Attribute-Based Encryption as a Service for Access Control in Large-Scale Organizations*

Johannes Blömer[1], Peter Günther[2], Volker Krummel[2], and Nils Löken[1]

[1] Paderborn University
{johannes.bloemer,nils.loeken}@uni-paderborn.de
[2] Diebold Nixdorf
{peter.guenther,volker.krummel}@dieboldnixdorf.com

Abstract In this work, we propose a service infrastructure that provides confidentiality of data in the cloud. It enables information sharing with fine-grained access control among multiple tenants based on attribute-based encryption. Compared to the standard approach based on access control lists, our encryption as a service approach allows us to use cheap standard cloud storage in the public cloud and to mitigate a single point of attack. We use hardware security modules to protect long-term secret keys in the cloud. Hardware security modules provide high security but only relatively low performance. Therefore, we use *attribute-based encryption with outsourcing* to integrate hardware security modules into our micro-service oriented cloud architecture. As a result, we achieve elasticity, high performance, and high security at the same time.

1 Introduction

Cloud computing [9] has become a popular computing model that allows enterprises to outsource their IT infrastructure. Resource pooling, multi-tenancy, and elasticity enable high availability of services at low costs. A major concern with cloud computing is security [4]. Particularly, a Cloud Service Provider (CSP) should not learn confidential information from the data it stores or processes. Such restrictions are imposed by data privacy laws, industry-specific regulation and standards, or by companies willing to outsource data but worried about the confidentiality of their secrets.

Many solutions for protecting data confidentiality in the cloud have emerged over time. Attribute-Based Encryption (ABE) has often been proposed as a solution, but is not deployed in practice. Solutions not based on ABE are already deployed. Cloud storage providers, e. g. Dropbox,[3]

* 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).

[3] https://www.dropbox.com

encrypt data before storage ensuring confidentiality from outsiders. Since the provider encrypts the data, confidentiality against insiders is not guaranteed. Protection from insiders can be achieved by client-side encryption, e. g. Cryptomator,[4] or encryption as a service, e. g. Ciphercloud.[5]

The latter two solutions are sufficient for individuals, but fall short of large organizations' needs. Typically, large organizations have members with heterogeneous rights to access data with rather complex policies describing such access rights in a fine-grained manner. Both, encryption as a service and client-side encryption do not provide fine grained access control. Additionally, with encryption as a service the encryption provider has access to plaintext data, requiring trust in the provider. Client side-encryption partially negates the benefits of resource pooling in the cloud.

Our contribution. In this paper we present a novel approach to fine-grained access control as a service. Our solution leverages the cloud's resources, while our design ensures confidentiality of data from the CSP. It is based on three components that are integrated into a system for access control as a service. Particularly, we have (1) a cloud-based service for Access Control with Encryption (ACE), (2) a cloud design that complements our ACE service, and (3) an infrastructure for identity and key management.

Our ACE service achieves fine-grained access control via Attribute-Based Encryption (ABE) with outsourcing as defined in [7]. We exploit outsourcing to design a cloud infrastructure complementing ACE such that ACE can take advantage of the cloud's resources to achieve efficiency. At the same time, our solution ensures confidentiality of data even against insider attacks. We furthermore show how to adapt the existing infrastructure for identity and key management of large-scale organizations to the specifics of access control via ABE. This enables user revocation without re-encryption through the application of standard mechanisms.

Related work and discussion. Our approach to ACE is based on ABE [12,1] with outsourcing, as introduced by Green et al. [7]. In [7] outsourcing enables mobile devices to perform ABE decryption by limiting their computations to the security critical part of the decryption, while the computationally expensive but non-critical part is performed by the cloud. We use this mechanism within the cloud to execute the security critical part of decryption on so-called hardware security modules, thus achieving high security. We use standard cloud services for non-critical computations, thus achieving efficiency.

[4] https://cryptomator.org
[5] https://ciphercloud.com

Numerous papers identify issues with ABE that hinder its deployment in the cloud, e. g. [14,15,16,17,18]. Issues include that (1) data integrity is not protected, (2) fine-grained access control for write access is typically ignored, and (3) user's access cannot be revoked once granted. Zhao et al. [18] suggest attribute-based signatures to address the first two issues. This proposal is rather generic and can also be applied in our scenario. However, this is beyond the scope of this paper.

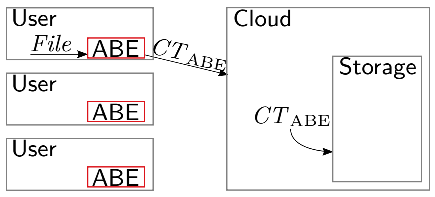
Means to revoke users' access rights are proposed in numerous papers [14,15,16,17]. These papers specifically suggest modifications to ABE schemes that allow user's access rights to be revoked. Particularly, they propose re-encrypting all ciphertexts that the revoked user had access to, as well as updating all the remaining users' cryptographic keys. This approach is very costly for all parties involved, especially for the party who re-encrypts the ciphertexts. Therefore, additional features are suggested to reduce the load of some parties. For example, Yang et al. [15] and Yu et al. [16] suggest to delay the re-encryption of a particular ciphertext until some user actually requests the ciphertext. Yu et al. additionally use a technique similar to outsourcing [7] in order to reduce users' loads when updating their keys: only the non-critical parts of the users' keys need to be updated, so key updates can be applied by a cloud server.

