mathcal{G}$ an indexable set of groups of users from the set of users $\mathcal{U}$. HAC-SSE is a set of polynomial time algorithms HKeygen, HBuildIndex, HAddUser, HRevokeUser, HTrapdoor, and HSearch, where HKeygen $\left(1^k, n\right)$ is the same as MKeygen $\left(1^k\right)$.

- MBuildIndex $\left(K_O, \mathcal{D}, \mathcal{G}, \left\{\Delta_1, \Delta_2, \dots, \Delta_{|\mathcal{G}|}\right\}\right)$ is run by O to construct indexes. It takes, as input, the owner's secret key K_O, a document collection $\mathcal{D}$, the set of groups $\mathcal{G}$, and the set of dictionaries, $\left\{\Delta_1, \Delta_2, \dots, \Delta_{|\mathcal{G}|}\right\}$. The function returns an index, $\mathcal{I}$, that forces the hierarchical access control.

- $\mathsf{AddUser}(K_O, U, g)$ is run by O whenever O wishes to add a user U to the group $g \in \mathcal{G}$. It takes the owner's secret key K_O, a user, and the group as input. The function then returns the group key to the user.
- $\mathsf{RevokeUser}(K_O, U, g)$ (*optional*) is run by O whenever O wishes to revoke a user U from group $g \in \mathcal{G}$. It takes the owner's secret key K_O, a user U, and a group g as inputs. The function then revokes the user's searching privileges.
- $\mathsf{MTrapdoor}(K_U, w)$ is run by a user (including O) to generate a trapdoor for a given word. It takes a user U's secret key K_U and a word $w \in \Delta_i$ as inputs, and returns a trapdoor $T_{U,w}$.
- $\mathsf{MSearch}(\mathcal{I}_\mathcal{D}, T_{U,w})$ is run by the server S in order to search for the documents in $\mathcal{D}$ that contain word w. It takes the index $\mathcal{I}_\mathcal{D}$ for collection $\mathcal{D}$ and the trapdoor $T_{U,w}$ for word w in some dictionary Δ_g as inputs, and returns $\mathcal{D}(w)$ if user U belongs into a specific $g \in \mathsf{G}$ and $\perp$ if user $U \notin \mathsf{G}$.

For HAC-SSE to be secure we must have the following property satisfied:

Property 1: A user $u \in g_i$ can not, successfully, query for any word $w \in \Delta_j$ for all $j > i$ with more than a negligible probability.

4. Background

4.1 Key Regression

Our system relies on a construct by Fu et. al. called Key Regression [12], originally desinged to allow a content owner to manage dynamic group membership in a Content Distribution Network (CDN). The idea is that a content owner encrypts a document, for a group of users, with a key K_i at time i. All users belonging to the access group are given a member state stm_i, which allows them to derive the key K_i. At time j if a member of the group is evicted, then all documents will be re-encrypted with a new key K_j. All users remaining in the group are given a state stm_j which can be used to derive key K_j. They can now forget about stm_i. This can be done, due to the property of key regression that from state stm_j one can derive all previous states (and thus all previous keys). However, it is impossible for a user possessing state stm_j to accurately predict future states. We call $(i, \mathsf{stm}_1, \mathsf{stm}_2, \ldots, \mathsf{stm}_n)$ a publisher state. Formally, Key Regression [12] is defined as follows:

Definition 3: (Key Regression (KR)) A KR scheme consists of four polynomial time algorithms setup, wind, unwind, and keyder, where

- $\mathsf{setup}(1^\lambda, n)$ returns a publisher state stp; where 1^λ (in unary) is a security parameter and n is an integer representing the number of evictions the system should support. This algorithm may be probabilistic.
- $\mathsf{wind}(\mathsf{stp})$ returns a tuple $(\mathsf{stp}', \mathsf{stm}_i)$, where stp' is the new publisher state and stm_i is the new member state. This algorithm may be probabilistic, and may return $(\perp, \perp)$ if the number of winds has exceeded the n used in setup.
- $\mathsf{unwind}(\mathsf{stm}_j)$ returns the previous member state stm_i for $i < j$. If previous member states do not exist, the algorithm returns $\perp$.
- $\mathsf{keyder}(\mathsf{stm}_i)$ returns a key K_i in some keyspace.

