Multi-level Searchable Symmetric Encryption

Sarah Louise Renwick
Information Security Group, Royal Holloway, University of London, Egham, United Kingdom
sarahlouise.renwick.2012@live.rhul.ac.uk

James Alderman
Information Security Group, Royal Holloway, University of London, Egham, United Kingdom
james.alderman@rhul.ac.uk

Keith M. Martin
Information Security Group, Royal Holloway, University of London, Egham, United Kingdom
keith.martin@rhul.ac.uk

ABSTRACT

Remote storage delivers a cost effective solution for data storage. If data is of a sensitive nature, it should be encrypted prior to outsourcing to ensure confidentiality; however, searching then becomes challenging. Searchable encryption is a well-studied solution to this problem. Many schemes only consider the scenario where users can search over the entirety of the encrypted data.

In practice, sensitive data is likely to be classified according to an access control policy and different users should have different access rights. It is unlikely that all users have unrestricted access to the entire data set. Current schemes that consider multi-level access to searchable encryption are predominantly based on asymmetric primitives. We investigate symmetric solutions to multi-level access in searchable encryption where users have different access privileges to portions of the encrypted data and are not permitted to search over, or learn information about, data for which they are not authorised.

Keywords

Searchable Symmetric Encryption, Multi-level, Access Control, Information Flow Policy

1. INTRODUCTION

Searchable encryption (SE) enables a user to search over encrypted data that has been outsourced to a remote server. In some schemes [4, 5, 9, 18, 19, 20], the data owner may authorise multiple users to make search queries — in such cases, a querier is either authorised to search over the entirety of the data or not at all, in which case (ideally) no information about the outsourced data should be revealed. In practice, the access control requirements of outsourced data sets are likely to be more fine-grained than this binary ‘all or nothing’ approach; hence existing schemes do not suffice.

We study the problem of enforcing a multi-level access control policy (MLA) in the context of searchable symmetric encryption (SSE). As a notable example of this form of data classification, the UK government uses three levels of data classification: official, secret and top secret [16]. In our model, a user with ‘secret’ clearance should be unable to learn any information about data items classified as ‘top secret’, such as whether they contain searched keywords or not. This is an example of an information flow policy with a total order of security labels [2].

More precisely, consider a (possibly large) data set which is to be outsourced to an external storage provider, which could be outside of the data owner’s trusted zone. Although the provider has a business incentive to provide a storage and search service to the client (and to any other users authorised by the data owner), the provider may attempt to learn information about the sensitive data stored; in short, the storage provider may be honest-but-curious. Hence, the data must be encrypted prior to outsourcing, and the search procedure should not reveal unintended information to the storage provider or to other unauthorised entities. Each data item within the data set may be associated with some keywords, over which searches may be performed. Furthermore, each data item may differ in sensitivity and have different access control requirements. The data owner may authorise additional users to search the data set and, again, each user may have different access control clearance and therefore be able to access or search different sets of data items. Let us define a set of security labels \( L \), which forms a totally ordered set \( (L, \leq) \) to reflect the inheritance of access rights. Each user \( u \) and data item \( d \) is assigned one of these labels, denoted \( \lambda(u) \) and \( \lambda(d) \) respectively. A user \( u \) may search a data item \( d \) if and only if \( \lambda(u) \geq \lambda(d) \).

Public-key encryption (PKE), especially functional encryption, has previously been used to achieve MLA in SE [3, 11, 15, 21]. In general, PKE is computationally more intensive than symmetric key encryption (SKE), perhaps making SKE more suitable for practical systems. The enforcement of MLA policies in symmetric SE has, up to now, remained relatively unexplored. Kissel et al. [14] presented a SKE-based scheme in which users are divided into groups that each have a specified dictionary of keywords they may search over. These groups are arranged hierarchically so that each group may also search for all keywords in dictionaries assigned to groups at lower levels in the hierarchy. Although this scheme presents a form of hierarchical access in SSE, users may still search over the entire data set. In most access control scenarios, we are concerned with protecting a data item (i.e., the complete content of a data item), not just a single keyword describing the data item. Furthermore,
it may be difficult to correctly administer an access control policy expressed only in terms of authorised keywords; data items may gain their classification level due to semantic meaning regarding their contents (for example, the subject to which they pertain), which may not trivially be captured through the associated keywords. For example, consider two data items containing information about company spending: one providing a public report of company-wide spending, whilst the other pertains specifically to the research department. Whilst both items may be labelled by a keyword such as ‘finance’, detailed knowledge of research spending may be deemed more sensitive than a generalised report. Simply authorising users to search for keywords, such as ‘finance’, does not suffice in this instance as not all users that can search the public report should also be able to view the specific report. The access control policy in this case must be managed carefully — perhaps additional, more granular, keywords must be defined e.g. ‘finance-public’ (leading to an increase in the size of the searchable encryption index and a subsequent loss of efficiency) or a (less efficient) SE scheme that supports ‘conjunctive keyword-only access control’ would be required such that one can be authorised to search for (‘finance’ AND ‘public’) and only data items with both keywords would be returned. In this work, we consider the problem of fine-grained classification of data items directly and gain a more efficient solution.

In this work, we consider Multi-level Searchable Symmetric Encryption (MLSSE). We begin in Section 2 by reviewing background material, before defining our system and security models in Sections 3.1 and 3.2. In Section 3.3, we introduce an instantiation based on the constructions of [9, 13], and then show, in Section 3.5, how to extend our construction to support a dynamic data set using techniques from [15]. Section 3.6 discusses the efficiency of our scheme. The full security proofs of our constructions are omitted but are available in the full version of our paper [1].

2. BACKGROUND

We aim to enforce information flow policies within searchable encryption, which encompass a wide range of access control policies that are of practical interest, including the Bell-LaPadula model, temporal, role-based and attribute-based access control [8].

Definition 1. An information flow policy is a tuple \( \mathcal{P} = (\{L, \leq\}, U, D, \lambda) \), where \( \{L, \leq\} \) is a partially ordered set (poset) \(^1\) of security levels, \( U \) is a set of users, \( D \) is a set of objects (data items), and \( \lambda : U \cup D \rightarrow L \) is a function mapping users and objects to security levels in \( L \). We say that \( u \in U \) is authorised to read (search) an object \( d \in D \) if \( \lambda(d) \leq \lambda(u) \).