Achieving outsourced decryption [7] and user revocation, the scheme of Zhang et al. [17] at a first glance looks most similar to our approach to access control with encryption in the cloud. They consider the cloud's structure when outsourcing computations: decryption is outsourced to fog nodes, i. e. cloud resources close (e. g. in a geographical sense) to the user. Hence, Zhang et al. increase ABE's efficiency using cloud resources, but in contrast to us, do not fully deploy ABE in the cloud in a secure way.

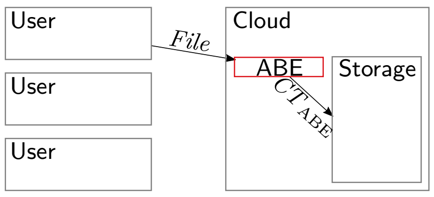
Paper organization. In Section 2 we discuss how ABE can be used for access control in the cloud. Section 3 gives a brief introduction to ABE and some of its properties. Section 4 presents ACE and its complementing cloud design. In Section 5 we present the infrastructure for identity and key management. Section 6 presents proofs of concept concerning the ABE schemes underlying ACE and the core technologies that we have used to implement ACE. Finally, the paper is concluded in Section 7.

2 Approaches to ABE for Cloud Infrastructures

As a starting point, we consider two basic approaches for realizing access control in the cloud via ABE. In the *security-oriented approach* (see



(a) Security-oriented approach: ABE performed by users.



(b) Service-oriented approach: ABE performed in the cloud.

Figure 1: ABE performed in a cloud environment.

Figure 1a), we store ABE encrypted data at a Cloud Service Provider (CSP). Encryption and decryption of the data is handled by the user. In the *service-oriented approach* (see Figure 1b), we also store ABE encrypted data at a CSP, but encryption and decryption of the data is handled by an encryption service in the public cloud on the user's behalf.

In the security-oriented method, users do not have to trust the CSP, because all data is encrypted by the users. Since users provide own resources for ABE, this approach does not benefit from resource pooling and elasticity in the cloud. This method also prohibits ABE's tight integration into complex applications where many services process the decrypted data.

The service-oriented approach offers resource pooling and elasticity. Furthermore, it can be coupled with services for data processing. However, this method requires users to ultimately trust the CSP.

In this work, we enhance the service-oriented approach with mechanisms to obtain security similar to the security-oriented method while preserving the elasticity of the service-oriented approach. To achieve this, we partition the computations of ABE encryption and decryption into sub-services according to their security and elasticity requirements.

3 Attribute Based Encryption with Outsourcing

We describe ABE-OS-KEM, a primitive that underlies the architecture for our ACE service. We also cover security guarantees and requirements.

3.1 ABE-OS-KEM

Access control can be cryptographically enforced by Attribute-Based Encryption (ABE) [1]. In ABE, ciphertexts are associated with access structures. Access structures represent Boolean formulas consisting only of AND and OR operators. Users hold attributes, e. g. role descriptions, represented by secret keys. Attribute sets represent interpretations of the variables in Boolean formulas, and satisfy an access structure if their interpretations satisfy the policy's formula. Only attributes satisfying a ciphertext's policy are able to decrypt that ciphertext.

Our approach to protect the confidentiality of outsourced data follows the KEM/DEM paradigm [8, Ch. 11.3]. The Data Encapsulation Mechanism (DEM) with algorithms $(\mathsf{Encrypt}, \mathsf{Decrypt})$ takes keys from some key space $\mathcal{K}$. Our Key Encapsulation Mechanism (KEM) is based on ABE:

Definition 1. *An ABE-KEM consists of algorithms*

Setup: *given a security parameter Λ, output public parameters pub_{ABE} and a master secret msk_{ABE}.*
KeyGen: *given pub_{ABE}, msk_{ABE} and an attribute set A_{ID}, output a user key $sk_{ABE,ID}$.*
Encaps: *given pub_{ABE} and a policy $\mathbb{A}$, output a symmetric key $k \in \mathcal{K}$ and a ciphertext CT_{ABE}.*
Decaps: *given pub_{ABE}, $sk_{ABE,ID}$ and ciphertext CT_{ABE}, output a symmetric key k.*

We require for all correctly set up systems, policies $\mathbb{A}$, tuples $(k, CT_{ABE}) \leftarrow \mathsf{Encaps}(pub_{ABE}, \mathbb{A})$, and user keys $sk_{ABE,ID}$, if the attributes in $sk_{ABE,ID}$ satisfy $\mathbb{A}$ then $\mathsf{Decaps}(pub_{ABE}, sk_{ABE,ID}, CT_{ABE}) = k$.

