Feasibility of End-To-End Encryption using Attribute Based Encryption in Health Care

Dennis Schroer
g.h.schroer@student.utwente.nl

University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands

2016

Supervisors:
dr. A. Peter
dr. A. Vasenev

UNIVERSITY OF TWENTE.

topicus

Abstract

Attribute-Based Encryption (ABE) is an encryption technique which allows to provide end-to-end encryption in situations where data is shared with multiple users, based on their roles or properties of the data. This is especially useful in situations where data is stored in an online environment, like cloud storage services.

We propose a construction using ABE which allows to meet certain requirements that are, in our view, required for ABE to be practically feasible in an application. We provide an implementation of this construction, as well as implementations of several ABE schemes. Multiple experiments are performed to investigate the feasibility and performance of four selected ABE schemes in this construction, by means of an example case of an health care application.

Our analysis shows that the application of ABE is feasible, as long as several conditions are met. Not all of the selected schemes turn out to be feasible, and the use of devices with limited computational resources yield unfeasible results. However, when these conditions are met, the overall performance is acceptable.

Contents

1 Introduction

For my Master's thesis, I investigate the feasibility and performance of Attribute-Based Encryption (ABE). This relatively new approach to encryption interests me because it offers to provide end-to-end encryption, while at the same time it offers high expressiveness in defining who has access to the data. Especially in the area of cloud storage, this can be useful. My curiosity on the practical application of ABE and on the performance of such encryption leads to this research.

This thesis is the result of the research, which I conducted in cooperation with Topicus. Topicus is an innovative ICT service provider, active in multiple sectors and specialized in integrations, Software as a Service (SaaS) and process management. One of the sectors they focus on is Health Care, offering services to improve information exchange between different health care parties.

Attribute-Based Encryption ABE is an encryption technique in which data can be encrypted based on attributes. Attributes are assigned to users, and the data is protected using an access policy over these attributes. Only when the attributes of the user satisfy the access policy, the user is able to decrypt the ciphertext.

As an example, Alice encrypts a document with the access policy $Doctor \wedge Heart$, indicating that only classified cardiologists are allowed to access the file. Bob's attributes include $Doctor$ and $Heart$, and thus satisfy the access policy. Bob is therefore able to decrypt the ciphertext. The example is visualized in Figure 1.

The attributes are issued to the users by one or more trusted authorities. Each attribute has an associated public and secret key. The public keys are used during encryptions, while the secret keys are used to generate the user keys.

ABE is useful when data is stored in an online environment, like a cloud storage service. Especially in situations where data has to be shared with multiple users based on user roles or data properties, ABE promises to offer fine grained access control while still offering end-to-end encryption. Even when the stored data is out in the open, malicious users are still unable to access the data.

In these situations, ABE also provides less key overhead compared to traditional encryption methods. Traditional encryption methods can provide end-to-end encryption, but for shared data either users share the same decryption keys or the data is stored in multiple instances, encrypted with different keys. Both results are undesirable.

Much research has been performed to ABE and the possible uses of it. However, the practical feasibility of ABE in a realistic scenario with write access control, authentication and access revocation has not been tested. In

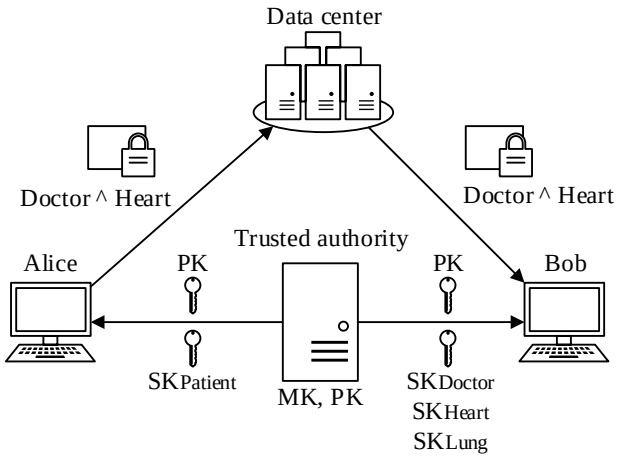


Figure 1: Example of ABE. Alice encrypts a document with the access policy $Doctor \wedge Heart$. Because Bob's attributes satisfy the access policy, he is able to decrypt the ciphertext.

a realistic scenario, users are able to securily exchange data using end-to-end encryption, while the data itself is stored in the cloud. Furthermore, users are able to grant write access to stored data, such that these users are able to securely update already stored data. Moreover, users are able to grant access to other users, while they can also revoke access.

In this paper, we investigate the feasibility of ABE in such a realistic scenario, in the context of an example case developed with Topicus. This example case allows users to securely share data with health care professionals employed by an insurance company, while employees of the insurance company are unable to access the data.

For this, we propose a construction which enables to satisfy these requirements of write access control, authentication and access revocation and show that the performance allows for practical use.

Contribution Our contribution comprehends four parts:

- We investigate the requirements which are, in our view, needed for ABE to be applicable in practice, in the context of the healt care data center.
- We combine several solutions in one new approach to create a scheme able to provide end-to-end encryption in a scenario in which the requirements of write access control and access revocation are met.
- We provide implementations or improvements to implementations for several ABE schemes.

- We run several experiments in order to analyze the performance and scalability in a health care data center scenario in which the requirements are coped with.

Paper organization The remainder of the paper is organized as follows. At first, we look in more details at ABE in section 2 and related work in section 3. Next, the problem statement and research goal are given in section 4. In this section, we also describe the requirements for ABE to be practically feasible. The requirement of write access control and possible approaches are discussed in section 5, while section 6 covers approaches on access revocation. Section 7 describes the example case, and section 8 compares several ABE schemes on their usability for the example case. In section 9, we propose our construction. The implementation of both the construction and the ABE schemes are described in section 10. Experiments and the results are discussed in section 11, while the conclusions and discussion can be found in sections 12 and 13.

2 Overview on Attribute-Based Encryption

Attribute-Based Encryption (ABE) is a technique in which data is encrypted and decrypted using a set of attributes ω. Attributes are assigned to users, and the secret keys of the users are based on these attributes. A user is only able to decrypt a ciphertext if his assigned attributes fulfil the access policy γ of the ciphertext.

The access policy is a collection of non-empty subsets of the attributes [5]. In most schemes, the access policy is represented by a tree containing logical operators in the nodes, while the attributes are the leaves in the tree. However, other representations exist.

An access policy γ is satisfied by a set of attributes ω if at least one of the sets in the access policy is a subset of the attributes ω of the user. Using access policies provides a way to incorporate access control in the encryption, as only users who possess enough attributes needed to fulfil the access policy are able to decrypt the ciphertext.

The attributes are issued by an entity called the *attribute authority*. The attribute authority is trusted to only hand out attributes to users who are eligible.

An important aspect of ABE is collusion prevention: it should not be possible for multiple users to combine their keys in order to gain access to data they did not have access to before. In practice, this is achieved by embedding a random component in the secret keys of a user, which is cancelled out when used in decryption.

2.1 Types of Attribute-Based Encryption

Two types of Attribute-Based Encryption exist: Key Policy based Attribute-Based Encryption (KP-ABE) [9] and Ciphertext Policy based Attribute-Based Encryption (CP-ABE) [6]. In KP-ABE, the access policy is associated with the users, while the data is encrypted using a set of descriptive attributes. In CP-ABE, this is the other way around. The data is encrypted using an access policy, while the users possess a set of attributes. In the first case, the access policy determines which data the user is able to decrypt. In the latter case, the access policy associated with the data determines which users are able to decrypt the ciphertext.

We focus on CP-ABE, because this approach offers the best control of data access for the users. Furthermore, in this approach the attributes can be used to express the roles of the users, which is best suited for our example case.

2.2 Multiple authorities

In the first ABE schemes, distributing the task of the attribute authority among multiple parties was not possible. This single attribute authority possessed master keys, with which it was able to generate keys for attributes as well as the secret user keys. This introduced the problem of *key escrow*: a single authority is able to generate keys for all attributes, and therefore can access all data. Were the authority to be breached, malicious users could also access all data.

Furthermore, using a single attribute authority often does not fit a real scenario. Consider a scenario where there are multiple parties, for example, a hospital managing its employees and a national database containing registered doctors. Having a single authority would mean there is one party responsible for both roles.

Therefore, most recent researches allow to use multiple authorities. Instead of one authority, there are multiple authorities which are each responsible for a disjoint subset of the attributes. When encrypting data using attributes from multiple attribute authorities, breach of a single attribute authority does not allow malicious users to decrypt the data.

To avoid collusion of user keys, a random element can no longer be applied to the secret keys. Otherwise, the user keys from multiple authorities can not be used simultaneously. To mitigate this problem, a global user identifier is introduced and embedded in the secret keys issued to the users.

2.3 Algorithms

In Multi-Authority Attribute-Based Encryption (MA-ABE), a central authority is required to generate global public parameters. These parameters make sure that the keys of the attribute authorities can be used together in the encryption. The central authority is only required in the setup phase.

In general, the schemes consist of the following steps:

Setup The central party generates global public parameters GP. These public parameters ensure that keys from the attribute authorities can be used together, and are required for the setup of the authorities.

Authority setup The attribute authority AA_i generates public keys PK_i and secret keys SK_i for the attributes ω_i this authority is responsible for, using the global public parameters GP.

Encryption The message M is encrypted using an access policy γ and the public keys PK_i for each attribute authorites AA_i of which at least one attribute occurs in the access policy γ. It outputs the ciphertext C.

KeyGen The attribute authority AA_i creates secret keys $UK_{u,i}$ for the set of attributes $\omega_{u,i}$ managed by this authority and owned by user $u \in U$. The secret keys SK_i and public keys PK_i of the authority, as well as a global user identifier GID_u of the user are used in the creation of these keys. The user identifier is included to avoid collusion of the user keys. It outputs a decryption key $UK_{u,i}$.

Decryption The algorithm takes as input the ciphertext C encrypted with access policy γ and, for each attribute authority AA_i of which at least one attribute is used in the policy, the decryption keys $UK_{u,i}$ of the user and the public keys PK_i of the attribute authorites. If the attributes ω_u used in the generation of UK_u (union of all $UK_{u,i}$ for user u) satisfy the accesspolicy γ of the ciphertext, the algorithm is able to decrypt the ciphertext and outputs the message M. If the access policy can not be fulfilled, the algorithm outputs an error symbol $\perp$.

2.4 Attribute Universe

Another way to categorize ABE schemes is to look to the attribute universe. Most schemes support a *small attribute universe*, which means that the possible number of attributes is limited, because each attribute requires its own private and public key which have to be initialized during setup.

However, in a *large attribute universe* each attribute authority has only a constant number of public and private keys. By using a hash function, each string can be used as an attribute.

3 Related work

In this section, related work is analyzed. First, the related work on ABE is investigated, with a focus on schemes supporting multiple authorities. Next, ABE schemes supporting access revocation are explored, after which existing approaches to authentication in ABE are analyzed. Finally, proposed applications using ABE are investigated.

3.1 Multiple Authorities in Attribute-Based Encryption

The Fuzzy Identity-Based Encryption (FIBE) scheme proposed by Sahai et al. [22] can be seen as the predecessor of ABE. In this scheme, a user has a private key for the attribute set ω and is able to decrypt a ciphertext encrypted with the attribute set $\omega^{'}$ only when the two sets overlap with a certain threshold value, thus introducing some fault tolerance.

Chase et al. [8] extend this scheme to support multiple authorities. The scheme only allows to define that the decryptor should possess at least d_k attributes for each authority k, without specifiying which attributes.

As opposed to this limitation, the multi-authority scheme proposed by Lewko et al. [13] supports any boolean access policy and does not require a central authority, except for setup. Authorities do not have to be aware of each other and can generate their own public and private keys, based on some predetermined public parameters. However, users have to reveal their identifier and the scheme does not support a large attribute universe.

Rouselakis et al. [20] improve the scheme of Lewko et al. by proposing a scheme which both supports a large attribute universe and improves the performance. Instead of the authorities having a public and private key for every attribute, each authority has just one public and private key. By using a hash function a random string can be mapped to a group element, which makes it possible to use any attribute.

Each attribute is uniquely linked to an authority by appending the identifier of the authority to each attribute name. This way, different authorities can still issue an attribute with the same id, while the keys are different.

