MK-SEAC: Multi-Keyword Searchable Encryption with Access Control

Riccardo Longo[1][a], Enrico Sorbera[1][b] and Valeria Vicard[2][c]
[1]*Fondazione Bruno Kessler, Center for Cybersecurity, Trento, Italy*
[2]*Adverity GmbH, Vienna, Austria*

Keywords: Searchable Encryption, Attribute-Based Encryption, Multi-Keyword Search.

Abstract: In this work we present MK-SEAC, a scheme that allows to perform outsourced keyword-based search queries on an encrypted document batch, with native support to multi-keyword queries, fine-grained access control on the results, and full verifiability. The scheme conceals both the query and the results from the server, and minimizes the information revealed about documents to which users do not have access. Compared to other schemes, such as SEAC by Nils Löken, MK-SEAC offers extended improved efficiency with a reduced leakage profile.

1 INTRODUCTION

Searchable Encryption (SE) (Song et al., 2000), introduced by Song et al. in 2000, consists of a set of algorithms that allow users to delegate the storage of encrypted documents on a remote location while retaining the ability to search for specific data without the need first to decrypt them. Since then, various schemes have been presented, each introducing valuable properties. In particular, we are going to focus on a context in which each document is described by a set of keywords, and the search operation is performed utilizing these keywords. More precisely, we would like to include multiple keywords in a single query and find only documents that feature all the specified keywords. Additionally, we would like our scheme to guarantee the verifiability of the search results, that is, in response to a search query, the user receives from the server a set of results that includes all the information necessary to prove its completeness and correctness.

Moreover, we would also like to enforce fine-grained access control on the set of results. In fact, in several practical scenarios, the data should be accessible by a wide variety of users, with a potentially vast number of different access requirements. A common solution to this problem is provided by Attribute-Based Encryption (ABE). This type of encryption scheme was introduced in (Sahai and Waters, 2004) and is used to cryptographically enforce access control (i.e., users can access a document if and only if they can decrypt it) in a setting where access policies are defined on a set of *attributes* and with formulas on these attributes. The two main paradigms of ABE are: Key-Policy Attribute-Based Encryption (KP-ABE) (Goyal et al., 2006), where the documents are described by sets of attributes, and the user keys embed access policies that dictate which documents they will be able to decrypt; and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) (Bethencourt et al., 2007), where conversely the user keys embed attributes that describe the users themselves, while the ciphertexts embed an access policy that restricts which users can access them. In our construction, we use CP-ABE, so we will have a central trusted authority that assigns attributes to users and provides them with the corresponding keys.

1.1 Related Work

During the last decade, several attempts have been made to combine SE and ABE, one of the most interesting being (Li et al., 2020), which achieves collusion resistance between the cloud server and search users and ensures both backward and forward privacy. Unfortunately, it does not guarantee verifiability and has a computational complexity of $O(n^2)$, where n is the number of documents. Another important work is (Yin et al., 2023), as it supports numeric attribute comparison in access policies. Its drawbacks

[a] https://orcid.org/0000-0002-8739-3091
[b] https://orcid.org/0009-0009-0973-512X
[c] https://orcid.org/0009-0002-4334-1641

Longo, R., Sorbera, E. and Vicard, V.
MK-SEAC: Multi-Keyword Searchable Encryption with Access Control.
DOI: 10.5220/0013529200003979
In *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, pages 443-454
ISBN: 978-989-758-760-3; ISSN: 2184-7711

are mainly the lack of backward privacy, verifiability, and fork consistency.

Considering the downsides of these results, we took inspiration mainly from Nils Löken's SEAC (Löken, 2017), as it guarantees the verifiability of the search results in a static environment.

The idea behind SEAC is to pre-compute all search results and insert them in a structure that is outsourced to an online cloud provider. Following the structure's pre-computed paths, the provider can access only the ciphertexts of the documents the user requested. In particular, the structure presented in SEAC can be thought of as a series of filters, each concealing a specific element from the server. For instance, the first one is a Hash Table (`HT`) where all the utilized keywords are inserted masked through a pseudo-random function. Each entry in `HT` is then mapped to an encrypted linked list in the next structure (`A`), where we have the ABE policies of the documents that are described with that keyword.

SEAC has been extended by Löken and Blömer to the dynamic case (i.e., allowing the addition of documents to the initial document collection), resulting in dSEAC (Blömer and Löken, 2019), which ensures Forward and Backward Privacy, as well as Fork Consistency.

Our work aims to improve SEAC by adding native support for searches over multiple keywords at once and simultaneously reducing the leakage profile while retaining full verifiability with a reduced total number of signatures. Intuitively, we filter immediately with the access policy: this eliminates the leakage of documents beyond access rights and allows us to be more expressive with keyword filtering.

Organization. In Section 2 we give some preliminaries and background information on the primitives used in our construction, which is presented in detail in Section 3. In particular, Section 3.1 describes the supporting structures enabling the encrypted search and their contents, Section 3.2 describes setup and user enrollment, Section 3.3 presents how queries are created and processed, and Section 3.4 shows how the results can be verified. Finally, in Section 4 we draw our conclusions, showing how MK-SEAK improves on SEAC and some future directions.

2 PRELIMINARIES

2.1 CP-ABE with Key Customization

In a Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme, given a universe of attributes $\mathcal{U}$, users are associated with an identifier *uid* and a set of attributes $S \subseteq \mathcal{U}$. A trusted entity performs the setup, obtaining the master secret key *msk* and publishing the public parameters *pp* and the public key *pk*. This entity then gives to each user a key κ_S^{uid}. We denote by $\text{ABE}.Enc_{\mathbb{A}}(r)$ the encryption of message r for the policy $\mathbb{A} \subseteq \mathcal{P}(\mathcal{U}) \setminus \emptyset$. A user with key κ_S^{uid} can correctly decrypt a ciphertext and obtain r if and only if $S \in \mathbb{A}$ (i.e. the attributes satisfy the policy).

In our MK-SEAC protocol, we use an instance of CP-ABE that allows users to customize their keys. In this kind of CP-ABE protocol (e.g. (Agrawal and Chase, 2017; Rouselakis and Waters, 2015)) keys can be seen as composed of individual keys for each attribute. This allows users to easily derive less powerful keys that do not present every attribute, and therefore they can delegate some of their access privileges without having to disclose their full key. Our construction requires an IND-CPA CP-ABE scheme.

2.2 Symmetric Encryption and Digital Signature

Since asymmetric encryption schemes (and ABE in particular) are quite expensive, a well-established hybrid approach (Weber, 2013) is used: the actual contents of the documents are encrypted with a symmetric cipher, using a key $k = \text{KDF}(r)$ that is derived by applying an appropriate key derivation function on a short random message r, and only r is encrypted with CP-ABE. In particular, our construction uses an authenticated IND-CCA2 (Sahai, 1999) symmetric cipher, and we denote with $\text{Sym}.Enc_k(m)$ the encryption of the message m with the symmetric key k. Sometimes we will consider separately the authentication tag of a message m produced with the authenticated symmetric cipher and the key k, in these cases, we denote it with $\texttt{tag}_k(m)$.

To ensure integrity and verifiability of the construction (see (Blömer and Löken, 2019) for more details about verifiability), we exploit an EUF-CMA (Goldwasser et al., 1988) digital signature. We will denote with $\text{Sig}_{sk}(m)$ the signing of message m with secret key *sk*, and with $\text{Ver}_{pk}(m, \sigma)$ the verification whether σ is a valid signature for the message m and the public key *pk*.

2.3 Pseudo-Random Function

Let *KW* be a fixed universe of keywords. Following the original SEAC (Löken, 2017), we employ a pseudo-random function $f_{k_U}()$ which takes as input the description of an ABE policy $\mathbb{A}$, a set of keywords $L \subseteq KW$ (ordered lexicographically to have unique

representations), and outputs two-byte strings of fixed length, in particular, the second output has the same length of a key for the chosen symmetric encryption scheme.

