Searchable Encryption with Access Control[*]

Nils Löken[†]
Paderborn University

Abstract

Outsourcing data to the cloud is becoming increasingly prevalent. To ensure data confidentiality, encrypting the data before outsourcing it is advised. While encryption protects the secrets in the data, it also prevents operations on the data. For example in a multi-user setting, data is often accessed via search, but encryption prevents search. Searchable encryption solves this dilemma. However, in a multi-user setting not all users may be allowed to access all data, requiring some means of access control. We address the question how searchable encryption and access control can be combined. Combining these technologies is required to achieve strong notions of confidentiality: if a ciphertext occurs as a search result, we learn something about the underlying document, even if access control does not let us access the document. This illustrates a need to link search and access control, so that search results presented to users only feature data the users are allowed to access. Our searchable encryption scheme with access control establishes that link.

1 Introduction

Searchable Encryption [23] enables data to be securely outsourced to the cloud while maintaining the ability to search the data efficiently. If the data is outsourced by a company, multiple users need to be considered. Typically not all are granted access to all the company's files. Hence, there is a need for access control. Access control can be applied independently of searchable encryption. Indeed, such schemes have been proposed [28], but naturally, they require filtering of data according to the user's access rights, which potentially incurs significant information leakage to the party performing the filtering. We aim at integrating searchable encryption and access control to achieve better data protection.

Imagine Bob, an accountant, to search for "Kryptonite," and search yields hundreds of recently modified files. Even if access control prevents Bob from accessing the files, Bob can deduce what his company's R&D department is researching. That information should be hidden from Bob! Filtering the result with respect to access

[*]This work was partially supported by the Federal Ministry of Education and Research (BMBF) within the collaborate research project Securing the Financial Cloud (SFC), grant 16KIS0058K, and the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly-Computing (SFB 901).

[†]nils.loeken@uni-paderborn.de

control might reduce the amount of knowledge that Bob gains. However, the party responsible for filtering still gets to see the original result to Bob's query, so the information that Bob is not supposed to obtain is obtained by some other party instead.

Our goal is to eliminate this leakage by integrating searchable encryption and access control such that the server performing search on Bob's behalf only outputs files that Bob can access and that contain Bob's search term. Moreover, even the server cannot learn which of the files not accessible to Bob contain Bob's search term. We call this paradigm *searchable encryption with access control.*

Related Work Based on the seminal work of Song et al. [23], various flavors of searchable encryption have been developed, providing searchable encryption in different settings. *Searchable symmetric encryption* (SSE), influenced by Curtmola et al. [10, 9], originally provides searchable encryption to single users. SSE [15, 6, 5, 18] satisfies different non-equivalent security notions [12, 7, 10, 9], including UC-security [16]. Generic techniques to achieve multi-user SSE are (proxy) re-encryption [26] and broadcast encryption [10]. SSE has been combined with oblivious RAM [24, 20], private information retrieval [13] and blind storage [21] to limit what servers or data owners can learn from participating in search.

Public key encryption with keyword search (PEKS) [2] provides searchable encryption in settings with multiple data creators and a single recipient, such as e-mail. Using proxy re-encryption, PEKS allows for multiple recipients [11]. Due to the public key setting, the security notion for PEKS is rather weak: the server performing search can create a searchable ciphertext on its own and apply old search requests to it, revealing what keywords have been searched for. Another drawback with PEKS is that typically search requires time linear in the number of document–keyword pairs which is inefficient—the optimal time is linear in the size of the result set.

Concerning *multiple recipients and access control*, several schemes have been proposed, often separating searchable encryption and access control, relying on third parties for filtering search results [14] or formulating search queries [19, 17]. Recently, *attribute-based encryption with keyword search* [29, 25] has been suggested, using *attribute-based encryption* (ABE) to achieve access control.

For a comprehensive survey of searchable encryption in its various flavors, see [4]. Also, see [27] for several generic attacks on searchable encryption based on document collection dynamics.

Our contribution We present *searchable encryption with access control.* That is, the searchable data's access policy determines who is allowed to search the data. We consider a *single data owner* outsourcing a *static document collection* to a server, making the collection searchable to *many users* with different access rights.

The server and the users are considered together when it comes to security. We want an adversarial server to learn as little as possible about the searchable document collection. This must hold, even if the adversary corrupts some users. We capture this in a *leakage-based security definition* in the spirit of semantic security. Our security notion covers different threats—document confidentiality, keyword secrecy, and security against chosen keyword attacks—that are typically considered separately. However, we assume the server to answer honest users' search queries correctly.

We give a *generic construction* of searchable encryption with access control that is secure under our definition. Our construction is based on *multi-authority attribute-based encryption* with *authority key customization*. Due to authority key customization, *search query formulation requires no interaction*. We use techniques from searchable symmetric encryption, resulting in a very *efficient* search process: during search, we do not check all document–keyword pairs. Finally, we show that our construction can be *realized* based on the Rouselakis/Waters multi-authority ABE scheme [22].

Comparison to other schemes The literature offers four schemes that are of particular interest for a comparison with our construction. The scheme of Kaci et al. [14] realizes searchable encryption with access control based on the SSE-1 scheme of Curtmola et al. [10], much like our construction. However, their scheme heavily relies on trusted third parties and interaction. Particularly, in their scheme users need assistance from a third party when formulating search queries. Another third party is employed to filter search results based on access rights, for which the party requires the user's keys. We drop the need for interaction for query formulation, and get rid of explicit filtering, as our index structure used for search only allows the server to output search results matching the user's access rights. In [14] the server performing search is not presented as a potential threat in any way. However, the setup of the scheme implies that the server is assumed to be honest-but-curious.

Alderman et al. [1] also present a scheme based on the SSE-1 construction [10]. Their scheme is capable of serving multiple users and integrates access control, while only using symmetric primitives. Their proposal is restricted to access policies where the set of access rights is totally ordered. This results in smaller runtime of their scheme in comparison to ours. However, we realize more expressive policies, which Alderman et al. stated as an open problem.

Sun et al. [25] realize searchable encryption with access control in a setting with multiple data owners, as well as multiple users. Due to allowing multiple data owners, the scheme inherently allows data to be dynamically added to the document collection. In the scheme, search is based on the PEKS-approach. Their construction features user revocation via re-encryption of indexes and data. For security, Sun et al. assume the server to be honest, i. e. it answers queries correctly, but curious, i. e. it tries to learn as much as possible about the document collection. On the other hand, users are assumed to be malicious and to collude to access data that they are not allowed to access. This model strikes us as odd: it is hard to argue that an honest-but-curious entity rejects the offer to gain additional information (here: user keys), if taking advantage of the offer cannot be detected. In particular, we think that the server should be allowed to collude with users. But then, the means of revocation proposed by Sun et al. fails, because it relies on the server's complete cooperation. If the server does not cooperate, which cannot be detected, user revocation is useless. All in all, the scheme from [25] is more advanced than ours in some respects (multi-owner, dynamic addition of documents), but lacks our construction's security against more reasonable adversaries as well as its efficiency.

The fourth scheme for our comparison is by Zheng et al. [29]. A major difference to us is in how they model access to search. Particularly, they directly associate keywords with policies, thus restricting what keywords a particular user is allowed to search for. Hence, their scheme does not implement searchable encryption with access control in the sense of our definition. In their scheme, the search results presented to users are independent of access policies of the data contained in the search result. In terms of our example from the introduction, they can prevent Bob from searching for "Kryptonite," but if there is good reason for Bob to be allowed to search for it, then Bob will also get to see the problematic documents from the R&D department that we do not want him to see. For Zheng et al., this is not a problem, since only servers are seen as a threat, whereas users may be honest-but-curious and do not collude. The scheme from [29] relies on verifiability of search results to ensure that the server returns complete search results. In our model, the server is assumed to return complete results. The construction from [29] can be modified such that users obtain search results that only feature data accessible to the user, but then our scheme is more efficient, due to its underlying data structures.