Typically, the Decaps algorithm is computationally expensive [7]. Therefore, [7] splits the Decaps algorithm into a computationally expensive part that is not security critical, and a security critical part that is rather efficient. Splitting Decaps yields a variant of ABE-KEM that we call ABE-OS-KEM, as it allows to partially outsource computations to the cloud.

Definition 2. *An ABE-OS-KEM consists of the following algorithms*

Setup, KeyGen *and* **Encaps**, *are as in Definition 1.*
TransKey: *given pub_{ABE} and $sk_{ABE,ID}$, output a transformation key tk_{ID} and a decapsulation key $sk_{PK,ID}$.*
Transform: *given pub_{ABE}, tk_{ID} and CT_{ABE}, output partially decrypted ciphertext CT_{PK}.*
Decaps: *given pub_{ABE}, $sk_{PK,ID}$ and CT_{PK}, output a symmetric key k.*

We require for all correctly set up systems, policies $\mathbb{A}$, tuples $(k, CT_{ABE}) \leftarrow \mathsf{Encaps}(pub_{ABE}, \mathbb{A})$, user keys $sk_{ABE,ID}$, and key pairs $(tk_{ID}, sk_{PK,ID}) \leftarrow \mathsf{TransKey}(pub_{ABE}, sk_{ABE,ID})$ if $sk_{ABE,ID}$ satisfies $\mathbb{A}$ then

$$\mathsf{Decaps}(pub_{ABE}, sk_{PK,ID}, \mathsf{Transform}(pub_{ABE}, tk_{ID}, CT_{ABE})) = k.$$

The original Decaps algorithm is split into two algorithms: Transform and Decaps. The additional algorithm TransKey is required to produce separate keys required by algorithms Transform and Decaps.

For our proof of concept implementation, we constructed an ABE-OS-KEM based on the ciphertext-policy ABE scheme of Rouselakis and Waters [11] and the outsourcing technique of Green et al. [7]. We present our ABE-OS-KEM in Section 6.1.

3.2 Security of ABE-OS-KEM and protection of keys

For the security of our ABE-OS-KEM, we adopt the notion of security against replayable chosen-ciphertext attacks (RCCA, see [7]): no reasonable adversary learns any information about the key encapsulated in a certain ciphertext, nor can that key be modified unnoticeably. These properties even hold for adversaries with access to arbitrary transformation keys.

In an ABE-OS-KEM, we consider several types of keys for decryption. Keys are categorized as critical and non-critical for security. Transformation keys (tk_{ID}) are non-critical. Critical keys are further classified as user-specific and file-specific. Keys $sk_{\mathrm{ABE},ID}$ and $sk_{\mathrm{PK},ID}$ are user-specific, while the data encapsulation mechanism's symmetric keys are file-specific.

Knowledge of a *user key* allows decryption of ciphertexts with policies satisfied by the user key. Hence, such a key is security critical. An ABE-OS-KEM being a KEM, each file is encrypted under an individual *symmetric key k*. Thus, if a symmetric key leaks, the corresponding file leaks. Thus, also the symmetric keys have to be considered as critical to security. In an ABE-OS-KEM, *transformation keys* tk_{ID} are meaningless without their respective *decapsulation keys* $sk_{\mathrm{PK},ID}$. Together the transformation keys and decapsulation keys have the same decapsulation capabilities as the user keys they are derived from. The order in which operation Transform and Decaps are performed then give the classification of transformation keys to be non-critical and decapsulation keys to be critical and user-specific.

4 Access Control with Encryption and Cloud Design

In this section, we describe our service for access control with encryption in the cloud (ACE). Our goal is a service that provides the security of the security-oriented approach from Section 2 and the elasticity of the service-oriented approach. We first split ACE into sub-services based on our ABE-OS-KEM and its keys. Then we present our model of a cloud. Subsequently, we match ACE sub-services to the components of our cloud design. Finally, we discuss our approach in terms of security and elasticity.

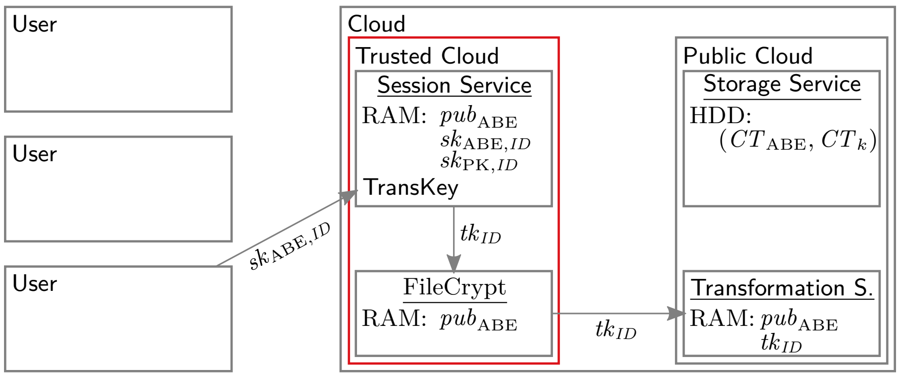


Figure 2: The interaction during session initialization and the sub-services' internal states after initialization. The user sets up the session with the session service that computes a transformation key given to FileCrypt, which relays the transformation key to the transformation service.