Yang et al. [23] propose data access control for multi-authority cloud storage based on ABE. The scheme allows to partially outsource decryption to the cloud. The server calculates a decryption token using the attribute keys of the user. To make sure that the server is not able to decrypt the ciphertext, a public and private key are issued to each user by the central authority. To avoid the central authority to be able to decrypt ciphertexts, each attribute authority calculates its own keys independently of the other authorities.

The scheme also allows to efficiently revoke attributes. An attribute authority can revoke an attribute by updating the keys for this attribute, while at the same time computing update keys for non-revoked uses and

ciphertexts. The latter can be used for proxy re-encryption.

Although the scheme has many features, it does not offer a solution to control write access. Furthermore, due to the possiblity to outsource decryption, additional keys are added to the system.

Rao et al. [19] propose a scheme with fast decryption. As opposed to other schemes, the size of the ciphertext is not linear to the number of attributes in the policy, but the number of authorized sets.

In this scheme, the access policy should be expressed in disjuctive normal form, in other words as a single OR gate over multiple AND gates. This makes the scheme somewhat limited, while it still it has the same expressiveness as any binary tree of other schemes. As a result of this, and the fact that for each authorized set the messages is re-encrypted, the size of the ciphertext can be large.

A temporal attribute based access control scheme is proposed by Yang et al. [24]. Data is encrypted for one or more time periods. Secret keys for attributes are issued only once to each user, and special update keys are published for each timeslot. The scheme is useful when access grants are in most cases limited to a certain time period only.

For each attribute, a binary tree is created in which users are assigned to the leaves. The keys of users are based on the values on the nodes from root to leaf. This allows to efficiently revoke attribute from users by simply only updating the minimum set of nodes covering all non-revoked users. On the other hand, it also increase the amount of keys, and thus the size of storage required.

For each time period, update keys have to be calculated. Furthermore, the number of users who could possess an attribute is limited to the size of the binary tree, which is determined during setup.

All of these schemes offer various features, but none of the schemes provides a complete solution which allows access revocation, write access control and multiple authorities.

3.2 Access revocation

Ibraimi et al. [12] introduce a mediated form of the ABE. In this scheme, the secret keys of the users are split in two parts. A semi-trusted mediator owns the first part of the secret key, while the user owns the second part. Using an Attribute Revocation List, The mediator checks whether attributes are revoked and, if the non-revoked attributes still satisfy the policy, outputs a new ciphertext which is decrypted (or rather re-encrypted) with the mediator's part of the secret key.

Yu et al. [25] combine CP-ABE with proxy re-encryption in order to be able to revoke attributes at any time. The access policy is represented by a single *and* gate, while each attribute can have three occurrences: *positive*, *negative* and *don't care*. When an attribute is revoked, the trusted party

updates the public key, generates proxy re-key's and transmits these to semi-trusted proxy servers. These proxy servers then can re-encrypt existing ciphertexts and update user secret key components if necessary. A version number is used to track the version of the current keys. The proxies keep a list of re-keys, which also makes it possible to aggregate multiple updates. As a result, the proxy is also able to re-encrypt the ciphertext only when this ciphertext is requested.

A drawback of this scheme is that users are required to have a key for every existing attribute, even though they do not possess all attributes. Furthermore, each ciphertext contains a part for every existing attribute.

A disadvantage of both schemes is that they introduce another party which has to be online at all time. Another disadvantage of both schemes is that all chipertexts have to be re-encrypted when an attribute is revoked. For [12], this re-encryption is required for each decryption. In [25], the re-encryption is required only when attributes are revoked. This leads to a lot of computational overhead. Furthermore, the need to send the updated ciphertexts introduces additional traffic and delay.

Attrapadung et al. [4] introduce a so called Broadcast Attribute-Based Encryption (bABE) scheme. The scheme has a single authority and can be used for both KP-ABE and CP-ABE. The access policy is combined with a user set $S \subseteq U$, where U is the universe of user identifiers. Users in S have access to the data, given that their attributes satisfy the other part of the access policy. Revocation is possible by setting $S = U \setminus R$, where R contains the identifiers of users whose access is revoked

However, as the policy is updated when a user is revoked, there is need to re-encrypt the data for which the user is revoked. Furthermore, the identity of each user has to be known.

Although the scheme does not support multiple authorities, it is possible to construct a disjunctive multi-authority ABE from the scheme at the cost of removing the revocation from the scheme. Moreover, this requires that the underlying schemes supports access delegation.

In all these schemes, re-encryption is required on revocation. Existing schemes attempt to move this re-encryption to the cloud, by introducing an additional mediator or proxy which should be online at all times. The mentioned bABE scheme is not sufficient either, as it does not support multiple authorities.

3.3 Authentication

Ruy et al. [21] propose a decentralized scheme which combines ABE with Attribute-Based Signatures (ABS) to authenticate the user. Attributes are used in ABS to sign a message, proving that the user owns certain attributes according to a claim policy.

A user first has to obtain a token from a trusted party. Using this token, he is able to request keys for encryption, decryption and signing from the authorities. Replay attacks are prevented by adding a timestamp to the token. This also makes sure that a revoked user is unable to create or update data when his access is revoked.

However, revocation other than simply no longer issuing this timed keys is not possible. This means that the possibility to authenticate can be revoked in some sense, but not the ability to decrypt data. Furthermore, the scheme is not implemented, so a practical application of the scheme is not tested.

Li et al [14] propose a scheme which uses signatures with a time period embedded to authenticate a user in order to control write access. Only within these time periods, the user is allowed to update the data. The signatures are provided by the user's organization or the data owner.

Although the data owner does not always has to be online, he still has to know in advance when the other should have write access, and should come online periodically to re-grant access. Moreover, the data owner has to know exactly who has write access, while for read access a description using attributes is sufficient. Revocation of write access is only possible by simply not providing signatures any longer.

So although both approaches introduce authentication, both schemes also have their own drawbacks. Existing techniques using attributes for authentication lack the support for revocation, and are not tested in a practical application. Other approaches require the need to know the identities of users.

3.4 Applications of Attribute-Based Encryption

Proposed applications of ABE often involve Electronic Health Record (EHR) systems. This section discusses related work performed in this area, where the focus is on techniques applying ABE.

Akenyele et al. [1] propose an EHR system using dual ABE, a combination of CP-ABE and KP-ABE. They present a policy engine which proposes access policies, based on the author and the content of the record.

However, a single attribute authority is used, which is kept offline to narrow the attack model. Furthermore, they do not discuss how write access control is managed, even though the data records are created in a hierarchical structure.

EHR systems using ABE are also proposed in [3] and [18]. However, both constructions use a single attribute authority. In [3], an implementation is analyzed, but no details about the used ABE schemes are given. Furthermore, the experiments are only repeated several times.

In [18], KP-ABE is employed. To cope with the fact that the access policy is specified beforehand and not changeable, access delegation is applied.

EHR with keyword search Narayan et al. [16] construct an EHR system which uses bABE to guarantee confidentiality and privacy, while allowing to revoke access of users. This allows patients to share health data with health care providers. However, a single trusted authority is used.

Additionally, a primitive called *Secure Channel Free Public-Key Encryption with Keyword Search* is used to enable a keyword search over the encrypted data, without revealing information about the used keyword itself.

In this scheme, the health care providers manage the keys and attributes. After each session, all data on the client is deleted to prevent adversaries to retrieve data.

In the scheme, the patient is the only entity able to update the data or the policy. Furthermore, the given scheme is only proposed and not implemented, so the efficiency of the scheme is questionable.

Multiple domains Some proposed applications divide the system in two security domains, the *professional domain* and the *personal domain*. The first consists of users who access the data for professional use, such as doctors and medical researchers, while the second consists of users personally associated with the EHR owner, such as family and close friends. Examples of these are [11] and [14].

In [11] by Ibraimi et al., a single trusted authority is used for the professional domain and the attributes describe the characteristics of the users. For the personal domain each patient has his own trusted authority, while the attributes categorize the data. The scheme lacks support for access revocation.

Because of the use of a single trusted authority, the scheme is unable to cope with the *key escrow* problem. Furthermore, the use of two domains require each user to describe the data in two ways.

In [14] of Li et al., multiple authorities are applied in the professional domain. Attributes describe the roles of users and are divided in types which are divided among multiple authorities. To avoid key escrow, each access policy has to contain at least one attribute of each type, while wildcards are allowed. In emergency situations, temporarily read keys can be granted by the emergency department.

In the personal domain, KP-ABE is applied. The attributes are based on intrinsic properties of the records, while keys are managed and distributed by the user himself. Proxy re-encryption is applied to make attribute revocation possible. Moreover, the scheme employs write access control using signatures issued by data owners or organizations, as described earlier in Section 3.3

As with the other scheme, the user has to describe the data in two ways: one time by describing which roles the users should have, and one time to describe the properties of the data.

Most of the proposed applications employ only a single authority, thus

suffering from the *key escrow* problem. Other schemes use two separate domains with different policies, requiring the data to be described and encrypted twice. Moreover, none of the approaches provides a way to securely regrant access.

4 Research goal

ABE is investigated in many researches, but existing solutions lack several features which are, in our view, required for ABE to be practically feasible in most applications. There are several open problems, which should be coped with before ABE can be used in such an application.

The first open problem involves *write access control*. In most realistic scenarios, data should also be updatable. However, current research mainly focusses on the control of read access. Additional measures are required to enable secure control of write access.

Write access requires validation of the rights of the updating user. This means that there should be some way to tell which users are allowed to update the data, as well as a means to validate this. Likewise, such validation is required for updates of the access policy.

Another problem is the problem of access revocation. Several different approaches are proposed, but each approach comes at a cost. Determining which approach is best suited to enable access revocation is a problem which depends on the application and the requirements. This should be considered carefully.

We now state the goal of this research. Next, several topics of interest are investigated in more detail.

> *The goal of this paper is to investigate the feasibility and performance of ABE in a realistic scenario in which several actors securely exchange data, on a relatively small scale.*

4.1 Topics of interest

Realistic scenario In a realistic scenario, users want to securely share data with other users. Securely here means that the data is encrypted using end-to-end encryption. In a realistic scenario, users should not only be able to read data, but also update this data while other users with access rights can still access the updated data. Furthermore, users should be able to control who has rights to access the data. Access revocation should also be possible.

Feasibility ABE is considered feasible in the realistic scenario, when the scheme allows to securely perform all the required actions (data update, policy update and access revocation), while the performance is still acceptable. To enable this, several requirements should be met.

The first requirements (**R1**) involves *authentication and write access control*. Users should be able to update data, but only when they have the rights to do so. Validation of the permitted actions of a user is required to enable secure updating of data.

Secondly, it should be possible to grant access to other users when data is already shared with several users, in a secure manner (**R2**). This can be achieved by an update of the policy. However, the owner of the data should be the only entity allowed to update the access policy.

The third requirement (**R3**) is that *access revocation* should be possible. This is an important aspect in the application of ABE, as it both enables temporal access and the possibility to revoke attribute of users in case of job resignation or leaked credentials. Moreover, the robustness of the system is improved, and it allows to respond to breaches.

Furthermore, we require that in a feasible application multiple authorities are used (**R4**). Having only a single authority introduces the so called *key escrow* problem: the single authority has a master key, and is therefore able to generate keys for all attributes. Were the authority to be breached, malicious users could access all data.

By using multiple authorities, this problem can be mitigated by enforcing policies to contain attributes from multiple authorities. No single authority has a master key for all attributes, and data can be encrypted using attributes from multiple authorities. This way, all authorities should be breached before any data can be accessed.

The last requirement is therefore that the application should utilize *multiple authorities*.

None of the currently proposed constructions using ABE both meets al these requirements and is tested on feasibility and performance in a realistic scenario.

The requirements for a feasible application of ABE are summarized in Table 1.

Table 1: Requirements for an application of ABE to be feasible.

Requirement	Motivation
R1 Authentication and write access control	Users should be able to securely update data
R2 Access regranting	Users should be able to grant access to additional users
R3 Access revocation	Temporal access and changes in user's attributes should be possible
R4 Multiple authorities	Avoids key escrow and better represents real world

Performance We analyze performance of the applied schemes regarding required time and resources that the algorithms take to run. We focus on time duration and storage requirements. However, the CPU usage, network traffic and memory usage can also be of interest.

We aim attention at the performance and resource usage of the schemes. The main goal is to provide insight into the current performance of the most suitable ABE schemes, and compare them. We aim to give insight in whether the performance is acceptable, although this can change in the long term and it being acceptable highly depends on the type of application. Using this, we can also see where there is room for improvement, for example because a certain operation requires much time. Moreover, we can state which schemes are preferred.