Paper organization In Section 2, we present multi-authority ciphertext-policy attribute-based encryption and searchable encryption with access control. Section 3 presents authority key customization. In Section 4 we provide our searchable encryption scheme with access control. We conclude our paper in Section 5.

2 Preliminaries

In this section we introduce our notation regarding attribute-based encryption, describe multi-authority ciphertext-policy attribute-based encryption, and define searchable encryption with access control. We also provide security definitions for these primitives.

2.1 Policies, attributes and keys

A file's access policy—or *access structure*—describes who is allowed to access that file. The description is a Boolean formula over certain attributes a user must have in order to get access. For example, attributes can state that the user has reached a certain age or holds a particular position in a company. We use the & operator to denote the conjunction of access structures.

In Section 2.2, we present attribute-based encryption (ABE). In such schemes, users hold attributes in the form of cryptographic keys that are given to users by an attribute authority. We assume that we can break down a user's key into smaller sub-keys consisting only of single attributes. The key given to user uid specific to attribute u is denoted $uk_{uid,u}$ and we write $uk_{uid,Attr_{uid}}$ to denote all attribute keys given to user uid, while $Attr_{uid}$ denotes uid's attribute set. We often use attributes and the respective keys interchangeably.

In the multi-authority setting of ABE that we use as a technique, multiple attribute authorities exist. By $\mathrm{Auth}(u)$ we denote the authority that manages attribute u. Different authorities may hand out attributes that have identical names.

We thus prepend attributes with the name of the authority that manages the particular attribute, e. g. “AA:CEO” denotes the attribute CEO managed by authority AA.

2.2 Attribute-based encryption

We now present a definition of multi-authority attribute-based encryption [8] in the variant that we use to construct a searchable repository of ciphertexts that provides access control.

Definition 1. *A* multi-authority ciphertext-policy attribute-based encryption (MA-CP-ABE) scheme *consists of the following probabilistic polynomial time algorithms [22]:*

GlobalSetup *takes security parameter 1^κ, and outputs public parameters pp*

AuthSetup *takes pp and authority identifier θ, and outputs public key pk_θ and authority secret key mk_θ*

KeyGen *takes pp, mk_θ, user identifier uid and attribute identifier u, and outputs attribute-specific user key $uk_{uid,u}$*

Enc *takes pp, set $\{pk\}$ of authorities' public keys, access structure $\mathbb{A}$ and message msg, and outputs ciphertext ct*

Dec *takes pp, set $uk_{uid,Attr_{uid}}$ of attribute-specific user keys and ct, and outputs message msg*

We require for all correctly set up systems and users with message msg, ciphertext $ct \leftarrow \mathsf{Enc}(pp, \{pk\}, \mathbb{A}, msg)$ and user key $uk_{uid,Attr_{uid}}$, if attribute set $Attr_{uid}$ satisfies access structure $\mathbb{A}$ then $\Pr[\mathsf{Dec}(pp, uk_{uid,Attr_{uid}}, ct) = msg] = 1$.

Note that KeyGen produces a key that only holds one attribute, but can trivially be extended to operate on attribute sets. The user uid's key is the set containing keys for all her attributes $Attr_{uid}$.

Security of MA-CP-ABE We now consider the security for multi-authority ciphertext-policy attribute-based encryption. We present here a game-based definition due to Rouselakis and Waters [22]. It considers a static adversary, i. e. the adversary has to make all its queries at the same time. In the game, the adversary can choose the participating attribute authorities, of which the adversary statically corrupts some. The adversary can query for user keys, but the queried keys can only contain attributes managed by non-corrupt authorities; the adversary can compute the other attributes by itself. The adversary chooses two messages of equal length and an access structure and has to distinguish which message was encrypted under the chosen access structure without holding a user key able to decrypt, i. e. the challenge access structure cannot be satisfied by attributes managed by corrupted authorities alone, or by adding such attributes to queried user keys.

Particularly, let Π be an MA-CP-ABE scheme Π. Consider the following experiment $\mathbf{MultABE}^{\text{static}}_{\Pi,\mathcal{A}}(\kappa)$ between a challenger and an adversary $\mathcal{A}$:

Trusted setup Compute $pp \leftarrow \mathsf{GlobalSetup}(1^\kappa)$ and give pp to $\mathcal{A}$.

Queries $\mathcal{A}$ computes and outputs

- set $\{\theta\}$ of honest authorities' identifiers,
- set $\{pk_\theta\}$ of corrupt authorities' public keys,
- sequence $Q_K = \{(uid_i, Attr_{uid_i})\}_{i=1}^{m_K}$ of queries for user keys for user identifiers uid_i and attribute sets $Attr_{uid_i}$, where no uid_i can occur in more than one query and every attribute in $Attr_{uid_i}$ is managed by an honest authority,
- two messages msg_0, msg_1 of equal length and an access structure $\mathbb{A}$ such that $\mathbb{A}$ is not satisfied by any attribute set $U \cup Attr_{uid_i}$, where $Attr_{uid_i}$ occurs in a KeyGen query and U is the set of attributes managed by the corrupt authorities.

Replies Pick $b \leftarrow_\$ \{0,1\}$ and reply with

- public key pk_θ from $(pk_\theta, mk_\theta) \leftarrow \mathsf{AuthSetup}(pp, \theta)$ for every honest authority θ output by $\mathcal{A}$,
- user keys $uk_{uid} \leftarrow \mathsf{KeyGen}(pp, mk_{\mathrm{Auth}(u)}, uid, Attr_{uid})\}$ for all queried user identifiers and attribute sets $(uid, Attr_{uid}) \in Q_K$,
- challenge ciphertext $ct \leftarrow \mathsf{Enc}(pp, \{pk\}, \mathbb{A}, msg_b)$ generated using the public keys of all honest and corrupt authorities.

Guess $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The outcome of the experiment is 1 if $b = b'$ and 0 otherwise.

Definition 2. *An MA-CP-ABE scheme* Π *is* statically secure *if for sufficiently large* κ *and all probabilistic polynomial time adversaries* $\mathcal{A}$ $\Pr[\boldsymbol{MultABE}^{static}_{\Pi,\mathcal{A}}(\kappa) = 1]$ *is negligible, where the probability is taken over the random choices of the adversary and the experiment.*

2.3 Searchable encryption with access control

We now present a primitive for searchable encryption with access control. Our definition captures scenarios where a single data owner makes a static document collection available to a larger group of registered users. Not all users can access and search all documents, so access control is required. Nevertheless, we assume that all registered users know which keywords are present in the document collection, just not in which documents. The document collection is expected to be stored on some publicly accessible server, e. g. in the cloud. Our discussion of searchable encryption with access control includes a security definition, in which a (static) adversary controls the storage server and can corrupt arbitrary users.

Definition 3. *A* searchable encryption scheme with access control *consists of six probabilistic polynomial time algorithms:*

Setup *takes security parameter 1^κ, and outputs public parameters pp, master secret mk and owner key ok*

KeyGen *takes pp, mk, user identifier uid and attribute set $Attr_{uid}$, and outputs user secret uk_{uid}*

Enc *takes pp, ok and document collection DC, and outputs index structure $Index$ and chiphertext set CT*

Trpdr *takes pp, uk_{uid} and keyword kw, and outputs search trapdoor $t_{uid,kw}$*

Search *takes pp, $Index$ and $t_{uid,kw}$, and outputs result $X \subseteq CT$*

Dec *takes pp, uk_{uid} and ciphertext ct, and outputs document doc*

For all correctly set up systems and search results $X \leftarrow \mathsf{Search}(pp, Index, t_{uid,kw})$, we require for each $doc \in DC$:

- *$kw \notin doc$, or*
- *$kw \in doc$ and $Attr_{uid}$ does not satisfy doc's access structure $\mathbb{A}(doc)$, or*
- *there is $ct \in X$ such that $\mathsf{Dec}(pp, uk_{uid}, ct) = doc$.*

The correctness property ensures *completeness* of search results: each document is either present in the result (via its ciphertext) or is excluded from the result set due to not containing the searched keyword or the user not being allowed to access the document.