These algorithms can be based on the SHA-1 hash function, the AES symmetric cipher, the RSA public key cipher, and a generic construction based on any Forward-Secure Pseudorandom Generator. For our purposes we will simply use Key Regression as a black box.

4.2 Tries

Our keyword indexing mechanism uses a data structure called trie, devised by Fredkin in [11], It supports two main operations Insert and Search, both take a word $w \in \Sigma^*$ as input. A trie is a $|\Sigma \cup \{\$\}|$-ary tree, where each node of the tree is labeled with an element of $\Sigma \cup \{\$\}$. Moreover, a root-to-leaf path through the tree denotes a word $w \in \Sigma^*$, which is terminated by a special character $\$ \notin \Sigma$.

The Insert appends a \$ to the input w. Starting at the root node of the tree, we use w to create a path. The first time we reach a node that does not have the current corresponding letter in w, we add a subpath as a child to the current node. Moreover, we label this subpath appropriately with the remaining letters of w, terminating the path with a \$.

The Search function uses input w as a path through the tree. The function first adds a \$ to the path. If that path ends in a leaf, i.e., the path is a root-to-leaf path, the search is successful. Otherwise, the word does not exist in the dictionary.

In what follows we will denote a trie by T and a node by $T_{i,j}$, where i is the depth of the node and j the left to right placement of the node. We will denote the access to values stored in the node of T by $T_{i,j}[s]$, where s denotes the name of the field.

5. Construction of HAC-SSE and Security

Our system associates with each group $g_i \in \mathcal{G}$ a member state stm_i from a key regression system and a dictionary $\Delta_i \subset \Delta$. The dictionary Δ_i contains all the words in dictionary Δ_j for all $j < i$. Recall that in Key Regression, given a member state stm_i one can derive previous member states stm_j for all $j < i$. We will ensure that members of group g_i can search for any keyword in dictionary Δ_i, but not in dictionary Δ_k with $k > i$.

Our index structure, $\mathcal{I}$, is based on idea of secure trie [2]. The root to leaf paths through our trie, unlike the trie in [2], are secured in such a way that the hierarchical access policy is enforced. We insert into the trie the words in the set of dictionaries $\{\Delta_1, \Delta_2, \ldots, \Delta_{|\mathcal{G}|}\}$. Once this process is complete, we annotate the trie with the keys used to secure each letter along the root-to-leaf path. We start by annotating every root-to-leaf path in Δ_1 with K_1. Note that in a trie, a word is a root-to-leaf path. Starting at $i = 2$ we iteratively apply the following process for each word $w \in (\Delta_i - \Delta_{i-1})$. Walk through the trie according to w and look at each annotation. If the node is un-annotated, then annotate the node with key K_i (see Figure 1 for a completely annotated trie). We use a completely annotated trie for each dictionary Δ_i to

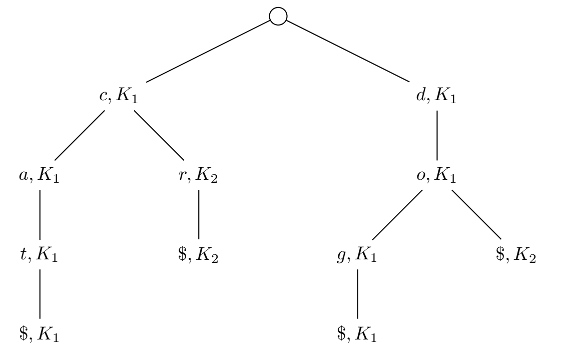

Fig. 1: An annotated trie for dictionaries $\Delta_1 = \{\text{cat}, \text{dog}\}$ and $\Delta_2 = \{\text{car}, \text{do}\} \cup \Delta_1$