4.1 Access Control with Encryption via ABE

We aim to realize ACE via ABE using our ABE-OS-KEM. As discussed in Section 3.2, several categories of keys exist in an ABE-OS-KEM. For each of the categories, we establish a separate sub-service: a *session service* handling user-specific keys, a *FileCrypt* service for file-specific keys, and a *transformation service* handling non-critical keys. Hence, the session service implements algorithms TransKey and Decaps, the transformation service implements Transform and FileCrypt implements Encaps as well as algorithms Encrypt and Decrypt of the data encapsulation mechanism.

We further explore our services' interactions. Particularly, we discuss how users set up a session with ACE and access files. We omit file uploads, because they only involve the FileCrypt service.

The interactions for setting up a session of ACE are shown in Figure 2. The session service uses the user's key $sk_{\mathrm{ABE},ID}$ to compute transformation key tk_{ID} and decapsulation key $sk_{\mathrm{PK},ID}$ via algorithm TransKey. While the service keeps the $sk_{\mathrm{PK},ID}$ secret, tk_{ID} is handed over to FileCrypt and forwarded to the transformation service in the public cloud.

For decryption, the sub-services interact as shown in Figure 3. When the user requests an encrypted file, stored as $(CT_{\mathrm{ABE}}, CT_k)$ in cloud storage, the transformation service receives a copy of CT_{ABE} and applies algorithm Transform. The resulting CT_{PK} is given to the session service. The session service applies the Decaps algorithm and gives the obtained symmetric key k to FileCrypt. Applying the data encapsulation mecha-

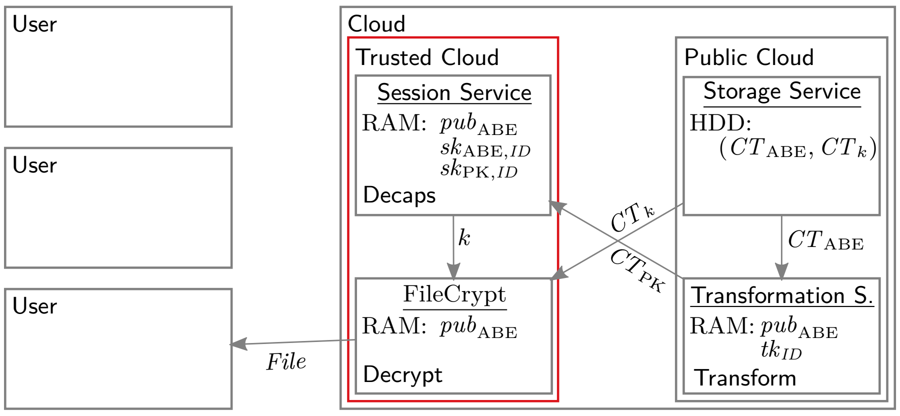


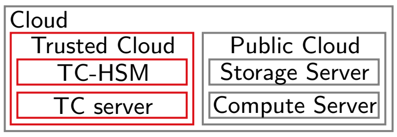
Figure 3: Interactions during file access. ABE ciphertext CT_{ABE} is transformed into partially decrypted ciphertext CT_{PK} by the transformation service using the transformation key. The session service performs a Decaps operation on CT_{PK} using the decapsulation key $sk_{\mathrm{PK},ID}$ and obtains k which is used by FileCrypt to decrypt CT_k. The user receives the result.

nism's Decrypt algorithm to k and CT_k received from the cloud's storage yields the plaintext data *File* that is given to the user.

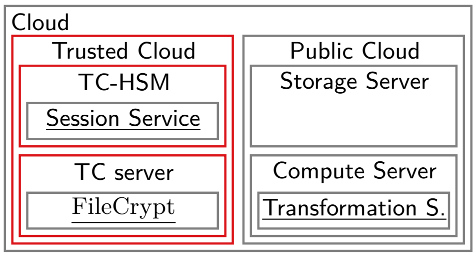
4.2 Cloud design for ACE

As described in Section 2, our architecture for access control in the cloud aims at realizing strong security as if access control and decryption were performed at the user while leveraging the cloud's computational resources for those tasks. In order to achieve this goal, our cloud model must reflect the security requirements imposed by the sub-services of ACE. Figure 4a presents our cloud model. The design considers storage and computational resources of the public cloud and complements them with a trusted cloud, which consists of two components, a TC server and a TC-HSM.

A *TC server* is a server with dedicated technical and organizational measures like remote attestation and memory encryption to protect the integrity of executed software and to protect the confidentiality of processed data. A *TC-HSM* is a tamper-resistant Hardware Security Module (HSM) that provides a restricted and well-defined Application Programming Interface (API). An HSM protects integrity and confidentiality of stored data also against attackers with physical access to the device. Furthermore, the TC-HSM API supports run-time initialization with user keys.



(a) The layout of our cloud: a trusted cloud with a TC-HSM as trust anchor, and non-trusted public resources.



(b) The mapping of ACE sub-services to the components of our cloud based on security considerations.

