Searchable Encryption with Access Control*

Nils Löken
Paderborn University
nils.loeken@uni-paderborn.de

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.

KEYWORDS

Searchable encryption, inverted index, access control, authority key customization, multi-authority ABE

1 INTRODUCTION

Searchable Encryption [21] 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 [27], 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 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. [21], various flavors of searchable encryption have been developed, providing searchable encryption in different settings. *Searchable symmetric encryption* (SSE), influenced by Curtmola et al. [8, 9], originally provides searchable encryption to single users. SSE [4, 5, 14, 24] satisfies different non-equivalent security notions [6, 8, 9, 11], including UC-security [15]. Generic techniques to achieve multi-user SSE are (proxy) re-encryption [25] and broadcast encryption [8]. SSE has been combined with oblivious RAM [18, 22], private information retrieval [12] and blind storage [19] 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 [10]. 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 [13] or formulating search queries [16, 17]. Recently, *attribute-based encryption with keyword search* [23, 28] has been suggested, using *attribute-based encryption* (ABE) to achieve access control.

For a comprehensive survey of searchable encryption in its various flavors, see [3]. Also, see [26] 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

*The full version of this paper can be found at http://eprint.iacr.org/2017/679.

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).
ARES '17, Reggio Calabria, Italy

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 [20].

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. [13] realizes searchable encryption with access control based on the SSE-1 scheme of Curtmola et al. [8], 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 [13] 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 [8]. 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. [23] 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 [23] 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. [28]. 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. 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 [28] 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 [28] 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 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. Thus, we prepend each attribute's name with the name of the authority that manages that particular attribute, e. g. "AA:CEO" is the attribute CEO managed by authority AA. We often express access structures in terms of attributes.

2.2 Attribute-based encryption

We now present a definition of multi-authority attribute-based encryption [7] in the variant that we use to construct a searchable repository of ciphertexts that provides access control.

Definition 2.1. A *multi-authority ciphertext-policy attribute-based encryption (MA-CP-ABE) scheme* consists of the following probabilistic polynomial time algorithms [20]:

- **GlobalSetup** takes security parameter 1^κ; outputs public parameters pp
- **AuthSetup** takes pp and authority identifier θ; outputs public key pk_θ and authority secret key mk_θ
- **KeyGen** takes pp, mk_θ, user identifier uid and attribute identifier u; outputs attribute-specific user key $uk_{uid,u}$
- **Enc** takes pp, set $\{pk\}$ of authorities' public keys, access structure $\mathbb{A}$ and message msg; outputs ciphertext ct
- **Dec** takes pp, set $uk_{uid,Attr_{uid}}$ of attribute-specific user keys and ct; outputs message msg

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

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. Due to space limitations, we only present an intuition of the *static security* model for MA-CP-ABE. For a formal description, we refer to Rouselakis and Waters[20]. The security notion considers a static adversary, i. e. it makes all its queries at the same time. The adversary chooses the participating attribute authorities, of which some are statically corrupted. The adversary can query for user keys, which 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.

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

- **Setup** takes security parameter 1^κ; outputs public parameters pp, master secret mk and owner key ok
- **KeyGen** takes pp, mk, user identifier uid and attribute set $Attr_{uid}$; outputs user secret uk_{uid}
- **Enc** takes pp, ok and document collection DC; outputs index structure $Index$ and chiphertext set CT
- **Trpdr** takes pp, uk_{uid} and keyword kw; outputs search trapdoor $t_{uid,kw}$
- **Search** takes pp, $Index$ and $t_{uid,kw}$; outputs result $X \subseteq CT$
- **Dec** takes pp, uk_{uid} and ciphertext ct; outputs document doc

We require for all correctly set up systems and search results $X \leftarrow \mathrm{Search}(pp, Index, t_{uid,kw})$, 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 $\mathrm{Dec}(pp, uk_{uid}, ct) = doc$.

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.

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

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

(1) Setup: run $(pp, mk, ok) \leftarrow \mathrm{Setup}(1^\kappa)$; 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 $(Index, CT) \leftarrow \mathrm{Enc}(pp, ok, DC)$, 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.

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

(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 2.3. 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 $|\Pr[\mathbf{Real}^{\text{static}}_{\Pi,\mathcal{A}}(\kappa) = 1] - \Pr[\mathbf{Sim}^{\text{static}}_{\Pi,\mathcal{A},\mathcal{S}}(\kappa) = 1]|$ 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 3.1. 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; outputs customized authority key $sk_{\theta,uid}$
CustKeyGen takes pp, $sk_{\theta,uid}$ and attribute identifier u; 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 θ', (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 trivially.