We require this function to be collision resistant (Hirose, 2022), and the second output should be indistinguishable from a byte string generated uniformly at random. Furthermore, we require that some collisions on the first outputs happen with low probability, more precisely that, for any $L_1, L_2 \subseteq L$ with $\emptyset \neq L_1 \neq L_2 \neq \emptyset$ and $\mid L \mid \leq T$, we have that $\mathbb{P}(f_{k_U}(\mathbb{A}, L_1)_1 = f_{k_U}(\mathbb{A}, L_2)_1) < \varepsilon$.

Note that such a function can be constructed by exploiting a hash function with a sufficiently long output. In particular, if we model the pseudo-random first output as a uniformly random string of ℓ bytes, then by the birthday attack (Mosteller, 2006) we have that the collision probability given 2^T inputs is upper-bounded by $2^{2T-8\ell-1}$, so it is sufficient to take $\ell > \frac{2T-1-\log_2(\varepsilon)}{8}$.

This function is necessary to prevent the Cloud Provider from getting the set of keywords L_{doc} describing a document, as we will discuss in the next section. This is possible as $f_{k_U}()$ can be calculated by all enrolled users since they receive the secret key k_U from the Certification Authority, but not by the Cloud Provider unless it corrupts a user (see Appendix A for the formal proof of the leakage of the scheme).

3 SCHEME DESCRIPTION

First, let us describe the entities involved:

- *Data Owner* (DO): a user that owns the *Document Batch* (*DB*) to be outsourced encrypted on the cloud.
- *Data User* (DU): a client of the service, they generate queries based on keywords and their attributes to be executed by the service provider.
- *Cloud Provider* (CP): a service provider, it receives the encrypted database alongside an additional structure called *Index*, which allows to correctly answer the DU's queries.
- *Certification Authority* (CA): a trusted authority that is in charge of authenticating the other entities and distributing the cryptographic keys.

The idea behind MK-SEAC, inspired by SEAC, is that the DO encrypts the set of documents to be outsourced. The flow of the protocol is as follows:

1. Each document is characterized by one or more keywords and a policy that prescribes which attributes are required to access the document.
2. To enable efficient searches on this data, the DO builds an index I, consisting of multiple linked structures. Basically, the index pre-computes the possible searches (queries) that a DU may do and allows the CP to correctly process and answer these queries without having access to the plaintext data.
3. To participate, the DUs have to enroll: the CA assigns them the attributes that describe which documents they are entitled to access, and provides them with the cryptographic material needed to construct the queries and decrypt the results.
4. To search for documents, the DUs select a set of keywords and a subset of their attributes to generate a query.
5. The CP, upon receiving a query from a DU, exploits the index to return the ciphertexts of the documents that are associated with the selected keywords and whose policy is satisfied by the given attributes.
6. The DU can verify the correctness and completeness of the results, by exploiting data given by the CP and CA, which is signed by the DO.

3.1 The Index

As we remarked, the Index is the core structure of our scheme, that enables searching on the encrypted *DB*. We now present the building blocks of the index (which are essentially encrypted dictionaries), in the order they are visited by the CP when processing a query:

1. `A`: the first structure enforces attribute-based access control. Each node in `A` serves as an entry point to all documents that have a given access policy. By placing this structure right at the beginning (in contrast with SEAC), we limit the leakage: a DU will learn nothing about documents to which they do not have access.
2. `K`: the second structure is a set of encrypted linked lists that is in charge of filtering the results depending on the keywords specified in the query.
3. `R`: the third structure acts as a sort of router, that enables efficient multi-keyword searches and deduplicates data by routing to the same result all the paths that lead to files that can be searched in multiple ways.
4. `D`: the fourth structure is a set of encrypted linked lists, each item containing the index of an encrypted document fulfilling the query and the CP-ABE encryption of the symmetric key necessary to decrypt it.

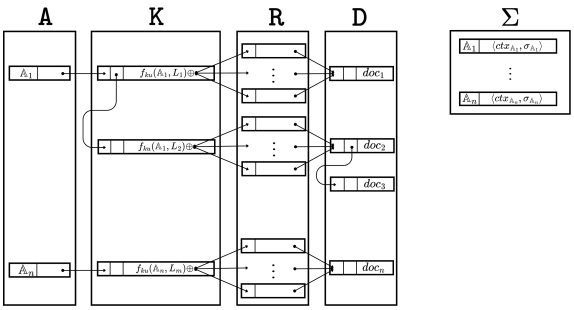

Figure 1: High-level description of the index I.

5. Σ: the last structure contains all the signatures, and allows the DU to verify the correctness and completeness of CP's answers to their queries.

Figure 1 presents a high-level scheme of the index's structures and how they are linked.

3.1.1 Contents of the Index's Structures

Each document doc in DB is associated with an anonymous identifier id_{doc}, is described by a set of keywords $L_{doc} \subseteq KW$ (we suppose KW to be known in advance to the DO and each of the DUs), and its access is regulated by a policy $\mathbb{A}_{doc} \subseteq \mathcal{P}(\mathcal{A}) \setminus \emptyset$. Denoting with ptx_{doc} the actual contents of the document, we have that the complete description is:

$$doc = (id_{doc}, ptx_{doc}, L_{doc}, \mathbb{A}_{doc}). \tag{1}$$

Let us now describe in detail what is contained in each of the structures composing the Index, starting with A, which is the first structure to be accessed when evaluating a query.

For each access policy $\mathbb{A}$ for which there is a document $doc \in DB$ such that $\mathbb{A} = \mathbb{A}_{doc}$, we define an entry in A:

$$n^{\mathtt{A}}_{\mathbb{A}} = (\mathbb{A}, \text{ABE.}Enc_{\mathbb{A}}(r_{\mathbb{A}}), \text{Sym.}Enc_{\text{KDF}(r_{\mathbb{A}})}\left(id^{K}_{L_{\mathbb{A},1}}\right)), \tag{2}$$

where $id^{K}_{L_{\mathbb{A},1}}$ is an index in K. Basically, A maps every policy to a hybrid ABE encryption of a pointer to a linked list inside the next structure K, more precisely to the index of the first element of the linked list. Note that the CP can decrypt this index only if it is given an ABE secret key that satisfies the policy: this assures that the DUs that do not have access to the documents with this policy will not be able to produce queries that give the CP the ability to decrypt this index, and thus they will not learn anything. Note that our notations explicit the link between the identifier $id^{K}_{L_{\mathbb{A},1}}$, a policy $\mathbb{A}$, and a set of keywords $L_{\mathbb{A},1}$, but in reality, the value of the index does not reveal anything about this association.

The linked lists K are organized following the partitions on DB induced by the distinct access policies and sets of keywords associated with the documents. More precisely, in K there is a linked list for each distinct access policy (and we saw how A helps the CP to navigate to the start of the linked list given a policy), then each element inside a single list is associated to a different set of keywords. Formally, let us enumerate the distinct sets of keywords associated with the documents with a given access policy: $\{L_{\mathbb{A},1}, \dots L_{\mathbb{A},t_{\mathbb{A}}}\} = \{L : \exists\ doc \in DB\ s.t.\ \mathbb{A}_{doc} = \mathbb{A},\ L_{doc} = L\}$, then for $i \in \{1, \dots, t_{\mathbb{A}}\}$ we have that the i-th entry of the linked list associated to $\mathbb{A}$ has index $id^{\mathtt{K}}_{L_{\mathbb{A},i}}$ and value:

$$id^{\mathtt{K}}_{L_{\mathbb{A},i}} : \text{Sym.}Enc_{\text{KDF}(r_{\mathbb{A}})}\left(id^{\mathtt{K}}_{L_{\mathbb{A},i+1}}, \mathtt{pad}_{L_{\mathbb{A},i},0}, \mathtt{pad}_{L_{\mathbb{A},i},1}\right), \tag{3}$$

where $id^{\mathtt{K}}_{L_{\mathbb{A},i+1}}$ is the identifier of the next element in the list, that is associated with another set of keywords $L_{\mathbb{A},i+1}$. To signal the end of the list we set $id^{\mathtt{K}}_{L_{\mathbb{A},t_{\mathbb{A}}+1}} = \perp$, where $\perp$ is a special value. The values $\mathtt{pad}_{L_{\mathbb{A},i},0}, \mathtt{pad}_{L_{\mathbb{A},i},1}$ are two randomly generated byte strings with the same length as the pseudo-random function $f_{k_U}()$, they will be used to derive links and keys to access the following structure R. Note that the entire entry is encrypted with the same key derived from the ABE encryption found in A.