Figure 4: The components of our cloud and their relation to ACE.

On the one hand, the TC server provides high computational power and is able to dynamically assign its resources to executed services based on their workload. On the other hand, the TC server is unable to protect the confidentiality of data against adversaries with physical access. This protection is provided by a TC-HSM at the cost of restricted flexibility and power. In consequence, the components of our trusted cloud achieve security by different means and thus establish different levels of security. Both the TC-HSM and the TC server are secure. However, its stronger guarantees make the TC-HSM fit to serve as a trust anchor for our cloud.

4.3 Security, execution of services

Our division of ABE-OS-KEM's algorithms into services reflects the security levels required by the algorithms based on the threat that exposing their key inputs poses. Thus, the assignment of services to the components of our cloud model must also reflect the expected levels of security. As a result, we map our ACE sub-services, which implement the ABE-OS-KEM, to the components of our cloud as shown in Figure 4b.

As discussed in Section 4.2, the TC-HSM provides the highest level of security. It is thus fit to run the session service that works on security critical user specific secrets. The TC server provides a level of security that is sufficient to run the FileCrypt service that operates on file-specific secrets. The public cloud's compute servers provide no security guarantees, so they may only operate on non-critical keys. The transformation service can be run on such servers. For this assignment, the security needed by the three categories of ABE-OS-KEM keys, and thus our services, match the three levels of security provided by the components of our cloud model.

4.4 Elasticity

The partitioning of our ACE service into sub-services does not only reflect the sensitivity of keys. It also supports elasticity because each sub-service can be scaled individually. This provides four dimensions of elasticity: elasticity with respect to data storage, simultaneous access to multiple files, the complexity of access policies, and the number of active users.

Data is always stored encrypted at the storage server in the public cloud. Hence, storage can be dynamically (de-)provisioned based on the needed amount, the expected reliability, and the acceptable latency.

For simultaneous processing of multiple files, we distinguish between encryption (write access) and decryption (read access). Encryption of data does not involve user keys and is performed on the TC server by the FileCrypt service. Depending on the number of files to encrypt, the TC server provisions resources for the FileCrypt service based on standard load balancing mechanisms. Decryption of files additionally involves the transformation service and the session service. The transformation service is hosted in the public cloud with high elasticity. The session service is executed on the TC-HSM with limited resources, but due to the initialization (see Section 4.1), additional TC-HSMs can temporarily be initialized.

For decrypting files with complex access policies, we benefit from the ABE-OS-KEM with a separated transformation and decapsulation step. The complexity of Decaps at the constrained TC-HSM is independent of the policy. Hence, additional resources for complex policies only need to be provisioned for $\mathsf{Transform}$ that is executed in the public cloud.

Finally, the number of active users determines the amount of provisioned HSMs for executing the session service. Resources for the FileCrypt and transformation service are provisioned during active encryption or decryption and are freed while no data is being accessed.

5 Infrastructure for Identity and Key Management

In this section, we describe identity management, rights management, and key management for our ABE-based ACE service. Identity management provides entities like users or hardware components with an identity and revokes identities. Rights management is the assignment of access rights, respectively attributes, to identities according to their roles. Key management is the technical task of enforcing those rights using ABE and includes key generation, key storage, and key revocation.

5.1 Tasks

Identity and rights management. As a building block, our system uses a classical Public Key Infrastructure (PKI) with a root certificate and a corresponding Certification Authority (CA). In a technical sense, an identity ID is a public key with a certificate. The CA provides an entity with an identity by issuing a certificate for the entity's public key. Standard mechanisms like revocation lists (see [10, Chapter 13]) are used to revoke an identity. An entity can prove its identity or establish a secure channel with another entity based on its certificate.

An identity possesses a set of rights according to its role. In our case, these rights correspond to a set of ABE-attributes (see Section 3). We encode these attributes into the certificate of the identity during certificate generation. This allows entities to check the rights of an identity.

ABE key storage. For our system, we apply client-side key management to put the individual organizations in control of decryption keys (see [4]). Therefore, in our system, each organization operates a dedicated service for ABE key storage. The ABE decryption key $sk_{\mathrm{ABE},ID}$ of the user with identity ID is then stored at the key storage of the user's organization.

Based on the key storage, we modify the initialization of our ACE service from Section 4.1 (see Figure 5, compare Figure 2). No user can have direct access to her key. Instead, the user with identity ID authenticates at the key storage to obtain a ticket for her key. The ticket is only granted to the user if she has not been revoked by the CA. The user forwards the ticket to the session service, who, via an authenticated channel, presents the ticket to the key store and obtains the user's key $sk_{\mathrm{ABE},ID}$ in return.

ABE key generation. For the generation of ABE user keys, we operate a global key generation service that has access to the ABE master secret key. Take note that the creation of user secret keys via algorithm KeyGen is independent of any user identifier. Thus, organizations can request user keys from the key generation service without providing user identifiers. Organizations can store the obtained user secret keys in their respective key storages and bind the keys to users later on. Note that the key generation service itself is beyond our considerations.