Scale We test our goal using an example case in which the stated requirements are present. Furthermore, the effect of other inputs on the performance is investigated, to see how well the schemes scale. For example, the influence of the number of attributes or the size of the policy on durations and storage is investigated.

The example case itself is on a relative small scale. In this case, a user is enabled to securely share data with health care professionals acquired by an insurance company. The professionals act as reviewers, which have temporal access to the data shared by users in order to judge the medical need of a surgery.

As we focus on the feasibility of performance of ABE, we only consider such a case with a limited number of authorities and users. The scalability of ABE is another problem. However, we expect ABE to scale well, and also discuss what the expected implications on a large scale will be.

5 Write access and authentication

In this section, the problem of authentication and write access control is investigated. Authentication means that the claimed identity of a user is verified. Write access control means that the rights to write access, for instance to update some data, can be controlled and verified. In an attribute-based scenario, users are not identified by a user identifier, but attributes are used instead.

We assume that the write access is enforced with a *write policy*. This write policy determines whether a user is allowed to update the data; Only when the user has enough attributes to satisfy the write policy, he is allowed to update the data. The access policy can therefore be separated in two access policies: the *read policy* and the *write policy*.

To enable write access control, the updating user should prove that he owns enough attribute to satisfy the write policy. This authentication is therefore required to enable write access control.

Firstly, traditional access control mechanisms are discussed, as well as why these are not sufficient for an attribute based scenario. Next, alternative approaches are discussed. The focus is on signatures schemes, as these provide both *authenticity* and *integrity*. Authenticity means that the origin of a message can be verified, while integrity concerns the verification that a message is unchanged.

5.1 Traditional access control

In a traditional server-client scenario, the client proves his identity by presenting credentials like a password, key or uniquely generated code to the server. The server has a great responsibility, as it has to validate these credentials and, if valid, enable the user to perform certain actions like accessing or updating data. Both the authentication (validating the user's identity) and authorization (determining the access of the user) are performed by the server, so much trust in the server is required.

However, an advantage of ABE is that it allows to provide security in cases where the server is trusted as little as possible. It is preferable that for the control of write access, the required trust in the server is also limited as much as possible. Therefore, the server should perform as few checks as possible. Traditional access control approaches are thus not sufficient for our case.

5.2 Signatures

The most used algorithm for creating signatures uses a hash of the message and a public and private key-pair. When a signature is created, the hash of the message is calculated using a predetermined hashing function. The

Message Authentication Code (MAC) is computed based on this hash and the secret key.

The receiver can validate the signature by using the corresponding public key to retrieve the hash from the signature and comparing this hash to the calculated hash of the message. If the hashes are equal, the receiver can be sure that the signature was created by a user having the private key.

5.3 Private key with write policy

In order to apply such a signature to prove that the signing user possesses enough attributes to satisfy the write policy, the private key can be encrypted using ABE with the write policy. When a data record is created, the creator of the data record generates a random public and private key, and signs the record using the private key. The public key is added to the data record in plain text, while the private key is encrypted using ABE with the write policy and added to the data record.

When a user wants to update the data, he first encrypts the new data. He then decrypts the encrypted secret key for the signature using his attributes satisfying the write policy, and signs the data using the obtained secret key. The updated encrypted data and the signature are sent to the storage server.

The receiver validates the update by verifying the signature using the updated data and the already stored public key. If the check is successful, the receiver knows the data is signed with the private key corresponding to the public key, which can only be accessed by users with enough attributes to satisfy the write policy. Thus, the data is updated by a user with sufficient rights.

5.4 Attribute-Based Signature

Another approach is using the attributes directly to create a signature. This so called Attribute-Based Signatures (ABS) [15] is a relatively new technique which enables a user to sign a message using a subset of the attributes he owns. The signature assures that the message is sent by a user whose attributes satisfy a certain predicate.

The attributes are handed to the users in the form of secret keys, and are distributed by attribute authorities, comparable to the authorities in ABE.

The predicate is in the form of a boolean formula and can be used by the user to express a certain claim. The predicate can, depending on the used scheme, contain *and*, *or* and threshold operations. For example, a user can use the predicate $Doctor \vee Dentist$ to sign a message, while he only uses his attribute *Doctor* in the signature. The predicate itself can be of any size, as long as the attributes of the user satisfy the predicate and the predicate is acceptable by the other party.

An important property of ABS is *anonymity* [15]. This means that the signature does not reveal the identity of the user, or the way in which the signature was created. It only shows to the verifier that the user's attributes satisfy the predicate, and nothing more.

Equal to ABE, collusion prevention is an important requirement of ABS. Users should not be able to create signatures they could not create individually by colluding their keys.

The main advantage of using ABS in a health care application is that it allows authentication of users without the need to store and validate the identity of individual users. The latter is, for instance, needed in public key signatures, where for each user the public key has to be known.

Another advantage is that it overlaps with ABE, and can use the same architecture or even the same attributes, as long as both schemes allow this. Also, users can be authenticated based on the capabilities they possess. As the attributes already need to be distributed to the users for the encryption, ABS can be combined with ABE.

5.5 Conclusion

However, using public key signatures is a more feasible solution at the moment. ABS is relatively new, and the performance has not yet been analyzed with an implementation. Only formal constructions exists. On the other hand, public key signatures have been around for a while now and have proven to be feasible. We want to focus on the performance and feasibility of ABE, and leave the analysis of ABS to further research.

6 Access revocation

Revocation of access is an important aspect of the application. Access revocation allows to enable temporal access, as well as the possibility to revoke users in case of leaked credentials or changes in job status. This improves the security of the system, and allows to respond to breaches.

Within the ABE landscape, there exist several approaches to access revocation. These approaches are described first, after which the selected approaches for our example case are discussed.

6.1 Attribute revocation

In the first approach, attributes are revoked from users, by removing an attribute from the set of attributes assigned to a user. As these attributes are issued in the form of secret keys, the revocation of attributes comes down to updating the secret keys of users.

Attribute revocation is needed when the access rights of a specific user, or a group of users, should be updated. Attribute revocation can be realized by re-creating the keys for the attribute to be revoked. Only the users who still possess this attribute receive the new secret key, while the users for whom the attribute is revoked do not receive an updated key.

Attribute revocation can be separated in two categories, *direct* and *indirect* revocation. *Direct* revocation means that a revocation has immediate effect. From the moment the revocation is applied, the user does not have access anymore to the involved records. In *indirect* revocation, on the other hand, it can take some time before the revocation has effect.

It should be clear that *direct* revocation is the preferred approach. *Direct* revocation, however, has some drawbacks. It is hard or impossible to enforce a key update for every user. Malicious users could simply not update their keys, after which they would still be able to access the data, despite the fact that they do not have sufficient attributes. Thus, re-encryption of the data is required to ensure that only the updated attribute keys can be used.

6.1.1 Indirect attribute revocation

In the *indirect* approach, revocation is realized by embedding a time period in the secret keys of users. Only in the embedded time period, the secret key can be used to decrypt the ciphertext. This enforces the user keys to be updated regularly. Attribute revocation is realized when the secret key for an attribute is no longer issued to the user.

This revocation only has effect when the secret keys are updated, so depending on the time period this takes some time. Until the update, the user still is able to access the data, which introduces some vulnerabilities. Another drawback of *indirect* revocation is that it introduces a bottleneck,

as there are many keys which should be updated at the same time when a time period has ended. Furthermore, depending on the granularity of the time periods the amount of keys increases, as each time period requires new keys which need to be calculated and distributed.

Some schemes support the encryption of a message for multiple time periods. However, when data should be accessible for many multiple consecutive time periods, at some point re-encryption is still needed.

The indirect approach is used, among others, in [6], [7] and TAAC [24]. Although not all ABE schemes support indirect attribute revocation, it can easily be added by appending a time period to each attribute.

6.1.2 Direct attribute revocation

As opposed to the *indirect* approach, in a *direct* approach the access revocation has immediate effect. Direct revocation can always be realized by updating attribute keys, re-encrypting all ciphertexts using these keys and only sending the updated keys to non-revoked users. However, this is not really efficient. Moreover, re-encryption requires decryption of the ciphertext, and thus can not be performed by servers if end-to-end decryption is desired.

To avoid the need for re-encryption on the client side, other approaches have been designed. In these approaches, the user's secret keys are split in two parts. The user receives the first part, while the second part is stored by a third party, called a proxy or a mediator. Only the combination of both parts enables to decrypt the ciphertext.

The third party, let us call it the *mediator* maintains a revocation list. Each time a user wants to decrypt a ciphertext, he has to contact this mediator. The mediator executes part of the decryption by using its share of the secret key, as long as the user is not on the revocation list. The user can complete the decryption with his share of the secret keys.

An attribute can be revoked by adding the user to the revocation list of the attribute. An example of a scheme using this approach is [12].

In a slightly different approach, upon revocation new keys for the attributes are generated, and re-encryption keys are calculated. These re-encryption keys are used to update the ciphertexts to the new keys. The re-encryption key does not reveal any information about the secret keys. As the mediator acts as a proxy between the storage server and the user, this is also called *proxy re-encryption*.

A scheme which utilizes proxy re-encryption is [25].

Both approaches introduce a new party which should be online at all time. In addition, this solution puts a large burden on the mediators, as all the data has to pass through mediators and has to be processed by the mediators. This basically doubles the time needed for decryption and increases the amount of network traffic.

6.2 User revocation

In another approach, the access of a single user can be revoked per data record. This is achieved by adding the identifiers of non revoked users to the access policy of a data record. When a user's access is revoked, the identity of the user is removed from the access policy. This type of encryption is called Broadcast Attribute-Based Encryption (bABE) [4].

An advantage of this approach is that the access of single users can be revoked. However, to accomplish this the revoking party should know the identities of the users. Furthermore, re-encryption is required when the policy is updated.

6.3 Policy update

An update of the policy allows to change which users have access to a certain data record. This means that, if updating the policy is possible, either the access policy can be loosened allowing more users to access the data, or be made stricter, allowing less users to access the data.

Updating the access policy is the most convenient solution when the access to a single data record should be revoked for a group of users.

An update of the access policy always has an immediate effect. From the moment the policy is updated, involved users no longer have access. They can, however, still access the 'old' data record using their old keys. Furthermore, being able to update the policy also allows to grant access to other users.

When the new policy is more restrictive than the old one was, the data itself has to be re-encrypted. Otherwise, users who had access under the old policy could still access the decryption key and decrypt the data. In cases when new access is granted however, re-encryption is not required.

6.4 Selected approach

The main problem in all the approaches is that, to be really secure, re-encryption is required after access is revoked to ensure that users who had access before revocation no longer have access after the revocation. Only the indirect approach really avoids this problem, at the cost of restricting encryption to predetermined time periods.

However, in our example case, most data access is only temporal. Direct revocation gives much overhead, as it introduces a third party which should be online at all time to partially decrypt data. We want to avoid introducing such additional parties.

For these reasons, we apply a combination of timed attributes and policy updates for our example case. Timed attributes allow to give temporal access, while the possibility for policy updates is added for both direct revocation and the possibility to share data with more users. Updating the

policy allows to revoke access from a single data record, which is needed for most cases where access should be revoked. Furthermore, to enable sharing the data with more users it should be possible to extend the access policy. Both use cases can be performed when it is possible to update the access policy.

6.5 Ownership proof

There is one consideration left. We assume that only the owner of a data record is allowed to update the access policies of this data record. In order to check whether it indeed was the owner who updated the access policy, the owner has to prove his identity. However, we do not want to introduce unique user identifiers or a public key dictionary for all possible data owners. This would both increase the needed storage space as well as decrease the anonymity of the individual users.

Another approach to prove ownership is to introduce a unique attribute for each possible data owner. An update of the policy then has to be signed with this attribute, while the read and write policy have to be converted to always allow the owner access. However, this approach increases the amount of attributes dramatically. Furthermore, this would enable the authority responsible for the owner attributes to update all data.

Instead, we propose an approach in which each owner has one or more public and private key-pairs. On creation of a new data record, one of the public keys is added to the data record to indicate ownership. This removes the need for verifiers to know the public keys of all users.

When the access policy is updated, the data owner signs the new policies using the same public-private key pair. The receiver verifies the update by checking whether the new policies are indeed signed using the private key belonging to the already stored public key.

Whether the data owner uses a new or an existing key depends on the desires of the data owner. Decreasing the storage costs by reusing keys is at the cost of decreased anonymity and unlinkability of data records, and vice versa.

