MASTER

Policy enforcement using attribute-based encryption in distributed environments

Vangjeli, D.

Award date:
2014

Link to publication
Policy Enforcement using Attribute-Based Encryption in Distributed Environments

Master Thesis

Deni Vangjeli

Eindhoven, August 2014
The work for this master thesis was conducted as part of a research project in Philips Research Eindhoven and submitted at the University of Technology Eindhoven in partial fulfillment of the requirements for the degree of Master of Science in Information Security Technology.

**Supervisor:** Prof. Dr. Milan Petkovic  
**Mentor:** Dr. Saeed Sedghi

**Assessment Committee**  
Prof. Dr. Milan Petkovic  
Dr. Saeed Sedghi  
Dr. Ir. Berry Schoenmakers
Abstract

There is a high tendency for a large variety of companies and organizations to outsource their IT requirements, particularly data storage, to cloud computing systems. One of the reasons behind the popularity of cloud computing is that most of the responsibilities and expenditures related to handling hardware and software requirements of the infrastructure are transferred to the cloud computing administrative system. However, next to all the benefits that this feature provides for data owners, it raises concerns over the security of the stored data on the cloud especially when the data is sensitive e.g. patients’ medical records or critical financial information. In general, to ensure that the data is accessible by right users only, the data storage infrastructure needs to authorize any data requestor, according to applicable policies. However, this requires that the authorization unit to be fully trusted in such way that only requestors specified by the policies can access the data. There are several trust models on the authorization unit of cloud computing. In our model, we assume that the cloud storage server cannot be fully trusted, i.e. that intentionally or unintentionally it does not enforce the access policy correctly. This assumption makes it essential for developing security mechanisms that preserve the privacy of the data owner yet without giving up the benefits of cloud services.

The problem that this thesis will address is how to develop security mechanisms that allow data owners to have full control over the authorization decision of their data stored on the cloud. To address this problem, we study state of the art on how to enforce the access control policies on the cloud focused on attribute-based encryption (ABE) methods. In general, in ABE techniques, the data is encrypted in such a way that data owners can specify with which attributes data requestors can decrypt the data. This problem will be considered in across domain environments where multiple organizations with different security share information with each other.
Acknowledgements

For these past two years of my studies TU/e imposed a significant and qualitative transformation on my professional and personal values, something for which I feel highly grateful. I am grateful to all my professors here at TU/e who in my eyes are an excellent example of high academic and personal integrity.

It would have been much more difficult for me to write this thesis without the help of my mentor Saeed whose weekly consults helped me every time understand something new or see it differently and my supervisor Prof. Milan who supported me and showed trust in my ability to fulfill this project from the beginning.

I also am thankful towards both my parents and their persistent contribution for my education, especially my mother Irma from whom I learned to always be positive and helpful and my father Ilir who taught me to think critically and freely.
# Contents

Abstract ........................................................................................................................................................... iv
Acknowledgements ........................................................................................................................................ v
Contents .......................................................................................................................................................... vi
Abbreviations .............................................................................................................................................. viii

1. Introduction ................................................................................................................................................ 1
   1.1 Motivation .................................................................................................................................... 2
   1.2 Introduction to Attribute-Based Encryption .......................................................................... 3
   1.3 Thesis Organization .................................................................................................................... 3

2. Background and Formal Definitions ....................................................................................................... 5
   2.1 ABE Algorithms .......................................................................................................................... 5
   2.2 Security Requirements ................................................................................................................ 5
   2.3 Traditional Architecture ............................................................................................................ 6
   2.4 Multi-Authority Architecture.................................................................................................... 7
   2.5 Trust Model .................................................................................................................................. 8
   2.6 Security Model ............................................................................................................................. 9
   2.7 Conclusion ................................................................................................................................... 9

3. Related Works ........................................................................................................................................... 11
   3.1 Introduction ............................................................................................................................... 11
   3.2 Key-Policy ABE ......................................................................................................................... 11
   3.3 Ciphertext-Policy ABE ............................................................................................................. 12
   3.4 Reference Architectures ........................................................................................................... 12
   3.5 Predicate Encryption schemes [14, 15] .................................................................................. 13
   3.6 Analysis of ABE Schemes ......................................................................................................... 14
   3.7 Conclusion ................................................................................................................................... 16

4. Attribute Revocation ................................................................................................................................ 19
   4.1 Introduction ............................................................................................................................... 19
   4.2 Intuitions .................................................................................................................................... 19
       4.2.1 Access Structures .................................................................................................................. 19
       4.2.2 Access Policy Tree ................................................................................................................. 20
       4.2.3 Secret Sharing Schemes ........................................................................................................ 20
       4.2.4 Bilinear Maps ........................................................................................................................ 21
   4.3 Related Works ............................................................................................................................ 23
       4.3.1 Proxy Mediator Approach .................................................................................................. 23
       4.3.2 Key-Expiration Approach ................................................................................................... 24
   4.4 Our Proposed Solution ................................................................................................................. 25
       4.4.1 Construction ......................................................................................................................... 26
Abbreviations