Our system explicitly supports sharing of data between identities of different organizations. Therefore, the ABE master secret key msk_{ABE} and the corresponding public parameters pub_{ABE} (see Definition 1 and Definition 2) are used globally for all participating organizations.

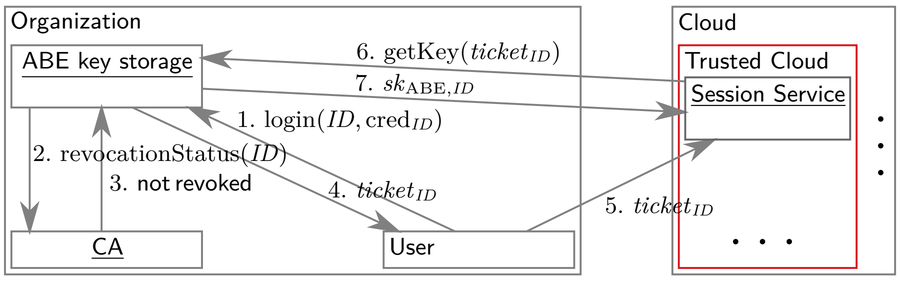


Figure 5: ACE initialization with key storage: The user *ID* uses her credentials (e. g. password) to log in at the key storage, which checks the user's revocation status. If the user is not revoked, she is given a ticket $ticket_{ID}$ that she presents to the ACE session service. The session service uses the ticket to obtain the user's ABE key from the key store.

5.2 Discussion

Our system is *dynamic*, i. e. it is possible to add and revoke entities, as well as changing entities' access rights. Our system also provides *separation of duties* among multiple parties. It distinguishes the technical aspects of key management from the administrative task of rights management, and it splits responsibilities between participating organizations.

Dynamic. To *add users* to the system, we generate an identity *ID* consisting of a public/private key pair with a PKI-based certificate. New identities are provided with secret ABE decryption keys according to their role and corresponding attributes. To *revoke/remove users* from the system we revoke their *ID* based on classical revocation mechanisms of the PKI. This effectively revokes the user from the ACE service because it prevents access to the user's secret key $sk_{\mathrm{ABE},ID}$ at the key storage. Hence, we implicitly add a key revocation mechanism to ABE by combining ABE with classical PKI and a dedicated ABE key storage. *Changing the access rights* of an entity can be realized by first revoking its identity and then providing it with a new certificate that reflects the updated rights.

Separation of duties. Our setup supports *distinguishing administrative from technical duties* by providing distinct services for identity and rights management on the one side, and key storage and ABE key generation on the other side. To *separate responsibilities between organizations,* each organization implements its own identity management and operates its own service for key storage. Nevertheless, an organization could operate the key management service at a trusted third party or in the cloud.

The CA and the ABE key generation service are operated by a trusted global provider. To split responsibilities between organizations, we propose a hierarchical PKI with an inter-organizational master CA and additional organization-specific CAs. ABE key generation with access to the master secret key msk_{ABE} is a single point of attack. To mitigate this risk, we propose to apply techniques for distributing msk_{ABE} as in [6].

6 Proofs of concept — ABE-OS-KEM and Infrastructure

In this section we present proofs of concept for our definition of ABE-OS-KEM and our cloud architecture. Particularly, we provide a description of an ABE-OS-KEM and complement it a description of how to implement our cloud architecture using cloud technology.

6.1 Our construction of RCCA secure ABE-OS-KEM

Our ABE-OS-KEM applies the ideas of outsourced decryption from [7] to to the Rouselakis/Waters ciphertext-policy ABE scheme [11]. As in [7], we achieve security against replayable chosen-ciphertext attacks (RCCA, [7]) via the Fujisaki-Okamoto transform [5]. Our architecture from Section 4.1, and especially the separation between the session and FileCrypt services relies on the properties of a key encapsulation mechanism. Therefore, we have modified the original encryption scheme [7] to adhere to the definition of a key encapsulation mechanism. Furthermore, we explicitly describe our scheme in the efficient type-III setting of bilinear groups [3].

For convenience, we assume the data encapsulation mechanism used in conjunction with our ABE-OS-KEM to use keys from $\{0,1\}^{\Lambda}$. The scheme is defined as follows:

Setup(1^{Λ})**:** compute RW master secret $msk^{\mathrm{RW}}_{\mathrm{ABE}} = \alpha$ and public parameters $pub^{\mathrm{RW}}_{\mathrm{ABE}} = ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e), \{g_i, u_i, h_i, v_i, w_i\}_{i\in\{1,2\}}, e(g_1, g_2)^{\alpha})$, where p is a Λ-bit prime, $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map and generator $g_1 \in \mathbb{G}_1$, generator $g_2 \in \mathbb{G}_2$ and $\alpha \in \mathbb{Z}_p$ are chosen uniformly at random. Parameters $a, b, c, d \in \mathbb{Z}_p$ are chosen unifomly at random and are used to compute $\forall i \in \{1, 2\} : u_i = g_i^a, \quad h_i = g_i^b, \quad v_i = g_i^c, \quad w_i = g_i^d$. Choose hash functions $F : \{0,1\}^* \to \mathbb{Z}_p$, $H_1 : \mathbb{G}_T \times \{0,1\}^{\Lambda} \to \mathbb{Z}_p$ and $H_2 : \mathbb{G}_T \to \{0,1\}^{\Lambda}$. Output $pub_{\mathrm{ABE}} = (pub^{\mathrm{RW}}_{\mathrm{ABE}}, F, H_1, H_2)$ and $msk_{\mathrm{ABE}} = msk^{\mathrm{RW}}_{\mathrm{ABE}}$.