If a DU searches for documents matching a set of keywords L', then a document associated with the set $L \supseteq L'$ should show up in the results if the user can access the document. This means that a document with keywords L_{doc} should be found starting from any set $L \subset L_{doc}$. However, for efficiency reasons, we want to link each $doc \in DB$ only once in the pre-computed search result characterized by the pair $(\mathbb{A}_{doc}, L_{doc})$. This is a major difference with respect to SEAC, which gives a consistent efficiency gain. In order to enable the retrieval with any subset, we exploit the routing structure R which has an entry for each subset but then makes each of them re-converge to the same entry in the following structure D. This allows multiple search paths with minimal duplicate data. In particular we exploit the pads $\mathtt{pad}_{L_{\mathbb{A},i},0}, \mathtt{pad}_{L_{\mathbb{A},i},1}$ and the pseudo-random function $f_{k_U}()$ to define for each $L' \in \mathcal{P}(L_{\mathbb{A},i}) \setminus \emptyset$ a pair $(ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L'}, k^{\mathtt{R}}_{L_{\mathbb{A},i},L'})$ that gives a pointer in R and a symmetric key to decrypt the corresponding entry:

$$(ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L'}, k^{\mathtt{R}}_{L_{\mathbb{A},i},L'}) := f_{k_U}\left(\mathbb{A}, L'\right) \oplus (\mathtt{pad}_{L_{\mathbb{A},i},0}, \mathtt{pad}_{L_{\mathbb{A},i},1}), \tag{4}$$

where the xor operation $\oplus$ is performed componentwise. Note that this construction implies that indexes in R are byte strings of fixed length and that the properties of $f_{k_U}()$ (see Section 2.3) assure that we derive secure keys and that, with high probability, $ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L_1} \neq ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L_2}$ for $L_1, L_2 \in \mathcal{P}(L_{\mathbb{A},i}) \setminus \emptyset, L_1 \neq L_2$.

As anticipated, the third structure R routes all the

different ways to obtain the documents associated with a set of keywords as a search result (i.e., any search that specifies a subset of this set) to the same pointer to these documents. In particular, the information necessary to retrieve and decrypt the documents is stored (encrypted) in D, so R maps the pointers extracted from K as in Equation (4) to the correct elements of D, simultaneously providing the key to decrypt it. So, for every $ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L'}$ where $\emptyset \neq L' \subseteq L_{\mathbb{A},i}$, in R we have the association:

$$ptr^{\mathtt{R}}_{L_{\mathbb{A},i},L'} : \mathrm{Sym}.\widehat{Enc}_{k^{\mathtt{R}}_{L_{\mathbb{A},i},L'}}\left(ptr^{\mathtt{D}}_{L_{\mathbb{A},i},1}, k^{\mathtt{D}}_{L_{\mathbb{A},i}}\right), \quad (5)$$

where $id^{doc}_{L_{\mathbb{A},i},1}$ is the pointer to the first element of an encrypted linked list in D, $k^{\mathtt{D}}_{L_{\mathbb{A},i}}$ is the key to decrypt the elements of that list, and $\mathrm{Sym}.\widehat{Enc}()$ is the symmetric encryption function modified with the CTX construction (Chan and Rogaway, 2022) (where the authentication tag is substituted by a hash of key, nonce, and tag) to have key commitment. This variant assures that a valid decryption is obtained only with the correct key, in particular, we are assured that we will not succeed in decrypting entries in R with the wrong key, and this will allow the CP to process queries correctly (see Section 3.3.2).

Finally, in D we have an entry for each $doc \in DB$, organized in linked lists where each list gathers all the documents that share the same access policy and set of keywords (and thus a correct response to any search query will always return either none of them or all of them together). Formally, let us enumerate the ids of the documents with access policy $\mathbb{A}$ associated with the set of keywords $L_{\mathbb{A},i}$:

$$\left\{id^{doc}_{L_{\mathbb{A},i},1}, \ldots, id^{doc}_{L_{\mathbb{A},i},t_{L_{\mathbb{A},i}}}\right\} = \left\{id_{doc} : \exists doc \in DB \, s.t. \, \mathbb{A}_{doc} = \mathbb{A}, \, L_{doc} = L_{\mathbb{A},i}\right\}, \quad (6)$$

then for $j \in \{1, \ldots, t_{L_{\mathbb{A},i}}\}$ we have that the j-th entry of the linked list is:

$$id^{doc}_{L_{\mathbb{A},i},j} : \mathrm{Sym}.Enc_{k^{\mathtt{D}}_{L_{\mathbb{A},i}}}\left(id^{doc}_{L_{\mathbb{A},i},j+1}, \mathtt{abe}_{L_{\mathbb{A},i},j}, \mathrm{Sym}.Enc_{\mathrm{KDF}\left(r_{L_{\mathbb{A},i}}\right)}\left(k_{L_{\mathbb{A},i},j}\right)\right), \quad (7)$$

where $\mathtt{abe}_{L_{\mathbb{A},i},j}$ is a byte string of fixed length (equal to the longest ABE encryption among all access policies actually used in *DB*) that, for $j > 1$ is simply a string of zeros, while for $j = 1$ it encodes the (possibly padded) encryption $\mathrm{ABE}.Enc_{\mathbb{A}^+_{doc}}\left(r_{L_{\mathbb{A},i}}\right)$. $\mathbb{A}^+ = \mathbb{A}$ & read is a stricter policy enriched with the attribute read that prevents the CP to decrypt the documents (see Section 3.3.1). This string serves to avoid duplicate encryptions, and, without revealing (with the length of the ciphertext) which entries of D are the start of the lists, it lets the DU recover the random value $r_{L_{\mathbb{A},i}}$ from which they can derive the symmetric key to decrypt all the per-document keys in the list: in fact in the encrypted *DB* we find the documents saved as:

$$id^{doc}_{L_{\mathbb{A},i},j} : \mathrm{Sym}.Enc_{k_{L_{\mathbb{A},i},j}}\left(ptx_{id^{doc}_{L_{\mathbb{A},i},j}}\right). \quad (8)$$

As in K, we signal the end of the list by setting $id^{doc}_{L_{\mathbb{A},i},t_{L_{\mathbb{A},i}}+1} = \perp$.

3.1.2 Signatures

The last structure of the index, Σ, contains the verification data, used to check the accuracy and completeness of the search results. In Σ we have, for every policy $\mathbb{A}$ mentioned in *DB*:

$$\mathbb{A} : (\mathtt{check}_{\mathbb{A}}, \mathrm{Sig}_{sk_{uid_{\mathrm{DO}}}}(\mathtt{check}_{\mathbb{A}})), \quad (9)$$

where uid_{DO} identifies the DO that uploaded the *DB*, $sk_{uid_{\mathrm{DO}}}$ its private signing key, and

$$\mathtt{check}_{\mathbb{A}} := \left(\mathrm{ABE}.Enc_{\mathbb{A}^+}(r^{\mathtt{c}}_{\mathbb{A}}), \mathrm{Sym}.Enc_{\mathrm{KDF}(r^{\mathtt{c}}_{\mathbb{A}})}\left(uid_{\mathrm{DO}}, \left(L_{\mathbb{A},i}, \mathtt{t\_d}_{\mathbb{A}[i]}\right)_{i=1}^{t_{\mathbb{A}}}\right)\right),$$
$$\mathtt{t\_d}_{\mathbb{A}[i]} := H\left(\mathtt{tag}_{k_{L_{\mathbb{A},i},j}}(ptx_{id^{doc}_{L_{\mathbb{A},i},j}})\Big\|_{j=1}^{t_{L_{\mathbb{A},i}}}\right) \quad (10)$$

is basically an ABE hybrid encryption that allows the DUs to see which sets of keywords are associated with at least one document with access policy $\mathbb{A}$, and for each of these sets gives the hash of the concatenation of all the authentication tags of each document that is associated to that set of keywords and policy. As we will see in Section 3.4, these values will enable verifiability. Note that a straightforward encryption leaks the length of the plaintext, which in this case gives away how many different sets of keywords are associated with every policy. To avoid this we pad the plaintexts so that the size of each $\mathtt{check}_{\mathbb{A}}$ is the same for every $\mathbb{A}$.