generate a perfect hash table $\mathcal{H}_i$ with hash function h_i. Each entry in $\mathcal{H}_i$ contains a "Key Schedule" that represents what key is used to secure each node along the root-to-leaf path for the hash of the corresponding word. Finally, the trie is walked and each node is secured, using a keyed hash function, according to its key annotation. This new value is the hash of the value the node represents together with the hash of the path to the nodes predecessor. After a node is secured, the key annotation is removed (see Figure 2). Though we did not explicitly outline how, we note that every leaf node of the trie contains a list of document identifiers that denote the documents that contain the word specified by the given root-to-leaf path. The trie, which is the index $\mathcal{I}$, is then sent to the cloud.

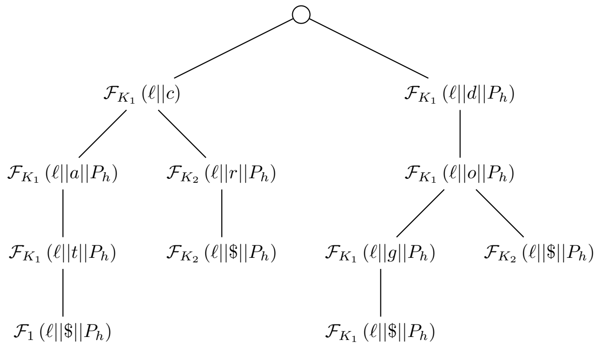
$\mathcal{F}_{K_1}(\ell||c)$
$\mathcal{F}_{K_1}(\ell||d||P_h)$
$\mathcal{F}_{K_1}(\ell||a||P_h)$
$\mathcal{F}_{K_2}(\ell||r||P_h)$
$\mathcal{F}_{K_1}(\ell||o||P_h)$
$\mathcal{F}_{K_1}(\ell||t||P_h)$
$\mathcal{F}_{K_2}(\ell||\$||P_h)$
$\mathcal{F}_{K_1}(\ell||g||P_h)$
$\mathcal{F}_{K_2}(\ell||\$||P_h)$
$\mathcal{F}_1(\ell||\$||P_h)$
$\mathcal{F}_{K_1}(\ell||\$||P_h)$

Fig. 2: Final trie based on Figure 1. The values P_h denotes the parents hash value and ℓ denotes the current nodes level.

When a user (call him Bob) $\in g_j$ wants to search for a specific word $w \in \Delta_j$, Bob looks up the appropriate entry $\mathcal{H}_j[h_j(w)]$, and determines the keys used to encrypt the root-to-leaf path specified by w. Bob then encrypts each character of w according to this "key schedule" and submits the encrypted query to the cloud. Upon receiving a query, the cloud uses the encrypted path to trace through the trie. If the cloud reaches a leaf node in the trie, then it returns the document identifiers to Bob; otherwise, it returns $\perp$ to Bob.

Placing our system in the formal model of Section 3.1 we achieve the system in Figure 3.

5.1 Security Guarantees of HAC-SSE

We first note that each document is encrypted separately. This means that nothing is leaked about the encrypted documents except their sizes. The privacy preserving queries are a collection of hashes of prefixes with confidentiality protected by the underlying hash function. The only information leaked by a query is the length of the query.

Turning our attention to confidentiality of the encrypted trie, we observe that each node contains a hashed value r_3. We know that, although it is improbable for an attacker to derive the original value from r_3, it is possible that the attacker will try to learn statistical information regarding r_3 when considering all the nodes in T. We show that this is impossible using the following theorem on the uniqueness of hash values in nodes of a trie [2].

