# Multi-level Access in Searchable Symmetric Encryption

James Alderman*, Keith M. Martin,
and Sarah Louise Renwick†

Information Security Group, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, United Kingdom
{James.Alderman, Keith.Martin}@.rhul.ac.uk
{SarahLouise.Renwick.2012}@live.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.

## 1 Introduction

Searchable encryption (SE) enables a user to search over encrypted data that has been outsourced to a remote server. In some schemes [3, 4, 8, 17–19], 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 [15]. 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 [1].

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.

* Supported by the European Commission under project H2020-644024 "CLARUS" and acknowledges support from BAE Systems Advanced Technology Centre.
† Supported by Thales UK and EPSRC under a CASE Award.

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 $\mathbb{L}$, which forms a totally ordered set $(\mathbb{L}, \leqslant)$ 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 $\lambda(u) \geqslant \lambda(d)$.

Public-key encryption (PKE), especially functional encryption, has previously been used to achieve MLA in SE [2, 10, 14, 20]. 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. [13] 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 our instantiation based on the constructions of [8, 12], and then show, in Section 3.5, how to extend our construction to support a dynamic data set using techniques from [12]. Section 3.6

discusses the efficiency of our scheme. The full security proofs of our constructions are omitted but will be available in the full online version of our paper.

## 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 [7].

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

In this paper, we will focus on the case where $(\mathbb{L}, \leq)$ 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 $2^X$.

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

- $K_O \xleftarrow{\$} \mathsf{MSSE.KeyGen}(1^\kappa)$*: A probabilistic algorithm run by the data owner that takes a security parameter* $\kappa \in \mathbb{N}$ *and outputs a secret key* $K_O$*.*
- $(\mathcal{I}_\mathcal{D}, st_O, st_S) \xleftarrow{\$} \mathsf{MSSE.BuildIndex}(\mathcal{D}, \mathcal{G}, K_O)$*: A probabilistic algorithm run by the data owner that takes a set of data items* $\mathcal{D}$*, a set of authorized users* $\mathcal{G}$ *and the secret key* $K_O$*. It outputs an index* $\mathcal{I}_\mathcal{D}$*, and server and owner states* $st_S$ *and* $st_O$*.*
- $K_u \xleftarrow{\$} \mathsf{MSSE.AddUser}(u, K_O, st_O)$ *: 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* $K_u$*.*
- $t_\omega \leftarrow \mathsf{MSSE.Query}(\omega, K_u)$[2]*: A deterministic algorithm run by a user that takes a keyword* $\omega$ *and the user's secret key, and outputs a search token.*
- $R_\omega \leftarrow \mathsf{MSSE.Search}(t_\omega, \mathcal{I}_\mathcal{D}, st_S)$*: 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* $R_\omega$ *of identifiers of data items containing* $\omega$*.*
- $(st_O, st_S) \xleftarrow{\$} \mathsf{MSSE.Revoke}(u, K_O, st_O)$*: 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 $\mathcal{D}$ and keyword $\omega \in \varDelta$ (where $\varDelta$ is a dictionary of possible keywords), let us denote by $\mathcal{D}_\omega$ the expected results of searching for $\omega$ in $\mathcal{D}$ (in the plain); informally we say that an MSSE scheme is correct if it also produces the output $\mathcal{D}_\omega$. More formally, a MSSE scheme $\mathsf{MSSE}$ is correct if for all $k \in \mathbb{N}$, for all $K_O$ output

[1] A poset is a set of labels $L$ and a binary order relation $\leqslant$ on $L$ such that for all $x, y$ and $z \in L$, $x \leqslant x$ (reflexivity), if $x \leqslant y$ and $y \leqslant x$ then $x = y$ (antisymmetry), and if $x \leqslant y$ and $y \leqslant z$ then $x \leqslant z$ (transitivity). If $x \leqslant y$ then we may write $y \geqslant x$.

[2] This algorithm is sometimes referred to as $\mathsf{MSSE.Trapdoor}$ in the literature, however to maintain consistent notation throughout this paper we refer to it as $\mathsf{MSSE.Query}$

by $\mathsf{MSSE.KeyGen}(1^k)$, for all $\mathcal{D} \in 2^{\Delta}$, for all $\mathcal{G} \in 2^{\mathcal{U}}$, for all $(\mathcal{I}_{\mathcal{D}}, st_O, st_S)$ output by $\mathsf{MSSE.BuildIndex}(K_O, \mathcal{G}, \mathcal{D})$, for all $\omega$ in $\Delta$: $\mathsf{Search}(\mathsf{MSSE.Query}(K_u, \omega), \mathcal{I}_{\mathcal{D}}, st_S) = \mathcal{D}_{\omega}$.

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

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

$\mathsf{BE}$ *is correct if $\forall k \in \mathbb{N}$, for all $PP$ and $K_{\mathsf{BE}}$ output by* $\mathsf{BE.KeyGen}(1^k, m)$*, for all $M$ in the plaintext space, all sets of users $\mathcal{G} \in \mathcal{U}$, every $K_U$ output by* $\mathsf{BE.Add}(u, K_{\mathsf{BE}})$ *and all $C$ output by* $\mathsf{BE.Enc}(M, \mathcal{G})$ *where $u \in \mathcal{G}$ we have: $M \leftarrow$* $\mathsf{BE.Dec}(C, K_u)$.