For security, we consider an adversary with full control over the server, that can additionally corrupt users. Our goal is to minimize what such adversaries can learn. What can be learned from our system is expressed using stateful leakage functions. In our security notion, we use leakage as input to a simulator, such that no probabilistic polynomial time adversary can distinguish whether it interacts with the real world, or with the simulator.

We break down the leakage given to the simulator to the different interactions in the system, i. e., deploying the encrypted document collection, corrupting a user and performing search. The respective leakage functions share a common state to capture that interactions may not be independent. Notationwise, the statefulness is implicit throughout the paper. The concrete leakage functions for our construction are given in the analysis of our scheme.

Consider the following experiments with a searchable encryption scheme with access control Π, an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$, respectively.

$\mathbf{Real}^{\text{static}}_{\Pi,\mathcal{A}}(\kappa)$

1. Setup: run $(pp, mk, ok) \leftarrow \mathsf{Setup}(1^\kappa)$ and give pp to $\mathcal{A}$.
2. Queries: $\mathcal{A}$ outputs
   - document collection DC,
   - sequence $Q_U = \{(uid_i, Attr_i)\}_{i=1}^{m_U}$ of user creation queries, where no uid can occur more than once,

- sequence $Q_C = \{uid_i\}_{i=1}^{m_C}$ of user corruption queries, where each uid must also occur in a user creation query and no user can be corrupted more than once,
- sequence $Q_T = \{(uid_i, kw_i)\}_{i=1}^{m_T}$ of trapdoor queries, where each uid must also occur in a user creation query and no uid can refer to a corrupt user.

3. Replies: compute $(Index, CT) \leftarrow \mathsf{Enc}(pp, ok, DC)$, private keys for the created users using the uids and attribute sets from Q_U, and the requested honest user's trapdoors; give $(Index, CT)$, the trapdoors and corrupted user's keys to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ outputs a bit b. The experiment outputs b.

$\mathbf{Sim}^{\text{static}}_{\Pi,\mathcal{A},\mathcal{S}}(\kappa)$

1. Setup: $\mathcal{S}$ gives pp to $\mathcal{A}$.
2. Queries: $\mathcal{A}$ outputs a document collection DC and sequences of queries Q_U, Q_C, Q_T as before.
3. Replies: given setup leakage $\mathcal{L}_1(DC)$, user corruption leakage $\mathcal{L}_2(uid)$ and query leakage $\mathcal{L}_3(uid, kw)$, $\mathcal{S}$ computes $(Index, CT)$, the honest user's trapdoors and the corrupted user's keys; $\mathcal{S}$ gives $(Index, CT)$ and query responses to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ outputs a bit b. The experiment outputs b.

Definition 4. *A searchable encryption scheme with access control* Π *is* $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$-semantically secure against static adversaries, *if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *there is a probabilistic polynomial time simulator* $\mathcal{S}$ *such that* $|\Pr[\boldsymbol{Real}^{static}_{\Pi,\mathcal{A}}(\kappa) = 1] - \Pr[\boldsymbol{Sim}^{static}_{\Pi,\mathcal{A},\mathcal{S}}(\kappa) = 1]|$ *is negligible, where the probabilities are taken over the random bits of* $\mathcal{A}$*,* $\mathcal{S}$ *and the experiments.*

3 Authority key customization

Our searchable encryption scheme with access control that we present in the next section users multi-authority ciphertext-policy attribute-based encryption as a building block. For users to be able to formulate search queries, they must hold an attribute for every keyword that exists in a document collection. However, we neither want search queries to be formulated in an interactive process with an attribute authority, nor do we want long user keys.

Our notion of authority key customization for MA-CP-ABE solves the dilemma. Authority key customization restricts an attribute authority's secret key in such a way that it can be used to produce user keys for all the attributes managed by that authority, but only for a single user. *In our scenario* customized authority keys for a particular authority are given to users, so they can produce the required keyword-specific attributes themselves, without being able to produce arbitrary keys for themselves or other users.

Definition 5. *An MA-CP-ABE scheme provides* authority key customization *via two probabilistic polynomial time algorithms*

Customize *takes pp, authority secret key mk_θ and user identifier uid, and outputs customized authority key $sk_{\theta,uid}$*

CustKeyGen *takes pp, $sk_{\theta,uid}$ and attribute identifier u, and outputs attribute-specific user key $uk_{uid,u}$*

We require keys derived via CustKeyGen *to be functionally equivalent to keys derived via* KeyGen.

Security We consider authority key customization to be *secure* if it is computationally infeasible to compute an attribute $uk_{uid',u}$ from a polynomially large set of customized authority secrets $\{sk_{\theta,uid_i}\}$ with $uid_i \neq uid'$ for all i. For a formal treatment, consider the following game $\mathbf{AuthCust}_{\Pi,\mathcal{A}}(\kappa)$:

Trusted Setup Give $pp \leftarrow \mathsf{GlobalSetup}(1^\kappa)$ to $\mathcal{A}$.

Authorities The adversary $\mathcal{A}$ outputs

- set $\{\theta\}$ of honest authorities' identifiers,
- set $\{pk_\theta\}$ of corrupt authorities' public keys.

Replies Compute (pk_θ, mk_θ) for every honest authority θ output by $\mathcal{A}$; give the computed public keys to $\mathcal{A}$.

Queries $\mathcal{A}$ adaptively queries customized authority secrets for honest authorities and user identifiers of its choice. The experiment replies with the queried secrets.

Output $\mathcal{A}$ outputs a user key uk_{uid}, and an access structure $\mathbb{A}$. The experiment outputs 1 if (1) uk_{uid} satisfies $\mathbb{A}$, (2) some attribute u from uk_{uid} is managed by an honest authority $\theta' = \text{Auth}(u)$, (3) $\mathcal{A}$ has never queried a customized authority secret for (θ', uid), (4) $uk_{uid} \setminus \{u\}$ does not satisfy $\mathbb{A}$ and (5) for every message msg $\mathsf{Dec}(pp, uk_{uid}, \mathsf{Enc}(pp, \{pk\}, \mathbb{A}, msg)) = msg$, where $\{pk\}$ includes the public keys of all authorities. Otherwise, the experiment outputs 0.

The conditions guarantee functional equivalence of the output key to KeyGen-derived keys, and ensure that $\mathcal{A}$ cannot win the experiment trivially.

Definition 6. *An MA-CP-ABE scheme* Π *with authority key customization provides* secure authority key customization *if for sufficiently large κ and all probabilistic polynomial time adversaries $\mathcal{A}$* $\Pr[\boldsymbol{AuthCust}_{\Pi,\mathcal{A}}(\kappa) = 1]$ *is negligible, where the probability is over the random bits of the experiment and the adversary.*

Notice that the security definition for MA-CP-ABE schemes does not consider authority key customization. However, we could adapt the definition of MA-CP-ABE static security to consider authority key customization, i. e. give the adversary access to customized authority secrets created by honest authorities. Then we need to require that the adversary did not query user secret keys or customized authority secrets such that the set of attributes contained in the queried user keys or managed by the corrupt authorities and the attributes managed by honest authorities for which customized authority secrets were queried satisfy the challenge access structure $\mathbb{A}$. Secure authority key customization then implies that a statically secure MA-CP-ABE scheme with authority key customization is also secure under this modified security notion, because customized authority secrets are no help in creating keys for users other than the one to whom the secret is customized.