Theorem 1 (Uniqueness of Node Values): Given a trie T of depth L with $C \geq \frac{|\Sigma|^{L+1}-1}{|\Sigma|-1}$, we have

$$\begin{aligned}\mathsf{Pr}\left(T_{j,q}[r_3] = T_{\hat{j},\hat{q}}[r_3] \,|\, (q,j) \neq \left(\hat{q},\hat{j}\right)\right) \\ \approx 1 - \left(\frac{2^z-1}{2^z}\right)^{\frac{C(C-1)}{2}}\end{aligned} \quad (1)$$

Theorem 1 holds when the prefix path values for the nodes are computed with the same keyed hash function. While our system has $|\mathcal{G}|$ possible keyed hash functions, due to the construction of keyed cryptographic hash functions, the theorem still holds.

It is straightforward to see that Figure 3 satisfies Property 1, because no users, given a member state stm_i, can determine stm_j for any $j > i$ with more than a negligible probability.

We can further prove that our system is non-adaptively secure, stated in Theorem 2, by constructing search patterns appropriately. The detailed proof of this theorem is given in Appendix.

Theorem 2: The construction shown in Figure 3 is non-adaptively secure.

6. Adding Revocation and Verification

We extend our construction in Section 5 to include verification of search results and revocation of access. By adding verification we can place our system under the SHBC model. To do this we must modify MBuildIndex by augmenting the nodes of the trie in a fashion similar to the methods used in [2]. We also extend our model to contain a polynomial-time function MVerify to verify that the cloud has returned the results correctly. To add revocation of access we make use of the revocation method of Curtmola et. al. [1].