## 3 Multi-level Access in Searchable Symmetric Encrytion

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 search all data items at their own access level as well as all data items that are classified at lower access levels.

### 3.1 System Model

Consider a *data owner* $O$, a *server* $S$, and a set of $m$ data *users* $\mathcal{U}=\{u_1, ..., u_m\}$. The data owner possesses a set of data items $\mathcal{D}=\{d_1, ..., d_n\}$ which they wish to encrypt and outsource to $S$ whilst authorising other users to search over some data items within $\mathcal{D}$. Each data item $d_i \in \mathcal{D}$ is associated with an identifier $id_{d_i}$.

To enable searching over the encrypted data, $O$ must upload some encrypted metadata to the server. It first defines a dictionary of keywords, denoted $\Delta = \{\omega_1, ..., \omega_{|\Delta|}\}$, and assigns a set $\delta_{d_i} \subseteq \Delta$ of keywords to each data item $d_i \in \mathcal{D}$. We refer to the set of keywords for all data items as $\delta_{\mathcal{D}} = (\delta_{d_1}, ..., \delta_{d_n})$. The data owner then produces an encrypted *index* $\mathcal{I}_{\mathcal{D}}$ based on $\delta_{\mathcal{D}}$, over which searches will be performed.

$O$ also defines an information flow policy $\mathcal{P}$ with a labelling function $\lambda$ mapping each user $u_i \in \mathcal{U}$ and data item $d_j \in \mathcal{D}$ to an access level, denoted $\lambda(u_i)$ and $\lambda(d_j)$ respectively, in the totally ordered set $\mathbb{L} = \{a_1, ..., a_l\}$. Access control in our model is enforced at data item level — users are restricted in the data items that they may search, not the keywords they may search for [13]. A user with clearance $\lambda(u_i)$ is authorised to search a data item with classification $\lambda(d_j)$ if and only if $\lambda(d_j) \leq \lambda(u_i)$. To search for a keyword $\omega \in \Delta$, a user $u_i$ (with clearance $\lambda(u_i)$) generates a search query $T_{\omega,\lambda(u_i)}$. Let $\mathcal{D}_{\omega}$ be the set of identifiers of all data items assigned the keyword $\omega$, and denote by $\mathcal{D}_{\omega,\lambda(u_i)} \subseteq \mathcal{D}_{\omega}$ the search results that user $u_i$ is authorised to view; in other words, the set of identifiers of all data items $id_{d_j}$ assigned $\omega$ where $\lambda(d_j) \leq \lambda(u_i)$.

To add and revoke users, we use *broadcast encryption* (BE) (Definition 3) as per [8]; a user may only produce a valid search query if they are authorized in the BE scheme.

To ease notation, we define the tuple ${d_i}^{aug} = (d_i, id_i, \delta_{d_i}, \lambda(d_i))$ to completely describe a data item $d_i \in \mathcal{D}$ (being the data itself, the identifier, the associated keywords and the security classification). We denote the information regarding all data items by $\mathcal{D}^{aug} = \{{d_1}^{aug}, ..., {d_n}^{aug}\}$.

We present a *structure only* MLSSE system — we only consider the data structure (index) and do not encrypt the data items themselves; data items may be encrypted separately and retrieved based on the search results, which comprise a set of data item identifiers that fulfil the query. We 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:*

- $(K_O, k_S, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \mathcal{P}, S)$*: A probabilistic algorithm run by the data owner* $O$ *that takes the security parameter* $\kappa$*, policy* $\mathcal{P}$ *and the server identity* $S$*, and outputs* $O$*'s secret key* $K_O$*, a server key* $K_S$ *and public parameters* $PP$*.*
- $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}^{aug}, K_O, PP)$*: A probabilistic algorithm run by* $O$*. It takes the description of the data set* $\mathcal{D}^{aug}$ *and* $O$*'s secret key, and outputs the index* $\mathcal{I}_\mathcal{D}$*.*
- $K_u \xleftarrow{\$} \mathsf{AddUser}(u, \lambda(u), K_O, PP)$*: A probabilistic algorithm run by* $O$ *to enrol a new user into the system. It takes the new user's identity and access level, and* $O$*'s key, and outputs a secret key for the new user.*
- $T_{\omega,\lambda(u)} \leftarrow \mathsf{Query}(\omega, K_u)$*: A deterministic algorithm run by a user with clearance* $\lambda(u)$ *to generate a search query. It takes as input a keyword* $\omega \in \Delta$ *and the user's secret key and outputs a query token* $T_{\omega,\lambda(u)}$*.*
- $\mathcal{R}_{\omega,\lambda(u)} \leftarrow \mathsf{Search}(T_{\omega,\lambda(u)}, \mathcal{I}_\mathcal{D}, k_S)$*: A deterministic algorithm run by* $S$ *to search the index for data items containing a keyword* $\omega$*. It takes a search query and the index, and returns the search results* $\mathcal{R}_{\omega,\lambda(u_i)}$*, comprising either a set of identifiers of data items* $d_j$ $\mathcal{D}_{\omega,\lambda(u)}$ *containing* $\omega$ *such that* $\lambda(d_j) \leq \lambda(u)$ *(where* $\lambda(u_i)$ *is the access level of the user that submitted the search query), or a failure symbol* $\perp$*.*
- $(K_O) \xleftarrow{\$} \mathsf{RevokeUser}(u, K_O, PP)$*: 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* $\mathsf{KeyGen}(1^k, \mathcal{P})$*, for all* $\mathcal{D}^{aug}$*, for all* $\mathcal{I}_\mathcal{D}$ *output by* $\mathsf{Buildindex}(\mathcal{D}^{aug}, K_O)$*, for all* $\omega \in \Delta$*, for all* $u \in \mathcal{U}$*, for all* $K_u$ *output by* $\mathsf{AddUser}(K_O, u, \lambda(u), PP)$*, Search*$(\mathcal{I}_\mathcal{D}, T_{\omega,\lambda(u)}) = \mathcal{D}_{\omega,\lambda(u)}$*.*