Proof-of-concept Authority key customization can be added to the MA-CP-ABE scheme due to Rouselakis and Waters [22]. The scheme uses bilinear groups of prime order p with generator g and bilinear map e; it uses hash functions F, H that map bitstrings to the bilinear group. The attribute-specific key for attribute u is of the form $(K_{uid,u}, L_{uid,u})$ with $K_{uid,u} = g^{\alpha_\theta} H(uid)^{y_\theta} F(u)^t$ and $L_{uid,u} = g^t$, where t is chosen uniformly at random from $\mathbb{Z}_p$ and $(\alpha_\theta, y_\theta) \in \mathbb{Z}_p \times \mathbb{Z}_p$ are authority θ's master secret. Authority key customization works as follows:

Customize(pp, mk_θ, uid) Output $sk_{\theta,uid} = g^{\alpha_\theta} \cdot H(uid)^{y_\theta}$,

CustKeyGen$(pp, sk_{\theta,uid}, u)$ Pick $t \leftarrow_\$ \mathbb{Z}_p$, set $K_{uid,u} = sk_{\theta,uid} F(u)^t$ and $L_{uid,u} = g^t$, output $uk_{uid,u} = (K_{uid,u}, L_{uid,u})$.

It is easy to see that keys derived from the customized authority secret are functionally equivalent to KeyGen-derived user secrets. The security of this construction is proven in Appendix A.2.

4 Search with access control

Aiming at realizing a searchable encryption scheme with access control, we rely on an index data structure to perform search efficiently. Our scheme uses a data structure influenced by the data structure underlying the SSE-1 scheme of Curtmola et al. [10].

Intuition As in the SSE-1 scheme, we precompute all potential search results and store these results in *encrypted linked lists.* Such lists are symmetrically encrypted node by node, using a fresh key for each node. With each node, we store a pointer to its successor, as well as the successor's symmetric key. The nodes themselves are stored at random locations—determined upon node creation—in a memory array that leaves room for dummy entries. Dummy entries are symmetrically encrypted bit strings that are indistinguishable from not yet decrypted list nodes. The addresses and keys of list heads are stored separately.

For each keyword–access structure pair $kw, \mathbb{A}$ from a document collection, we create encrypted list $DL[kw, \mathbb{A}]$, that stores the partial result consisting of documents that contain keyword kw and have access structure $\mathbb{A}$. Note that the list only stores pointers to documents. For each keyword kw, we create encrypted list $AL[kw]$ that stores ABE-encrypted addresses and keys of lists $DL[kw, \mathbb{A}]$. The policy for the head of list $DL[kw, \mathbb{A}]$ is based on $\mathbb{A}$. The address and key of the head of list $AL[kw]$ is stored in hash table HT.

When a server executes search on users' behalf, it needs to decrypt the ABE ciphertexts stored in list $AL[kw]$. However, the server cannot be allowed to decrypt document ciphertexts. Therefore we use MA-CP-ABE to realize multiple functionalities (MA-CP-ABE authorities) that are controlled by a single authority in the sense of searchable encryption with access control. *The functionalities split attributes into three classes, based on attribute semantics.* Functionality Usr manages attributes that originally describe users, e. g. their roles. Functionality Sys manages attributes that determine users' interactions with the system. Particularly, we use attributes "Sys:dec" allowing file decryption, and "Sys:srch" allowing search. Functionality $Srch$ is used for parameters of the interaction; when searching for keyword kw, a key for attribute "$Srch$:$f_\mathrm{a}(kw)$" is derived via some function f_a. The function serves to hide kw from the server. We apply authority key customization to functionality $Srch$, so users themselves can derive the keyword-specific attributes.

Scheme Based in this intuition, we now construct SEAC, a searchable encryption scheme with access control. Besides an MA-CP-ABE scheme ABE with authority key customization and a symmetric encryption scheme Sym, our construction uses three pseudorandom functions f_l, f_k, f_a as its underlying primitives.

Setup(1^κ) Sample keys $k_\mathrm{l}, k_\mathrm{k}, k_\mathrm{a} \leftarrow_\$ \{0,1\}^\kappa$ for the pseudorandom functions, set up ABE via $pp' \leftarrow \mathsf{ABE.GlobalSetup}(1^\kappa)$ and the functionalities via $\{(pk_\theta, mk_\theta) \leftarrow \mathsf{ABE.AuthSetup}(pp', \theta)\}_{\theta \in \{Usr, Sys, Srch\}}$. Output (pp, mk, ok), where

- $pp = (pp', pk_{Usr}, pk_{Sys}, pk_{Srch})$,
- $mk = (mk_{Usr}, mk_{Sys}, mk_{Srch}, k_\mathrm{l}, k_\mathrm{k}, k_\mathrm{a})$,
- $ok = (k_\mathrm{l}, k_\mathrm{k}, k_\mathrm{a})$.

KeyGen$(pp, mk, uid, Attr)$ Ensure that all attributes in $Attr$ are managed by functionality Usr. Set $sk_{Srch,uid} \leftarrow \mathsf{ABE.Customize}(pp, mk_{Srch}, uid)$ and

$$\begin{aligned} uk'_{uid} \leftarrow & \{\mathsf{ABE.KeyGen}(pp', mk_{Usr}, Attr)\} \\ & \cup \{\mathsf{ABE.KeyGen}(pp', mk_{Sys}, \{\text{“}Sys\text{:dec”}, \text{“}Sys\text{:srch”}\})\}. \end{aligned}$$

Output $uk_{uid} = (uk'_{uid}, sk_{Srch,uid}, k_\mathrm{l}, k_\mathrm{k}, k_\mathrm{a})$.

Enc(pp, ok, DC) For each document doc from DC with access structure $\mathbb{A}(doc)$, create document ciphertext

$$ct_{doc} \leftarrow \mathsf{ABE.Enc}(pp', \{pk_{Usr}, pk_{Sys}\}, \mathbb{A}(doc)\&\text{“}Sys\text{:dec”}, doc).$$

Let CT be the set of all generated document ciphertexts.

Create encrypted linked lists $DL[kw, \mathbb{A}]$ as outlined in the intuition. Note that these lists store pointers to CT rather than documents. Let $\mathtt{D}$ be the memory array of appropriate size that stores the list nodes. Let $\langle p_{kw,\mathbb{A}}, k_{kw,\mathbb{A}}\rangle$ be the address and key of the head of list $DL[kw, \mathbb{A}]$. ABE encrypt $\langle p_{kw,\mathbb{A}}, k_{kw,\mathbb{A}}\rangle$ under policy $\mathbb{A}$&"Sys:srch"&"Srch:$f_{\mathrm{a}}(kw)$" and store the ciphertext in encrypted list $AL[kw]$. Let $\mathtt{A}$ be the memory array of appropriate size that stores the list nodes. Before symmetric encryption of the lists, all ABE ciphertexts and dummy entires are padded to the same length. Let $\langle p_{kw}, k_{kw}\rangle$ be the address and key of the head of list $AL[kw]$. Add tuple $(f_{\mathrm{l}}(kw), \langle p_{kw}, k_{kw}\rangle \oplus f_{\mathrm{k}}(kw))$ to hash table HT. Add an appropriate number of dummy entries to HT that are indistinguishable from the other HT entries. Set $\mathit{Index} = (HT, \mathtt{A}, \mathtt{D})$. Output (Index, CT).