To add verification to our system we add a set, $\mathcal{B}$, of $|\mathcal{G}|$ bitmaps of size $|\Sigma|$ to every node of the trie construction algorithm given in Figure 3. We require that the bitmap $b_i \in \mathcal{B}$ represent all children reachable by a member of group $i \in \mathcal{G}$. Under step 5 for the BuildIndex algorithm we add verification tags to each node according to the algorithm given in figure 4.

```
$\mathsf{HKeygen}\left(1^k, n\right)$ Set up a key regression system, $\mathcal{KR} = (\mathsf{setup}, \mathsf{wind}, \mathsf{unwind}, \mathsf{keyder})$. Set $\mathsf{stp} = \mathsf{setup}\left(1^k, n\right)$, construct the hierarchical dictionaries, $\Delta_1, \Delta_2, \ldots, \Delta_n$, and return $K_O = (\mathsf{stp}, \{\Delta_1, \Delta_2, \ldots, \Delta_n\}, \mathcal{KR})$.
$\mathsf{MBuildIndex}\left(K_O, \mathcal{D}, \mathcal{G}, \left\{\Delta_1, \Delta_2, \ldots, \Delta_{|\mathcal{G}|}\right\}\right)$
  1) Create a full $|\Sigma|$-ary tree.
  2) Set $(r_0, r_1, r_2) = (0, 0, 0)$ for every node, $T_{0,0}[r_1] = 0$, and $q_0 = 0$.
  3) For each word $\mathbf{w} = (w_1, w_2, \ldots)$ in document $D_i$ (where $1 \leq i \leq n$)
     a) Find an $\ell$ such that $w \in \Delta_\ell$, if one doesn't not exist skip the word.
     b) For $j = 1$ to $|\mathbf{w}|$
        i) Find a $q_j \in [q_{j-1} \times |\Sigma| + 1, (1 + q_{j-1}) \times |\Sigma|]$, where $T_{j,q_j}[r_1] = w_j$. If such a $q_j$ can't be found, find a $q_j$ such that $T_{j,q_j}$ is empty and set $T_{j,q_j}[r_1] = w_j$ and $T_{j,q_j}[r_0] = \ell$.
     c) Find an appropriate $q_{j+1} \in [q_j \times |\Sigma| + 1, (1 + q_j) \times |\Sigma|]$ at level $j + 1$ such that $T_{j+1,q_{j+1}} =$ "\$". If one can not be found, find a $q_{j+1}$ such that $T_{j+1,q_{j+1}}$ is empty and set its $r_1$ value to "\$" and $r_2$'s value to $\ell$. Where $\$ \notin \Sigma$. Add $\mathsf{mem} \leftarrow \mathsf{mem}||\mathsf{id}(D_i)$, since $\mathbf{w} \in D_i$ to node $T_{j+1,q_{j+1}}$.
  4) From $T$ build the set of perfect hash tables $\{\mathcal{H}_i | 1 \leq i \leq |\mathcal{G}|\}$.
  5) For each node $T_{j,q_j}$ in $T$
     a) If $T_{j,q_j}$ is a leaf node, $\mathsf{mem} \leftarrow \mathsf{mem}||\mathcal{F}_{\mathsf{keyder}(\mathsf{stm}_1)}(\mathsf{mem})$.
     b) If $T_{j,q_j}$ is not a leaf node, $T_{j,q_j}[r_3] = \mathcal{F}_{\mathsf{keyder}\left(\mathsf{stm}_{T_{j,q_j}[r_0]}\right)}\left(j||T_{j,q_j}[r_1]||\mathsf{parent}\left(T_{j,q_j}\right)[r_3]\right)$.
  6) Pad all remaining $r_3$ fields in nodes with bit-strings of the same length as the other non-zero nodes of the trie. As well as clearing all values in $r_0$ and $r_1$.
  7) Return $\mathcal{I} = (T, \{\mathcal{H}_i | 1 \leq i \leq |\mathcal{G}|\})$. The value, $\{\mathcal{H}_i | 1 \leq i \leq |\mathcal{G}|\}$ is given to the data owner and not the cloud.
$\mathsf{HAddUser}(K_O, U, g)$
  1) Parse $K_O$ as $\mathsf{stp}$.
  2) If $1 \leq g \leq |\mathcal{G}|$, if not return $\perp$.
     a) set $\mathsf{newstp} = \mathsf{stp}$
     b) For $i = 1$ to $g$, $(\mathsf{newstp}, \mathsf{stm}) = \mathsf{wind}(\mathsf{newstp})$
  3) Return $K_U = (\mathsf{stm}, \mathcal{H}_g)$.
$\mathsf{HTrapdoor}(K_u, w = (w[1], w[2], \ldots))$
  1) Parse $K_u$ as $(\mathsf{stm}_g, \mathcal{H}_g)$.
  2) Set $s = \mathcal{H}_g[h_g(w)]$ and $\mathsf{curstm} = \mathsf{stm}_g$, $\pi[0] = 0$, and $w[|w| + 1] =$ "\$".
  3) For $j = 1$ to $|w| + 1$
     a) For $i = 1$ To $i < s[j]$, $\mathsf{curstm} = \mathsf{unwind}(\mathsf{curstm})$.
        i) If $\mathsf{curstm} = \perp$, return $\perp$.
     b) Set $\pi[j] = \mathcal{F}_{\mathsf{keyder}(\mathsf{curstm})}(j||w[j]||\pi[j-1])$
  4) Return the privacy preserving trapdoor $T_{u,w} = \pi$.
$\mathsf{HSearch}(\mathcal{I}_\mathcal{D}, T_{U,w})$
  1) Parse $T_{U,w}$ as $\pi[1 \ldots n]$ and set $q_0 = 0$.
  2) For $j = 1$ to $n$
     a) Find a $q_j \in [q_{j-1} \times |\Sigma| + 1, (1 + q_{j-1}) \times |\Sigma|]$ such that $T_{j,q_j}[r_3] = \pi[j]$. If found, go back to top of loop, otherwise return $\perp$.
  3) If $T_{j,q_j}$ has no child, then return the document collection $T_{j,q_j}[r_2]$.
```

Fig. 3: A complete HAC-SSE system

To verify the results returned by the cloud we utilize the verification algorithm given in Figure 5. This algorithm is a slightly modified version of Verify in [2].