In this paper, we will focus on the case where \( \{L, \leq\} \) is a total order (chain) giving a simple hierarchy of security levels and, without loss of generality, we assume that each user and object is assigned to at most one security label. Given a set \( X \), we denote the power set of \( X \), comprising all combinations of elements in \( X \), by \( 2^X \). Throughout this paper we refer to ‘security levels’ and ‘security labels’ as access levels.

\(^1\) A poset is a set of labels \( L \) and a binary order relation \( \leq \) on \( L \) such that for all \( x, y, z \in L \), \( x \leq x \) (reflexivity), if \( x \leq y \) and \( y \leq x \) then \( x = y \) (antisymmetry), and if \( x \leq y \) and \( y \leq z \) then \( x \leq z \) (transitivity). If \( x \not\leq y \) then we may write \( y \geq x \).

Definition 2. A Multi-User Searchable Symmetric Encryption (MSSE) scheme is a set of six polynomial time algorithms defined as follows:

- \( K_0 \leftarrow \text{MSSE.KeyGen}(1^\lambda) \): A probabilistic algorithm run by the data owner that takes a security parameter \( k \in \mathbb{N} \) and outputs a secret key \( K_0 \).
- \( (I_D, st_O, st_S) \leftarrow \text{MSSE.BuildIndex}(K_O, D, G) \): A probabilistic algorithm run by the data owner that takes a set of data items \( D \), a set of authorized users \( G \) and the secret key \( K_O \). It outputs an index \( I_D \), and server and owner states \( st_S \) and \( st_O \).
- \( K_u \leftarrow \text{MSSE.AddUser}(u, K_O, st_O) \): A probabilistic algorithm run by the data owner that takes the identity, \( u \), of a user to be enrolled in the system along with the owner’s secret key and state. It outputs a secret key for the new user \( K_u \).
- \( T_u \leftarrow \text{MSSE.Query}(\omega, K_u)^2 \): A deterministic algorithm run by a user that takes a keyword \( \omega \) and the user’s secret key, and outputs a search token.
- \( R_u \leftarrow \text{MSSE.Search}(T_u, I_D, st_S) \): A deterministic algorithm run by the server that takes as input a search token, an encrypted index and the server state, and outputs a set \( R_u \) of identifiers of data items containing \( \omega \).
- \( (st_O, st_S) \leftarrow \text{MSSE.Revoke}(u, K_O, st_O) \): A probabilistic algorithm run by the data owner that takes a user identity of a user to be revoked along with the data owner’s secret key and state. It outputs new server and owner states.

For a data set \( D \) and keyword \( \omega \in \Delta \) (where \( \Delta \) is a dictionary of possible keywords), let us denote by \( D_\omega \) the expected results of searching for \( \omega \) in \( D \) (in the plain); informally we say that an MSSE scheme is correct if it also produces the output \( D_\omega \). More formally, a MSSE scheme \( \text{MSSE} \) is correct if for all \( k \in \mathbb{N} \), for all \( K_0 \) output by \( \text{MSSE.KeyGen}(1^\lambda) \), for all \( D \in 2^\Delta \) for all \( G \in 2^\Delta \), for all \( (I_D, st_O, st_S) \) output by \( \text{MSSE.BuildIndex}(K_O, D, G) \), for all \( \omega \in \Delta \): Search(MSSE.Query(K_\omega, I_D, st_S)) = D_\omega.

Definition 3. A Broadcast Encryption (BE) scheme is a set of four polynomial time algorithms as follows, where \( U \) is the user space of all possible user identities:

- \( (PP, K_{BE}) \leftarrow \text{BE.Keygen}(1^\lambda) \): A probabilistic algorithm that takes a security parameter \( k \) outputs public parameters \( PP \) and a master secret key \( K_{BE} \).
- \( C \leftarrow \text{BE.Enc}(M, G) \): A probabilistic algorithm that takes a plaintext \( M \), a set of users \( G \in U \) authorized to decrypt and produces a ciphertext \( C \).
- \( K_u \leftarrow \text{BE.Add}(K_{BE}, u) \): A probabilistic algorithm that takes as input the master secret key \( K_{BE} \) and a user identifier \( u \in U \), and outputs a user key \( K_u \).

\(^2\)This algorithm is sometimes referred to as MSSE.Trapdoor in the literature, however to maintain consistent notation throughout this paper we refer to it as MSSE.Query.
BE is correct if $\forall k \in \mathbb{N}$, for all PP and $K_{BE}$ output by $\text{BE.KeyGen}(1^k, m)$, for all $M$ in the plaintext space, all sets of users $G \subseteq U$, every $K_u$ output by $\text{BE.Add}(u, K_{BE})$ and all $C$ output by $\text{BE.Enc}(M, G)$ where $u \in G$ we have: $M \leftarrow \text{BE.Dec}(C, K_u)$.

3. MULTI-LEVEL ACCESS IN SEARCHABLE SYMMETRIC ENCRYPTION

A MLSSE scheme permits searching over encrypted data in the symmetric key setting for multiple users that have varying access rights to the set of data items. The access levels are hierarchical (totally ordered), meaning a user may vary their access rights to the set of data items. The access in the symmetric key setting for multiple users that have $\delta$ as $d$ index over some data items within $D$ and outsource to $d$ all data items by security classification. We denote the information regarding data itself, the identifier, the associated keywords and the $\lambda$ keywords they may search for [14]. A user with access level $\lambda$ search query if they are authorized in the BE scheme.

To enable searching over the encrypted data, we load some encrypted metadata to the server. It first de-envelops all data items at their own access level as well as all levels are hierarchical (totally ordered), meaning a user may have $\delta$ varying access rights to the set of data items. The access levels are hierarchical (totally ordered), meaning a user may have $\delta$ varying access rights to the set of data items. The access in the symmetric key setting for multiple users that have $\delta$ as $d$ index over some data items within $D$ and outsource to $d$ all data items by security classification. We denote the information regarding data itself, the identifier, the associated keywords and the $\lambda$ keywords they may search for [14]. A user with access level $\lambda$ search query if they are authorized in the BE scheme.