Trpdr(pp, uk_{uid}, kw) Let $u = \mathsf{ABE.CustKeyGen}(pp', sk_{\mathit{Srch},uid}, \text{"}\mathit{Srch}\text{:}f_{\mathrm{a}}(kw)\text{"})$. Let U be the set of attributes from uk_{uid}. Set $sk_{kw} \leftarrow (U \setminus \{\text{"}\mathit{Sys}\text{:dec"}\}) \cup \{u\}$. Output $t_{uid,kw} = (f_{\mathrm{l}}(kw), f_{\mathrm{k}}(kw), sk_{kw})$.

Search$(pp, \mathit{Index}, CT, t_{uid,kw})$ Parse $t_{uid,kw} = (\ell, k, sk)$. Initialize $X := \emptyset$. Access HT entry ℓ. If no such entry exits, output X. Otherwise parse $HT[\ell]\oplus k$ as the address and key of the head of list $AL[kw]$ and decrypt the list node by node. Decrypt the contained ABE ciphertexts using sk_{kw}. If ABE decryption fails, continue to the next list node. Otherwise, access list $DL[kw, \mathbb{A}]$ referenced in the ABE ciphertext and add all referenced document ciphertexts from CT to X. Finally, return X.

Dec(pp, uk_{uid}, ct) Let U be the set of attribute keys from uk_{uid}. Output $doc = \mathsf{ABE.Dec}(pp', U, ct)$.

SEAC is correct: all potential search results are precomputed and search simply recovers those partial results that are relevant to the searched keyword and accessible to the searching user.

The efficiency of SEAC heavily depends on the number of keyword–access structure pairs. Let a_{kw} be the number of such pairs for keyword kw, and let n_{kw} be the number of documents containing kw. Furthermore, let $a_{\max}$ be the size of the largest access structure. Then our scheme creates an *Index* structure of size $\mathcal{O}(\sum_{kw}(a_{kw}a_{\max} + n_{kw}))$ (not counting overhead due to dummy entries). Search for keyword kw is performed in time $\mathcal{O}(a_{kw}a_{\max} + n_{kw})$, which is only slightly worse than the trivial lower bound $\mathcal{O}(n_{kw})$. However, the additional costs allow for small leakage.

The leakage that SEAC incurs *to an adversary that controls the server and may corrupt users* can be described by three leakage functions, that we discuss next. By $id(doc)$ we denote the label of document doc's ciphertext in CT used for reference in the encrypted lists $DL[kw, \mathbb{A}]$. We write $id(kw)$ to refer to any identifier of keyword kw from $\{f_{\mathrm{l}}(kw), f_{\mathrm{k}}(kw), f_{\mathrm{a}}(kw)\}$.

Leakage $\mathcal{L}_1(DC)$ from deploying the encrypted document collection includes upper bounds on the number of distinct keywords, the number of keyword–access structure pairs and the number of keyword–document pairs. Additionally, for every document an identifier, its bit length and its access structure is leaked.[1] All this information can be directly extracted from *Index* and CT. Thus,

$$\mathcal{L}_1(DC) = (\#HT,\ \#\mathtt{A},\ \#\mathtt{D}, \{(id(doc), |doc|, \mathbb{A}(doc)) : doc \in DC\}).$$

[1]We do not consider policy hiding for ABE.

When corrupting user uid, leakage $\mathcal{L}_2(uid)$ occurs. It includes the corrupted user's identifier uid and attribute set $Attr_{uid}$, all documents stored at the server such that the documents' access structures are satisfied by $Attr_{uid}$, and all keyword–access structure pairs occurring in the document collection. This leakage occurs because corrupting a user reveals the user's key from which uid and $Attr_{uid}$ can be extracted. Given the user's key, the adversary is able to decrypt ciphertexts that the user is allowed to access, revealing the corresponding documents. As mentioned, we assume users to know the set of keywords occurring in the document collection, so corrupting a user relates keywords kw to their identifiers $id(kw)$. Using algorithm Trpdr, all access structures of documents containing kw can be revealed. This revelation is only possible due to the combined knowledge of the server and the corrupt user. Formally,

$$\mathcal{L}_2(uid) = \begin{pmatrix} (uid, Attr_{uid}), \\ \{(doc, id(doc), KW(doc) : doc \in DC \wedge Attr_{uid} \text{ satisfies } \mathbb{A}(doc)\}, \\ \{(id(kw)), kw, \mathbb{A}(doc)) : doc \in DC \wedge kw \in doc\} \end{pmatrix}.$$

Processing a query for keyword kw on behalf of user uid leaks $\mathcal{L}_3(uid, kw)$, which includes the user's identifier uid and attribute set, $id(kw)$, the access structures of documents that contain the searched keyword and the identifiers of documents that both are accessible to user uid and contain the keyword kw. The user identifier, the user's attribute set and $id(kw)$ can be extracted from the trapdoor. Access structures of documents containing kw are used in list $AL[kw]$ to protect references to DL lists. Those of the DL lists accessible due to the trapdoor reveal the identifiers of documents containing kw and being accessible by user uid. As a result,

$$\mathcal{L}_3(uid, kw) = \begin{pmatrix} (uid, Attr_{uid}), id(kw), \\ \{\mathbb{A}(doc) : doc \in DC \wedge kw \in doc\}, \\ \{id(doc) : doc \in DC \wedge Attr_{uid} \text{ satisfies } \mathbb{A}(doc) \wedge kw \in doc\} \end{pmatrix}.$$

Now, that we know the leakage of SEAC and why that leakage occurs, we can prove the security of SEAC relative to this leakage.

Theorem 7. *SEAC is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$-semantically secure against static adversaries if instantiated with statically secure multi-authority ciphertext-policy attribute-based encryption ABE with authority key customization and eavesdropping-secure symmetric encryption Sym.*

Proof. We construct a simulator $\mathcal{S}$ such that for every adversary $\mathcal{A}$ we have: if $\mathcal{A}$ can distinguish between experiments $\mathbf{Real}^{\mathsf{static}}_{\mathsf{SEAC},\mathcal{A}}(\kappa)$ and $\mathbf{Sim}^{\mathsf{static}}_{\mathsf{SEAC},\mathcal{A},\mathcal{S}}(\kappa)$ with non-negligible probability, then $\mathcal{A}$ breaks ABE's static security or its secure authority key customization or Sym's indistinguishability under eavesdropping attacks. The proof structure is as follows: we first present the simulator, then we explore several hybrid experiments, where part of the computations are performed as in the **Real** experiment and the other part is performed as in the **Sim** experiment. We show that no adversary can distinguish between any pair of these hybrids, unless the adversary breaks the security of ABE or Sym. The security of SEAC then follows from the experiments **Real** and **Sim** being the extreme cases of the hybrids. The simulator $\mathcal{S}$ can be constructed as follows:

Setup. Compute $(pp, mk, ok) \leftarrow \mathsf{Setup}(1^\kappa)$ and give pp to $\mathcal{A}$.

Index. If no $\mathcal{L}_2$ leakage occurs, create bit strings bs_{kw} for each keyword identifier $id(kw)$ occurring in any $\mathcal{L}_3$ leakage, such that the bitstrings bs_{kw} are distinct and of length κ. If $\mathcal{L}_2$ leakage occurs, set $bs_{kw} = kw$ for each keyword that occurs in $\mathcal{L}_2$. Remember the mapping from $id(kw)$ to bs_{kw}.

For each keyword–access structure pair $(kw, \mathbb{A})$ that occurs in $\mathcal{L}_2$ or implicitly in $\mathcal{L}_3$, create an encrypted linked list $DL[kw, \mathbb{A}]$. Add to this list references to document ciphertexts from CT such that the access structure of the corresponding plaintext document is $\mathbb{A}$ and the document contains kw according to $\mathcal{L}_2$ or $\mathcal{L}_3$. Randomly map the list nodes to an array $\mathtt{D}'$ of size $\#\mathtt{D}$. Remember list heads' locations and keys. Fill empty cells of array $\mathtt{D}'$ with dummy entries.