3.2 Enrollment

The CA is responsible for authenticating and enrolling the users (the DO and the DUs), and maintaining a public structure (CL) that acts as a public key infrastructure and allows the CP to authenticate users. After running the Setup of the CP-ABE scheme (Section 2.1, the CA initializes CL with the public parameters *pp* and the public key *pk*.

When a user asks to enroll in the system, first of all, they are associated with a unique identifier *uid*. The user is supposed to have autonomously generated a signing key pair (sk_{uid}, pk_{uid}), and to send pk_{uid} with the enrollment request. The CA associates the user to an appropriate set of attributes $\mathcal{A}_{uid}$ that describes the user's roles and permissions and generates the user's ABE secret key $\kappa^{uid}_{\mathcal{A}_{uid}\cup\{\texttt{read}\}}$. Finally, the CA sends back to the user the shared user key k_U and the CP-ABE secret key $\kappa^{uid}_{\mathcal{A}_{uid}\cup\{\texttt{read}\}}$, and publishes in `CL` the association $uid : pk_{uid}$.

The contents of `CL` should be signed by the CA to ensure authenticity. A simple approach would be to sign each entry separately, however, more complex designs akin to Certificate Transparency Logs (Laurie, 2014) may bring desirable features such as batch signing, or monitoring and accountability (i.e., independent monitors and co-signers can help in keeping the CA honest).

3.3 Search Queries

Noticing that the set of used access policies $\{\mathbb{A} : \exists doc \in DB\, s.t.\, \mathbb{A}_{doc} = \mathbb{A}\}$ is publicly available as the indexes of `A`, we order its elements into a *Policy List* `PL` and assume that it is distributed to every enrolled user (signed by the CA to ensure authenticity). This does not worsen the leakage profile, since CP-ABE ciphertexts already leak the associated policy.

At a high level, a query is characterized by a set of keywords $L \subseteq KW$ and the set of attributes $\mathcal{A} \subseteq \mathcal{U}$ that the DU wants to use for the search, looking for all documents that are associated to those keywords and that can be accessed with that attribute set:

$$\{doc \in DB\, s.t.\, L \subseteq L_{doc} \wedge \mathcal{A} \in \mathbb{A}_{doc}\}. \quad (11)$$

Our privacy requirements are that the CP should not learn the contents of the documents that match the query, nor the set of keywords L.

3.3.1 Query Generation

Let $\left(\mathbb{A}_{\mathcal{A},i}\right)_{i=1}^{n_{\mathcal{A}}}$ be the list of policies that are satisfied by the set of attributes $\mathcal{A} \subseteq \mathcal{U}$, taken in the same order as they appear in `PL`. A query payload for the set of keywords $L \subseteq KW$ and the set of attributes $\mathcal{A} \subseteq \mathcal{U}$ is computed as:

$$Q = \left(\kappa^{uid}_{\mathcal{A}}, f_{k_U}\left(\mathbb{A}_{\mathcal{A},1}, L\right), \ldots f_{k_U}\left(\mathbb{A}_{\mathcal{A},n_{\mathcal{A}}}, L\right)\right), \quad (12)$$

where *uid* is the identifier of the DU that is making the query. Note that the properties of CP-ABE (see Section 2.1) assure that the DU is only able to generate queries for attribute sets $\mathcal{A}$ such that $\mathcal{A} \subseteq \mathcal{A}_{uid}$, i.e., users can search only for documents to which they have access. Note also that in order to prevent the CP from accessing the documents, we always assume that $\texttt{read} \notin \mathcal{A}$. To request the search, the DU sends to the CP the signed query: $(Q, \mathrm{Sig}_{sk_{uid}}(Q))$.

3.3.2 Query Processing

When the CP receives a query $(Q, \mathrm{Sig}_{sk_{uid}}(Q))$ from a DU with identifier *uid*, first of all it verifies that the query comes from an authorized user. The CP obtains from `CL` the verification key pk_{uid} associated with the user (if the id is not found, the query is ignored because the user is not enrolled), then uses this key to verify the signature $\mathrm{Sig}_{sk_{uid}}(Q)$.

Once the request has been authenticated, the CP follows the order given by `PL` to visit the entries in `A` and decrypt those that can be accessed with $\kappa^{uid}_{\mathcal{A}}$. Note that usually the set of attributes $\mathcal{A}$ is encoded inside the ABE private key, so it is not necessary to try to decrypt every entry in `A`, and that the CP decrypts the entries associated to $(\mathbb{A}_{\mathcal{A},1}, \ldots, \mathbb{A}_{\mathcal{A},n_{\mathcal{A}}})$ in order.

Each of these successful decryptions gives the index to an entry in `K` (see Equation (2)), which (once decrypted) eventually gives a list of pair of pads (see Equation (3)): one for each of the sets of keywords associated to that policy.

Let $\mathbb{A}$ be the policy currently used, i.e., we have decrypted the entry in `A` associated with $\mathbb{A}$ and from `K` we have obtained the list of pairs $\Big((\texttt{pad}_{L_{\mathbb{A},1},0}, \texttt{pad}_{L_{\mathbb{A},1},1}), \ldots, (\texttt{pad}_{L_{\mathbb{A},t_{\mathbb{A}},0}}, \texttt{pad}_{L_{\mathbb{A},t_{\mathbb{A}},1}})\Big)$. For each of these pairs, the CP computes $f_{k_U}(\mathbb{A}, L) \oplus (\texttt{pad}_{L_{\mathbb{A},i},0}, \texttt{pad}_{L_{\mathbb{A},i},1})$, where $f_{k_U}(\mathbb{A}, L)$ is taken from Q, and interprets the result as a pointer to an entry in `R` and the key to decrypt it (see Equation (4)). Then, the CP searches in `R` the entry corresponding to this pointer and (if it exists) tries to decrypt it with the aforementioned key. By construction of `R` (see Equation (5)), the CP obtains a valid pointer and a valid decryption if and only if $L \subseteq L_{\mathbb{A},i}$ (i.e. the set of keywords is a match), and the value obtained by the decryption encodes a pointer to an entry in `D` and the key to decrypt it.

By decrypting that entry (and following the linked list), the CP obtains a CP-ABE encryption of the key-encrypting key and a list of pairs consisting of a pointer to an encrypted document in the *DB* and the wrapping of the key to decrypt it (see Equation (7)). The CP fetches the ciphertexts from the *DB* (see Equation (8)) and appends to the list of results the CP-ABE-wrapped Key-Encrypting Key (KEK) and all the ciphertexts together with the corresponding wrapped key. When all the results have been retrieved, the CP adds, for verifiability reasons, all the entries in Σ corresponding to the visited policies $(\mathbb{A}_{\mathcal{A},1}, \ldots, \mathbb{A}_{\mathcal{A},n_{\mathcal{A}}})$.

Note that the DU is able to obtain the KEK from the CP-ABE wrapping, and with this to obtain all the document keys, therefore to decrypt all the documents in the results.