An MLSSE scheme is correct if for all $K_O, k_S$ output by $\text{KeyGen}(1^k, S, P)$, for all $D^{aug}$, for all $I_D$ output by $\text{BuildIndex}(D^{aug}, K_O, PP)$, for all $\omega \in \Delta$, for all $u \in U$, for all $K_u$ output by $\text{AddUser}(u, \lambda(u), K_O, PP)$, $\text{Search}(\omega, K_u)$, $I_D, k_S = D_{\omega, \lambda(u)}$.

3. Security model

A secure MLSSE scheme would, ideally, reveal no information regarding the data set $D$ to the server (i.e. a curious server cannot learn information about the data it stores) and reveal no information to users regarding data items that they are not authorised to search. However, most SSE schemes leak additional information to gain efficiency. For example, the search results $\{R_{u_1, a_1}, ..., R_{u_p, a_p}\}$ for a set of queries $\{T_{u_1}, ..., T_{u_p}\}$ could be revealed. This is referred to as the access pattern (Definition 5) and defines the link between permit data items to be of any format and the sets of keywords can be arbitrarily chosen from the dictionary — they may not necessarily correspond to the actual content of the data, but could be descriptive attributes of the data item. This may help minimise the risk of a statistical attack on the index as the frequency of a certain word in a document is not necessarily reflected in the set of keywords chosen to index the data item.

Definition 4. A Multi-level Searchable Symmetric Encryption Scheme (MLSSE) scheme consists of six algorithms defined as follows:

- $(K_O, k_S, PP) \xleftarrow{\$} \text{KeyGen}(1^k, S, P)$: A probabilistic algorithm run by the data owner $O$ that takes the security parameter $k$, policy $P$ and the server identity $S$, and outputs $O$'s secret key $K_O$, a server key $k_S$ and public parameters $PP$.
- $I_D \xleftarrow{\$} \text{BuildIndex}(D^{aug}, K_O, PP)$: A probabilistic algorithm run by $O$. It takes the description of the data set $D^{aug}$ and $O$'s secret key, and outputs the index $I_D$.
- $(K_u, PP) \xleftarrow{\$} \text{AddUser}(u, \lambda(u), K_O, PP)$: A probabilistic algorithm run by $O$ to enrol a new user into the system. It takes the new user's identifier $u$ and access level $\lambda(u)$, and $O$'s key, and outputs a secret key for the new user.
- $T_{\omega, \lambda(u)} \xleftarrow{\$} \text{Search}(\omega, K_u)$: A deterministic algorithm run by a user with access level $\lambda(u)$ to generate a search query. It takes as input a keyword $\omega \in \Delta$ and the user's secret key and outputs a search query $T_{\omega, \lambda(u)}$.
- $R_{\omega, \lambda(u)} \xleftarrow{\$} \text{RevokeUser}(u, K_O, PP)$: A probabilistic algorithm run by $O$ to revoke a user from the system. It takes the user's id, the data owner's and server's secret keys, and outputs updated owner and server keys.

An MLSSE scheme is correct if for all $k \in \mathbb{N}$, for all $K_O, k_S$ output by $\text{KeyGen}(1^k, S, P)$, for all $D^{aug}$, for all $I_D$ output by $\text{BuildIndex}(D^{aug}, K_O, PP)$, for all $\omega \in \Delta$, for all $u \in U$, for all $K_u$ output by $\text{AddUser}(u, \lambda(u), K_O, PP)$, $\text{Search}(\omega, K_u)$, $I_D, k_S = D_{\omega, \lambda(u)}$.

3.2 Security model

A secure MLSSE scheme would, ideally, reveal no information regarding the data set $D$ to the server (i.e. a curious server cannot learn information about the data it stores) and reveal no information to users regarding data items that they are not authorised to search. However, most SSE schemes leak additional information to gain efficiency. For example, the search results $\{R_{u_1, a_1}, ..., R_{u_p, a_p}\}$ for a set of queries $\{T_{u_1}, ..., T_{u_p}\}$ could be revealed. This is referred to as the access pattern (Definition 5) and defines the link between
a search query and the search results it produces; it may be thought of as a database where each row stores a search query and a corresponding set of identifiers of data items that satisfies the search query.

Most efficient SSE schemes also leak the search pattern (Definition 6), which reveals the set of search queries made to the server. In most single-user SSE schemes [6, 7, 9, 10, 12, 13], search queries are formed deterministically; the server can therefore ascertain whether a search query has been made previously.

**Definition 5.** For a sequence of $q$ search queries
\[
\Omega = \{(T_{\omega_1,a_1}, \ldots, T_{\omega_q,a_q})\} \text{ for which } 1 \leq i, j \leq q: \omega_i \text{ and } \omega_j \text{ or } \lambda(u_i) \text{ and } \lambda(u_j) \text{ are not necessarily distinct for } i \neq j, \text{ the access pattern is defined as:}
\]
\[
AP(I_D, \Omega) = \{(T_{\omega_1,a_1}, R_{\omega_1,a_1}), \ldots, (T_{\omega_q,a_q}, R_{\omega_q,a_q})\}.
\]

**Definition 6.** For a sequence of $q$ search queries
\[
\Omega = \{(T_{\omega_1,a_1}, \ldots, T_{\omega_q,a_q})\} \text{ for which } 1 \leq i, j \leq q: \omega_i \text{ and } \omega_j \text{ or } \lambda(u_i) \text{ and } \lambda(u_j) \text{ are not necessarily distinct for } i \neq j, \text{ the search pattern is defined as a } q \times q \text{ symmetric binary matrix}
\]
\[SP(I_D, \Omega) \text{ such that for } 1 \leq i, j \leq q:
\]
\[SP(I_D, \Omega)_{i,j} = 1 \iff T_{\omega_i,a_i} = T_{\omega_j,a_j}.
\]

Intuitively, the search pattern reveals when the ith and jth queries are the same, which happens when queries are issued for the same keyword by users with the same access level.

**Definition 7.** For an index $I_D$ we define the setup leakage $L_{\text{Setup}}(I_D)$ to be all the information that is leaked by the index $I_D$.