For each keyword kw that occurs in $\mathcal{L}_2$ or implicitly in $\mathcal{L}_3$, create an encrypted list $AL[kw]$. For each pair $(kw', \mathbb{A})$ encrypt the tuple $\langle p_{kw',\mathbb{A}}, k_{kw',\mathbb{A}} \rangle$ under policy $\mathbb{A}\&$"Sys:srch"$\&$"$Srch$:$f_\mathrm{a}(bs_{kw'})$", where $\langle p_{kw',\mathbb{A}}, k_{kw',\mathbb{A}} \rangle$ is the address and key of the head of list $DL[kw', \mathbb{A}]$ if that list was previously created and the list is not empty, or a bit string of 0 of appropriate length. Pad all list nodes to the same length before encryption. Randomly map the list nodes to an array $\mathtt{A}'$ of size $\#\mathtt{A}$. Remember list head's locations and keys. Fill the empty cells of array $\mathtt{A}'$ with dummy entries.

For each keyword kw that occurs in $\mathcal{L}_2$ or implicitly in $\mathcal{L}_3$, create a hash table entry $(f_\mathrm{l}(bs_{kw}), f_\mathrm{k}(bs_{kw}) \oplus \langle p_{kw}, k_{kw} \rangle)$ for HT', where $\langle p_{kw}, k_{kw} \rangle$ is the address and key for the head of list $AL[kw]$. Fill HT' with dummy entries up to capacity $\#HT$. Set $Index' = (HT', \mathtt{A}', \mathtt{D}')$.

Ciphertext collection. For each plaintext document doc contained in any $\mathcal{L}_2(uid)$, encrypt doc under policy $\mathbb{A}(doc)\&$"Sys:dec". For each document doc that occurs in $\mathcal{L}_1(DC)$ but not in any $\mathcal{L}_2(uid)$, encrypt $0^{|doc|}$ under policy $\mathbb{A}(doc)\&$"Sys:dec". Let CT' be the set of ciphertexts generated from documents or based on document lengths.

Keys. For each distinct $(uid, Attr_{uid})$ in $\mathcal{L}_2$ or $\mathcal{L}_3$, compute user secret key $uk_{uid} \leftarrow \mathsf{SEAC.KeyGen}(pp, mk, uid, Attr_{uid})$.

Trapdoors. For each $\mathcal{L}_3(uid, kw)$ compute $t_{uid,kw} \leftarrow \mathsf{SEAC.Trpdr}(pp, uk_{uid}, bs_{kw})$, where uk_{uid} was created before and bs_{kw} is the same as for index generation.

Output. The simulator $\mathcal{S}$ outputs $(Index', CT')$, the set of user secret keys generated due to $\mathcal{L}_2$ and the set of trapdoors generated due to $\mathcal{L}_3$.

Hybrids We denote game $\mathbf{Real}^{\mathsf{static}}_{\mathsf{SEAC},\mathcal{A}}(\kappa)$ by Hyb_0. Starting in hybrid Hyb_1, the simulator $\mathcal{S}$ gets to compute pp. From Hyb_2 onwards, we have $\mathcal{S}$ compute corrupted user's secret keys. Similarly, starting with Hyb_3, $\mathcal{S}$ produces requested honest user's trapdoors. Computations of $\mathtt{D}$ are performed by $\mathcal{S}$ from Hyb_4 onwards. Starting with Hyb_5, $\mathcal{S}$ gets to compute the references to lists $DL[kw, \mathbb{A}]$ for $\mathtt{A}$. $\mathcal{S}$ takes over the remaining computations of $\mathtt{A}$ from Hyb_6 onwards. From Hyb_7 onwards, $\mathcal{S}$ computes hash table HT. Computation of CT is given over to $\mathcal{S}$ in Hyb_8, so hybrid Hyb_8 resembles experiment $\mathbf{Sim}^{\mathsf{static}}_{\mathsf{SEAC},\mathcal{A},\mathcal{S}}(\kappa)$.

Indistinguishability of hybrids We first note that $\mathcal{A}$ does not know keys k_{l}, k_{k}, k_{a} for pseudorandom functions f_{l}, f_{k}, f_{a}, unless $\mathcal{A}$ corrupts some user. In case $\mathcal{A}$ corrupts a user, $\mathcal{S}$ knows all keywords occurring in DC from the $\mathcal{L}_2$ leakage. Then, $bs_{kw} = kw$ for all keywords kw. In case $\mathcal{A}$ does not corrupt a user, it cannot evaluate functions $f_{\mathrm{l}}, f_{\mathrm{k}}, f_{\mathrm{a}}$ on its own. Nevertheless, the unique choice and consequent use of bs_{kw} ensures consistency of the simulator's answers to queries.

Hybrids Hyb_0 and Hyb_1 cannot be distinguished, because both 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 $\mathcal{L}_2$ 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 input as the real experiment does. 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.

Hybrid Hyb_4 is computationally indistinguishable from Hyb_3, since Sym has indistinguishable encryptions under eavesdropping adversaries. Hence no polynomial time algorithm can distinguish dummy entries of $\mathtt{D}$ and $\mathtt{D}'$ from non-dummy entries unless the algorithm holds keys for such entries—replacing $\mathtt{D}$ by $\mathtt{D}'$ leaves the computations of such entries to which $\mathcal{A}$ can obtain keys unchanged and replaces every other entry by a dummy. Similar arguments apply to the computational indistinguishability of Hyb_5 from Hyb_4, applying ABE's static security instead of Sym's eavesdropping indistinguishability. Additionally, ABE's secure authority key customization ensures that $\mathcal{A}$ cannot use a corrupted user's secret key to add a second keyword-specific attribute to an honestly created trapdoor. In case the adversary corrupts a user, there is no difference in the real experiment's remaining computations of $\mathtt{A}$ in Hyb_5 and the simulator's corresponding computation of $\mathtt{A}'$ 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.

For the indistinguishability of hybrids Hyb_6 and Hyb_7, we observe that f_{k} is a pseudorandom function and the bitwise XOR of a bitstring x with the image of a pseudorandom function is a CPA-secure symmetric encryption scheme. Therefore, dummy entries cannot be distinguished from non-dummy entries in HT and HT', unless keys for such entries—images of f_{k}—are known. $\mathcal{A}$ 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 $\mathcal{A}$ to corrupt a user (except with negligible probability), in which case $\mathcal{A}$ can distinguish all dummy entries from non-dummy entries for both HT and HT' 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, $\mathcal{A}$ lacks the keys to f_{l} and f_{k}, so it cannot check that the image it holds is an image of the keyword known to $\mathcal{A}$ but not to $\mathcal{S}$.

Finally, $\mathcal{A}$ cannot distinguish CT from CT', and thus Hyb_7 from Hyb_8, because, again, ABE is statically secure and encryptions of 0 bit strings of appropriate length are computationally indistinguishable from encryptions of documents under the same policy, unless a relevant key is known. Some documents from DC are associated with policies that are satisfied by the attributes of some corrupt user. Such documents are directly encrypted in CT as well as CT'. For all other documents their respective ciphertexts in CT are computationally indistinguishable from the encryptions of 0 bit strings of the relevant length that take their place in CT'.

To sum up the discussion, all our hybrids are computationally indistinguishable. Thus, for sufficiently large κ, no probabilistic polynomial time adversary $\mathcal{A}$ can distinguish between experiments $\mathbf{Real}^{\text{static}}_{\mathsf{SEAC},\mathcal{A}}(\kappa)$ and $\mathbf{Sim}^{\text{static}}_{\mathsf{SEAC},\mathcal{A},\mathcal{S}}(\kappa)$ with probability non-negligible in κ. □

We point out that the proof covers the property of keyword secrecy, i. e. a server's inability to identify plaintext keywords encoded into trapdoors and the index structure. This can be seen from our use of random bitstrings bs_{kw} instead of keywords kw in case no $\mathcal{L}_2$ leakage occurs, i. e. no user is corrupted. SEAC achieves this through its use of pseudorandom functions to hide keywords from the server.