To handle the need for revocations in our basic system we again modify our construction. The idea is to use a traditional broadcast encryption system and a keyed pseudo-random permutation ϕ to manage group membership. Instead of sending trapdoor T_w to the cloud, we send $\phi(T_w)$. When the query arrives at the cloud, the cloud inverts the permutation (computes $\phi^{-1}(T_w)$) to recover the trapdoor T_w. To enable dynamic membership, this pseudo-random permutation is indexed by a key v. In [1] the value v is changed, via a broadcast encryption, each time a user leaves the system. In our system we will make use of a second Key Regression system. Recall that Key Regression allows for a content owner to share access to data with users. Moreover, it allows

```
1) For $y = 1$ To $|\mathcal{G}|$
    a) Set $T_{i,j}[r_4]$ as:
       $T_{i,j}[r_4] = T_{i,j}[r_4] || \mathcal{E}_{K_{\mathsf{keyder}(\mathsf{stm}_v)}}(b_y || T_{i,j}[r_3])$   (2)
```

Fig. 4: Modification to the BuildIndex algorithm to add verification support to the trie

```
HVerify $(w = (w_1, w_2, \ldots), \{\mathsf{proof}_i | 1 \leq i \leq |w|\}, \pi)$:
1) Determine the key schedule $s$ from the hash table $\mathcal{H}_g$
2) If the cloud responded positively we walk the list of proofs $\mathsf{proof}_t$, For $t = 1$ to $r - 1$ we,
    a) Parse $\mathsf{proof}_t$ as $e_1 || e_2 || \ldots || e_{|G|}$
    b) Decrypt the the $g$-th entry in $\mathsf{proof}_t$ to obtain, $b_g || \sigma$. Where $b_g$ denotes the children accessible from $w_t$ and $\sigma$ is the prefix signature for the node.
    c) If $w_{t+1}$ is not one of the children listed in bit vector $b_g$ or $\pi[t] \neq \sigma$, return false.
3) return true.
```

Fig. 5: The HVerify algorithm

a finite number of revocations of access. To add revocation to our system, as described in Figure 3, we modify, HSetup, HSearch, HTrapdoor, and define the routine HRevokeUser from the HAC-SSE model. We start by modifying HSetup to take an additional parameter that describes the maximum number or revocations, ρ, that the system should permit. The content owner, can now, run setup to initialize a new Key Regression system. The HSearch function should apply the inverse pseudo-random permutation, $\phi^{-1}_{\mathsf{keyder}(\mathsf{stm}_*)}$, to the trapdoor $T_{u,w}$. Where stm_* denotes the current member state. We modify the HTrapdoor algorithm to return the trapdoor, $T_{u,w}$, with the pseudo-random permutation $\phi_{\mathsf{keyder}(\mathsf{stm}_*)}$ applied. The revocation algorithm is designed to be run by the owner to revoke a user, u, from any group in the entire system. The algorithm appears in figure 6. We note here, that our

```
HRevokeUser $(K_O, u, g)$:
1) Parse $K_O$ as $(\mathsf{stp}, \mathsf{stp}^r)$
2) Run $(\mathsf{stm}_*, \mathsf{stp}) = \mathsf{wind}(\mathsf{stp})$ to obtain the next key material
3) Sent to the cloud and all users, except $u$, the new memberstate $\mathsf{stm}_*$.
```

Fig. 6: The HRevokeUser algorithm

method for revoking access to searches relies on the trust that the cloud will not give away the key for the pseudo-random permutation.

6.1 Security Guarantees

We emphasize that revoked users are not able to issue successful queries after they are revoked. This is due to the assumption that a member state is not given to any user by the cloud. This directly implies that revoked users cannot generate a trapdoor, for they are unable to apply the correct pseudo-random permutation.

We now discuss the unforgeablility of verification tags in the system. In order for the cloud to successfully forge a verification tag it would need to be able recover at least one encryption key, since the cloud can not forge a valid encryption. Moreover, the cloud can not use a different verification tag from the tree as the hash of the prefix of the query is included in the verification tag, thus binding the verification tags to specific nodes.