3.4 Verifiability

As previously mentioned, a desirable property is to allow the DU to verify the completeness and correctness of the results returned by the CP on processing their search queries. In practice, the DU should be able to verify that in response to the query Q as in Equation (12), the CP sends a result that gives the set of documents defined in Equation (11). Let us now prove that the data taken from Σ ensures the verifiability of the results.

First, note that the unforgeability of the signatures assures that the CP cannot tamper with the data in Σ, and that the DU knows, thanks to $\texttt{PL}$, which elements of Σ should be returned within the query response. Then, note that the DU should be able to correctly decrypt the $\texttt{check}_{\mathbb{A}}$, and the corresponding plaintext allows to verify that it is the correct entry (CP-ABE encryptions contain the policy) and that it has been created by the DO (the symmetric encryption provides the user identifier, that can be used to get the verification key form $\texttt{CL}$).

At this point, the DU organizes the documents received according to the partition induced by the distinct access policies and sets of keywords associated with them. For each policy $\mathbb{A}$ let $(L_{\mathbb{A},1},\ldots,L_{\mathbb{A},t_{\mathbb{A}}})$ be the list of set of keywords given by the decryption of $\texttt{check}_{\mathbb{A}}$ (see Equation (10)). Now, for each $L_{\mathbb{A},i}$ such that $L \subseteq L_{\mathbb{A},i}$, the DU concatenates the authentication tags (from the authenticated symmetric encryptions) of all the documents in the results that are associated with $L_{\mathbb{A},i}$. This concatenation follows the same (arbitrary, but pre-determined) order of the document identifiers used in Equations (6), (7) and (10). This byte string is then hashed and the result is compared with the digest contained in $\texttt{check}_{\mathbb{A}}$: if the two correspond then we are sure that the results included all the documents with policy $\mathbb{A}$ and set of attributes $L_{\mathbb{A},i}$, and also that these documents have not been modified after the DO has uploaded them and created the entry in Σ. Therefore, by performing the checks for every policy and corresponding sets of attributes, the DU can indeed check the correctness and completeness of the results.

Finally, we require the DO to sign the whole index and all encrypted documents. This signature will be kept by the CA and gives to the DO full responsibility on the data (a malicious DO could blame the CP for data malformed at origin).

3.5 Security

MK-SEAC has a quite permissive threat model, where only one entity is supposed truly honest. Specifically, our assumptions are the following:

- The DO is assumed to delete its data once the encrypted batch and the Index have been uploaded on the cloud. To search for data it should act as a simple DU. Note that the DO signs everything, so it can be blamed for a poorly computed index or corrupted encrypted data.
- The DUs can be corrupted, the leakage profile is analyzed in Appendix A. Note that colluding users cannot access more data than the union of what they could read individually.
- The CP can be malicious, in the sense that any deviation from the proscribed behaviour is detectable.
- The CA is assumed to be honest. This assumption could be mitigated by decentralizing this role through secure multiparty computation techniques, which are outside of our scope.

This model is stronger compared to the one in (Löken, 2017), where the CP has to be honest but curious to ensure the completeness of search results, and a malicious DO could upload corrupted data and blame the CP for the incorrectness of search result.

4 CONCLUSIONS

We have presented a novel protocol that combines Searchable Encryption and Attribute-Based Encryption to enable easier retrieval with fine-grained privacy and access control in remote storage. Similarly to SEAC, MK-SEAC allows the outsourcing of a searchable document collection to a third party without compromising data confidentiality. Both static protocols ensure the completeness of the search results and the secrecy of the keywords. The main contributions of our work are native support for searches over multiple keywords at once, rather than the single-keyword query permitted by SEAC, while considering a stronger threat model (see Section 3.5) and reducing the leakage profile of the structure (see Appendix A). These improvements come at the cost of a new structure $\texttt{R}$, whose size grows exponentially with the number of keywords that describe a single document, although every single entry has a quite small size. The construction also retains full verifiability while reducing the total number of signatures required. The overall efficiency in terms of cryptographic operations needed to upload a batch, process-

ing a query operation, and verifying a set of results is similar to the one in SEAC (see Appendix B).

Optimizations. The multi-keyword searches enabled by MK-SEAC are effectively queries where the individual keywords are combined with a logical AND. In SEAC, the same result can be achieved by making a query for each single keyword and then combining the results by intersecting the individual responses. Besides the straightforward efficiency gains offered by MK-SEAC, obtained by performing a single query instead of multiple ones followed by post-processing, another benefit is the reduced leakage. In fact, MK-SEAC does not reveal anything about which documents are associated with a single keyword (not even in a masked form as in SEAC) unless a user specifically searches for a single keyword, and also in this case the CP is not able to understand that the search concerns a singleton instead of a bigger set of keywords.

When considering combined queries that use the logical OR instead of the AND, MK-SEAC still offers some advantages, but to a lesser extent. With the OR, the individual results should be combined with a union instead of an intersection, and indeed this is the standard approach both for SEAC and MK-SEAC. However, in MK-SEAC it is possible to avoid some redundant computations by exploiting the partition on the documents induced by the policies. In fact, since the control of the policies is performed immediately in MK-SEAC (contrarily to SEAC where the masked keyword is checked first), we can exploit the fact that the sets of documents associated with different access policies are disjoint, and fully parallelize the search over each policy. Moreover, when combining searches for documents sharing the same policy $\mathbb{A}$, but characterized by different sets of keywords L_1 and L_2, we can note that if, by XORing the pads found in an entry of $\mathtt{K}$ with $f_{k_U}(\mathbb{A}, L_1)$, we obtain a pointer and a key that lead to successful decryption of an entry in $\mathtt{R}$, then it is useless to xor same pads with $f_{k_U}(\mathbb{A}, L_2)$. In fact, we would either obtain nothing (if L_2 is not a match) or the same result as before. Therefore it is sufficient to check again only entries in $\mathtt{K}$ which did not match with previously checked sets.

Future Work. The first natural extension, on which we are already working, is to allow the update of document batches in a dynamic setting. While an approach similar to the one used in dSEAC would fit our case too, there are hints that many optimizations are possible, e.g., by craftfully reusing parts of the indexes in subsequent updates. We are also working on a proof-of-concept implementation to empirically test and validate the construction and compare it with SEAC.

Another feature we would like to introduce is to optimize searches in a scenario where there is a second set of keywords that are supposed to be less sensitive and therefore would require less protection. In particular, less stringent privacy on these keywords could allow for more expressive queries on them, such as interval searches and regular expressions.

An interesting line of work could be exploring other possibilities regarding the structures of the index, to improve the efficiency or reduce the overhead. However it should be considered that improvements are likely to come at the cost of increased leakage, and there will be time-space tradeoffs. For example, the aggregation of data in the encrypted linked list $\mathtt{K}$ could reduce overhead but would likely leak the number of keywords per policy corresponding to at least one document in the batch. Padding could preserve time efficiency and privacy, at the cost of increased space. We envision more possibilities in improving the router $\mathtt{R}$, e.g., by exploiting more sophisticated cryptographic primitives or via clever tricks. However, note that currently the structure is already quite light and the XOR operations are arguably as efficient as it gets.

Finally, one could investigate the insertion of an LLM module to improve the labeling of documents and aid users in generating more accurate and efficient queries by selecting the best sets of keywords.

ACKNOWLEDGEMENTS

This work was partially supported by the project ”METAfora: Metodologie e tecnologie di rappresentazione per il metaverso” (CUP code B69J23000190005), funded by the Italian Ministero delle Imprese e del Made in Italy and coordinated by BIT4ID S.r.l., and by the project SERICS (PE00000014), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

Agrawal, S. and Chase, M. (2017). Fame: Fast attribute-based message encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’17, page 665–682, New York, NY, USA. Association for Computing Machinery.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *2007*

IEEE Symposium on Security and Privacy (SP '07), pages 321–334.

Blömer, J. and Löken, N. (2019). Dynamic searchable encryption with access control. In *Foundations and Practice of Security: 12th International Symposium, FPS 2019, Toulouse, France, November 5–7, 2019, Revised Selected Papers*, page 308–324, Berlin, Heidelberg. Springer-Verlag.