KeyGen$(pub_{\mathbf{ABE}}, msk_{\mathbf{ABE}}, A_{ID})$**:** compute RW secret key for attribute set A_{ID}, i. e. pick $r_{ID}, r_{a_1}, \ldots, r_{a_{|A_{ID}|}} \xleftarrow{\$} \mathbb{Z}_p$. Let $K_0 = g_1^{\alpha} w_1^{r_{ID}}$, $K_1 = g_1^{r_{ID}}$, and for all $a_i \in A_{ID}$: $K_{a_i,2} = g_1^{r_{a_i}}, K_{a_i,3} = \left(u_1^{F(a_i)} h_1\right)^{r_{a_i}} v_1^{-r_{ID}}$. Output $sk_{\mathrm{ABE},ID} = (K_0, K_1, \{K_{a_i,2}, K_{a_i,3}\}_{a_i \in A_{ID}})$.

Encaps$(pub_{\mathbf{ABE}}, \mathbb{A})$**:** parse $\mathbb{A}$ as (M, ρ) with $M \in \mathbb{Z}_p^{\ell \times n}$ and row labelling $\rho : [\ell] \to \{0,1\}^*$. Pick $R \xleftarrow{\$} \mathbb{G}_T, k \xleftarrow{\$} \{0,1\}^{\Lambda}$. Set $s := H_1(R, k)$ and $r := H_2(R)$. Let $C' = k \oplus r$. Pick $y_2, \ldots, y_n, t_1, \ldots, t_\ell \xleftarrow{\$} \mathbb{Z}_p$. Set $\lambda := M \cdot (s, y_2, \ldots, y_n)^\top$; denote by λ_i the ith component of λ. Let $C := R \cdot e(g_1, g_2)^{\alpha s}$, $C_0 := g_2^s$ and for all $i \in [\ell]$: $C_{i,1} := w_2^{\lambda_i} v_2^{t_i}$, $C_{i,2} := \left(u_2^{F(\rho(i))} h_2\right)^{-t_i}$, $C_{i,3} := g_2^{t_i}$. Define $CT_{\mathrm{ABE}} := ((M, \rho), C, C', C_0, \{C_{i,1}, C_{i,2}, C_{i,3}\}_{i \in [\ell]})$. Output (k, CT_{ABE}).

TransKey$(pub_{\mathbf{ABE}}, sk_{\mathbf{ABE},ID})$**:** pick $z \xleftarrow{\$} \mathbb{Z}_p$ and set $sk_{\mathrm{PK},ID} := z$ and $tk_{ID} := (K'_0 = K_0^{1/z}, K'_1 = K_1^{1/z}, \{(K'_{a_i,2} = K_{a_i,2}^{1/z}, K'_{a_i,3} = K_{a_i,3}^{1/z})\})$. Output $(sk_{\mathrm{PK},ID}, tk_{ID})$.

Transform$(pub_{\mathbf{ABE}}, tk_{ID}, CT_{\mathbf{ABE}})$**:** if the ciphertext is malformed or tk_{ID} does not satisfy $\mathbb{A} = (M, \rho)$, output $\perp$ and exit. Otherwise, let $I \subseteq [\ell]$ be a satisfying set of $\mathbb{A}$ with respect to tk_{ID}, i. e. there are $b_i \in \mathbb{Z}_p$ such that $\sum_{i \in I} b_i M_i = (1, 0, \ldots, 0)$, where M_i denotes the ith row of M. Compute $B' := \prod_{i \in I} \left(e(K'_1, C_{i,1}) e(K'_{\rho(i),2}, C_{i,2}) e(K'_{\rho(i),3}, C_{i,3})\right)^{b_i}$ and $B := e(K'_0, C_0)/B'$. Output $CT_{\mathrm{PK}} = (C, C', B)$.

Decaps$(pub_{\mathbf{ABE}}, sk_{\mathbf{PK},ID}, CT_{\mathbf{PK}})$**:** parse $CT_{\mathrm{PK}} = (T_0, T_1, T_2)$ and compute $R := T_0 / T_2^z$, $k := T_1 \oplus H_2(R)$ and $s := H_2(R, k)$. Check whether $T_0 = R \cdot e(g_1, g_2)^{\alpha s}$. If the check fails, output $\perp$, otherwise output k.

The scheme's correctness and selective RCCA security follow from the respective properties of RW ciphertext-policy ABE [11] and the Fujisaki-Okamoto transformation [5], after applying the obvious modifications required due to outsourced decryption.