A drawback of introducing an owner signature is that the size of a data record is increased, and thus additional storage capacity is required. Nonetheless, this overhead is minimal relative to the size of the data record.

7 Example case

To investigate the practical feasibility of the proposed construction, we developed an example case of a practical situation involving a health care application, in cooperation with Topicus. In this example case, users are able to securely share data with certain employees of an insurance company, in particular with health care professionals acting as reviewers for the insurance company. These reviewers judge the medical need for certain surgeries based on data presented by the patient, and thereby determine whether the surgery is covered by the insurance of this patient.

To further illustrate the example case, we give a use case which illustrates different actions and requirements in this example case. A graphical representation can be found in Figure 2.

Bob wants to undergo surgery for correcting his eyelids. The insurance company only reimburses such a cosmetic surgery when there is a medical need. To qualify for a compensation, Bob has to send a picture of his face to his insurance for approval of the operation. Bob sends his picture, but allows only the reviewers of the picture to access the image. The picture is encrypted to make sure nobody else has access to it, and Bob sends it to the insurance company.

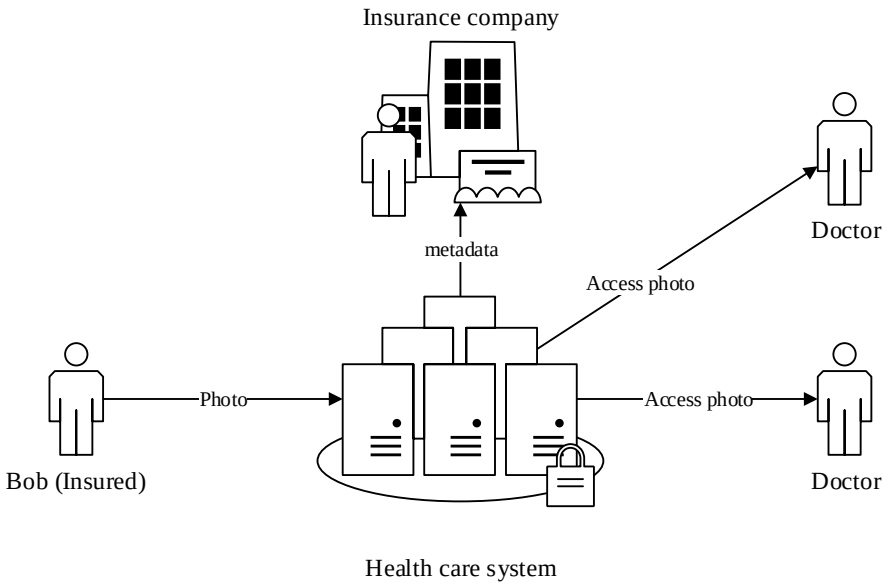


Figure 2: Example use case. Bob sends a photo via a health care system to be investigated by doctors.

The staff of the insurance company does not judge the picture, but outsources this task to a doctor. The employee of the insurance company only needs access to the meta data of the picture: he is allowed to know that there is a picture and where he can find it, but he is not allowed to view the picture himself.

The employee shares the picture with a doctor. The doctor then is able to decrypt the picture and take a look at it. The doctor gives an advise, and the insurance company processes this advice. Afterwards, the access of the doctor to this picture is revoked.

As the request of Bob is rejected because of a negative advise from the doctor, he requests a second opinion. A second doctor is allowed to access the picture, and he also gives an advice. The advice is processed by the insurance company, and the permissions on the photo are revoked again. At this moment, none of the doctors has access to the picture.

However, the possibility exists that the insurance company gets another request to make the data available for a other purposes. Thus, on a later moment it should be possible to grant permissions for data access to other parties.

This example case, access is regranted to the second doctor, thus support for access granting is required. Moreover, access is revoked after the doctor has reviewed the photo. Access revocation is required, and we note that most access is only temporal and has to be revoked on a later moment. Although not specifically present in the example use case, write access control is also a requirement. An example of this is when Bob wants to send a new photo, or when he sends some medical data which has been updated in the meantime.

We require multiple authorities to avoid the *key escrow* problem, but in the example case multiple authorities are also required. At least two are needed: one for the insurance company and one which issues attributes to qualified health care professionals.

8 Selected Attribute-Based Encryption schemes

In this section, existing ABE schemes are analyzed, in order to find schemes which can be feasible in our construction. At first, requirements for the schemes are described. Based on this, a comparison is given and possibly feasible schemes are selected.

8.1 Requirements for schemes

Our first requirement ($\mathbf{R}_{ABE}\mathbf{1}$) is that the scheme should be a CP-ABE scheme. This approach, where data is encrypted under access policies and attributes are assigned to users, is more suited to the use case than KP-ABE. In the example case, users have certain roles based on their profession. CP-ABE allows to express these roles in the form of attributes. Moreover, it allows users to be more expressive about the access to the data. Instead of categorize the data using attributes, the access is granted based on the roles of the users.

A second requirement ($\mathbf{R}_{ABE}\mathbf{2}$) for an ABE scheme to be feasible, is that the scheme should support multiple authorities. This is a requirement, as a single authority suffers the problem of *key escrow*, which can be mitigated by the use of multiple authorities. Moreover, in most cases, including our example case, the real world can be represented better using multiple authorities.

Furthermore, we compare the schemes on a couple of additional features, which are not strictly required for our example case. These *weak* requirements make the scheme more suitable for our use case and could be relevant in other cases.

The first of these features is the support for a large attribute universe ($\mathbf{W}_{ABE}\mathbf{1}$). The introduction of time periods attached to attributes greatly increases the number of attributes in the system, and support for a large attribute universe could result in a decrease of attribute key size.

Another weak requirement is the possibility to efficiently revoke attributes. In our construction, access revocation is enabled by applying timed attributes and policy updates, so support for other revocation approaches are not strictly required. However, being able to directly revoke ($\mathbf{W}_{ABE}\mathbf{2}$) attribute greatly improves the security and robustness of the system. Furthermore, support for indirect revocation is not strictly required ($\mathbf{W}_{ABE}\mathbf{3}$) as it can be added to every scheme, although native support for time periods could have a positive impact on the performance.

The last weak requirement ($\mathbf{W}_{ABE}\mathbf{4}$) is the support of re-encryption without the need to decrypt the ciphertext. Re-encryption can both enable the use of a mediator to enable direct attribute revocation and alleviate the burden imposed on client applications when an attribute key is updated.

Re-encryption of a ciphertext does no longer have to be performed by the client, but can be outsources to a proxy server.

8.2 Comparison of schemes

We compared several existing ABE schemes on how they fulfil the stated requirements. The result is given in Table 2 on page 30.

In the table, each column contains a requirement or feature, and the cells indicate whether a scheme supports the given aspect. A checkmark ✓ means that the scheme on that row supports the aspect. When a cell is empty, the scheme does not support that aspect. If there are different types of the given aspect, the cell contains a description of the type supported by the scheme instead of a checkmark.

As can be seen in the table, there are several ABE schemes which satisfy all our requirements. On the other hand, has not yet been published an ABE scheme which satisfies all the stated features, including weak requirements, at the same time. Each scheme supports different aspects, and each scheme has its own advantages and disadvantages.

Two of the schemes, [10] and [24], utilize binary trees to manage additional keys for each attribute. Using this, attributes can efficiently be revoked from users, at the cost of additional storage requirements. Users are assigned to leaves, and the path from root to leaf define the user's key.

Much of the schemes ([17], [6], [12], [25], [10]) are not suited for our use case, as they do not support multiple attribute authorities. Furthermore, [4] does not support multiple authorities by default. Multiple authorities are only supported by using access delegation, which means that there also should be a central authority to 'delegate' access to the authorities, so the authorities can delegate the access again to the users. This central authority possesses a master key, which grants him access to every encrypted file, thus the scheme is not suited for our case.

Only two of the investigated MA-ABE schemes support a large attribute universe. However, the scheme of [8] is not sufficient, as this scheme requires a fixed amount of attributes from each authority in each access policy, and only supports the of operator with a fixed threshold. Furthermore, this scheme is not strictly a CP-ABE scheme, as a set of attributes is related to both the user keys and ciphertext a set of attributes. This only leaves [20] when a large attribute universe is required.

When we focus on attribute revocation, the scheme by Yang et al. [23] is the most favorable candidate. This scheme does not only support multiple authorities and direct attribute revocation, but also has the possibility to outsource decryption and re-encryption.

Another scheme by Yang et al. [24] uses ciphertexts which are bound to one or more time slots. However, the user's secret keys are only issued once, after which update keys for each period are published. Although the

Table 2: Comparison of existing ABE schemes. A gray background marks the selected schemes

Work	CP-ABE ($\mathbf{R}_{ABE}$**1**)	Multiple authorities ($\mathbf{R}_{ABE}$**2**)	Large attribute universe ($\mathbf{W}_{ABE}$**1**)	Direct Attribute revocation ($\mathbf{W}_{ABE}$**2**)	Indirect Attribute revocation ($\mathbf{W}_{ABE}$**3**)	Re-encryption ($\mathbf{W}_{ABE}$**4**)
Chase et al. [8]		✓	✓			
Ostrovsky et al. [17]			✓			
Bethencourt et al. [6]	✓		✓		✓	
Ibraimi et al. [12]	✓			✓		✓
Attrapadung et al. [4]	✓		✓[a]			
Yu et al. [25]	✓			✓		✓
Hur et al. [10]	✓		✓	✓		
Lewko et al. [13]	✓	✓				
Yang et al. [24] (TAAC)	✓	✓			✓	
Yang et al. [23] (DAC-MACS)	✓	✓		✓		✓
Rao et al. [19] (RD-DABE)	✓	✓				
Rouselakis et al. [20] (RW-ABE)	✓	✓	✓			

[a] Although there is support for a large attribute universe, this support is bounded. This means that the size of the access policy is limited to a predetermined boundary.

scheme requires more storage for these update keys and the used binary trees, the scheme is potential beneficial for our use case since it was created with timed attributes at its base.

Although the scheme of Roa et al. [19] does not offer a large attribute universe or support for attribute revocation, it promises to offer fast decryption. Compared to the other schemes, this could be so much more efficient that it may outweigh the benefits of supporting a large attribute universe or having native attribute revocation.

Finally, the scheme of Lewko et al. [13] is a CP-ABE scheme with support for multiple authorities. However, the scheme by Rouselakis et al. [20] scheme is an improvement of this scheme. Furthermore, it does not provide any more advantages compared to the other multiple authority schemes.

So there are four schemes which could be suited for our cases and of which the feasibility will be investigated. RW-ABE [20] offers a large attribute universe, DAC-MACS [23] offers additional features in the form of attribute revocation and outsourcing of re-encryption and decryption, RD-DABE [19] provides fast decryption and TAAC [24] has timestamps embedded in the algorithms, removing the need to send update keys periodically. From now on, we will refer to these schemes with the abbreviations.

9 Construction

We now describe our proposed construction. The construction consists of several steps, which will be described first. Next, some practical considerations are discussed, and we analyze how the proposed construction allows to meet the requirements. Finally, the application of the construction in the example case is described.

9.1 Steps

setup In the *setup* phase, a central authority determines the global parameters. These are required to allow to use the attributes of multiple authorities in the same encryption.

authsetup In the *authsetup* step, the different authorities generate the public and private keys for the attributes they are responsible for.

register For the DAC-MACS scheme, it is required for each user to register itself to the central authority, in order to receive special keys. In the *register* step, this registration is performed. In the other schemes, this step is skipped.

keygen In this step, the secret keys of attributes are issued to the users who are eligible. Determining which attributes should be issued to the users is something we assume is performed outside the scheme. The keys are representations of the roles of the users, and are linked to a time period.

update keys For the TAAC scheme, next to the secret keys of the user, special update keys are required. These update keys are linked to a time period, and are updated every time period for the non-revoked users. In the *update keys* step, these update keys are generated and published to the users.

In the other schemes, this step is skipped.

encrypt In the *encrypt* step, a user encrypts a file using a read policy and a write policy. The user fetches the public keys representing these policies. For efficiency, the data is encrypted using a symmetric key encryption scheme. Any symmetric key encryption scheme can be used here.

The key used in the encryption is encrypted both using ABE with the read policy and the public key of the owner. This way, both rightful users and the owner have access to the data. The public key of the owner is added to the data record to facilitate owner proof.