Chan, J. and Rogaway, P. (2022). On committing authenticated-encryption. In *European Symposium on Research in Computer Security*, pages 275–294. Springer.

Goldwasser, S., Micali, S., and Rivest, R. L. (1988). A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). *Attribute-based encryption for fine-grained access control of encrypted data*. CCS '06. Association for Computing Machinery, New York, NY, USA.

Hirose, S. (2022). Collision-resistant and pseudorandom function based on merkle-damgård hash function. In Park, J. H. and Seo, S.-H., editors, *Information Security and Cryptology – ICISC 2021*, pages 325–338, Cham. Springer International Publishing.

Laurie, B. (2014). Certificate transparency: Public, verifiable, append-only logs. *Queue*, 12(8):10–19.

Li, H., Yang, Y., Dai, Y., Yu, S., and Xiang, Y. (2020). Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data. *IEEE Transactions on Cloud Computing*, 8(2):484–494.

Löken, N. (2017). Searchable encryption with access control. *Proceedings of the 12th International Conference on Availability, Reliability and Security*.

Mosteller, F. (2006). *Understanding the Birthday Problem*, pages 349–353. Springer New York, New York, NY.

Rouselakis, Y. and Waters, B. (2015). Efficient statically-secure large-universe multi-authority attribute-based encryption. In Böhme, R. and Okamoto, T., editors, *Financial Cryptography and Data Security*, pages 315–332, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sahai, A. (1999). Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 543–553.

Sahai, A. and Waters, B. (2004). Fuzzy identity based encryption. Cryptology ePrint Archive, Paper 2004/086.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55.

Weber, S. G. (2013). Designing a hybrid attribute-based encryption scheme supporting dynamic attributes. Cryptology ePrint Archive, Paper 2013/219.

Yin, H., Li, Y., Deng, H., Zhang, W., Qin, Z., and Li, K. (2023). Practical and dynamic attribute-based keyword search supporting numeric comparisons over encrypted cloud data. *IEEE Transactions on Services Computing*, 16(4):2855–2867.

APPENDIX

A Leakage in MK-SEAC

In this section, we want to describe the leakage that our scheme incurs, considering an adversary that can take control of the server and is able to corrupt users. We describe such leakage through three leakage functions.

The first one (Equation (13)) depicts the leakage of enrolling an honest user, that simply coincides with the user identifier *uid*, which is published by the CA.

In MK-SEAC the leakage of corrupting an honest user or enrolling a dishonest one coincides (Equation (14)), and it contains the private information of the user and all the data the adversary can derive. In particular, the adversary obtains the user's identifier *uid* and their set of attributes $\mathcal{A}_{uid}$. Using the user's private ABE keys, the server can decrypt part of the index, discovering the linking between a document *doc*, its identifier id_{doc}, and its keywords L_{doc}, for each document *doc* in the batch that is accessible by the corrupted user. Similarly, taking advantage of the knowledge of the private key $\kappa^{uid}_{\mathcal{A}_{uid}}$, the adversary can decrypt some nodes in Σ, finding out, for each pair of policy and set of keywords, the digest of the concatenation of the tags of the corresponding documents. Moreover, since the adversary knows the private key k_U, it can generate the pseudo-random function $f_{k_U}(\mathbb{A}_{doc}, L_{doc})$.

The next leakage function (Equation (15)) describes the leakage caused by the DO updating a document batch. Here, the adversary learns the encrypted index, and therefore the cardinality of its structures. In particular, from the cardinality of $\mathtt{D}$, the adversary learns the number of documents in DB[1].

The last leakage function (Equation (16)) describes the leakage caused by searches. By sending a query, the user leaks their identifier *uid* and the subset of their attributes used in the query, $\mathcal{A}$. Then, by processing the query, the adversary discovers the evaluation $f_{k_U}(\mathbb{A}_i, L_i)$ and the binding between the identifier of the document id_{doc} and its access policy $\mathbb{A}_{doc}$, for each document *doc* in the batch that satisfies the query.

The proof of the following theorem is inspired by (Löken, 2017) and constructs a real experiment **Real** and a simulated one **Sim**. The idea is to prove that an adversary is unable to distinguish between them unless it is able to break either the symmetric or the ABE encryption. The strategy consists of having

[1]This leakage could be mitigated by inserting dummy entries in $\mathtt{D}$.

$$\mathcal{L}_{HJ}(U) = uid \tag{13}$$

$$\mathcal{L}_{COR}(uid) = \begin{pmatrix} (uid, \mathcal{A}_{uid}), \\ \{doc,\ id_{doc},\ L_{doc_i} \ :\ doc \in DB\} \ \wedge\ \mathcal{A}_{uid} \text{ satisfies } \mathbb{A}_{doc}\ \}, \\ \{\mathbb{A}, L, H(\{\mathtt{tag}_k(ptx_{id^{doc}}) \in DB :\ \mathbb{A}_{doc} = \mathbb{A},\ L_{doc} = L\}) \ :\ \mathcal{A}_{uid} \text{ satisfies } \mathbb{A}\} \\ \{f_{k_U}(\mathbb{A}_{doc}, L_{doc}),\ \mathbb{A}_{doc},\ L_{doc} :\ doc \in DB\} \end{pmatrix} \tag{14}$$

$$\mathcal{L}_{UP}(DC) = \begin{pmatrix} \#\mathtt{A},\quad \#\mathtt{K},\quad \#\mathtt{R},\quad \#\mathtt{D},\quad \#\Sigma, \\ \{\mathbb{A}\ :\ \exists\, doc \in DB,\ \mathbb{A}_{doc} = \mathbb{A}\} \end{pmatrix} \tag{15}$$

$$\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}}) = \begin{pmatrix} (uid, \mathcal{A}), \\ \{f_{k_U}(\mathbb{A}_i, L_i)\ :\ i \in \{1,\ldots,k\}\}, \\ \{f_{k_U}(\mathbb{A}_{doc}, L_{doc}),\ id_{doc},\ \mathbb{A}_{doc}\ :\ doc \in DB \ \wedge\ L_{doc} \supset L_i \ \wedge\ \mathcal{A}_{uid} \text{ sat. } \mathbb{A}_{doc}\} \end{pmatrix} \tag{16}$$

the simulated experiment gradually take over computations originally performed by the real one, in a series of hybrid scenarios.

Theorem A.1. *MK-SEAC is* $(\mathcal{L}_{UP}, \mathcal{L}_{COR}, \mathcal{L}_{SE})$*-semantically secure against static adversaries if instantiated with IND-CPA ciphertext-policy attribute-based encryption, and IND-CCA2 symmetric encryption.*

Proof. We start by constructing a simulator $\mathcal{S}$ such that, for any polynomial-time adversary $\mathcal{A}_{dv}$, if $\mathcal{A}_{dv}$ is able to distinguish between $\mathbf{Real}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv}}(k)$ and $\mathbf{Sim}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv},\mathcal{S}}(k)$ with non-negligible probability, then $\mathcal{A}_{dv}$ breaks the static security of ABE or Sym's indistinguishability against eavesdropping attacks. The simulator $\mathcal{S}$ can be constructed in this way:

Setup. Compute $(pp, pk, msk) \leftarrow Setup(1^k)$ and give pp to $\mathcal{A}_{dv}$.

Index. If no $\mathcal{L}_{COR}(uid)$ leakage occurs, create bit strings bs_{L_i} for each set of keywords occurring in $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$ leakage, such that the bit strings bs_{L_i} are distinct and of length k. If $\mathcal{L}_{COR}(uid)$ leakage occurs, set $bs_{L_i} = L_i$ for each keyword in L_i that occurs in $\mathcal{L}_{COR}(uid)$. Remember the mapping from $id(L_i)$ to bs_{L_i}.