## 3.2 Security model

A secure MLSSE scheme would, ideally, reveal no information regarding the data set $\mathcal{D}$ 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 $\{\mathcal{R}_{\omega_1,\lambda(u)}, ..., \mathcal{R}_{\omega_p,\lambda(u)}\}$ for a set of queries $\{T_{\omega_1,\lambda(u)}, ..., T_{\omega_p,\lambda(u)}\}$ 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 data item identifier of a data item 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 [5, 6, 8, 9, 11, 12], 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 $\Omega = \{T_{\omega_1,\lambda(u_1)}, ..., T_{\omega_q,\lambda(u_q)}\}$ where for $1 \leq i, j \leq q$: $\omega_i$ and $\omega_j$ or $\lambda(u_i)$ and $\lambda(u_j)$ are not necessarily distinct for $i \neq j$, the* access pattern *is $AP(\mathcal{I}_\mathcal{D}, \Omega) = \{\mathcal{R}_{\omega_1,\lambda(u_1)}, ..., \mathcal{R}_{\omega_q,\lambda(u_q)}\}$.*

**Definition 6.** *For a sequence of $q$ search queries $\Omega = \{T_{\omega_1,\lambda(u_1)}, ..., T_{\omega_q,\lambda(u_q)}\}$ where for $1 \leq i, j \leq q$: $\omega_i$ and $\omega_j$ or $\lambda(u_i)$ and $\lambda(u_j)$ are not necessarily distinct for $i \neq j$, the* search pattern *is defined as a $q \times q$ symmetric binary matrix $SP(\mathcal{I}_\mathcal{D}, \Omega)$ such that for $1 \leq i, j \leq q$: $SP(\mathcal{I}_\mathcal{D}, \Omega)_{i,j} = 1 \iff T_{\omega_i,\lambda(u_i)} = T_{\omega_j,\lambda(u_j)}$. Intuitively, the search pattern reveals when the $i$th and $j$th 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 $\mathcal{I}_\mathcal{D}$ we define the* setup leakage *$\mathcal{L}_{Setup}(\mathcal{I}_\mathcal{D})$ to be all the information that is leaked by the index $\mathcal{I}_\mathcal{D}$.*

**Definition 8.** *For an index $\mathcal{I}_\mathcal{D}$ and set of $q$ search queries $\Omega = (T_{\omega_1,\lambda(u)}, ..., T_{\omega_q,\lambda(u)})$ we define the* query leakage *$\mathcal{L}_{Query}(\mathcal{I}_\mathcal{D}, \Omega)$ to be all the information leaked by evaluating the queries in $\Omega$ on the index $\mathcal{I}_\mathcal{D}$,*

We now formalise the notions of security we require in MLSSE. We use cryptographic games to formalize our notions of security. For each game, a challenger $\mathcal{C}$ instantiates a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ whose inputs are chosen to reflect the information available to a realistic adversary.

**Multi-level Access** Our first security notion, in Figure 1.1, 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 $\lambda(u_i)^\star$* on $\mathcal{I}_\mathcal{D}$ to be $\mathcal{L}_{Query}^{\lambda(u_i)^\star}(\mathcal{I}_\mathcal{D}, \{T_{\omega_i,\lambda(u_i)^\star}\}_{\omega_i \in \Delta})$ — this is the leakage resulting from every possible keyword search with the maximal security level available.

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, $\lambda^{max}$ is the maximal security label of any corrupted user, and $\mathsf{chall}$ 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 security label 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 security label 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 security label 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 security label 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 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 query can distinguish the data sets. Note that Van Rompay et al. [16] 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 above 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, $\mathcal{P}$ is an information flow policy, and $\mathcal{A}$ a PPT adversary. The advantage of $\mathcal{A}$ is:*

$$Adv_{\mathcal{A}}^{MLA}(\mathcal{ML}, 1^{\kappa}, \mathcal{P}) = |\Pr[\mathbf{Exp}_{\mathcal{A}}^{MLA}[\mathcal{ML}, 1^{\kappa}, \mathcal{P}] = 1] - \frac{1}{2}|.$$

*We say that $\mathcal{ML}$ is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against adaptive chosen keyword attacks in the sense of Game 1.1 if for all $\mathcal{A}$, all $k \in \mathbb{N}$ and all $\mathcal{P}$, $Adv_{\mathcal{A}}^{MLA}(\mathcal{ML}, 1^{\kappa}, \mathcal{P}) \leq \mathrm{negl}(k)$ for a negligible function* negl.