**Definition 8.** For an index $I_D$ and set of $q$ search queries $\Omega = (T_{\omega_1}, \ldots, T_{\omega_q})$ we define the query leakage $L_{\text{Query}}(I_D, \Omega)$ to be all the information leaked by evaluating the queries in $\Omega$ on the index $I_D$.

We now formalise the notions of security we require in MLSSE. We use cryptographic games to formalise our notions of security. For each game, a challenger $C$ instantiates a probabilistic polynomial time (PPT) adversary $A$ whose inputs are chosen to reflect the information available to a realistic adversary. Our notion of adaptive security is based on that of IND-CCA2 presented in [9]. In the following we represent the dictionary of keywords as $\Delta$, $\lambda$ defines the mapping function as described in Section 3.

### 3.2.1 Multi-level Access

Our first security notion, in Figure 1, is that of multi-level access which requires that a user, $u$, cannot receive search results or learn information relating to data items $d_i$ such that $\lambda(u) < \lambda(d_i)$. More specifically, a server colluding with several users cannot learn anything about the index beyond the specified leakage according to the corrupt users’ access rights.

We define a maximal query leakage with access level $\lambda_{\text{max}}$ on $I_D$ to be $L_{\text{Qmax}}^{\lambda_{\text{max}}}(I_D, (T_{\omega_i,\lambda_{\text{max}}})) \text{ — this is the leakage resulting from every possible keyword search with the maximal access level available, in Game 1 we denote this as } L_{\text{max}}^{\lambda}(I_D)$. The challenger sets up the system, including instantiating several global variables (which the challenger can use in the main game and in oracle functions, but which the adversary cannot see): $L$ is a list of users that have been corrupted, $\lambda_{\text{max}}$ is the maximal access level of any corrupted user, and $\text{call}$ is a Boolean flag to show whether the challenge parameters have been generated yet. The adversary is given the security parameter, access control policy, server key and the public parameters, as well as providing access to the following oracles.

The AddUser oracle allows the adversary to enrol a user into the system, and the adversary corrupts this user by receiving the user key. If the challenge has not yet been generated, then the challenger adds the requested user to the list $L$ of corrupted users, checks if the maximal access level of corrupted users needs updating, and runs the AddUser algorithm. Otherwise, if the challenge has been generated, the above procedure is carried out only if the maximal query leakage for the new user’s access level is equal on both challenge data sets — that is, providing the user key for the queried user cannot allow the adversary to trivially distinguish the two data sets.

The RevokeUser oracle first checks that the requested user has indeed been added previously. If so, it removes the user identity from $L$ and checks whether the maximal access level needs changing. It returns the server key resulting from running the RevokeUser algorithm.

The BuildIndex oracle simply runs BuildIndex and returns the output to the adversary.

After a polynomial number of queries, the adversary outputs two data sets which must have identical maximal query leakages for the maximal access level of any corrupted user. The adversary cannot choose data sets where a user that it has corrupted could make any query that legitimately distinguishes the data sets since this would count as a trivial win. Whilst this may appear to be a strong assumption, we believe it to be the minimal assumption necessary to avoid trivial wins in the multi-user setting. The main issue is that in the multi-user setting it is necessary to consider the server colluding with a set of users (but not the data owner); as such, the adversary is able to perform the roles of the server and of an authorised user, and therefore may produce arbitrary search queries and perform searches themselves. Thus, the challenger in the game is unable to monitor which searches have been performed and hence cannot determine whether the query leakages of the actual queries on both data sets are equal, and instead must rely on the stronger assumption that no possible authorised search query can distinguish the data sets. Note that Van Rompay et al. [17] deal with the multi-user case without this assumption since they deal with single word indexes and have a proxy through which all queries are made.

The challenger sets the challenge flag to true and chooses a random bit $b$ which determines the data set used to form an index. The adversary is given the index and oracle access as described in Game 1 and must determine which data set was used.

**Definition 9.** (Multi-level Access) Let $\mathcal{ML}$ be a multi-level searchable symmetric encryption scheme where $k \in \mathbb{N}$ is the security parameter, $P$ is an information flow policy, $S$ is the identity of the server and $A$ a PPT adversary. The advantage of $A$ is:

\[
Adv_A^{\text{ML}}(\mathcal{ML}, 1^k, P) = | \Pr[A^{\mathcal{ML}}(\mathcal{ML}, 1^k, S, P) = 1] - \frac{1}{2} |.
\]

We say that $\mathcal{ML}$ is $(L_{\text{Setup}}, L_{\text{Query}})$-secure against adaptive
chosen keyword attacks in the sense of Game 1 if for all \( A \), all \( k \in \mathbb{N} \), all \( S \) and all \( P \), \( Adv^\text{MLA}_{A}(\mathcal{ML}, 1^n, S, P) \leq \text{negl}(k) \) for a negligible function \text{negl}.

### 3.2.2 Revocation Security

In MLSSE, as with other multi-user SSE schemes, we need to consider user revocation to remove a user’s ability to submit valid search queries to the server, and hence receive search results. We capture this in Game 2. The adversary is given the public parameters and selects a data set (along with associated access levels, keywords and identifiers). The challenger then creates the index. The adversary is given access to a set of oracles that perform the \( \text{AddUser}(u, \lambda(u), K_O, PP) \), \( \text{Search}(\mathcal{I}_D, k) \) and \( \text{RevokeUser}(u, K_O, PP) \) functions, where the parameters represented by \( \cdot \) are provided by the adversary, and the adversary is given the resulting user keys and search results. Once the adversary has completed his queries, the challenger revokes all users that were queried to the \( \text{AddUser} \) oracle but were not subsequently queried to the \( \text{RevokeUser} \) oracle (i.e. all users for which the adversary holds a valid user key). The adversary must then produce a search query \( T \) which, when used as input to the \( \text{Search} \) algorithm, does not produce \( \bot \) i.e. the adversary must produce a valid search query even though it does not hold a non-revoked key.