For each set of keywords–access structure pair $(L_i, \mathbb{A})$ that occurs in $\mathcal{L}_{COR}(uid)$ or implicitly in $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$, create an encrypted list $\mathtt{DL}[L_i, \mathbb{A}]$, using the same symmetric key through the variation of L_i. Add to this list references to document ciphertexts from $\mathtt{D}$ such that the access structure of the corresponding plaintext document is $\mathbb{A}$ and the document contains L_i according to $\mathcal{L}_{COR}(uid)$ or $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$. Randomly map the list of $\mathtt{DL}$ nodes in an array $\mathtt{D}'$ of size $|\mathtt{D}|$, and encrypt all nodes. Remember the list head's locations and key. Fill empty array cells $\mathtt{D}'$ with dummy entries.

For each of the list heads in $\mathtt{D}'$, consider the set of keywords describing the document, L_i. For each element in the power set of L_i, except the empty set, that occurs in $\mathcal{L}_{COR}$ or implicitly in $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i, \{P_j\}_{j\in J_i}\}_{i\in\{1,\ldots,k\}})$, create a node in $\mathtt{RL}[L_i, \mathbb{A}]$. Using jolly pointers, map all the new nodes in $\mathtt{RL}[L_i, \mathbb{A}]$ to the corresponding list head in $\mathtt{D}'$. Randomly map the list of $\mathtt{RL}$ nodes in an array $\mathtt{R}'$ of size $|\mathtt{R}|$, and encrypt all nodes. Remember the list head's locations and key. Fill empty array cells $\mathtt{R}'$ with dummy entries.

For each of the list heads in $\mathtt{D}'$, consider the set of keywords describing the document, L_i, that occurs in $\mathcal{L}_{COR}(uid)$ or implicitly in $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$. Generate an encrypted linked list $\mathtt{KL}[\mathbb{A}]$ and utilize the jolly pointers to map the list heads' to all the nodes in $\mathtt{RL}$ that contained an element of the power set of L_i beside $\emptyset$. Randomly map the list of $\mathtt{KL}$ nodes in an array $\mathtt{K}'$ of size $|\mathtt{K}|$, and encrypt all nodes. Remember the list head's locations and key. Fill empty array cells $\mathtt{K}'$ with dummy entries.

For each $\mathbb{A}$ linked to the set of keywords L_i by the leakage functions, or implicitly publicly known, generate a node in $\mathtt{AL}$ and map them to the corresponding entries of $\mathtt{K}'$. Then randomly map the list of $\mathtt{AL}$ nodes in an array $\mathtt{A}'$ that should have the exact size $|\mathtt{A}|$, and encrypt all nodes.

Set $Index' = (\mathtt{A}', \mathtt{K}', \mathtt{R}', \mathtt{D}')$.

Ciphertext collection: For each plaintext document doc contained in any $\mathcal{L}_{COR}(uid)$, encrypt doc under policy $\mathbb{A}^+_{doc}$. For each document doc that occurs in $\mathcal{L}_{UP}(DC)$ but not in any $\mathcal{L}_{COR}(uid)$, encrypt $0^{|doc|}$ under policy $\mathbb{A}^+_{doc}$. Let $\mathtt{D}'$ be the set of ciphertexts generated from documents or based on document lengths.

Keys: For each distinct $(uid, \mathcal{A}_{uid})$ in $\mathcal{L}_{COR}(uid)$ or in $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$, compute the user secret key

$uk_{uid} \leftarrow$ MK-SEAC.$KeyGen(pp, msk, uid, Attr_{uid})$.

Trapdoors: For each $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$ compute $(uid, L_i) \leftarrow$ MK-SEAC.$Trpdr(pp, uk_{uid}, bs_{L_i})$, where uk_{uid} was created before and bs_{L_i} is the same as in the index generation.

Output: The simulator $\mathcal{S}$ outputs $(Index')$, the set of user secret keys generated due to $\mathcal{L}_{COR}(uid)$ and the set of trapdoors generated due to $\mathcal{L}_{SE}(uid, \mathcal{A}, \{\mathbb{A}_i, L_i\}_{i\in\{1,\ldots,k\}})$.

Hybrids. We denote game $\mathbf{Real}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv}}(k)$ by Hyb_0, meaning a hybrid game, partially taken over by the simulator. Starting in hybrid Hyb_1, the simulator $\mathcal{S}$ gets to compute pp. From Hyb_2 onward, we have $\mathcal{S}$ compute corrupted user's secret keys. Similarly, starting with Hyb_3, $\mathcal{S}$ produces the trapdoors requested by honest users. Computations of $\mathtt{D}'$ are performed by $\mathcal{S}$ from Hyb_4 onward. Starting with Hyb_5, $\mathcal{S}$ gets to compute the references to lists $\mathtt{DL}[L_i, \mathbb{A}]$ for $\mathtt{RL}[L_i, \mathbb{A}]$. $\mathcal{S}$ takes over the computation of $\mathtt{R}$ from Hyb_6 onward. From Hyb_7 onward, $\mathcal{S}$ takes over the computation of the references to lists $\mathtt{RL}[L_i, \mathbb{A}]$ for $\mathtt{KL}[\mathbb{A}]$. The remaining computations of $\mathtt{K}$ are performed by $\mathcal{S}$ from Hyb_8 onward. From Hyb_9 onward, $\mathcal{S}$ computes the references to list $\mathtt{AL}$ for $\mathtt{KL}[\mathbb{A}]$. Finally, from Hyb_{10} onward, $\mathcal{S}$ takes over the entire computation of $\mathtt{A}$. Hence, hybrid Hyb_{10} resembles experiment $\mathbf{Sim}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv},\mathcal{S}}(k)$.

Indistinguishability of Hybrids. We first note that $\mathcal{A}_{dv}$ does not know the key k_U for the pseudo-random function $f_{k_U}()$ unless $\mathcal{A}_{dv}$ corrupts some user. In case $\mathcal{A}_{dv}$ corrupts a user, $\mathcal{S}$ knows all keywords occurring in DB from the $\mathcal{L}_{COR}$ leakage. Then, $bs_{L_i} = L_i$ for every leaked set of keywords L_i. In case $\mathcal{A}_{dv}$ does not corrupt a user, it cannot evaluate function $f_{k_U}()$ on its own. Nevertheless, the unique choice and consequent use of bs_{L_i} ensures the consistency of the simulator's answers to queries.

Hybrids Hyb_0 and Hyb_1 cannot be distinguished because the real experiment and $\mathcal{S}$ perform the same computations. For the indistinguishability of Hyb_1 and Hyb_2, we note that if no user is corrupted, i.e. no $\mathcal{L}_{COR}$ leakage occurs, the computations of both the real experiment and $\mathcal{S}$ are non-existent in this step. Otherwise, the simulator executes the same algorithm on the same inputs as the real experiment. In either case, Hyb_1 is indistinguishable from Hyb_2. Hybrids Hyb_2 and Hyb_3 are indistinguishable because the simulator in Hyb_3 performs the same computations as the real experiment in Hyb_2.

$\mathcal{A}_{dv}$ cannot distinguish $\mathtt{D}$ from $\mathtt{D}'$, and thus Hyb_4 from Hyb_3, because, both *ABE* and *Sym* are IND-CPA. Some documents from DB may be associated with policies that are satisfied by the attributes of some corrupt user. Such documents are directly encrypted in $\mathtt{D}$ as well as $\mathtt{D}'$ by utilizing hybrid encryption. For all other documents, their respective ciphertexts in $\mathtt{D}$ are computationally indistinguishable from the encryptions of 0-bit strings of the relevant length that take their place in $\mathtt{D}'$.

The computational indistinguishability of Hyb_5 from Hyb_4 follows, since Hyb_5 performs the same computations as the real experiment in Hyb_4.

The computational indistinguishability of Hyb_6 from Hyb_5 follows considering that *Sym* is IND-CPA. If the adversary corrupts a user, there is no difference between the real experiment's remaining computations of $\mathtt{R}$ in Hyb_5 and the simulator's corresponding computation of $\mathtt{R}'$ in Hyb_6. In case the adversary does not corrupt any user, the arguments for the indistinguishability of Hyb_3 and Hyb_4 analogously hold for the indistinguishability of hybrids Hyb_5 and Hyb_6.