7. Conclusion

We presented a secure SSE scheme with hierarchical access control for multiple groups of users under the SHBC model. Our system can support both addition and eviction of group members. One application of our system is tagging documents with security terms (e.g., *need-to-know*, *secret*, *top-secret*), and with efficient secure search that enforces the data hiding property. This data hiding property amounts to obscuring what documents are tagged with the *top-secret* designation. Another application is providing filters over search engine queries thus providing a child lock on Internet searches while still maintaining query privacy. Yet another application is providing protection for patient records allowing only certain classes of physicians to query for certain diseases or other medical tagging.

Future work in this area will be devoted to constructing both adaptively secure hierarchical access control system as well as investigating non-hierarchical access control systems. Additional future directions would be extending keyword searches to phrase searches, as well as other privacy-preserving queries over a search index.

Appendix

We prove Theorem 2 under the framework in [1]. We will use the following definitions.

Definition 4 (History, View, and Trace [1]): Let $\mathcal{D}$ be a collection of n documents, and Δ a dictionary. A *history* H_q is an interaction between a client and a server over q queries, which is denoted by $H_q = (\mathcal{D}, w_1, w_2, \ldots w_q)$.

An adversary's *view* of H_q under secret key K is defined by

$$V_K(H_q) = (\mathsf{id}(D_1), \ldots, \mathsf{id}(D_n), \mathcal{E}(D_1), \ldots, \mathcal{E}(D_n), \mathcal{I}, T_1, \ldots, T_q), \quad (3)$$

where $T_1, \ldots, T_q$ are a series of trapdoors and $\mathcal{I}$ is an index.

The *trace* of H_q is the following sequence:

$$\mathrm{Tr}(H_q) = (\mathsf{id}(D_1), \ldots, \mathsf{id}(D_n), |D_1|, \ldots, |D_n|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_q), \pi_q), \quad (4)$$

where π_q is the search pattern of the user.

For our index we define the search pattern by creating a matrix $T_{q\times k}$, where every word in the dictionary Δ is of length at most k. Each entry $T_{i,j}$ in the matrix is a $q \times k$ binary matrix E, constructed in a way such that $E_{u,v} = 1$ if the j^{th} letter of word i is the same as the u^{th} letter of word v.

Proof of Theorem 2: We recall the definition of non-adaptive security from [1] and proceed to prove non-adaptive security of HAC-SSE. We describe a probabilistic polynomial-time simulator, $\mathcal{S}$, such that for all $q \in \mathbb{N}$, all probabilistic polynomial-time adversaries, $\mathcal{A}$, all distributions $L_q = \{H_q | \mathsf{Tr}\,(H_q) = \mathsf{Tr}_q\}$, where Tr_q is some q sized trace. Simulator, $\mathcal{S}$ can construct a view V_q^* such that $\mathcal{A}$ can not distinguish from a genuine view $V_K\,(H_q)$. Given that $\mathcal{S}$ is provided with $\mathsf{Tr}\,(H_q)$ for any $q \in \mathbb{N}$.

For $q = 0$, the simulator $\mathcal{S}$ constructs a V^* that is indistinguishable from $V_K\,(H_0)$ for any $H_0 \overset{R}{\leftarrow} L_0$. In particular, $\mathcal{S}$ generates $V^* = \{1, \ldots, n, e_1^*, \ldots e_n^*, \mathcal{I}^*\}$, where $e_i^* \overset{R}{\leftarrow} \{0,1\}^{|D_i|}$ for all $1 \leq i \leq n$ and $\mathcal{I}^* = T^*$. The simulator $\mathcal{S}$ generates a complete $|\Sigma|+1$-ary trie T^* of height $\max_{|w_i|}\,(w \in \Delta)$, and fills each node with a random number from $\{0,1\}^z$. The leaf nodes should be filled with a series of random numbers from 1 to n. We claim that V^* and $V_K\,(H_0)$ are indistinguishable. By a standard hybrid argument we must show that $\mathcal{A}$ cannot distinguish any element in V^* from the corresponding element in $V_K\,(H_0)$. This is true simply because the document identifiers in V^* and $V_K\,(H_0)$ are computational indistinguishable. It remains to argue that index $\mathcal{I}^* = T^*$ and $\mathcal{I} = T$ are indistinguishable and that the encrypted documents are indistinguishable. To see that T^* and T are indistinguishable, we note that every node in T^* are binary string in $\{0,1\}^z$ and the nodes of T are either a hash value from the set $\{0,1\}^z$ or a random binary string from $\{0,1\}^z$. In either case, the nodes are indistinguishable. In the case of the encrypted documents we observe that since $(G, \mathcal{E}, D)$ is a semantically secure encryption scheme, we conclude that e_i^* is indistinguishable from the associated encryption in $V_K\,(H_0)$.