**Definition 10. (Revocation)** Let \( \mathcal{ML} \) be a multi-level searchable symmetric encryption scheme where \( k \in \mathbb{N} \) is the security parameter, \( S \) the server identity, \( P \) an information flow policy and \( A \) a PPT adversary. We define the advantage of \( A \) in Game 2 as:

\[
Adv^\text{Revoke}_{A}(\mathcal{ML}, 1^n, S, P) = |\mathbb{P}[\text{Exp}^\text{Revoke}_{A}(\mathcal{ML}, 1^n, S, P) = 1] - \frac{1}{2}|.
\]

### 3.3 Construction

Our construction MLSSE is an adaptation of the scheme of Kamara et al. [13], which is based on the construction of the influential inverted index scheme SSE-1 by Curtmola et al. [9].

Informally, MLSSE scheme uses an array \( A \) of linked lists, along with a look-up table \( T \) to index the encrypted data. This produces a sequential search that lends itself well to the hierarchical access rights on the data items that we require. For each keyword \( \omega \in \Delta \), we define a list \( L_{\omega} \) which stores the identifiers for all data items containing that keyword and is ordered according to the access level of the data items — data items with the highest classification are placed at the beginning of the list, and those with the lowest classification at the end. Each list \( L_{\omega} \) is encrypted and stored in \( A \) as a linked list. During the search phase the look-up table \( T \) is used to point the server to the correct node in the array depending on the information in the search query i.e. which keyword was searched for and what access rights the user that submitted the search query has. This node is decrypted using information in the search query and the node itself, revealing the address of the next node in the linked list.

The server may continue to decrypt all other relevant nodes in the linked list, obtaining the set of search results relevant to the user’s searched keyword and access level.
The key difference between our scheme and that of [13] is that, rather than pointing to the beginning of each linked list, the entry in $T$ will point to the appropriate position within the linked list according to the access rights of the querier (recall that the list is ordered by access levels). Since it is not possible to move backwards through the encrypted lists, the only search results available are those contained beyond this point in this list — that is, identifiers for those documents containing the keyword and whose classification is at most that of the querier, as required by the information flow policy.

Let $BE$ be an IND-CPA secure broadcast encryption scheme. We define the following pseudorandom functions (PRFs):

$$F : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k,$$

$$G : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^*,$$

$$P : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^k,$$

$$H : \{0,1\}^* \times \{0,1\}^k \rightarrow \{0,1\}^*,$$

and a pseudorandom permutation (PRP):

$$\phi : \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^k \rightarrow \{0,1\}^k \times \{0,1\}^* \times \{0,1\}^k,$$

$\Lambda$ is a $|\Delta| \times |L|$ array and $T$ is a dictionary of size $|\Delta| \times |L|$. We denote the address of a node $N$ in $\Lambda$ as $addr_{\Lambda}(N)$.

Let $\lambda$ map users and data items to their relevant access levels as described in Section 3.1. We define a function $\gamma$ which outputs three ordered lists $L_{\omega_i}, X_{\omega_i}$ and $N_{\omega_i}$ given the set of identifiers $D^{\omega i g}$ and the array $\Lambda$. We refer to the $n$th item in a list $L_{\omega_i}$ as $L_{\omega_i}[n]$. The list $L_{\omega_i}$ contains identifiers of data items in $D_{\omega_i}$ ordered from the identifiers with the highest to the lowest access levels, the list $N_{\omega_i}$ contains the addresses of $|L_{\omega_i}|$ nodes chosen randomly from $\Lambda$ and the list $X_{\omega_i}$ contains the indices of the identifiers in $L_{\omega_i}$ where each access level starts i.e. if we have an ordered list of identifiers $L_{\omega_i} = (id_1, id_2, id_3, id_4, id_5)$ we have that $X_{\omega_i}[3] = 4$, which says that the list of nodes with access level at most $a_3$ starts at the fourth entry in $L_{\omega_i}$. There is an entry per each access level in $X_{\omega_i}$, even if two access levels have the same starting point in $L_{\omega_i}$; from the example above we can see that $X_{\omega_i}[2] = X_{\omega_i}[3] = 4$. If an access level is not authorised to view any data items in $D_{\omega_i}$ then the entry corresponding to that access level (as well as the entries corresponding to all access levels below it) in $X_{\omega_i}$ is set to $\perp$. An identifier of a data item $d_i \in D_{\omega_i}$ will inherit the access level label of the respective data item, i.e. $\lambda(id_{d_i}) = \lambda(d_i)$.

**Algorithm 1 (K$_O$, K$_G$, PP) $\xleftarrow{}$ KeyGen(1$^n$, U, P)\**

1: for $i \in [1..|L|]$ do
2: \hspace{1em} $k_{a_i,1}, k_{a_i,2}, k_{a_i,3} \xleftarrow{} \{0,1\}^n$
3: \hspace{1em} $(PP_{\text{BE}}, k_{\text{BE}}) \xleftarrow{} \text{BE.KeyGen}(1^n, |\ell|)$
4: \hspace{1em} $st_O \xleftarrow{} \{0,1\}^n$
5: \hspace{1em} $S \xleftarrow{} \mathcal{U}$
6: \hspace{1em} $\mathcal{G} \leftarrow \{S\}$
7: \hspace{1em} $st_S \xleftarrow{} \text{BE.Enc}(st_O, \mathcal{G}, k_{\text{BE}})$
8: return
9: $K_{O} \leftarrow ((\{k_{a_i,1}\}_{i \in [1..|L|]}, \{k_{a_i,2}\}_{i \in [1..|L|]}, \{k_{a_i,3}\}_{i \in [1..|L|]}, k_{\text{BE}}, st_O)$
10: $PP \leftarrow (\mathcal{G}, st_S, PP_{\text{BE}})$
11: $K_S \xleftarrow{} \text{BE.Add}(k_{\text{BE}}, S)$