(Element) Denotes element of XACML policy e.g. <Rule>
ABE Attribute-Based Encryption
API Application Programming Interface
APT Access Policy Tree
CA Central Authority
CP-ABE Ciphertext-Policy Attribute-Based Encryption
JPBC Java Pairing Based Cryptography
KGA Key Generation Authority
KMS Key Management System
KP-ABE Key-Policy Attribute-Based Encryption
MA CP-ABE Multi Authority Ciphertext-Policy ABE
MA-ABE Multi-Authority ABE
PHR Personal Health Record
PKI Public Key Infrastructure
RBAC Role Based Access Control
SSL Secure Socket Layer
TA Trusted Authority
UCCMA Unanimous Consent Control by Modular Addition
Chapter 1

1. Introduction

Sharing data across different organizations and domains is gaining increasing popularity because of the benefits it provides for easier access to the data. For example, Electronics Health Records (EHR) systems allow hospitals to share medical records of their patients among each other, in such a way that makes the access of medical records to doctors more convenient. However, these benefits come at a cost that data sharing increases the risk of unauthorized access to the data. If the data is moved to a different domain that is not under the full control of the data owner, how can we ensure that unauthorized users cannot get access to the data? Therefore, collaborations bring along the challenges of ensuring users’ privacy and preventing misuse of their sensitive information according to the European Data Protection Directive [1].

Access policy management and data encryption techniques are the main solutions used in the literature to address problems of unauthorized access to the data problems. In general, the data is stored encrypted while only authorized users can decrypt data and/or encrypted data. To authorize users, applicable access policies should be evaluated in such a way that in case of a positive evaluation, access to the data is permitted.

Traditional access control solutions use server mediated policy enforcement systems, where a server is responsible for enforcing access policies by evaluating the data requestor attributes against access policies. However, these solutions are suitable for the applications where the enforcement server can be fully trusted in a sense that the access policies are correctly evaluated. Otherwise, if the server intentionally or unintentionally does not evaluate access policies, unauthorized users can get access to the data.

In data sharing applications, policy enforcement servers cannot be fully trusted, because these servers are usually run by different administrative domains that are not under the control of data owners. For example in the EHR system, hospitals cannot ensure that other hospitals correctly enforce policy enforcements to patient’s data. Therefore, the problem is how to ensure that the data remains protected when moved to domains that cannot be fully trusted. Based on this problem we express the research question that we address in this thesis as follows:
How can we efficiently achieve reliable access control enforcement for shared data stored on the cloud, under the assumption that the storage cloud server is not trusted for the enforcement?

We answer the research question using cryptographical policy enforcement techniques, which enforce access policies using cryptographic functionalities. In more detail our thesis has the following contributions to answer the research question:

- Review and analysis of the state-of-the-art on Attribute-based Encryption (ABE), which is the most popular cryptographically enforced access control technique. Based on this analysis we identify open problems and gaps.
- We suggest a data protection system that uses ABE to address the open problems identified in our analysis and we propose specifically a new attribute revocation technique for it.
- We provide a practical implementation of an ABE system and we conduct performance measurements with the implementation versus different complexities of access control policies.

1.1 Motivation

The motivation for the work in this thesis comes from the eHealth industry, specifically technology enabling Ambient Assisted Living (AAL). Especially more elderly people in Europe now are depending on these technologies that provide them remote assistance depending on their living circumstances and vital conditions. Monitoring patients’ vital conditions to offer them medical care implies the responsibility to deal with their sensitive information and therefore protect their privacy as much as possible. Therefore we researched ways to enforce strong privacy protection on those data, to enable patients as data owners to have full control on the disclosure of their personal health records (PHR).

Nevertheless it is also very important to enable multiple parties to make use of the collected patients’ data as a form of collaboration. This is particularly important for research labs and hospitals such that they obtain better knowledge out of the collective information. Protecting the privacy of these data is therefore of high importance and a higher level of protection is needed. Cryptographically enforced access control methods are naturally suitable for healthcare systems maintaining patients’ PHRs for several reasons. Firstly they offer a stronger level of enforcement assurance compared to classical authorization systems by enforcing access policies mostly by the end-user and not by an
authorization server. Secondly it follows that they can be applied to enable authorization in an untrusted environment. Such eAuthorization enables the patients as data owners to define their own access policies and thus have control over who accesses their data.

**1.2 Introduction to Attribute-Based Encryption**

Attribute Based Encryption (ABE) is a technique that addresses the problem using cryptographic techniques. It was inspired by Sahai and Waters in their 2005 seminal work “Fuzzy Identity-Based Encryption” [2] where they viewed identities as a set of descriptive attributes related to one user. In ABE, ciphertext and user decryption keys are associated with a set of attributes and possible formula among them. The data can be decrypted only if the attributes associated with the decryption key and the ciphertext match, and thereby enforcing the policy by the decryption function.