For $q > 0$, simulator $\mathcal{S}$ constructs V^* as $V_q^* = \{1, \ldots, n, e_1^*, \ldots, e_n^*, \mathcal{I}^*, T_1^*, \ldots, T_q^*\}$, where $\mathcal{I}^* = T^*$ is a complete $|\Sigma|$-ary trie. Each value in the trie is drawn randomly from $\{0,1\}^z$. The trapdoors T_i^* are constructed as root-to-leaf paths through the trie. They are constructed in such a way that they have the correct length. Briefly, to get the correct length for word w_i the simulator inspects $E_{j,i}$. If $T_{j,i}$ is the zero matrix, then set $|w_i| = j - 1$. The trapdoors T_i^* are also constructed in a way such that they will lead to a leaf node that contains the appropriate document set. The encrypted documents as well as the document identifiers, are still indistinguishable for the same reasons as the case when $q = 0$. The index is likewise indistinguishable. The trapdoors are indistinguishable as they consist of a sequence of random values that are indistinguishable from the function used to create genuine trapdoors. Finally, we note that they are indistinguishable in length as well. This is because the search pattern of the trace provides the simulator with sufficient information to construct a trapdoor of the correct size.

We have thus described a polynomial-time simulator, $\mathcal{S}$, that can create a view indistinguishable from a genuine view for any polynomial-time adversary $\mathcal{A}$. This completes the proof.

In a straightforward manner, one can extend the proof to handle verification tags. To do this one must augment the trace, both genuine and simulator constructed, to contain the tags. To see that this does not change the proof above, we note that the verification tags themselves are the results of a semantically secure symmetric encryption system and thus are indistinguishable from random.

Acknowledgment

This work was supported in part by the NSF under grants CNS-1018422, CNS-1248380, and CNS-1247875. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS ’06. New York, NY, USA: ACM, 2006, pp. 79–88.

[2] Q. Chai and G. Gong, “Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers,” in *IEEE International Conference on Communications, ICC’12*, June 2012.

[3] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, “Fuzzy keyword search over encrypted data in cloud computing,” in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM’10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 441–445.

[4] J. Wang, X. Chen, H. Ma, Q. Tang, J. Li, and H. Zhu, “A verifiable fuzzy keyword search scheme over encrypted data,” *Journal of Internet Services and Information Security (JISIS)*, vol. 2, no. 1/2, pp. 49–58, 2 2012.

[5] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, ser. SP ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 44–.

[6] Y. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proceedings of the Third international conference on Applied Cryptography and Network Security*, ser. ACNS’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 442–455.

[7] E. Goh, “Secure indexes,” Cryptology ePrint Archive, Report 2003/216, 2003, http://eprint.iacr.org/2003/216/.

[8] Y. Tang, D. Gu, N. Ding, and H. Lu, “Phrase search over encrypted data with symmetric encryption scheme,” *2012 32nd International Conference on Distributed Computing Systems Workshops*, vol. 0, pp. 471–480, 2012.

[9] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds. Springer Berlin / Heidelberg, 2004, vol. 3027, pp. 506–522.

[10] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[11] E. Fredkin, “Trie memory,” *Commun. ACM*, vol. 3, no. 9, pp. 490–499, Sep. 1960.

[12] K. Fu, S. Kamara, and Y. Kohno, “Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage,” in *Network and Distributed System Security Symposium (NDSS ’06)*, 2006.