Adaptive security We note that our security proof for SEAC can be easily adopted to fully adaptive adversaries in the random oracle model, i. e. the adversary can adaptively add users to the system and corrupt them. This requires the use of non-committing encryption for MA-CP-ABE as well as for symmetric encryption, which in turn requires the random oracle model. This restriction is because the simulator obtains the leakage over time instead of all at once, so it must produce encrypted output to the server that can be later decrypted to data unknown at the time of encryption, e. g. document ciphertexts. Additionally, the three pseudorandom functions must be modelled as random oracles so the server can answer queries consistently, i. e. the technique of choosing bs_{kw} when encrypting the document collection can be applied. Only when a user is corrupted the simulator learns the relation between bs_{kw} and kw. At this point the simulator programs the random oracles representing the three pseudorandom functions. This approach is feasible, because, as in the proof, the adversary can only evaluate the pseudorandom functions if it holds the respective keys, which it only holds after it has corrupted a user.

Realization We point out that our generic SEAC construction relies on the new notion of authority key customization for multi-authority attribute-based encryption. As shown in Section 3, authority key customization can be added to the Rouselakis/Waters MA-CP-ABE scheme [22]. Hence, the Rouselakis/Waters MA-CP-ABE lends itself for implementing SEAC.

5 Extensions and conclusion

We have shown how to generically construct a searchable document collection that can be outsourced to the cloud without compromising data confidentiality. In SEAC, access control is tightly integrated into search. As a result, SEAC searches efficiently even though search respects access rights and the entity performing search learns little about documents excluded from search results.

Our scheme uses multi-authority attribute-based encryption to split attributes for access control into three classes, based on their semantics. To one such class, *Srch*, we apply our notion of authority key customization. This allows users to produce search trapdoors without help from a third party. Particularly, search trapdoors contain a proper subkey of a user's key, so the server can search using the user's access rights. This may seem like a breach of the user's privacy. In Appendix A.3 we show how user anonymity—among the set of users with the set of identical access rights—can be achieved for the Rouselakis/Waters MA-CP-ABE scheme [22] that can be used to realize SEAC.

In our scheme, we assume a trusted authority to generate user secret keys. With the multi-authority property, we can also split responsibilities for key generation. For example, the responsibility of the *Usr* authority that manages the user attributes can be split into multiple separate authorities.

Our SEAC scheme only supports static document collections. Future research must address document dynamics, because other approaches clearly allow documents to be added to the collection over time. An interesting question is how server's answers can be made verifiable in order to force the server to execute search correctly. This is especially interesting in combination with dynamic document collections. *We are particularly interested in the price we need to pay for such features in terms of leakage.* A third question we ask is, whether techniques such as policy hiding for MA-CP-ABE are compatible with our approach to searchable encryption with access control and how they affect efficiency.

References

[1] James Alderman, Keith M. Martin, and Sarah Louise Renwick. "Multi-level Access in Searchable Symmetric Encryption". In: *IACR Cryptology ePrint Archive* (2017), p. 211.

[2] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. "Public Key Encryption with Keyword Search". In: *EUROCRYPT 2004*. Springer, 2004, pp. 506–522.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. Springer, 2001, pp. 514–532.

[4] Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. "A Survey of Provably Secure Searchable Encryption". In: *ACM Comput. Surv.* 47.2 (2014), 18:1–18:51.

[5] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation". In: *NDSS 2014*. The Internet Society, 2014.

[6] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries". In: *CRYPTO 2013*. Springer, 2013, pp. 353–373.

[7] Yan-Cheng Chang and Michael Mitzenmacher. “Privacy Preserving Keyword Searches on Remote Encrypted Data”. In: *ACNS 2005*. Springer, 2005, pp. 442–455.

[8] Melissa Chase. “Multi-authority Attribute Based Encryption”. In: *TCC 2007*. Springer, 2007, pp. 515–534.

[9] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. “Searchable symmetric encryption: Improved definitions and efficient constructions”. In: *Journal of Computer Security* 19.5 (2011), pp. 895–934.

[10] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *CCS 2006*. ACM, 2006, pp. 79–88.

[11] Changyu Dong, Giovanni Russello, and Naranker Dulay. “Shared and Searchable Encrypted Data for Untrusted Servers”. In: *Data and Applications Security XXII*. Springer, 2008, pp. 127–143.

[12] Eu-Jin Goh. “Secure Indexes”. In: *IACR Cryptology ePrint Archive* (2003), p. 216.

[13] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. “Outsourced symmetric private information retrieval”. In: *CCS 2013*. ACM, 2013, pp. 875–888.

[14] Abdellah Kaci and Thouraya Bouabana-Tebibel. “Access control reinforcement over searchable encryption”. In: *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014*. IEEE, 2014, pp. 130–137.

[15] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. “Dynamic searchable symmetric encryption”. In: *CCS 2012*. IEEE, 2012, pp. 965–976.

[16] Kaoru Kurosawa and Yasuhiro Ohtaki. “UC-Secure Searchable Symmetric Encryption”. In: *FC 2012*. Springer, 2012, pp. 285–298.

[17] Jia-Zhi Li and Lei Zhang. “Attribute-Based Keyword Search and Data Access Control in Cloud”. In: *Tenth International Conference on Computational Intelligence and Security, CIS 2014*. IEEE, 2014, pp. 382–386.

[18] Peter van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter H. Hartel, and Willem Jonker. “Computationally Efficient Searchable Symmetric Encryption”. In: *SDM 2010*. Springer, 2010, pp. 87–100.

[19] Yanbin Lu and Gene Tsudik. “Enhancing Data Privacy in the Cloud”. In: *IFIPTM 2011*. Springer, 2011, pp. 117–132.

[20] Muhammad Naveed. “The Fallacy of Composition of Oblivious RAM and Searchable Encryption”. In: *IACR Cryptology ePrint Archive* (2015), p. 668.

[21] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. “Dynamic Searchable Encryption via Blind Storage”. In: *S&P 2014*. IEEE, 2014, pp. 639–654.

[22] Yannis Rouselakis and Brent Waters. “Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption”. In: *FC 2015*. Springer, 2015, pp. 315–332.

[23] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. “Practical Techniques for Searches on Encrypted Data”. In: *S&P 2000*. IEEE, 2000, pp. 44–55.

[24] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. “Practical Dynamic Searchable Encryption with Small Leakage”. In: *NDSS 2014*. The Internet Society, 2014.

[25] Wenhai Sun, Shucheng Yu, Wenjing Lou, Y. Thomas Hou, and Hui Li. “Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud”. In: *INFOCOM 2014*. IEEE, 2014, pp. 226–234.

[26] Yanjiang Yang, Haibing Lu, and Jian Weng. “Multi-User Private Keyword Search for Cloud Computing”. In: *IEEE 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011*. IEEE, 2011, pp. 264–271.

[27] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. “All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption”. In: *USENIX 2016*. 2016, pp. 707–720.

[28] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. “Multi-User Keyword Search Scheme for Secure Data Sharing with Fine-Grained Access Control”. In: *Information Security and Cryptology - ICISC 2011*. Springer, 2011, pp. 406–418.

[29] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. “VABKS: Verifiable attribute-based keyword search over outsourced encrypted data”. In: *INFOCOM 2014*. IEEE, 2014, pp. 522–530.