The major performance advantage of an ABE-OS-KEM in our architecture results from splitting the Decaps algorithm of an ABE-KEM into two separate Transform and Decaps steps (cf. Definition 1 and Definition 2). The complexity of Decaps in the ABE-OS-KEM is now independent of $sk_{\mathrm{ABE},ID}$ and CT_{ABE}, and hence independent of the user's permissions and of the ciphertext's policy. This allows us to handle users with a large set of permissions and complex access policies in our cloud scenario that involves resource constrained HSMs.

In concrete instantiations, arithmetic in $\mathbb{G}_1$ is much more efficient than in $\mathbb{G}_2$ [3]. In our setting, the resource constrained HSM performs TransKey, while the TC server executes Encaps. Our ABE-OS-KEM accounts for this by placing user secrets $K_{i,j}$ as arguments of TransKey in group $\mathbb{G}_1$ and ciphertext components of $C_{i,j}$ as arguments of Encaps in $\mathbb{G}_2$.

6.2 Implementation

As a proof of concept, we have implemented our service architecture from Section 4 based on the following technologies:

Docker: We have implemented the sub-services of our ACE service as separate Docker[6]-based micro services. This supports elasticity and multi-tenancy because we can create and destroy service instances for each user, based on the current load situation.

Kubernetes: We use Kubernetes[7] for orchestrating, scheduling, and monitoring the ACE sub-services. We operate a Kubernetes cluster that consists of four nodes. Each node runs with an Intel Xeon E3 at 2.3 GHz and 8 GB RAM.

RabbitMQ: We have implemented a queue between the FileCrypt service and the session service based on RabbitMQ.[8] Then, our queuing mechanisms allows us to dynamically assign TC-HSMs to active users.

Amazon S3: The FileCrypt service is compatible with the Amazon S3[9] interface. Hence, we can use a commercial cloud storage provider to host the encrypted files.

WebDAV: We have implemented a WebDAV service as the user front-end to ACE, with the session service serving as its back-end. Our implementation extends the Go[10] WebDAV implementation to support ABE policies. The extension passes policies as so-called WebDAV dead properties to the FileCrypt service. Since standard WebDAV clients do not support this mechanism, we have implemented a graphical user interface that allows us to define policies for file upload (encryption).

Our implementation shows that it is practical to implement an ABE service with modern cloud technology.

7 Future Work

As part of future research, we will enhance our design by various services. In particular, we want to realize a service for searchable encryption as introduced by Song et al. [13], granting authorized users the ability to efficiently search encrypted data. Another line of future research aims at including the multi-authority feature of Chase [2] into our cloud. The

[6] `https://docker.com`
[7] `https://kubernetes.io`
[8] `https://www.rabbitmq.com`
[9] `https://aws.amazon.com/s3`
[10] `https://golang.org/`

multi-authority feature allows the cloud resources of to be pooled among multiple instantiations of the cloud, while keeping everything beyond the hardware separate. This can help with removing the single trusted global provider for key generation that we assume in Section 5.1.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy. pp. 321–334 (2007)
2. Chase, M.: Multi-authority attribute based encryption. In: 4th Theory of Cryptography Conference, TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer (2007)
3. Chatterjee, S., Menezes, A.: On cryptographic protocols employing asymmetric pairings — the role of Ψ revisited. Discrete Applied Mathematics 159(13), 1311–1322 (2011)
4. Cloud Security Alliance: SecaaS implementation guidance category 8: Encryption (2012), `https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_8_Encryption_Implementation_Guidance.pdf`, Checked 2017-07-06.
5. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. J. Cryptology 26(1), 80–101 (2013)
6. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptology 20(1), 51–83 (2007)
7. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: 20th USENIX Security Symposium. USENIX Association (2011)
8. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. Chapman and Hall/CRC Press (2015)
9. Mell, P., Grance, T.: The NIST definition of cloud computing (2011), `http://dx.doi.org/10.6028/NIST.SP.800-145`, Checked 2017-07-06.
10. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
11. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption. In: CCS '13. pp. 463–474. ACM (2013)
12. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer (2005)
13. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy. pp. 44–55. IEEE (2000)
14. Yang, K., Jia, X., Ren, K.: Attribute-based fine-grained access control with efficient revocation in cloud storage systems. In: ASIA CCS '13. pp. 523–528. ACM (2013)
15. Yang, Y., Liu, J.K., Liang, K., Choo, K.R., Zhou, J.: Extended proxy-assisted approach: Achieving revocable fine-grained encryption of cloud data. In: ESORICS 2015. LNCS, vol. 9327, pp. 146–166. Springer (2015)
16. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: INFOCOM 2010. pp. 534–542. IEEE (2010)
17. Zhang, P., Chen, Z., Liu, J.K., Liang, K., Liu, H.: An efficient access control scheme with outsourcing capability and attribute update for fog computing. Future Generation Computer Systems pp. – (2016)
18. Zhao, F., Nishide, T., Sakurai, K.: Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems. In: Information Security Practice and Experience ISPEC 2011. LNCS, vol. 6672, pp. 83–97. Springer (2011)