**Algorithm 2 $\mathcal{D}_T \xleftarrow{}$ BuildIndex($D_{\omega i g}, K_O, PP$)\**

1: $free \leftarrow \{\text{addr}(N_i)\}_{i \in [\Lambda]}$
2: for $1 \leq i \leq |\mathcal{W}|$ do
3: \hspace{1em} $(L_{\omega_i}, X_{\omega_i}, N_{\omega_i}) \xleftarrow{} \gamma(\mathcal{D}_{\omega_i})$
4: \hspace{1em} $free \leftarrow free \setminus N_{\omega_i}$
5: for $1 \leq j \leq |N_{\omega_i}| - 1$ do
6: \hspace{2em} $r_j \xleftarrow{} \{0,1\}^n$
7: \hspace{2em} $\Lambda[N_{\omega_i}[j]] \leftarrow (L_{\omega_i}[N_{\omega_i}[j]], X_{\omega_i}[N_{\omega_i}[j]], N_{\omega_i}[j]) \oplus H(P_{\Lambda[X_{\omega_i}[j]], X_{\omega_i}[N_{\omega_i}[j]], r_j})$
8: \hspace{2em} $r_{N_{\omega_i}[j]} \xleftarrow{} \{0,1\}^n$
9: \hspace{2em} $\Lambda[N_{\omega_i}[|N_{\omega_i}|]] \leftarrow (L_{\omega_i}[|N_{\omega_i}|], 0, 0) \oplus H(P_{\Lambda[X_{\omega_i}[|N_{\omega_i}|]], X_{\omega_i}[|N_{\omega_i}|], r_{|N_{\omega_i}|})$
10: for $1 \leq \ell \leq |L|$ do
11: \hspace{2em} if $X_{\omega_i}[\ell] \neq \perp$ then
12: \hspace{3em} $\mathcal{T}[k_{a_i,1}(\omega_i)] \leftarrow (N_{\omega_i}[X_{\omega_i}[\ell]], G_{k_{a_i,2}(\omega_i)})$
13: \hspace{2em} else
14: \hspace{3em} $\mathcal{T}[k_{a_i,1}(\omega_i)] \leftarrow \perp$
15: return $\mathcal{I}_D \leftarrow (\mathcal{G}, T)$

**Algorithm 3 (K$_u$, PP) $\xleftarrow{}$ AddUser(u, $\lambda(u), K_O, PP$)\**

1: $\mathcal{G} \leftarrow \mathcal{G} \cup \{u\}$
2: $k_u \xleftarrow{} \text{BE.Add}(k_{\text{BE}}, u)$
3: $st_S \xleftarrow{} \text{BE.Enc}(st_O, \mathcal{G}, k_{\text{BE}})$
4: return
5: $PP \leftarrow (P, \mathcal{G}, st_S, PP_{\text{BE}})$
6: $K_u \leftarrow (k_u, k_{\lambda(u),1}, k_{\lambda(u),2}, k_{\lambda(u),3})$

**Algorithm 4 (K$_O$, PP) $\xleftarrow{}$ RevokeUser(u, K$_O$, PP)\**

1: $st_O \xleftarrow{} \{0,1\}^n$
2: $\mathcal{G} \leftarrow \mathcal{G} \setminus \{u\}$
3: $st_S \xleftarrow{} \text{BE.Enc}(st_O, \mathcal{G}, k_{\text{BE}})$
4: return $K_O \leftarrow (\{k_{a_i,1}\}_{i \in [1..|L|]}, \{k_{a_i,2}\}_{i \in [1..|L|]}, \{k_{a_i,3}\}_{i \in [1..|L|]}, k_{\text{BE}}, st_O)$
Alg. 5 \((T_{\omega,\lambda(u)}) \leftarrow \text{Query}(\omega, K_u)\)

1. \(s_{l_0} \leftarrow \text{BE.Dec}(k_u, s_{3})\)
2. if \(s_{l_0} = \perp\) then
3. return \(\perp\)
4. \(t_{\omega,\lambda(u)} \leftarrow (R_{\lambda(u),1}(\omega), G_{\lambda(u),2}(\omega), R_{\lambda(u),3}(\omega))\)
5. return
6. \(T_{\omega,\lambda(u)} \leftarrow \phi_{s_{l_0}}(T_{\omega,\lambda(u)})\)

Alg. 6 \(R_{\omega,\lambda(u)} \leftarrow \text{Search}(T_{\omega,\lambda(u)}, \mathcal{T}_P, K_S)\)

1. \(s_{l_0} \leftarrow \text{BE.Dec}(K_G, s_{3})\)
2. Parse \(\phi_{s_{l_0}}(T_{\omega,\lambda(u)})\) as \((\tau_1, \tau_2, \tau_3)\)
3. \(R_{\omega,\lambda(u)} \leftarrow \emptyset\)
4. if \(\mathcal{T}[\tau_1] = \perp\) then
5. return \(\perp\)
6. \(v_2 \leftarrow 1\)
7. while \(v_3 \neq 0\) do
8. Parse \(\mathcal{T}[\tau_1] \oplus \tau_2\) as \(y\)
9. Parse \(A[y]\) as \((z_1, z_2)\)
10. Parse \(z_1 \oplus H(\tau_3, z_2)\) as \((v_1, v_2)\)
11. \(R_{\omega,\lambda(u)} \leftarrow R_{\omega,\lambda(u)} \cup \{v_1\}\)
12. return \(R_{\omega,\lambda(u)}\)

The KeyGen algorithm initialises the system and generates the keys \(K_O, K_S\), along with the public parameters, PP. The key \(K_O\) includes the secret key for the BE scheme and the sets of [:] keys for each pseudo-random function:\(F, G\) and \(P\) and the key for the pseudo-random permutation \(\phi\) (referred to as the data owner’s state, \(s_{3})\). The server is enrolled as a user and its secret key is also generated (although it does not receive the necessary keys to form search queries). PP includes the information flow policy \(P\), the authorized user group \(G\), the server state \(s_{3}\) (which is an encryption of the owner state generated using BE) and the public parameters for BE, PPBE.