Definition 3.2. 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 $\mathcal{A}$ $\Pr[\mathbf{AuthCust}_{\Pi,\mathcal{A}}(\kappa) = 1]$ 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 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 Rouselakis/Waters MA-CP-ABE scheme [20]. 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 $L_{uid,u} = g^t$ for random $t \in \mathbb{Z}_p$ and $(\alpha_\theta, y_\theta) \in \mathbb{Z}_p \times \mathbb{Z}_p$ being 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$(pp, sk_{\theta,uid}, u)$ Set $K_{uid,u} = sk_{\theta,uid} F(u)^t$ and $L_{uid,u} = g^t$ for $t \leftarrow_\$ \mathbb{Z}_p$, 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 the full version.

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. [8].

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*:$f_a(kw)$" 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 PRF keys $k_l, k_k, k_a \leftarrow_\$ \{0,1\}^\kappa$, set up ABE and the functionalities $pp' \leftarrow \mathsf{ABE.GlobalSetup}(1^\kappa)$ and $\{(pk_\theta, mk_\theta) \leftarrow \mathsf{ABE.AuthSetup}(pp', \theta)\}_{\theta \in \{Usr, Sys, Srch\}}$. Output (pp, mk, ok), where $pp = (pp', pk_{Usr}, pk_{Sys}, pk_{Srch})$, $ok = (k_l, k_k, k_a)$, $mk = (mk_{Usr}, mk_{Sys}, mk_{Srch}, k_l, k_k, k_a)$.

KeyGen$(pp, mk, uid, Attr)$ Ensure that attributes in $Attr$ are managed by functionality *Usr*. Set $uk'_{uid} \leftarrow \{\mathsf{ABE.KeyGen}(pp', mk_{Usr}, Attr)\} \cup \{\mathsf{ABE.KeyGen}(pp', mk_{Sys}, \{\text{"}Sys\text{:dec"}, \text{"}Sys\text{:srch"}\})\}$. Let $sk_{Srch,uid} \leftarrow \mathsf{ABE.Customize}(pp, mk_{Srch}, uid)$ and output $uk_{uid} = (uk'_{uid}, sk_{Srch,uid}, k_l, k_k, k_a)$.

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 the lists store pointers to CT rather than documents. Let D be the memory array of appropriate size that stores the list nodes. Let $\langle p, k\rangle$ be the address and key of the head of list $DL[kw, \mathbb{A}]$. ABE encrypt $\langle p, k\rangle$ under policy $\mathbb{A}\&\text{"}Sys\text{:srch"}\&\text{"}Srch\text{:}f_a(kw)\text{"}$ and store the ciphertext in encrypted list $AL[kw]$. Let 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 $\langle p', k'\rangle$ be the address and key of the head of list $AL[kw]$. Add tuple $(f_l(kw), \langle p, k\rangle \oplus f_k(kw))$ to hash table HT. Set $Index = (HT, \mathsf{A}, \mathsf{D})$. Output $(Index, CT)$.

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

Search$(pp, Index, CT, t_{uid,kw})$ 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 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 $O(\sum_{kw}(a_{kw}a_{\max} + n_{kw}))$. Search is performed in time $O(a_{kw}a_{\max} + n_{kw})$, which is only slightly worse than the trivial lower bound $O(n_{kw})$. 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 $\{f_l(kw), f_k(kw), f_a(kw)\}$.

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. All this information can be directly extracted from *Index* and CT.

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.

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.

THEOREM 4.1. *SEAC is $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$-semantically secure against static adversaries when instantiated with statically secure multi-authority ciphertext-policy attribute-based ABE encryption and eavesdropping-secure symmetric encryption Sym.*

The proof is omitted due to space limitations, but is presented in the full version.

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 [20]. 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 the full version 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 [20] that can be used to realize SEAC.

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. 2017. Multi-level Access in Searchable Symmetric Encryption. *IACR Cryptology ePrint Archive* (2017), 211.

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

[3] Christoph Bösch, Pieter H. Hartel, Willem Jonker, and Andreas Peter. 2014. A Survey of Provably Secure Searchable Encryption. *ACM Comput. Surv.* 47, 2 (2014), 18:1–18:51.

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

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

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

[7] Melissa Chase. 2007. Multi-authority Attribute Based Encryption. In *TCC 2007*. Springer, 515–534.

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

[9] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.

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

[11] Eu-Jin Goh. 2003. Secure Indexes. *IACR Cryptology ePrint Archive* (2003), 216.

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

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

[14] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *CCS 2012*. IEEE, 965–976.

[15] Kaoru Kurosawa and Yasuhiro Ohtaki. 2012. UC-Secure Searchable Symmetric Encryption. In *FC 2012*. Springer, 285–298.

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

[17] Yanbin Lu and Gene Tsudik. 2011. Enhancing Data Privacy in the Cloud. In *IFIPTM 2011*. Springer, 117–132.

[18] Muhammad Naveed. 2015. The Fallacy of Composition of Oblivious RAM and Searchable Encryption. *IACR Cryptology ePrint Archive* (2015), 668.

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

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

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

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

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

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

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

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

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

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