To enable write access control, a public-private key pair is created which can be used in the creation of a signature, as described in section 5. The

(a) Data creation

(b) Data update

(c) Access policy update

Figure 3: Layout of a transferred data record for different steps in the construction. The second line in each block indicates whether the data in the block is encrypted, and which encryption technique is applied.

public key is added in plain text to the data record, while the private key is encrypted using ABE with the write policy.

The content of the data record sent to the server can be found in Figure 3a. This record is sent to the storage provider, in our case the insurance company. The storage provider is responsible for the management of the data storage, and returns a location pointer of the created data. Furthermore, the location pointer is sent to uses with whom the data is shared.

decrypt When a user wants to access a file, he requests the data record from the storage provider by sending the location of the file to the data service. The provider responds with the data record.

After the file is fetched, he can use the required attributes in his secret key to decrypt the ciphertext, as long as his attributes satisfy the read policy. If the user does not possess enough attributes to satisfy this policy, the decryption algorithm outputs an error.

data update An update of the data can only be performed by a user having enough attributes to satisfy the write policy. As the policies are not changed, none of the encryption keys has to be updated.

To update the data, the user first retrieves the symmetric encryption key from the original data record. He also retrieves the write private key, which is protected with the write policy. The updated data is encrypted using the symmetric key, and signed using the write private key. Both the encrypted data and the signature are sent to the storage provider. The contents of the update record can also be found in Figure 3b.

Upon receiving, the storage provider validates the signature over the updated data by using the already stored public write key. If the signature is valid, the encrypted data is updated. Otherwise, the server rejects the request and keeps the data record unchanged.

policy update The *policy update* step provides the ability for the data owner to update the access policies of a data record. The user re-encrypts the data under the new policy. Re-encryption is required to make sure that users whose access is revoked do not have access to data updates after revocation. In our construction, the data is always re-encrypted, although this is not required when the policy does not restrict access.

A user can prove that he is allowed to update the policy by signing th policies using the private key belonging to the public key in the original data record. The contents of a policy update record are shown in Figure 3c.

The receiver verifies the signature with the already stored public key of the owner, and only accepts the update when the policy is correctly signed.

9.2 Practical considerations

After creating a new data record, this data data record is sent to the storage provider, in our case the insurance company. The storage provider can perform additional checks to validate whether the user is allowed to create data, but this is outside the scope of this project.

A problem in the aforementioned system is that for the different parties to contact each other, the locations of the parties should be known. The service of the insurance company is available at some fixed location, but the users do not know how to reach the attribute authorities and vice versa. However, for the key distribution, communication between the authorities and the users is required.

In our approach, we simply assume that the users know the locations of the different authorities, and that these locations are valid. How the users know these locations and how this is verified, is outside the scope of this project.

Although we want to limit the required trust in the services as much as possible, each user still has to have a certain amount of trust in the different services.

Each user has to trust the attribute authorities to only hand out attributes to eligible users. However, a corrupt authority does not necessarily mean that an attacker can decrypt the ciphertext.

Furthermore, the users should trust the storage provider (the insurance company) to check the various signatures and to deny an update when the signature is invalid, although most of them could also be checked by the user. In other words, the storage provider is considered honest-but-curious.

The storage provider is assumed to try to gather as much information as possible, but still behaves as intended.

9.3 Meeting requirements

The construction described in this section meets the requirements to allow a feasible application of ABE, as stated in section 4 and summarized in Table 1. In this section, we described the problem and stated several requirements which should be met before an application of ABE is, in our view, considered feasible.

The first requirement (R1) is that users should be able to securly update data and control write access. This is achieved by introducing the *data update* step, as well as adding special write keys to the data record in order to allow write access control.

In the second requirement (R2), it is stated that the data owner should be able to update the policies over the data in order to allow other users access to the data. The third requirement (R3) states that it should be possible to revoke access.

The *policy update* step allows to grant more access or revoke access for a single data record. Furthermore, the use of timed attributes aids in the access revocation.

In the final requirement (R4), multiple authorities are required to avoid key escrow and to better fit the real world. This is achieved by only selecting ABE schemes which support multiple authorities.

9.4 Application in example case

We now describe how this construction can be applied in the example case as described in section 7. In the example case, patients are enabled to securely share data with health care professionals acting as reviewers for an insurance company.

Figure 4 visualizes the task of attribute authorities in the setup phase. The authorities generate keys for the different attributes for a certain time period. The public keys are stored and made publicly available. Two attribute authorities, an insurance company and a national registry, hand out keys to two doctors. The insurance company has employed one of the doctors as reviewer, and issues a key indicating this role to this doctor.

In Figure 5, Bob wants to send his photo to the insurance company. He encrypts the photo under the read policy $Reviewer \wedge Doctor$ for time period 1 using the public keys from the attribute authorities.

The encrypted data is sent to the insurance company, which stores the ciphertext on a storage server. The insurance company stores the location of the file in an index, links the file to the patient and responds with the

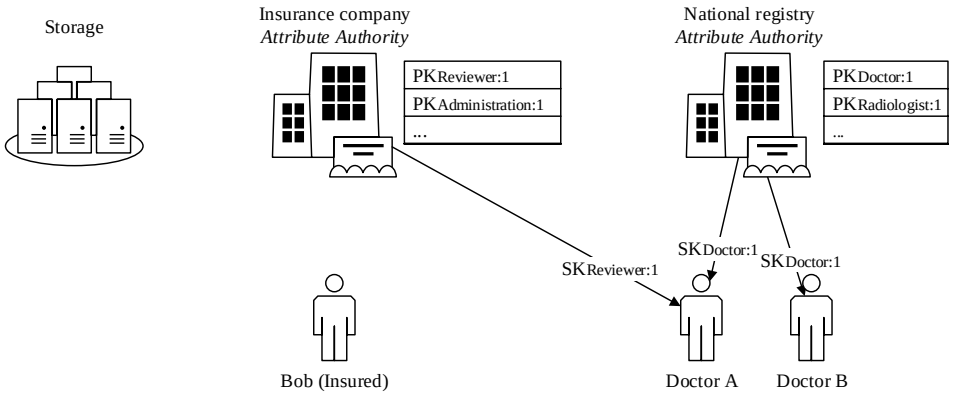


Figure 4: In the setup phase, timed attribute keys are generated by attribute authorities. Public keys are published, while secret keys are issued to eligible users.

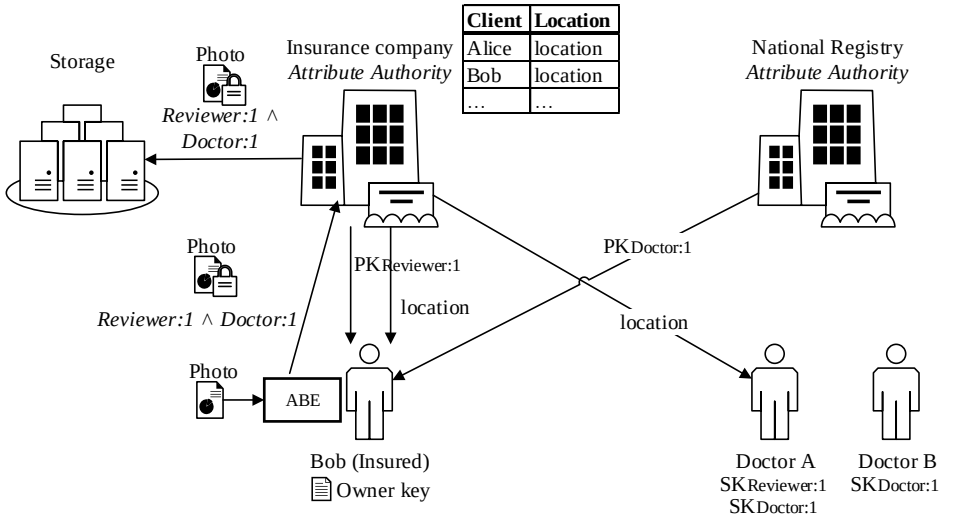


Figure 5: Bob encrypts his photo using the access policy $Reviewer \wedge Doctor$ for time period 1 by using the public attribute keys. After uploading, the storage provider (insurance company) shares the location of the data with the reviewing doctor.

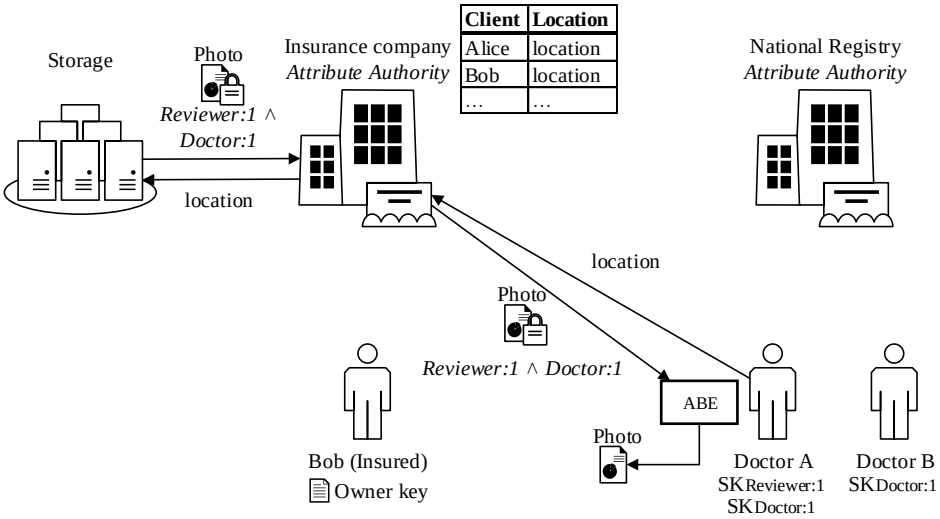


Figure 6: The doctor requests the ciphertext from the storage server and decrypts it using his secret keys.

location. Furthermore, the insurance company notifies a doctor that there is a picture to be reviewed, and sends the location of the file to the doctor.

Figure 6 visualizes the reviewing doctor requesting the photo from the storage server. The doctor decrypts the ciphertext using his attributes. His access is automatically revoked when the time period has elapsed.

Finally, in Figure 7 Bob shares his photo with another reviewer, this time an oculist. He re-encrypts the photo under the new policy, and sends the ciphertext with the original location to the insurance company. To prove that Bob is allowed to update the policy, he signs the data using his owner key.

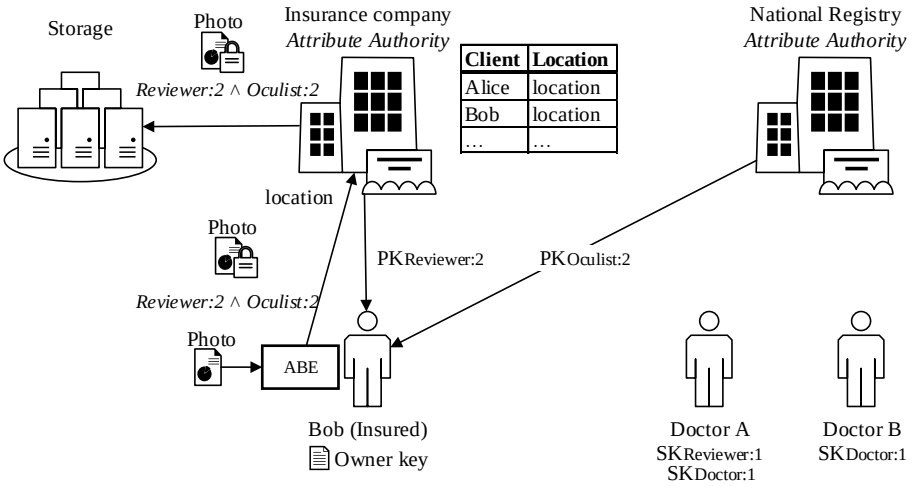


Figure 7: Bob updates the access policy of the photo to allow access to oculists in time period 2. To prove he is eligible to update the policy, he has to sign the data using his owner key. After the update, users who are both an oculist and a reviewer are able to decrypt the ciphertext the same way as it is shown in Figure 6 with Doctor A. This is not shown in the figure.

10 Implementation

We provide an implementation of the construction[1], as well as implementations for two of the compared ABE schemes. Furthermore, we improved the implementations of the two other schemes.

In the construction the different entities like users, attribute authorities and storage providers are modelled. The network connections between the parties are modelled as well, to enable measurements on the data exchanged between the parties.

All details related to a specific encryption schemes are wrapped in common interfaces to make them interchangeable. This way, when a different ABE scheme is used, as much of the implementation as possible remains equal, offering the best possible base for comparison. Furthermore, the used symmetric key and public key encryption schemes can easily be changed.