**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 1.2. The adversary is given the public parameters and selects a data set (along with associated

```
$\mathbf{Exp}_{\mathcal{A}}^{MLA}[\mathcal{ML}, 1^{\kappa}, S, \mathcal{P}]$:
$L = \emptyset, \lambda^{max} = \perp$
$\mathsf{chall} = \mathbf{false}$
$(K_O, k_S, PP) \leftarrow\$ \mathsf{KeyGen}(1^k, S, \mathcal{P})$
$(\mathcal{D}_0^{aug}, \mathcal{D}_1^{aug}, st) \leftarrow\$ \mathcal{A}^{\mathcal{O}}(1^k, \mathcal{P}, k_S, PP)$
if $\tau_{\lambda^{max}}^{max}(\mathcal{D}_0^{aug}) \neq \tau_{\lambda^{max}}^{max}(\mathcal{D}_1^{aug})$
  return $\perp$
$\mathsf{chall} = \mathbf{true}$
$b \leftarrow\$ \{0,1\}$
$\mathcal{I}_{\mathcal{D}} \leftarrow\$ \mathsf{BuildIndex}(\mathcal{D}_b^{aug}, K_O, PP)$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{O}}(\mathcal{I}_{\mathcal{D}}, st)$
if $b' = b$ return $1$
else return $0$
```

```
Oracle $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
if $\mathsf{chall} = \mathbf{false}$
  $L \leftarrow L \cup \{u\}$
  if $\lambda(u) > \lambda^{max}$
    $\lambda^{max} \leftarrow \lambda(u)$
  return $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
else if $\tau_{\lambda(u)}^{max}(\mathcal{D}_0^{aug}) = \tau_{\lambda(u)}^{max}(\mathcal{D}_1^{aug})$
  $L \leftarrow L \cup \{u\}$
  if $\lambda(u) > \lambda^{max}$
    $\lambda^{max} \leftarrow \lambda(u)$
  return $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
else return $\perp$
```

```
Oracle $\mathsf{RevokeUser}(u, K_O, PP)$
if $u \notin L$return $\perp$
$L = L \setminus \{u\}$
if $\lambda(u) = \lambda^{max}$
  for $u' \in L$
    if $\lambda(u') > \lambda^{max}$
      $\lambda^{max} \leftarrow \lambda(u')$
$(K_O, PP) \leftarrow\$ \mathsf{RevokeUser}(u, K_O, PP)$
return $PP$
```

Game 1.1: The Multi-level Access game

access levels, keywords and identifiers). The challenger then creates the index. The adversary is given access to a set of oracles that perform the $\mathsf{AddUser}(\cdot, \lambda(\cdot), K_O, PP)$, $\mathsf{Search}(\cdot, \cdot, \mathcal{I}_{\mathcal{D}}, K_S)$ and $\mathsf{RevokeUser}(u_i, K_O, K_S, PP)$ 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 $\mathsf{AddUser}$ oracle but were not subsequently queried to the $\mathsf{RevokeUser}$ oracle (i.e. all users for which the adversary holds a valid user key). The adversary must then produce a query token $T$ which, when used as input to the $\mathsf{Search}$ algorithm, does not produce $\perp$ 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, $\mathcal{P}$ is an information flow policy and $\mathcal{A}$ a PPT adversary. We define the advantage of $\mathcal{A}$ in Game 1.2 as:*

$$Adv_{\mathcal{A}}^{Revoke}(\mathcal{ML}, 1^{\kappa}, \mathcal{P}) = |\mathbb{P}[\mathbf{Exp}_{\mathcal{A}}^{Revoke}[\mathcal{ML}, 1^{\kappa}, \mathcal{P}] = 1] - \frac{1}{2}|.$$

*We say that $\mathcal{ML}$ achieves revocation if for all $\mathcal{A}$, all $k \in \mathbb{N}$ and all $\mathcal{P}$,*

$$Adv_{\mathcal{A}}^{Revoke}(\mathcal{ML}, 1^{\kappa}, \mathcal{P}) \leq \mathrm{negl}(k).$$

## 3.3 Construction

Our construction is an adaptation of the scheme of Kamara et al. [12], which is an adaptation of the influential inverted index scheme $\mathsf{SSE\text{-}1}$ by Curtmola et al. [8]. Our notion of adaptive security is based on that of IND-CKA2 presented in [8].

```
$\mathbf{Exp}_{\mathcal{A}}^{Revoke}\,[\mathcal{ML}, 1^{\kappa}, S, \mathcal{P}]$:
$(K_O, k_S, PP) \leftarrow_{\$} \mathsf{KeyGen}(1^k, S, \mathcal{P})$
$(\mathcal{D}^{aug}, st) \leftarrow \mathcal{A}(1^k, \mathcal{P}, PP)$
$\mathcal{I}_{\mathcal{D}} \leftarrow_{\$} \mathsf{BuildIndex}(\mathcal{D}^{aug}, K_O, PP)$
$st \leftarrow_{\$} \mathcal{A}^{\mathcal{O}}(st)$
for all non-revoked $u$ queried to $\mathcal{O}(\mathsf{AddUser})$
    $(K_O, PP) \leftarrow_{\$} \mathsf{RevokeUser}(u, K_O, PP)$
$T \leftarrow_{\$} \mathcal{A}(PP, st)$
$\mathcal{R} \leftarrow \mathsf{Search}(T, \mathcal{I}_{\mathcal{D}}, k_S)$
if $\mathcal{R} \neq \perp$ return $1$
else return $0$
```