The BuildIndex algorithm initialises a set \(f\) which consists of all nodes in the array \(A\). BuildIndex considers each keyword contained in the dataset in turn. For each keyword \(\omega\), the function \(\gamma\) generates \(L_{\omega}, N_{\omega}\) and \(N_{\omega}\). The free list is then updated according to which nodes have been chosen by \(\gamma\). The nodes in the array that form the linked lists consist of the identifier from \(L_{\omega}\), of a data item containing \(\omega\), the address in the array of the next node in the linked list, the key used to decrypt the following node in the linked list and a random bit string \(r_I \in \{0, 1\}^k\). The identifier, address of the next node and the key used to decrypt the following node in the linked list are XORed with the output of a PRF \(H\) in order to encrypt this information. For the first node in the linked list he input of \(H\) is the decryption key for the current node (which corresponds to an access level and keyword and forms part of the search query) along with \(r_I\), hence the information stored in the node can only be decrypted by the server if the server has a search query generated by a user who is authorized to view the data item whose identifier is stored at that node. The decryption key for all subsequent nodes is contained in the previous node of the linked list. BuildIndex then proceeds to create the lookup table \(T\). Unlike prior schemes [9], each user may have a different access level and thus the starting points for search results within the linked lists may vary; a search query made by a user with a higher access level should traverse more of the list than that of a user with lower access rights (the user is authorised to search more data items). Table \(T\) has an entry for each access level/keyword pair containing the address of a node in \(A\), which is the node in the linked list \(L_{\omega}\) from which the node with a specified access level is authorised to decrypt. If an access level is not authorised to view any part of the linked list then the value in \(T\) is set to \(\perp\). Finally the index \(\mathcal{I}_P = (A, T)\) is returned.

The AddUser algorithm grants a user \(u\) the ability to search the index at a specific access level. The user is added to the set \(G\) of authorized users and a BE key, \(k_u\), is derived for the new user. The new user is given \(k_u\) and the secret keys associated with their access level \(k_{\lambda(u),1}, k_{\lambda(u),2}\) and \(k_{\lambda(u),3}\) and PP is updated.

The RevokeUser algorithm revokes a user’s search privileges. A new value for \(s_{l_0}\) is selected and the user is removed from \(G\). This value is encrypted using BE to form the new server state \(s_{3}\). The updated versions of \(K_O\) and PP are output.

The Query algorithm generates a search query for a user \(u\) to search for a keyword \(w\). The server first attempts to decrypt the current server state \(s_{3}\) using their secret key \(k_u\); we denote the output of the decryption by \(s_{l_0}\). Note that if \(u\) is not authorised then decryption will return \(\perp\), if this is the case Query outputs \(\perp\). The query itself comprises three parts. The first is the output of the PRF \(F\) applied to the keyword \(\omega\), keyed with the secret key for \(F\) associated with the user’s access level \(k_{\lambda(u),1}\). This part of the query is used to locate the relevant entry in \(T\). The second part is the output of the PRF \(G\) applied to the keyword \(\omega\) and is used to mask the entry in \(T\) in order to locate the user’s relevant the starting position in the linked list corresponding to \(\omega\) in \(A\). The third part is the output of the PRF \(P\) applied to the keyword \(\omega\), which is used to decrypt the first relevant node in \(A\) according to the user’s access level. The PRF \(\phi\) is applied to the search query, using \(s_{l_0}\) as the key.

The Search algorithm finds data item identifiers associated with the searched keyword from the subset of data item identifiers the user is authorized to search. The server decrypts \(s_{3}\) and applies the inverse of the PRF \(\phi\) to the query it received; it parses the result as \((\tau_1, \tau_2, \tau_3)\). The server then looks up entry \(\mathcal{T}[\tau_1]\) and if that entry is not equal to \(\perp\), the server XORs the value with \(\tau_2\) and parses the resulting value as \(y\). The server looks up the node at \(A[y]\), parses the entry as \((z_1, z_2)\), and decrypts it by XORing \(z_1\) with the output of \(H\) (which takes as input \(\tau_3\) along with \(z_2\)).

The server is able to sequentially decrypt the rest of the list stored in \(A\) until they reach a node where the address stored in that node for the next item in the linked list is 0.

3.4 Security

In MLSSE search queries for the same keyword that are produced by users with different access levels are indistinguishable from one another. That is, a search query for a keyword \(\omega\) from a user \(u_i\) with access level \(\lambda(u_i)\) is indistinguishable from a search query for \(\omega\) from a user \(u_j\) with access level \(\lambda(u_j)\) for \(\lambda(u_i) \neq \lambda(u_j)\). This means that from the queries alone an adversary is unable to deduce how many times a certain keyword has been searched for overall, it can only deduce how many times the same keyword has been searched for within each access level. This information leakage is less than that of standard single or multi user SSE.
schemes such as [6, 7, 9, 10, 12, 13].

In terms of access pattern we also reduce the amount of information leakage compared with standard single user or multi-user SSE schemes. In particular we do not reveal whether a data item contains the keyword \( \omega \) associated with a search query unless the access level of the data item is less than or equal to that of the user \( u_i \) that generated the search query, meaning that an adversary cannot see a full set of search results.

However when a search query is paired with the search results it generates (the access pattern, Definition 5) then an adversary may be able to correlate which search queries are for the same keyword by looking at the intersections of the search results. For example if one set of search results is a subset of another set of search results then this may imply that the two search queries used to generate these results are for the same keyword. An adversary may eventually be able to build up a complete set of search results for a particular keyword, which is equivalent to the leakage produced by a search query in a single user SSE scheme. The server does not know, however, how many access levels there are altogether so a server would need to receive all possible search queries before it can ascertain whether or not a set of search results for a particular keyword is complete or not.

The hierarchal relationships between the data item identifiers i.e. which identifiers represent data items at higher access level than others could also be leaked in the same way. If an adversary has ascertained that two sets of search results \( \mathcal{R}_{\omega,a_i} \subset \mathcal{R}_{\omega,a_j} \) represent searches for the same keyword \( \omega \), then an adversary will be able to conclude that identifiers in the set \( \mathcal{R}_{\omega,a_j} \setminus \mathcal{R}_{\omega,a_i} \) are at a higher access level than those in \( \mathcal{R}_{\omega,a_i} \). We note that unless the search results are padded in some way this leakage is inevitable. Padding search results is not standard in SSE schemes as it requires post-processing of the search results by the user hence we do not pad the search results in our system model in order to maintain an efficient scheme.

From this we can see that initially our scheme leaks less information about the search pattern and access pattern than a single user SSE scheme, however over time as more queries are generated the information leakage tends to that of a single user SSE scheme. The information leakage relating to a keyword \( \omega \) i.e. the access patterns for search queries corresponding to \( \omega \) only reaches that of a single user SSE scheme once a search query has been generated at each possible access level, our leakage remains lower up until this point.