Next to the implementation of the construction, a framework for performing the experiments is created. The framework allows to specify a certain scenario and test case, which can be simulated an arbitrary amount of times. In each repetition the performance is monitored in separate runs, where in each run only one variable is measured. This way, possible influence of one measurement on the other is avoided. The output of the measurements are automatically written to several files.

10.1 Used techniques

For the implementation of both the construction and the ABE schemes, the Charm framework [2] is used. This framework is based on the Python language and targeted at rapid prototyping of advanced cryptosystems. Internally, native C modules and existing libraries are used for the intensive mathematical operations. The framework has been used for many implementations of state of the art ABE schemes.

The construction itself is also implemented in the Python language. For the symmetric encryption of the data, AES is used with a key size of 256 bits. The public key encryption scheme used for the signature is RSA with a key size of 1024 bits, the recommended minimum size. Different elliptic curves can be used in the ABE schemes. The Charm framework offers various curves, of which we use the *SS512* curve in the experiments. This is a super-singular symmetric curve with a base field of 512 bits.

10.2 ABE implementations

As the Charm framework has been used for many implementations of state of the art schemes, there is already a great number of ABE scheme imple-

[1] The implementation can be found at `https://github.com/denniss17/abe-healthcare`

mentations ready to be used. However, for some of the schemes which we consider possibly feasible, an implementation was not present. Part of our contribution is therefore that we provided implementations of the TAAC scheme of Yang et al [24], as well as the RD-DABE scheme of Rao et al. [19].

Furthermore, we improved the implementations of the other schemes. The DAC-MACS scheme, as described by Yang et al. [23], has support for multiple authorities. However, the implementation present in Charm only supports a single authority. We adapted the implementation to support multiple authorities as in the original definition, in order to make it usable for our case.

An implementation of RW-ABE by Rouselakis et al. [20] already existed, but did not quite follow the interface as specified by the Charm framework. We updated the implementation to follow this interface.

11 Experiments

In order to investigate the feasibility and performance of the implementations, several experiments are conducted. In the first experiment, a base scenario is investigated to settle a base case and to compare the different implementations. In the next experiments, we analyze the influence of the policy, the number of attributes and the size of the files on the performance of the construction.

In this section, the experiments are described in more detail and the results are discussed, but first the methodology and measured variables are outlined.

11.1 Methodology

Each experiment contains a specific scenario, in which various cases are repeated 100 times. Each case has a different input value, for example a different policy. For each case, various measurements are performed in multiple runs, to limit the mutual influence of the measurements.

Two different environments are used for the experiments. The first environment is a notebook machine with an Intel Core i7-2760QM CPU running at 2.4 GHz. The experiments are run on a virtual machine running Linux Mint 18 allocated with 2.0 GB of RAM.

The second environment is a Raspberry Pi in order to analyze the performance on devices with limited capabilities. The Raspberry Pi is a model B containing a Broadcom BCM2835 ARM processor with a frequency of 700 MHz, and 512 MB of RAM. The Raspberry Pi runs the Raspbian Jessie Lite[2] operating systems, dating from 27 May 2016.

In the experiments, the encrypted data and the meta information (policies and keys) are stored in two separate files. This allows to easily measure the size of the meta information.

11.2 Measured variables

Several variables are measured during the experiments. In this section, these variables are described, and possible external influences are discussed. Moreover, we discuss how the measurements are performed and, if a tool is used, which tool is used.

Time The first measurement factor is the time required for each operation. This is an important indicator of the performance and feasibility of the implementation. The time duration for each step in the algorithm is measured.

[2] `https://www.raspberrypi.org/downloads/raspbian/`

Although the algorithms and software influence the execution times for a major part, there are other factors too which could influence the execution time. For example, the used hardware has major impact. Despite this, the absolute execution times still give insight in the amount of time the encryption is expected to run. Furthermore, the relative differences between the algorithms and steps mostly remain equal, and using these relative differences the expensive operations can be easily detected.

Other processes running on the machine can also influence the execution times. When there are other processes which are very busy, the processing power is shared between the processes, and the execution time of the experiments can increase. To mitigate this influence, the number of parallel processes is limited as much as possible, while the experiments themselves are executed multiple times.

Python offers a module for deterministic profiling [3], which allows for monitoring the precise execution times of each function call. Although this adds some overhead to the execution, the overhead is minimal compared to the overhead caused by the interpreted nature of Python.

Storage The different implementations add meta information of variable size, like a number of attribute keys, to the encrypted data. All this information is stored on disk, and measuring the sizes of the output data gives an indication of the amount of data added by using encryption.

As the same symmetric encryption scheme is used throughout the experiments, the difference in file sizes is only the result of the different ABE implementations.

In addition to the required storage for the encrypted data, we also measure the required storage for the users' keys, the authorities' keys. The required storage at the authorities can differ greatly depending on, for example, whether the scheme supports a large universe construction for the attributes.

Network traffic Measuring the storage use does not capture the size of all data items used. For this reason, we also have to measure the transferred data not directly related to data stored on disk. This is for example the size of the requests for key generation, and the size of periodical update keys.

The amount of data transferred between the user and the authorities and between the user and the insurance company is measured. The data is serialized as if it were transferred over a network connection, and the size of the serialized data is determined. Possible overhead caused by the underlying network protocols is not included in the measurement.

[3] A more elaborate explanation can be found at `https://docs.python.org/3.5/library/profile.html`

11.3 Experiment 1: Base scenario

To analyze the difference between implementations and to settle a base case for other experiments, the first experiment is performed with a default scenario. In this scenario, two attribute authorities are instantiated with 10 attributes each. A file of 10 megabytes is created with random content. This size is selected to approximate the size of a large photo. The file is encrypted and decrypted. Furthermore, the data is updated with new random data of equal size, and the policy is updated using a policy of the same size and format, using different attributes.

The policies are created by selecting 3 pairs of attributes, where each pair contains an attribute from each authority. The attributes in a pair are combined with an *or* gate, while the pairs itself are combined with an *and* gate. So, the policies look like $(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge (a_3 \vee b_3)$, where each a_i is an attribute from the first authority and each b_i an attribute from the second. This format is selected as it is medium sized, while it still uses the different possible boolean operators.

Two users are created, one for encryption, modelling a client, and one for decryption, modeling a consultant doctor. Both users possess all attributes from each authority.

Results

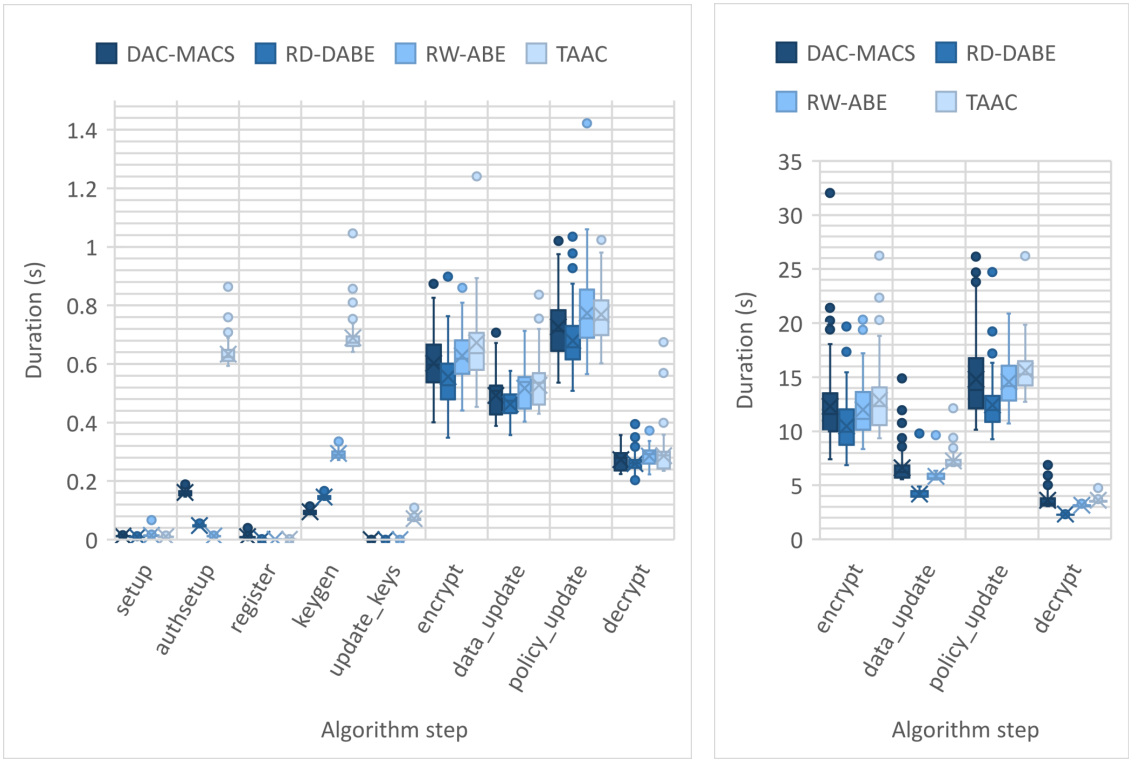
The experiment is executed in both environments. Figure 8 shows the execution times for the different steps in the algorithm as a boxplot. In Figure 8a, the durations for the notebook environment are displayed. Overall, the duration of all steps are below 1 second. With some exceptions, the differences between implementations are not very significant.

In encryption and decryption, RD-DABE is the fastest schemes with an average of 0.556 seconds for encryption and 0.260 seconds for decryption in the notebook environment. As this scheme focuses on fast decryption, we expected the scheme to be the fastest in decryption, but not in encryption.

In the setup phase, TAAC clearly requires the most setup time with an average of 0.634 seconds for the setup of attribute authorities, and 0.689 seconds for the generation of keys. This can be explained by the use of a binary user tree for each attribute. It is worth noting that these times involve the setup for one authority, for 10 attributes.

The *policy update* step requires the most time. This is not unexpected, as an update of the policy requires both decryption and encryption of the data.

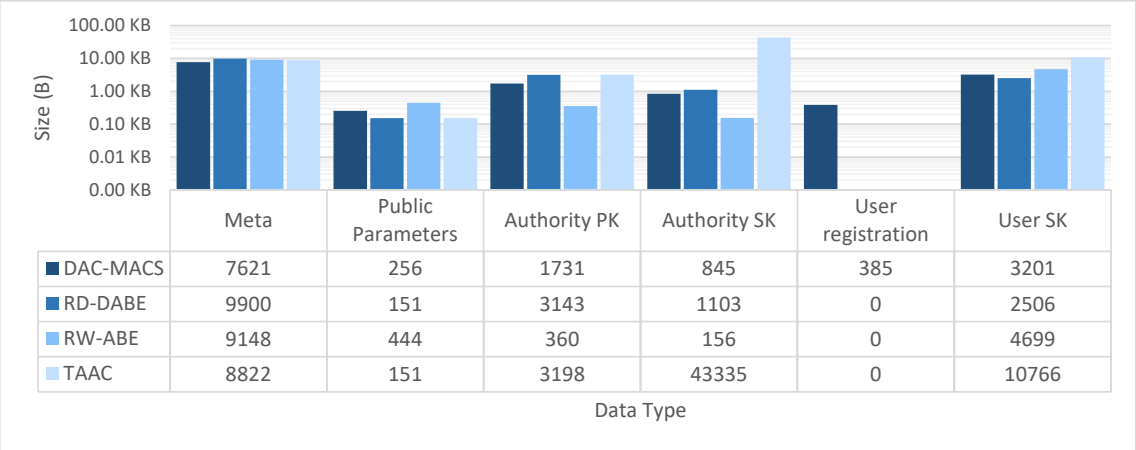
Figure 8b shows the results for the same experiment run on the Raspberry Pi. Only the *encrypt*, *data update*, *policy update* and *decrypt* steps are considered, as it is not likely that a mobile device will act as attribute authority.



(a) Notebook

(b) Raspberry Pi

Figure 8: Timings of the different steps in the default scenario in a boxplot. The borders of the box display the values at 1/4th and 3/4th of the series. The line in the boxplot displays the median, while the '×' marks the average value. The whiskers (lines) under and above the box display the smallest or largest values still within 1.5 times the length of the box respectively. The dots visualize outliers.



	Meta	Public Parameters	Authority PK	Authority SK	User registration	User SK
DAC-MACS	7621	256	1731	845	385	3201
RD-DABE	9900	151	3143	1103	0	2506
RW-ABE	9148	444	360	156	0	4699
TAAC	8822	151	3198	43335	0	10766

Figure 9: Storage usage of the implementations in the default scenario. Note that the vertical axis has a logarithmic scale.