Hybrids Hyb_7 and Hyb_6 are indistinguishable because the simulator in Hyb_7 performs the same computations as the real experiment in Hyb_6.

Hybrid Hyb_8 is computationally indistinguishable from Hyb_7 since *Sym* is IND-CPA. Additionally, $f_{k_U}()$ is a pseudo-random function and the bit-wise XOR of a bit-string x with the image of a pseudo-random function is a CPA-secure symmetric encryption scheme. Therefore, dummy entries cannot be distinguished from non-dummy entries in $\mathtt{K}$ and $\mathtt{K}'$, unless keys for such entries, images of $f_{k_U}()$, are known. $\mathcal{A}_{dv}$ can obtain such images either by computing them using the key of a corrupted user or by extracting the image from a trapdoor. The former option requires $\mathcal{A}_{dv}$ to corrupt a user (except with negligible probability), in which case $\mathcal{A}_{dv}$ can distinguish all dummy entries from non-dummy entries for both $\mathtt{K}$ and $\mathtt{K}'$ and the simulation ensures keywords underlying the labels and decryption keys are exactly the same in both cases. In case the adversary does not corrupt a user, $\mathcal{A}_{dv}$ lacks the keys to $f_{k_U}()$, so it cannot check that the image it holds is an image of the keyword known to $\mathcal{A}_{dv}$ but not to $\mathcal{S}$.

For the indistinguishability of hybrids Hyb_8 and Hyb_9, we observe that the adversary $\mathcal{A}_{dv}$ performs the same computations as the real experiment in Hyb_8.

The computational indistinguishability of Hyb_{10} from Hyb_9 follows considering that both *ABE* and *Sym* are IND-CPA. If the adversary corrupts a user, there is no difference between the real experiment's remaining computations of $\mathtt{A}$ in Hyb_9 and the simulator's corresponding computation of $\mathtt{A}'$ in Hyb_{10}. In case the adversary does not corrupt any user, the arguments for the indistinguishability of Hyb_3 and Hyb_4

analogously hold for the indistinguishability of hybrids Hyb_9 and Hyb_{10}.

In conclusion, all our hybrids are computationally indistinguishable. Thus, for sufficiently large k, no probabilistic polynomial time adversary $\mathcal{A}_{dv}$ can distinguish between experiments $\mathbf{Real}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv}}(k)$ by Hyb_0 and $\mathbf{Sim}^{\text{static}}_{MK-SEAC,\mathcal{A}_{dv},\mathcal{S}}(k)$ with probability non-negligible in k.

□

B Efficiency of Query Processing

In this section we report the cardinality of the structures composing the index and the number of cryptographic operations that the DO, the DU and the CP have to compute in order to upload a batch of documents and process a query operation respectively.
Let us introduce the following notation:

$$\mathcal{A}_{\text{DB}} = \{\mathbb{A}_{doc} \ : \ doc \in DB\},$$
$$\mathcal{AL}_{\text{DB}} = \{(\mathbb{A}_{doc}, L_{doc}) \ : \ doc \in DB\}$$
$$m_t = \sum_{i=1}^{k} |\ \{(\mathbb{A}_i, L) \in \mathcal{AL}_{\text{DB}}\}\ |,$$
$$m_d = |\{doc \in \text{DB} : \exists\ i \text{ s.t. } (\mathbb{A}_{doc}, L_{doc}) = (\mathbb{A}_i, L_i)\}|.$$

Then, for what concerns the cardinality of the structures in the index, we have:

Proposition B.1. *Let $\mathcal{A}_{\text{DB}}$ and $\mathcal{AL}_{\text{DB}}$ be defined as above, then the structures of the index of MK-SEAC have the following cardinalities:*

1. $|\ \mathtt{A}\ | = |\ \mathcal{A}_{\text{DB}}\ |$,
2. $|\ \mathtt{K}\ | = |\ \mathcal{AL}_{\text{DB}}\ |$,
3. $|\ \mathtt{D}\ | = |\ DB\ |$,
4. $|\ \Sigma\ | = |\ \mathcal{A}_{\text{DB}}\ |$,
5. $|\ \mathtt{R}\ | = \sum_{(L,\mathbb{A}) \in \mathcal{AL}_{\text{DB}}} \left(2^{|L|} - 1\right)$.

Proving this proposition should be straightforward. Indeed, as previously noted, the pairs of policies and maximal set of keywords induce a partition in the set of documents.

Proposition B.2. *The number of cryptographic operations required by a DO to update a batch are:*

$$(|\ \mathtt{A}\ | + |\ \mathcal{AL}_{\text{DB}}\ | + |\ \Sigma\ |) \cdot \text{E}_{ABE} +$$
$$+ (|\ \mathtt{A}\ | + |\ \mathtt{K}\ | + |\ \mathtt{R}\ | + 2\,|\ \mathtt{D}\ | + |\ DB\ | + |\ \Sigma\ |) \cdot \text{E}_{Sym} +$$
$$+ (|\ \Sigma\ |) \cdot \text{S}_g.$$

For the CP to process a query request are needed:

$$1 \cdot \text{S}_v + k \cdot \text{D}_{ABE} + (k + 2m_t + m_d) \cdot \text{D}_{Sym}$$

operations, while, for a DU to complete a search, we require:

$$1 \cdot S_g + (1+k) \cdot \text{S}_v + (|\ \mathcal{AL}_{\text{DB}}\ | + k) \cdot \text{D}_{ABE} +$$
$$+ (m_d + k) \cdot \text{D}_{Sym}$$

operations, where S_g and S_v are respectively the number of signature generations and verifications, D_{Sym} is the number of symmetric decryptions and D_{ABE} is the number of ABE decryptions. While E_{ABE} and E_{Sym} are the numbers of ABE and symmetric encryptions.

Proof. In updating a batch, the DO has to build the index and encrypt it. Note that each node in A and in Σ requires an ABE encryption (see Equations (2) and (9))[2]. Moreover, the structure D contains the ABE encryption of the keys required to retrieve the documents. Documents that share the same keywords and policy also share the same key (see Equation (7)), thus the DO has to compute an ABE encryption for each pair of policy and set of keywords occurring in the batch, that is $|\ \mathcal{AL}_{\text{DB}}\ |$.

Then, the DO has to symmetrically encrypt the remaining structures. All structures require a symmetric encryption per node, besides D, which requires two encryptions. The batch itself is symmetrically encrypted. Finally, it has to produce a signature for each node in Σ.

When processing a query, the CP has to check the signature of the issuer. Then, it decrypts with ABE the entries of A corresponding to the k policies in the query. To visit the inner layers of the index, it has to symmetrically decrypt the k identifiers $\text{Sym.}Enc_{\text{KDF}(r_{\mathbb{A}})}\left(id^{K}_{L_{\mathbb{A},1}}\right)$ in A to access the corresponding lists in K. Then, it decrypts the content of such lists in K, for a total of m_t decryptions. For each of these nodes, it tries to decrypt the pointed node in R, which requires m_t additional symmetric decryptions. Whenever this decryption succeeds, it then decrypts the corresponding nodes in D, for a total of m_d nodes.

Finally, the DU has to sign its query. Then, upon receiving the results, it has to check the signatures of the CP and one signature for each of the k policies occurring in the query. Then, it has to decrypt in ABE the first entry of the Σ nodes, which are again k, and the final entries of the D nodes that the CP visited during the search, that is, $\mathtt{abe}_{L_{\mathbb{A},i},j}$. Finally, it has to decrypt a symmetric ciphertext for each node in Σ - to retrieve the hash of the concatenations of the document tags - and each document received. □

[2]Since we are using hybrid encryption, note that each ABE (en/de)cryption requires a symmetric (en/de)cryption, which are already included in Proposition B.2.