A Secure authority key customization

In this section we show that secure authority key customization is feasible for multi-authority attribute-based encryption schemes. We first present the Rouselakis/Waters (RW) MA-CP-ABE scheme [22], then we review the two algorithms for authority key customization already given in the main text. We provide a formal proof of security for our construction. Finally, we discuss how the RW MA-CP-ABE scheme supports hiding a user’s identity when used to construct other primitives such as our SEAC scheme.

A.1 Rouselakis/Waters MA-CP-ABE

The RW MA-CP-ABE scheme is as follows:

GlobalSetup(1^κ) Select bilinear group $(p, \mathbb{G}, g, e)$ and pick hash functions $H, F : \{0,1\}^* \to \mathbb{G}$. Output $pp = (p, \mathbb{G}, g, e, H, F)$.

AuthSetup(pp, θ) Set $pk_\theta = (e(g,g)^{\alpha_\theta}, g^{y_\theta})$ and $mk_\theta = (\alpha_\theta, y_\theta)$ for $\alpha_\theta, y_\theta \leftarrow_\$ \mathbb{Z}_p$. Output (pk_θ, mk_θ).

KeyGen(pp, mk_θ, uid, u) Pick $t \leftarrow_\$ \mathbb{Z}_p$ and set $K_{uid,u} = g^{\alpha_\theta} H(uid)^{y_\theta} F(u)^t$, and $L_{uid,u} = g^t$. Output $uk_{uid,u} = (K_{uid,u}, L_{uid,u})$.

Enc$(pp, \{pk_\theta\}, \mathbb{A}, msg)$ Parse $\mathbb{A} = (M, \rho)$ as a monotone span program with $a \times b$ matrix M and row labelling ρ. If any attribute from $\mathbb{A}$ is not managed by an authority from $\{pk_\theta\}$, output $\perp$. Pick $z, v_2, \dots, v_b, w_2, \dots, w_b \leftarrow_\$ \mathbb{Z}_p$. Set $\mathbf{v} = (z, v_2, \dots, v_b)^\top$, and $\mathbf{w} = (0, w_2, \dots, w_b)^\top$. Denote by λ_x and ω_x the x^{th} component of $\lambda = M\mathbf{v}$ and $\omega = M\mathbf{w}$, respectively. Set $C_0 = msg \cdot e(g,g)^z$ and for each row x of M: pick $t_x \leftarrow_\$ \mathbb{Z}_p$ and set

$$C_{1,x} = e(g,g)^{\lambda_x} \cdot e(g,g)^{\alpha_{\mathrm{Auth}(\rho(x))} t_x}, \qquad C_{2,x} = g^{-t_x},$$
$$C_{3,x} = g^{y_{\mathrm{Auth}(\rho(x))} t_x} \cdot g^{\omega_x}, \qquad C_{4,x} = F(\rho(x))^{t_x}.$$

Output $ct = (\mathbb{A}, C_0, \{C_{1,x}, C_{2,x}, C_{3,x}, C_{4,x}\}_{x=1,\dots,m})$.

Dec$(pp, \{uk_{uid,u}\}, ct)$ Parse $\mathbb{A} = (M, \rho)$ as before. Let X be a subset of rows of M that are labelled with attributes from $\{uk_{uid,u}\}$ and span vector $(1, 0, \dots, 0)$, i. e. find $c_x \in Z_p$ such that $\sum_{x \in X} c_x M_x = (1, 0, \dots, 0)$. If no such set exists, output $\perp$ and exit. For each $x \in X$ compute

$$\begin{aligned} e(g,g)^{\lambda_x} \cdot e(H(uid), g)^{\omega_x} =& C_{1,x} \cdot e(K_{uid,\rho(x)}, C_{2,x}) \\ & \cdot e(H(uid), C_{3,x}) \cdot e(L_{uid,\rho(x)}, C_{4,x}). \end{aligned}$$

Reconstruct $e(g,g)^z = \prod_{x \in X} (e(g,g)^{\lambda_x} \cdot e(H(uid), g)^{\omega_x})^{c_x}$. Output $msg = C_0 / e(g,g)^z$.

A.2 Authority key customization

As a quick reminder, in Section 3 we gave the following two algorithms for authority key customization for RW MA-CP-ABE:

Customize(pp, mk_θ, uid) Output $sk_{\theta,uid} = g^{\alpha_\theta} \cdot H(uid)^{y_\theta}$,

CustKeyGen$(pp, sk_{\theta,uid}, u)$ Pick $t \leftarrow_\$ \mathbb{Z}_p$, set $K_{uid,u} = sk_{\theta,uid} F(u)^t$ and $L_{uid,u} = g^t$, output $uk_{uid,u} = (K_{uid,u}, L_{uid,u})$.

By RW MA-CP-ABE$^+$ we denote the RW MA-CP-ABE scheme with authority key customization.

Security of authority key customization For security of RW MA-CP-ABE$^+$, every adversary against secure authority key customization can be used to forge signatures of the Boneh/Lynn/Shacham (BLS) signature scheme [3]. The signature scheme is as follows:

Gen(1^κ) Select bilinear group $(p, \mathbb{G}, g, e)$ and hash function $H : \{0,1\}^* \to \mathbb{G}$. Set $pk = (p, \mathbb{G}, g, e, H, g^y)$ and $sk = y$ for $y \leftarrow_\$ \mathbb{Z}_p$. Output (pk, sk).

Sign(pk, sk, msg) Compute $\sigma = H(msg)^y$ and output (msg, σ).

Verify(pk, msg, σ) If $e(g^y, H(msg)) = e(g, \sigma)$, output 1, otherwise output 0.

Notice that in our algorithms for authority key customization the customized authority secret is essentially a BLS signature on the user identifier, augmented by the additional factor g^{α_θ}. This can be used to prove the following lemma.

Lemma 8. *RW MA-CP-ABE$^+$ provides secure authority key customization in the random oracle model under the gap-Diffie-Hellman assumption.*

In [3], the BLS signature scheme was proven secure under the gap-Diffie-Hellman assumption that is implied by the q-type assumption used to prove the security of the RW MA-CP-ABE scheme. So, via reduction, we can prove security of authority key customization without any additional assumptions. The proof of the lemma is based on standard techniques.

Note that, although the RW MA-CP-ABE scheme was only proven statically secure, RW MA-CP-ABE$^+$ has secure authority key customization even in an adaptive setting.

A.3 Anonymity

As noted in the conclusion, RW MA-CP-ABE$^+$ allows for hiding the user identifiers contained in user keys. For that, we apply the key re-randomization for RW MA-CP-ABE from [22]. Key re-randomization is done on a per-attribute basis: for each attribute-specific key $(K_{uid,u}, L_{uid,u})$ pick $t' \leftarrow_\$ \mathbb{Z}_p$ and compute the re-randomized attribute-specific key $(K_{uid,u}F(u)^{t'}, L_{uid,u}g^{t'})$.

In our application of MA-CP-ABE to the construction of SEAC, a subset of the attributes in user key uk_{uid} are used to create a trapdoor $t_{uid,kw}$ for search. For the server to be able to use this key as intended, i. e. for decrypting ABE ciphertexts contained in the *Index* data structure stored at the server, the server needs to know $H(uid)$, because it is combined with ciphertext component $C_{3,x}$ during decryption. In order to achieve anonymity from the server, we blind the user identifier using the above technique. Pick $k \leftarrow_\$ \mathbb{Z}_p$. Replace every attribute-specific key $(K_{uid,u}, L_{uid,u})$ in the trapdoor by a re-randomized version of $(K_{uid,u}g^{y_{\mathrm{Auth}(u)} \cdot k}, L_{uid,u})$. Additionally, together with the trapdoor, provide the blinded user identifier $H(uid)g^k$. It is easy to check that a user secret modified in this way is still functionally equivalent to the original user secret. However, new attributes cannot be added to the anonymized version of the user secret, unless the entity that adds the new attribute has knowledge of k.