As expected, the algorithms require more time on such a device with limited computational resources. The encryption takes somewhere between 10 and 14 seconds, while the decryption requires between 2 and 5 seconds. Again, here RD-DABE performs the fastest encryption and decryption. However, DAC-MACS seems to perform relatively worse than the other schemes, when comparing it to the experiment results in the notebook environment. This is probably due to a difference in the OS, which influences DAC-MACS more than the other schemes.

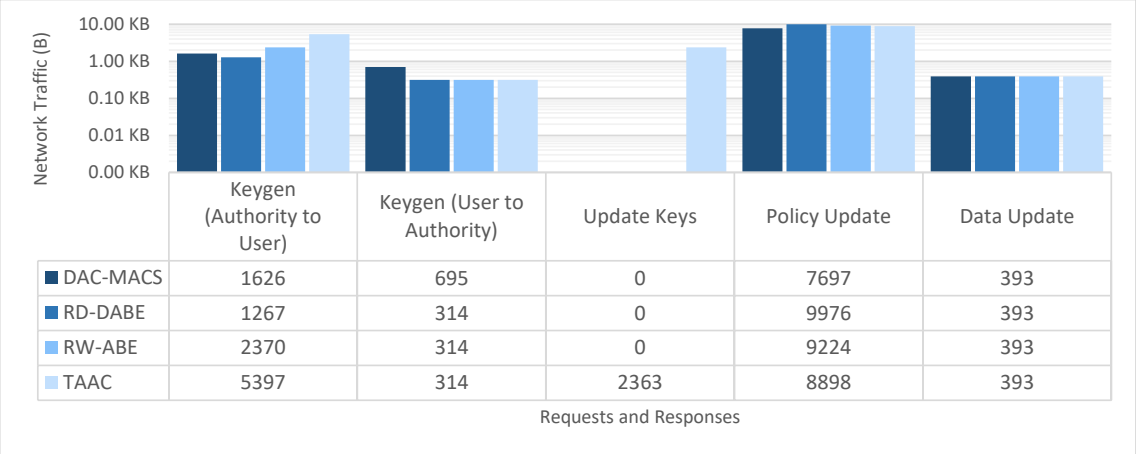
Figure 9 lists the size of the different created files. The size of the encrypted data itself is not listed, as this is the same for each implementation.

The most outstanding difference is in the keys size for the attribute authorities, both the public keys and the secret keys. For the RW-ABE scheme, the required storage is small, which can be explained by the support for a large attribute universe construction of the scheme. Opposed to that, TAAC requires a large amount of storage as it needs to keep track of additional states and binary user trees. This difference is also visible in the size of the user keys.

It is worth noting that RW-ABE does not result in the smallest user keys, even though the size of the attribute keys is very small thanks to the support for a large attribute universe.

Furthermore, the DAC-MACS scheme introduces the least overhead to each encrypted data record. An additional amount of 7621 bytes is required for each encrypted file, as opposed to the 9900 bytes added in the RD-DABE scheme.

In Figure 10, the sizes of data exchanged via the network is displayed.



	Keygen (Authority to User)	Keygen (User to Authority)	Update Keys	Policy Update	Data Update
DAC-MACS	1626	695	0	7697	393
RD-DABE	1267	314	0	9976	393
RW-ABE	2370	314	0	9224	393
TAAC	5397	314	2363	8898	393

Figure 10: Network traffic of the implementations in the default scenario. For the data update and policy update, this amount is the difference between the transferred bytes and the file size, in other words the overhead caused by the encryption. Note that the vertical axis has a logarithmic scale.

Only data not directly related to stored data is displayed. This means that the exchanged data for sending new records and requesting existing records is not listed, as well as the transfer of global parameters, public attribute keys and registration data.

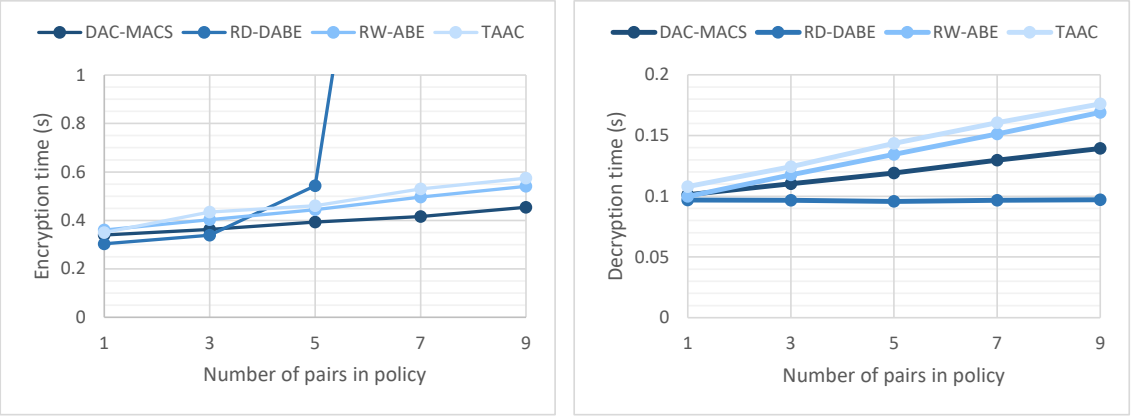
For the *data update* and *policy update* exchange, the difference between the size of the exchanged data and the file size is displayed. In other words, the amount of added bytes is displayed.

For TAAC , the size of the transferred user secret keys is with 5397 bytes more than twice as large as the other implementations, where RD-DABE has the smallest user secret keys with a size of 1267 bytes. In DAC-MACS , the user has to send his registration data when requesting the secret keys. This explains the difference in the size of the key generation request.

In addition to having the largest user keys, TAAC is the only scheme using update keys for each time period. This introduces another 2363 bytes of data exchanges, which has to occur at the start of each time period.

The overhead in the *data update* exchange is constant, which can be explained by the use of a signature, which is of constant size for each message. In the *policy update* the differences between the schemes are not very significant with an average overhead of around 9 kB.

The amount of overhead is small relative to the total network traffic. For a file with a size of 10 MB, the policy update still adds less than 0.1 percent overhead.



(a) Encryption

(b) Decryption

Figure 11: Average time durations to encrypt or decrypt for different policy sizes.

11.4 Experiment 2: Influence of policy size and format

In this experiment, the influence of size and format of the policy on the performance is investigated. In the experiment, the number of attribute pairs from the default policy is varied between 1 and 9 pairs, in steps of 2. That is, the first policy looks like $(a_1 \vee b_1)$, while the second looks like $(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge (a_3 \vee b_3)$ and so on.

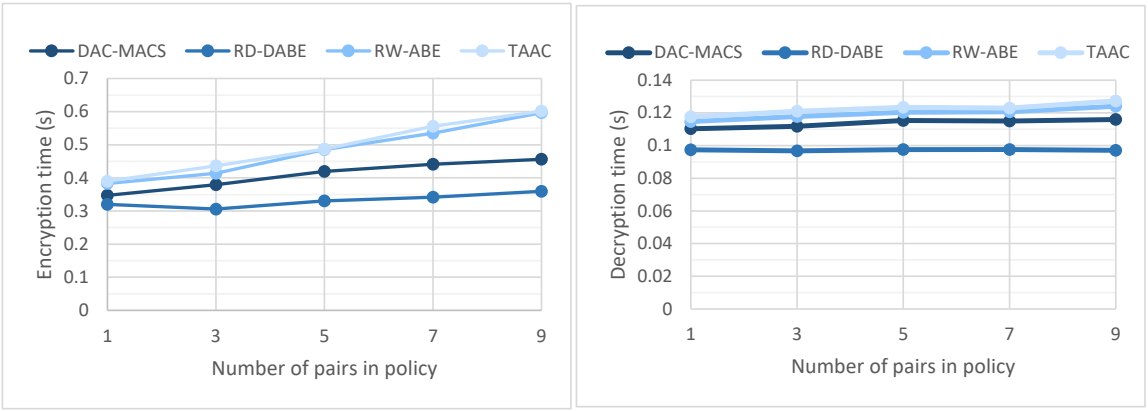
As a larger policy size involves more variables in the calculations performed during encryption and decryption, we expect that a larger policy size results in larger encryption and decryption times.

Results

The average durations of encryption and decryption as function of the policy size can be found in Figure 11. In Figure 11a, the duration for encryption are displayed. As expected, the encryption requires more time as the size of the policy increases. For RD-DABE , the increase is exponential, while the other schemes show a linear increase.

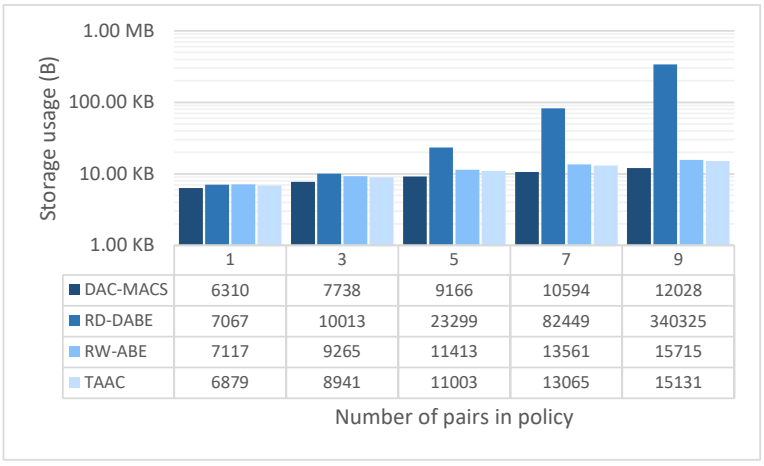
This exponential growth is also visible in the size of the encryption meta. Figure 13 shows that for RD-DABE the size of the meta information growths exponentially, up to 340 kilobytes for a policy consisting of 9 attribute pairs.

A possible explanation for this exponential growth can be found in the fact that the RD-DABE scheme requires each policy to be in disjunctive format (so as *or* operators combined with an *and* operator: $(a_1 \vee ... \vee a_k) \wedge ... \wedge (b_1 \vee ... \vee b_j)$), while in the default scenario the policy is given in conjunctive format. In the implementation, each policy is transformed to disjunctive normal form, which results in an exponential growth of policy size.



(a) Encryption (b) Decryption

Figure 12: Average time durations to encrypt or decrypt for different policy sizes for policies in disjunctive normal form instead of conjunctive format.



	1	3	5	7	9
DAC-MACS	6310	7738	9166	10594	12028
RD-DABE	7067	10013	23299	82449	340325
RW-ABE	7117	9265	11413	13561	15715
TAAC	6879	8941	11003	13065	15131

Figure 13: Size of meta information for different policy sizes. Note that the vertical axis has a logarithmic scale.

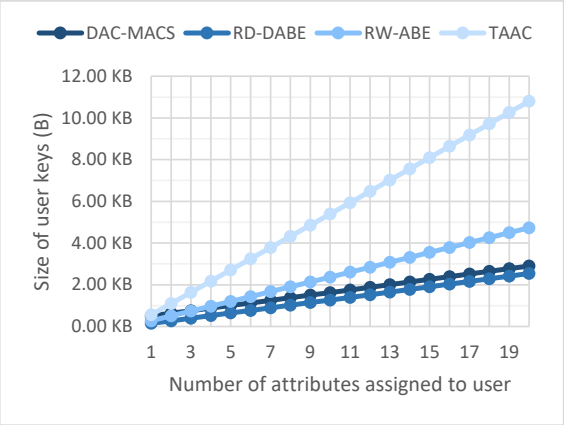


Figure 14: User key sizes for different amounts of attributes assigned to the user.

To see whether the growth of encryption duration and meta information is indeed caused by the policy format, we also run the experiment using policies in disjunctive normal form. The results can be found in Figure 12. This graph shows that for policies in disjunctive normal form, RD-DABE does no longer show an exponential growth in encryption duration and performs even faster than the other implementations, although it is not significantly faster than the others. Moreover, the decryption times appear to be nearly constant for each scheme.