Game 1.2: The Revocation game

Informally, our MLSSE scheme uses an array $\mathbb{A}$ of linked lists, along with a look-up table $\mathbb{T}$ 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 $\omega_i$, we define a list $\mathsf{L}_{\omega_i}$ which contains the identifiers for all data items containing that keyword 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 $\mathsf{L}_{\omega_i}$ is encrypted and stored in $\mathbb{A}$ as a linked list. During the search phase the look-up table $\mathbb{T}$ 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 and 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 [12] is that, rather than pointing to the *beginning* of each linked list, the entry in $\mathbb{T}$ 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 $\mathsf{BE}$ be an IND-CPA secure broadcast encryption scheme. We define the following pseudorandom functions (PRFs):

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

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

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

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

and a pseudorandom permutation (PRP):

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

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

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 $\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}$ and $\mathsf{N}_{\omega_i}$ given the set of identifiers $\mathcal{D}^{aug}$. We refer to the $n^{th}$ item in a list $\mathsf{L}_{\omega_i}$ as $L_i[n]$. The list $\mathsf{L}_{\omega_i}$ contains identifiers of data items in $\mathcal{D}_{\omega_i}$ ordered from the identifiers with the highest to the lowest access levels, the list $\mathsf{N}_{\omega_i}$ contains $|\mathsf{L}_{\omega_i}|$ nodes chosen randomly from $\mathbb{A}$ and the list $\mathsf{X}_{\omega_i}$ contains the indices of the identifiers in $\mathsf{L}_{\omega_i}$ where each access level starts i.e. if we have an ordered list of identifiers $\mathsf{L}_{\omega_i} = (id_1, id_2, id_3, id_4, id_5)$ where:

$$a_1 = \lambda(id_1) = \lambda(id_2) = \lambda(id_3) < \lambda(id_4) = \lambda(id_5) = a_3.$$