A search query for a keyword and access level pair is created deterministically we can think of the search query as a codeword for the combination of that keyword and access level. The index usually reveals these codewords as a search is carried out by matching search queries to relevant codewords in the index. A codeword for keyword \( \omega \) at access level \( a \) is denoted \( \text{id}(\omega, a) \).

We specify the leakage functions to precisely capture the leakage in MLSSE, where \( \Omega \) is a set of queries from users in the system that have been evaluated on the encrypted index by the server:

1. \( \mathcal{L}_{\text{Setup}}(\mathcal{T}_d) = \{(|\mathcal{A}|, |\mathcal{T}|, \text{id}(\omega, a))_{\omega \in \Delta, a \in |\mathcal{L}|} \}
2. \( \mathcal{L}_{\text{Query}}(\mathcal{T}_d, \Omega) = (AP(\mathcal{T}_d, \Omega), SP(\mathcal{T}_d, \Omega), \text{id}(\omega, a))_{\mathcal{T}_d, \omega \in \Omega} \)

**Theorem 1.** Given an IND-CPA secure broadcast encryption scheme \( \mathcal{BE} \), a pseudo-random permutation \( \phi \), and pseudorandom functions \( F, G, P, H \). Let MLSSE be the searchable symmetric encryption scheme with multi-level access defined in Algorithms 1-6. Then MLSSE is \((\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Query}})\)-secure in the sense of multi-level access and revocation.

We provide the intuitions of our security proofs here and refer the reader to the full online version of the paper for the full security proofs [1].

**Multi-level access:** To show multi-level access we reduce the security to that of the IND-CPA security of a symmetric encryption scheme which encrypts plaintexts with XORing them with the output of a PRF. We assume the possibility of an adversary \( \mathcal{A} \) that is able to break the multi-level security of our scheme then we construct a second adversary \( \mathcal{A}' \) that is able to use \( \mathcal{A} \) as a subroutine in order to break the IND-CPA security of the symmetric encryption scheme with non-negligible probability.

**Revocation:** In this proof we show that if we assume an adversary \( \mathcal{A} \) with non-negligible advantage \( \delta \) in Game 2 then \( \mathcal{A} \) can be used as a subroutine by an adversary \( \mathcal{A}_{\mathcal{BE}} \) to break the security of an IND-CPA secure broadcast encryption scheme \( \mathcal{BE} \).

3.5 Achieving dynamicity

We can extend MLSSE to support multi-level access on a dynamic data set by adding two new data structures to the index: a deletion table \( (\mathcal{T}_d) \) and a deletion array \( (\mathcal{A}_d) \). There are also four additional algorithms: AddToken, Add, DeleteToken, Delete. Array \( \mathcal{A}_d \) stores a list of nodes for each data item which point to nodes in \( \mathcal{A} \) that would need to be removed if the corresponding data item was deleted. This means that every node in \( \mathcal{A} \) will have a corresponding node in \( \mathcal{A}_d \), which is called its dual node. \( \mathcal{T}_d \) is a table with an entry for each data item which points to the start of the corresponding linked list in \( \mathcal{A}_d \). Given a valid delete token for that data item. In addition to these two new structures the index consists of a search array \( \mathcal{A}_s \) and a search table \( \mathcal{T}_s \) (as in the original construction) and a free list that keeps track of all the unused space in \( \mathcal{A}_s \).

In the dynamic scheme searching for a keyword is done similarly to the static construction in Section 3.3 and follows the concept of linked lists presented by [9].

To add a data item to the index, changes need to be made to \( \mathcal{T}_d, \mathcal{A}_s \) and \( \mathcal{A}_d \). The data owner creates an add token using AddToken and sends this to the server. The server then determines the free space available in \( \mathcal{A}_s \) using the free list and adds the relevant information to the free nodes and updates the free list. When adding a new data item the relevant nodes cannot be added to the end of each linked list; instead we have to insert in the appropriate place in the linked list according to the access level of the new data item. Information in the add token will allow the server to locate the correct point at which to insert the nodes in each linked list, so instead of the entry in \( \mathcal{T}_s \) just pointing to the end node of each linked list this is altered so that it points to the correct node in the linked list according to the access level of the new data item. The respective predecessor of each new node is modified to point to the new node instead of its previous ancestor.

In order to remove a data item, a deletion token is created which allows the server to locate and delete the correct entries in \( \mathcal{T}_d \). This, in turn, allows the server to locate and delete the correct entries in \( \mathcal{A}_s \). Some nodes will need to be updated in \( \mathcal{A}_s \) (as some of the linked lists will have nodes which point to nodes that have been deleted) and this is
done using homomorphic encryption.

3.6 Efficiency

In this section we discuss the efficiency of our multi-user, multi-level construction compared with the single-user construction of [13]. As our scheme is static and the scheme of [13] is dynamic, we ignore the structures and algorithms in [13] that apply to the dynamicity, such as the deletion table, the deletion array and algorithms AddToken, Add, DeleteToken, Delete.

The index is composed of a look-up table and a search array. No changes are made to the search array that effect the time needed to generate it or the search time, but the look-up table needs to be augmented by a factor of $|L|$: this will require more space on the server but does not effect the search time. The size of our index is $O(\omega,a)$ whereas the size of the index in the single user scheme is $O(\omega,a)$.

There search time of our scheme is $O(|D_{w,a}|)$ where $D_{w,a}$ is the set of data item identifiers satisfying the search query $T_{w,a}$. This is equivalent to the search time of [13], however in our scheme the size of $D_{w,a}$ is likely to be smaller, depending on the access level of the user who generated the search query.

The amount of computation required to generate the search queries as well as the size of the search queries is the same in both schemes, they are both constructed by evaluating three PRFs.

We note that in terms of efficiency our construction is very similar to that of [13]. This is also true for the dynamic version of our construction.

4. CONCLUSION

We have defined a new system, security models and a construction for symmetric solutions to searching on encrypted data in the multi-level setting. Users may search for keywords within a set of encrypted data items, restricting the search to data items they are authorised to view only. Future work will focus on increasing the range of query types beyond that of single keyword equality search and to expand the access control policies to arbitrary information flow policies.

5. REFERENCES