11.5 Experiment 3: Influence of number of attributes on user key size

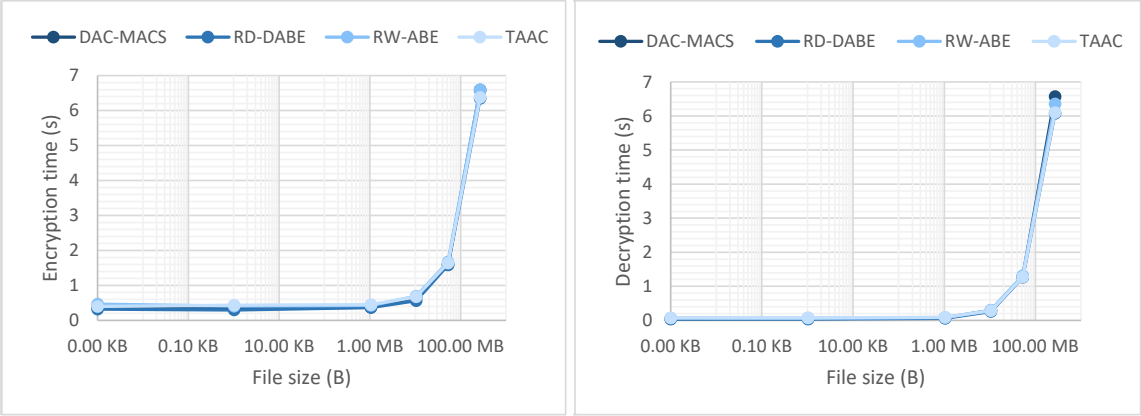
In the previous experiments, the user had a fixed amount of attributes. In this experiment, the effect of the number of attributes assigned to a user on the size of the user keys is investigated. The number of attributes is varied from 1 to 20, and the size of the keys is measured. For this, only one attribute authority is used.

We expect that the size of the keys growths with the number of attributes, but the rate at which it will grow is unknown. When the size of the keys grows very fast, the keys of the user could be too large to be feasible for a large number of attributes.

Results

The results can be found in Figure 14. For all implementations, the size of the keys is linear to the number of attributes. Overall, the size of the keys is not significantly large.

The size of the keys increases the fastest in TAAC, compared to the



(a) Encryption

(b) Decryption

Figure 15: Average time durations to encrypt or decrypt for different file sizes. Note that the horizontal axis has a logarithmic scale

other schemes, with a size of 563 bytes for 1 attribute to 10.8 kilobytes for 20 attributes. RW-ABE follows, while DAC-MACS and RD-DABE have the smallest user keys.

11.6 Experiment 4: Influence of file size

To investigate how the implementations handle different file sizes and to analyze whether the file size affects the performance and introduces possible side effects, files of random sizes are encrypted and decrypted in this experiment. The remaining inputs, like policy size and number of authorities, are kept constant.

Files of 1 byte, 1 kilobyte, 1 megabyte and 256 megabyte are used to cover a range of file sizes. Furthermore, file sizes of 10 and 50 megabytes are used to represent large photos and images.

Results

An increased file size leads to an increase in encryption and decryption times, as can be seen in Figure 15. The difference between the schemes are not significant in this regard. The size of the meta data remains equal for different file sizes. The only impact the file size has is thus an impact on the duration, as well as the storage for the encrypted data itself. These effects are as expected. No side effects were observed.

12 Conclusions

In this paper, we investigate the requirements which, in our view, should be met to make the application of ABE feasible. The goal of the research is to investigate the feasibility and performance of ABE in a realistic scenario in which several actors securely exchange data, on a relatively small scale.

We investigate several approaches to enforce write access control in an attribute based approach. We apply an approach where a data update is signed using a private key, which is encrypted with ABE under a special write policy.

Furthermore, we investigate different approaches to access revocation. We combine the use of timed attributes and access policy updates to meet the requirements of our example case. As most granted access is only temporal in the example case, this allows to cover most revocation cases without much overhead. Furthermore, enabling to securely update the access policy also allows to share your data with more users.

The approach of applying signatures to enable write access control and the approaches to allow access revocation are combined in one construction, that provides end-to-end encryption in a realistic scenario with the requirements of write access control and access revocation. The construction supports multiple authorities.

Multiple ABE schemes are compared on their compliance to requirements of multiple authorities, large attribute universe support, attribute revocation, proxy re-encryption and user revocation, resulting in four possibly usable schemes. For two of these schemes we provide an implementation, while for the other two schemes we provide improvements to the existing implementations. The implementations are tested in several experiments, targeted at analyzing the feasibility and performance of these implementations in our construction. Our main conclusion is:

> *The experiments show that on a relatively small scale ABE is feasible in a realistic scenario, in which several actors securely exchange data, as long as several conditions are met.*

The first condition is that devices with limited computation performance are not used. The results show that on the reference device used in the experiments, encryption requires a large amount of time, in the order of tens of seconds.

The second condition is that a feasible scheme is used. Of the tested schemes, DAC-MACS , RW-ABE and TAAC turn out to be feasible in a realistic scenario. However, we do not recommend using TAAC , as this scheme performs worse than the other schemes in most aspects. Especially the size of keys is significant larger than in the other schemes. Furthermore, in a scenario with many users, for example thousands of users, the amount

of authority keys and the time required for setup is expected to become large.

RD-DABE is feasible as long as a correct format of the policies is used. The access policy should be in a disjunctive normal form for the scheme to be feasible. Although this has the same expressiveness as any boolean policy, the disjunctive normal form can become super long, making the RD-DABE scheme not feasible in applications where such policies occur.

When these two conditions (no computationally limited devices and one of the feasible schemes) are met, the overall performance is acceptable. The execution times are in the order of hundreds of milliseconds, while the required storage and network traffic overhead is in the order of kilobytes.

13 Future work

There are still some differences between the implementations. RW-ABE for example, requires overall less storage space, while RD-DABE shows shorter encryption and decryption times. DAC-MACS , on the other hand, has support for more features, like proxy re-encryption and direct access revocation. In the example case we investigate, these differences are not significant. Nevertheless, these differences could be important in other cases.

Our example case considered a small number of users and attribute authorities. It would be interesting to investigate the feasibility of ABE in a scenario in which there are many users and authorities, or a scenario in which users are simultaneously communicating with the authorities.

In the example case, attributes authorities where created at the very beginning, and existed throughout the experiments. We did not consider the effect of adding and removing authorities in a later moment. This could for example occur when a new database should be linked to the system adding new attributes, or when existing authorities are merged into one. Investigating the impact of adding or removing an authority could be interesting.

Furthermore, in the experiment we focused on the encryption of data for one time period, as in the example only a short period of access is required. After this time period, the access is effectively revoked for users not having sufficient attributes in later time periods. Measuring the performance of the encryption of data for multiple time periods is a possible subject for future research. Moreover, the storage space required can increase over time, as new keys are required for each time period. For example, it would be interesting to find out how TAAC performs in this case, as this is an implementation tailored for the use of time periods.

For the control of write access, we focused on the use of signatures using private keys encrypted using ABE, as this is the most feasible option currently. In future work, the use of ABS can be investigated. This removes the need to use special write keys, which can result in smaller data records.

In our experiments we applied indirect revocation using attributes related to time periods, in combination with the ability to update the access policies. In future work, other approaches can be investigated. In particular the direct revocation approaches could be of interest because these always comes with additional costs. Some of the investigated schemes already support some of these other approaches.

We used the Charm framework for the implementations of the schemes. This framework is targeted at rapid development of prototypes of encryption schemes. For real production applications the efficiency can probably be improved. Improvements for devices with limited computational resources, such as mobile device, could make the application of ABE feasible on these devices too.

Acknowledgements

I would like to thank several people who helped me throughout the research and provided me with feedback. First of all, I want to thank dr. A. Peter and dr. A. Vasenev for supervising the research. In particular I want to thank dr. A. Peter for his feedback and questions, which helped me shape the research and allowed me to stay sharp and focused. Furthermore, I would like to thank B. G. Bakondi for his support. The meetings we had nearly every week combined with his detailed feedback really helped me throughout the research and aided in shaping up the thesis.

I wish to thank M. Gerritsen and M. Vloon, my supervisors at Topicus, for their support during the research. Both in the preparations and research they provided me with valuable feedback and support. Especially in defining the problem statement and example case, their support was invaluable.

Glossary

ABE Attribute-Based Encryption

ABS Attribute-Based Signatures

AES Advanced Encryption Standard

bABE Broadcast Attribute-Based Encryption

CP-ABE Ciphertext Policy based Attribute-Based Encryption

DAC-MACS Effective Data Access Control for Multiauthority Cloud Storage Systems [23]

EHR Electronic Health Record

FIBE Fuzzy Identity-Based Encryption

KP-ABE Key Policy based Attribute-Based Encryption

MA-ABE Multi-Authority Attribute-Based Encryption

MAC Message Authentication Code

RD-DABE Decentralized Ciphertext-Policy Attribute-Based Encryption Scheme with Fast Decryption [19]

RSA A Public Key encryption scheme designed by Rivest, Shamir and Adleman

RW-ABE Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption [20]

TAAC Temporal Attribute-based Access Control [24]

References

[1] J. Akinyele, C. Lehmann, M. Green, M. Pagano, Z. Peterson, and A. Rubin. Self-protecting electronic medical records using attribute-based encryption, 2010.

[2] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[3] S. Alshehri, S. Radziszowski, and R. K. Raj. Designing a secure cloud-based ehr system using ciphertext-policy attribute-based encryption. In *Proceedings of the Data Management in the Cloud Workshop, Washington, DC, USA*, 2012.

[4] N. Attrapadung and H. Imai. *Pairing-Based Cryptography – Pairing 2009: Third International Conference Palo Alto, CA, USA, August 12-14, 2009 Proceedings*, chapter Conjunctive Broadcast and Attribute-Based Encryption, pages 248–265. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[5] A. Beimel. *Secure schemes for secret sharing and key distribution.* Technion-Israel Institute of technology, Faculty of computer science, 1996.

[6] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 321–334, May 2007.

[7] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 417–426, New York, NY, USA, 2008. ACM.

[8] M. Chase. *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings*, chapter Multi-authority Attribute Based Encryption, pages 515–534. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[9] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[10] J. Hur and D. K. Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, July 2011.

[11] L. Ibraimi, M. Asim, and M. Petkovi. Secure management of personal health records by applying attribute-based encryption. In *Wearable Micro and Nano Technologies for Personalized Health (pHealth), 2009 6th International Workshop on*, pages 71–74, June 2009.

[12] L. Ibraimi, M. Petkovic, S. Nikova, P. Hartel, and W. Jonker. *Information Security Applications: 10th International Workshop, WISA 2009, Busan, Korea, August 25-27, 2009, Revised Selected Papers*, chapter Mediated Ciphertext-Policy Attribute-Based Encryption and Its Application, pages 309–323. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[13] A. Lewko and B. Waters. *Advances in Cryptology – EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, chapter Decentralizing Attribute-Based Encryption, pages 568–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[14] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):131–143, Jan 2013.

[15] H. K. Maji, M. Prabhakaran, and M. Rosulek. *Topics in Cryptology – CT-RSA 2011: The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, chapter Attribute-Based Signatures, pages 376–392. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[16] S. Narayan, M. Gagné, and R. Safavi-Naini. Privacy preserving ehr system using attribute-based infrastructure. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 47–52, New York, NY, USA, 2010. ACM.

[17] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 195–203, New York, NY, USA, 2007. ACM.

[18] B. Qin, H. Deng, Q. Wu, J. Domingo-Ferrer, D. Naccache, and Y. Zhou. Flexible attribute-based encryption applicable to secure e-healthcare records. *International Journal of Information Security*, 14(6):499–511, 2015.

[19] Y. S. Rao and R. Dutta. *Decentralized Ciphertext-Policy Attribute-Based Encryption Scheme with Fast Decryption*, pages 66–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[20] Y. Rouselakis and B. Waters. *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, chapter Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption, pages 315–332. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[21] S. Ruj, M. Stojmenovic, and A. Nayak. Privacy preserving access control with authentication for securing data in clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 556–563, May 2012.

[22] A. Sahai and B. Waters. *Advances in Cryptology – EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, chapter Fuzzy Identity-Based Encryption, pages 457–473. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[23] K. Yang, X. Jia, K. Ren, B. Zhang, and R. Xie. Dac-macs: Effective data access control for multiauthority cloud storage systems. *IEEE Transactions on Information Forensics and Security*, 8(11):1790–1801, Nov 2013.

[24] K. Yang, Z. Liu, Z. Cao, X. Jia, D. S. Wong, and K. Ren. Taac: Temporal attribute-based access control for multi-authority cloud storage systems. 2012.

[25] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 261–270, New York, NY, USA, 2010. ACM.