We have that $\mathsf{X}_{\omega_i}[3] = 4$, which says that the list of nodes with access level at most $a_3$ starts at the fourth entry in $\mathsf{L}_{\omega_i}$. There is an entry per each access level in $\mathsf{X}_{\omega_i}$, even if two access levels have the same starting point in $\mathsf{L}_{\omega_i}$; from the example above we can see that $\mathsf{X}_{\omega_i}[2] = \mathsf{X}_{\omega_i}[3] = 4$. If an access level is not authorised to view any data items in $\mathcal{D}_{\omega_i}$ then the entry corresponding to that access level (as well as the entries corresponding to all access levels below it) in $\mathsf{X}_{\omega_i}$ is set to $\perp$. An identifier of a data item $d_i \in \mathcal{D}_{\omega_i}$ will inherit the access level label of the respective data item, i.e. $\lambda(id_{d_i}) = \lambda(d_i)$.

```
$(K_O, k_S, PP) \leftarrow\$ \mathsf{KeyGen}(1^k, S, \mathcal{P})$

for $i \in |\mathbb{L}|$
  $k_{a_i,1}, k_{a_i,2}, k_{a_i,3} \leftarrow\$ \{0,1\}^k$
$(\mathrm{PP}_{\mathsf{BE}}, k_{\mathsf{BE}}) \leftarrow\$ \mathsf{BE.KeyGen}(1^{\mathsf{k}}, |\mathcal{U}|)$
$st_O \leftarrow\$ \{0,1\}^k$
$\mathcal{G} \leftarrow \{S\}$
$st_S \leftarrow\$ \mathsf{BE.Enc}(st_O, \mathcal{G}, k_{\mathsf{BE}})$
return
  $K_O \leftarrow (\{k_{a_i,1}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,2}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,3}\}_{i\in[|\mathbb{L}|]}, k_{\mathsf{BE}}, st_O)$
  $PP = (\mathcal{P}, \mathcal{G}, st_S, PP_{\mathsf{BE}})$
  $k_S \leftarrow\$ \mathsf{BE.Add}(k_{\mathsf{BE}}, S)$

$\mathcal{I_D} \leftarrow\$ \mathsf{BuildIndex}(\mathcal{D}_{aug}, K_O, PP)$

$free \leftarrow \{addr(N_i)\}_{[i\in|\mathbb{A}|]}$
for $1 \le i \le |\mathcal{W}|$
  $(\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}, \mathsf{N}_{\omega_i}) \leftarrow \gamma(\mathcal{D}_{\omega_i})$
  $free \leftarrow free \setminus \mathsf{N}_{\omega_i}$
  for $1 \le j \le |\mathsf{N}_{\omega_i}| - 1$
    $r_j \leftarrow\$ \{0,1\}^k$
    $\mathbb{A}[\mathsf{N}_{\omega_i}[j]] \leftarrow \Big(\big(\mathsf{L}_{\omega_i}[j], \mathsf{N}_{\omega_i}[j+1], P_{k_{\lambda(\mathsf{L}_{\omega_i}[j+1]),3}}(\omega_i)\big) \oplus H\big(P_{k_{\lambda(\mathsf{L}_{\omega_i}[j]),3}}(\omega_i), r_j\big), r_j\Big)$
  $r_{|\mathsf{N}_{\omega_i}|} \leftarrow\$ \{0,1\}^k$
  $\mathbb{A}[\mathsf{N}_{\omega_i}[|\mathsf{N}_{\omega_i}|]] \leftarrow \Big(\big(\mathsf{L}_{\omega_i}[|\mathsf{N}_{\omega_i}|], 0, 0\big) \oplus H\big(P_{k_{\lambda(\mathsf{L}_{\omega_i}[|\mathsf{N}_{\omega_i}|]),3}}(\omega_i), r_|\mathsf{N}_{\omega_i}|\big), r_{|\mathsf{N}_{\omega_i}|}\Big)$
  for $1 \leqslant \ell \leqslant |\mathbb{L}|$
    if $\mathsf{X}_{\omega_i}[a_\ell] \neq \perp$
      $\mathbb{T}[F_{k_{a_\ell},1}(\omega_i)] \leftarrow \big(\mathsf{N}_{\omega_i}\big[\mathsf{X}_{\omega_i}[a_\ell]\big] \oplus G_{k_{a_\ell},2}(\omega_i)\big)$
    else
      $\mathbb{T}[F_{k_{a_\ell},1}(\omega_i)] \leftarrow \perp$
return
  $\mathcal{I_D} \leftarrow (\mathbb{A}, \mathbb{T})$

$(K_u, PP) \leftarrow\$ \mathsf{AddUser}(u, \lambda(u), K_O, PP)$

$\mathcal{G} \leftarrow \mathcal{G} \cup \{u\}$
$k_u \leftarrow\$ \mathsf{BE.Add}(k_{\mathsf{BE}}, u)$
$st_S \leftarrow\$ \mathsf{BE.Enc}(st_O, \mathcal{G}, k_{BE})$
return
  $PP \leftarrow (\mathcal{P}, \mathcal{G}, st_S, PP_{\mathsf{BE}})$
  $K_u \leftarrow (k_u, k_{\lambda(u),1}, k_{\lambda(u),2}, k_{\lambda(u),3})$
```

```
$(K_O, PP) \leftarrow\$ \mathsf{RevokeUser}(u, K_O, PP)$

$st_O \leftarrow\$ \{0,1\}^k$
$\mathcal{G} \leftarrow \mathcal{G} \setminus \{u\}$
return
  $K_O \leftarrow (\{k_{a_i,1}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,2}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,3}\}_{i\in[|\mathbb{L}|]}, k_{\mathsf{BE}}, st_O)$
  $PP \leftarrow (\mathcal{P}, \mathcal{G}, st_S, PP_{\mathsf{BE}})$

$T_{\omega,\lambda(u)} \leftarrow \mathsf{Query}(\omega, K_u)$

$st'_O \leftarrow \mathsf{BE.Dec}(k_u, st_S)$
if $st'_O = \perp$
  return $\perp$
$t_{\omega,\lambda(u)} \leftarrow (F_{k_{\lambda(u),1}}(\omega), G_{k_{\lambda(u),2}}(\omega), P_{k_{\lambda(u),3}}(\omega))$
return
  $T_{\omega,\lambda(u)} \leftarrow \phi_{st'_O}(t_{\omega,\lambda(u)})$

$\mathcal{R}_{\omega,\lambda(u)} \leftarrow \mathsf{Search}(T_{\omega,\lambda(u)}, I_{\mathcal{D}}, k_S)$

$st'_O \leftarrow \mathsf{BE.Dec}(k_s, st_S)$
Parse $\phi^{-1}_{st'_O}(T_{\omega,\lambda(u)})$ as $(\tau_1, \tau_2, \tau_3)$
$\mathcal{R}_{\omega,\lambda(u)} \leftarrow \emptyset$
if $\mathbb{T}[\tau_1] = \perp$
  return $\perp$
$v_2 \leftarrow 1$
while $v_2 \neq 0$
  Parse $\mathbb{T}[\tau_1] \oplus \tau_2$ as $y$
  Parse $\mathbb{A}[y]$ as $(z_1, z_2)$
  Parse $z_1 \oplus H(\tau_3, z_2)$ as $(v_1, v_2)$
  $\mathcal{R}_{\omega,\lambda(u)} \leftarrow \mathcal{R}_{\omega,\lambda(u)} \cup \{v_1\}$
return $\mathcal{R}_{\omega,\lambda(u)}$
```

Fig. 1: MLSSE construction

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 and the sets of $|\mathbb{L}|$ 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, $st_O$). The server is enrolled as 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 $\mathcal{P}$, the authorized user group $\mathcal{G}$, the server state $st_S$ (which is an encryption of the owner state generated using $\mathsf{BE}$) and the public parameters for $\mathsf{BE}$, $PP_{\mathsf{BE}}$.

The $\mathsf{BuildIndex}$ algorithm initializes a set $free$ which consists of all nodes in the array $\mathbb{A}$. $\mathsf{BuildIndex}$ considers each keyword contained in the dataset in turn. For each keyword $\omega_i$, the function $\gamma$ generates $\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}$ and $\mathsf{N}_{\omega_i}$. The node at $addr_{\mathbb{A}}(|\mathsf{L}_{\omega_i}| + 1)$ is set to 0 to mark the end of each linked list in the array. The nodes in the array that form the linked lists consist of the identifier from $\mathsf{L}_{\omega_i}$ of a data item containing $\omega_i$, the address in the array of the next node in the linked list and a random bit string $r_i \in \{0, 1\}^k$. The identifier and address of the next node are XORed with the output of a random oracle $H$ in order to encrypt this information. The input of the random oracle is generated using the secret key corresponding to an access level and keyword (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. Once all the nodes for a particular list $\mathsf{L}_{\omega_i}$ have been created, the $free$ list is updated by removing all the nodes from it that have just been used to store $\mathsf{L}_{\omega_i}$. $\mathsf{BuildIndex}$ then proceeds to create the look-up table $\mathbb{T}$. Unlike prior schemes [8], each user may have a different level of access and thus the starting points for search results within the linked lists may vary; a query made by a user with a higher access level should traverse more of the list than that of a user with low access rights (the user is authorised to search more data items). Table $\mathbb{T}$ has an entry for each access level/keyword pair containing the address of a node in $\mathbb{A}$, which is the node in the linked list $\mathsf{L}_{\omega_i}$ from which the user 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 $\mathbb{T}$ is set to $\perp$. Finally the index $\mathcal{I}_{\mathcal{D}} = (\mathbb{A}, \mathbb{T})$ is returned.

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

The $\mathsf{RevokeUser}$ algorithm revokes a user's search privileges. The user is removed from $\mathcal{G}$ and a new value for $st_O$ is selected. This value is encrypted using $\mathsf{BE}$ to form the new server state $st_S$.

The $\mathsf{Query}$ algorithm generates a search query for a user $u_i$ to search for a keyword $w_i$. The user first attempts to decrypt the current server state $st_S$ using their secret key $k_{u_i}$; we denote the output of a successful decryption by $\theta$. Note that if $u_i$ is not authorised then decryption will return $\perp$. The query itself comprises three parts. The first is the output of the PRF $F$ applied to the keyword $\omega_i$, keyed with the secret key for $F$ associated with the user's access level $k_{\lambda(u_i),1}$. This part of the query is used to locate the relevant entry in $\mathbb{T}$. The second part is the output of the PRF $G$ applied to the keyword $\omega_i$ and is used to mask the entry in $\mathbb{T}$ in order to locate the relevant the starting position in the linked list corresponding to $\omega_i$ in $\mathbb{A}$. The third part is the output of the PRF $P$ applied to the keyword $\omega_i$, which is used to decrypt the relevant nodes in $\mathbb{A}$ according to the user's access level.

The $\mathsf{Search}$ 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 $st_S$ and applies the inverse of the PRP $\phi$ to the query it received; it parses the result as $(\tau_1, \tau_2, \tau_3)$. The server then looks up entry $\mathbb{T}[\tau_1]$ 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 $\mathbb{A}[y]$ and decrypts it using the output of the random oracle (which takes as input $\tau_3$ along with $r_i$).

The server is able to sequentially decrypt the rest of the list stored in $\mathbb{A}$ 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 our MLSSE scheme, the search queries are indistinguishable across access levels: 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. However this only holds in the absence of search results. An adversary who is able to see the search results generated by a particular search query (access pattern) will be able to correlate which search queries are for the same keyword and eventually build up a complete set of search results for a particular keyword. As 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, we use a different notation in our leakage functions to distinguish when we are referring to the presence of a search query or a codeword in the index; a codeword for keyword $\omega$ at access level $a_i$ is denoted $id(\omega, a_i)$.

This also leaks the hierarchical relationships between the data items i.e. which data items are encrypted at a higher access level than others. The adversary can do this by observing which sets are contained within other sets and which parts of the index are accessed. It is not always the case that access levels are considered private in access control systems so we do not view this as an issue.

In terms of access pattern, we reduce the amount of information leakage compared with standard single-user or multi-user SSE schemes such as [5, 6, 8, 9, 11, 12]. In particular we do not reveal whether a data item contains the keyword $\omega_i$ associated with a search query unless the access level of that data item is less than or equal to that of the user $u_i$ who generated the search query, meaning that an adversary cannot see a full set of search results. However if the server receives more search queries for the same keyword from users at different access levels then over time the server will be able to view a full set of search results. 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. Likewise, we do not pad the search results in our system model in order to maintain an efficient scheme.

We give the specific 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. $\mathcal{L}_{Setup}(\mathcal{I}_{\mathcal{D}}) = (|\mathbb{A}|, |\mathbb{T}|, [id(\omega, a_i)]_{\omega \in \Delta, i \in [|\mathbb{L}|]})$

2. $\mathcal{L}_{Query}(\mathcal{I}_{\mathcal{D}}, \Omega) = (AP(\mathcal{I}_{\mathcal{D}}, \Omega), SP(\mathcal{I}_{\mathcal{D}}, \Omega), [id(\omega, a)]_{\forall T_{\omega,a} \in \Omega}, \Omega)$

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

**Multi-level access**: To show multi-level access we reduce the security to that of the indistinguishability of the output of a PRF being indistinguishable from the output of a truly random function. We assume the possibility of a adversary $\mathcal{A}$ that is able to break the multi-level security of our scheme then we build a distinguisher $\mathcal{D}$ that is able to use $\mathcal{A}$ as a subroutine in order to distinguish between the output of a PRF and a truly random function 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 1.2 then $\mathcal{A}$ can be used as a subroutine by an adversary $\mathcal{A}_{\mathsf{BE}}$ to break the security of an IND-CPA secure broadcast encryption scheme BE.

## 3.5 Achieving dynamicity

The scheme of [12] achieves dynamicity by adding two new data structures to the index: a deletion table ($\mathbb{T}_d$) and a deletion array ($\mathbb{A}_d$). There are also four additional algorithms: AddToken, Add, DeleteToken, Delete. Array $\mathbb{A}_d$ stores a list of nodes for each data item which point to nodes in $\mathbb{A}$ that would need to be removed if the corresponding data item was deleted. This means that every node in $\mathbb{A}$ will have a corresponding node in $\mathbb{A}_d$, which is called its *dual* node. $\mathbb{T}_d$ is a table with an entry for each data item which points to the start of the corresponding linked list in $\mathbb{A}_d$, given a valid *delete token* for that data item. In addition to these two new structures the index consists of a search array $\mathbb{A}_s$ and a search table $\mathbb{T}_s$ (as in the original construction) and a *free list* that keeps track of all the unused space in $\mathbb{A}_s$.

In this scheme searching for a keyword is done similarly to our construction in Section 3.3 and follows the concept of linked lists presented by [8]. As it does not support multi-level access it is simpler than our construction and the look-up table only has one entry per keyword and directs the server to the starting node in each linked list during a search instead of having different starting points in the list for each access level.

To add a data item to the index, changes need to be made to $\mathbb{T}_d, \mathbb{A}_s$ and $\mathbb{A}_d$. The data owner creates an *add token* using AddToken and sends this to the server. The server then determines the free space available in $\mathbb{A}_s$ using the free list and adds the relevant information to the free nodes and updates the free list. These new nodes now need to be added to the appropriate linked lists in $\mathbb{A}_s$ according to the keywords the new data item contains; the new nodes will be the last nodes in the respective lists and the current last nodes in the linked list are modified to point towards the new node instead of terminating the search.

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

As our multi-level access construction is based on a modified version of this dynamic construction, we can easily apply our method for achieving multi-level access to this construction in order to achieve a dynamic SE scheme with multi-level access. We sketch the adaptations here and give a full construction in the the full online version of our paper.

Carrying out a search query is done analogously to our construction in Section 3.3.

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 $\mathbb{T}_s$ 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. The free list is then updated.

The removal of a data item is done analogously to that of the original construction.

We see that using the techniques from our construction only the AddToken and Add algorithms need to be changed in order to support a dynamic multi-level access SE scheme.

### 3.6 Efficiency

In this section we discuss the efficiency of our multi-user construction compared with the single-user construction of [12]. As our scheme is static and the scheme of [12] is dynamic, we ignore the structures and algorithms in [12] 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 $|\mathbb{L}|$; this will require more space on the server but does not effect the search time.

We note that in terms of efficiency our construction is very similar to that of [12]. 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.

## References

1. E. Bell and L. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, Mitre Corporation, 1976.

2. J. Benaloh, M. Chase, E. Horvitz, and K. E. Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009*, pages 103–114. ACM, 2009.
3. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
4. J. W. Byun, H. S. Rhee, H. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management, Third VLDB Workshop, SDM 2006*, volume 4165 of *Lecture Notes in Computer Science*, pages 75–83. Springer, 2006.
5. Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.
6. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.
7. J. Crampton. Cryptographic enforcement of role-based access control. In *Formal Aspects in Security and Trust*, volume 6561 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2010.
8. R. Curtmola, J. A. 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, CCS 2006*, pages 79–88. ACM, 2006.
9. E.-J. Goh. Secure indexes. IACR Cryptology ePrint Archive, Report 2003/216, 2003.
10. A. Kaci, T. Bouabana-Tebibel, and Z. Challal. Access control aware search on the cloud computing. In *2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014*, pages 1258–1264. IEEE, 2014.
11. S. Kamara and C. Papamonthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2013.
12. S. Kamara, C. Papamonthou, and T. Roeder. Dynamic searchable symmetric encryption. In *The ACM Conference on Computer and Communications Security, CCS'12*, pages 965–976. ACM, 2012.
13. Z. A. Kissel and J. Wang. Verifiable symmetric searchable encryption for multiple groups of users. In *Proceedings of the 2013 International Conference on Security and Management*, pages 179–185. CSREA Press, 2013.
14. M. Li, S. Yu, N. Cao, and W. Lou. Authorized private keyword search over encrypted data in cloud computing. In *2011 International Conference on Distributed Computing Systems, ICDCS*, pages 383–392. IEEE Computer Society, 2011.
15. C. Office. Goverment security classifications. Technical report, 2013.
16. M. Onen, R. Molva, and C. Van Rompay. Multi-user searchable encryption in the cloud. In *Information Security - 18th International Conference, ISC 2015*, volume 9290 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2015.
17. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE, 2000.
18. W. Sun, S. Yu, and W. Lou. Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 226–234. IEEE, 2014.
19. W. Sun, S. Yu, W. Lou, T. Hou, and H. Li. Protecting your right: Verifiable attribute-based keyword search with fine-grainedowner-enforced search authorization in the cloud. *IEEE Transactions on Parallel Distributed Systems*, 27(4):1187–1198, 2016.

20. Y. Yang. Attribute-based data retrieval with semantic keyword search for e-health cloud. *Journal of Cloud Computing: Advances, Systems and Applications*, 4, 2015.