One important feature that all ABE schemes share is user collusion prevention. This means that it is not possible for users to collude by joining their attributes to be able to decrypt some ciphertext given that not even one of them can decrypt it. Also it is possible to achieve fine-grained access control such that different records owned by the same person have different embedded access policies [3].

ABE comes in two main flavors, Key-Policy ABE and Ciphertext-Policy ABE. In Key-Policy ABE the decryption key is associated with an access policy and the ciphertext is associated with a set of attributes. Vice versa, in the Ciphertext-Policy ABE the ciphertext is associated with an access policy while the decryption key is associated with a set of attributes. In practice KP-ABE is more convenient in a scenario with an organization that has rules over who may read specific files but it is not able to specify policies in a per-message basis. Attaching policies to the ciphertexts as in CP-ABE is a more natural way of enforcing access control and similar to Role-Based Access Control (RBAC) where users have a list of their attributes and files have their own attached access policies. Security and efficiency are the main aspects of ABE schemes, where there is also a trade-off between these aspects. Here by efficiency we mean the complexity of creating a ciphertext, a decryption key and decrypting data.

**1.3 Thesis Organization**

This thesis is organized in the following way: Chapter 2 provides the necessary background and formal definitions that apply to generic Attribute-Based Encryption systems, their architectures, security models and requirements. In Chapter 3 is an analysis of exist-
ing ABE schemes. In the analysis we consider the efficiency, security and expressivity aspects of state-of-the-art ABE schemes for access control enforcement. In Chapter 4 we provide our solution to attribute revocation in ABE while in Chapter 5 we extend the security of ABE schemes by providing a scheme that supports multiple key authorities. In Chapter 6 we present the implementation details and results of the conducted measurements with the scheme presented in Chapter 5. Chapter 7 draws the conclusions of this thesis and gives some recommendations about possible improvements on the system.
Chapter 2

2. Background and Formal Definitions

In this chapter we give formal definitions and relevant background on Attribute-Based Encryption.

2.1 ABE Algorithms

As we mentioned in the Introduction chapter, there are two types of ABE, Ciphertext Policy ABE (CP-ABE) and Key Policy ABE (KP-ABE). Since in this thesis we use CP-ABE, we give formal definitions for CP-ABE only. A CP-ABE scheme consists of four algorithms below:

- **Setup**(s) → (Params, MSK), which given a security parameter \( s \) outputs public parameters \( \text{Params} \), and a master secret key \( \text{MSK} \).

- **KeyGen**(MSK, Attr) → SK\_Attr, which takes as input user’s attributes \( \text{Attr} \) taken from the universe of attributes \( \mathcal{A} \), and the master secret key \( \text{MSK} \) and outputs a decryption key for the user \( \text{SK}_{\text{Attr}} \).

- **Encrypt**(Params, AP, M) → CM\_AP, which takes as input public parameters \( \text{Params} \), the access policy \( \text{AP} \) (or a set of attributes only), and the message \( M \), and outputs a ciphertext \( \text{CM}_{\text{AP}} \).

- **Decrypt**(CM\_AP, SK\_Attr) → M, which takes as input a ciphertext \( \text{CM}_{\text{AP}} \) and a decryption key \( \text{SK}_{\text{Attr}} \) outputs the message \( M \) only if the attributes of the decryption key \( \text{Attr} \) satisfy the policy \( \text{AP} \) associated with the ciphertext.

2.2 Security Requirements

Each ABE scheme should meet some security requirements, which are explained below:

- **Collusion resistance** - Users should not be able to combine their attributes to decrypt encrypted data that is associated with attributes that each user partially possesses.

- **Data confidentiality** – The ciphertext of ABE should not reveal any information about the message encrypted.

- **Fine-grained access control** – Data owners should be able to specify different access policies for different parts of their data.
— **Policy updating** – Data owners should be able to efficiently update the access policy of their data.

Next to the security requirements that each ABE scheme should meet, each data protection system that uses ABE should satisfy attribute revocation requirements. This requirement states that:

— If a user does not possess some of his attributes anymore, he should not be able to use his old decryption key to decrypt data items that are encrypted using the revoked attributes.

### 2.3 Traditional Architecture

The traditional architecture of ABE is shown in Figure 1. As the figure shows, the architecture consists of the following actors: users, which can be either a data owner or a data consumer using storage services to store or retrieve encrypted data, a storage server, a Key Generation Authority, which generates appropriate decryption key for users, and a Central Authority, which generates master secret key and public parameters. In general the Central Authority and Key Generation Authorities can be the same, but for security reasons it is better that these entities are different, in such a way that separation of duties is achieved.

![Figure 1 - ABE Traditional Architecture and Workflow](image-url)
The typical workflow for data in ABE policy enforcement architecture is the following: First a Central Authority invokes the Setup algorithm which generates Params and the MSK. Then the Key Generation Authority given the MSK and a set of user attributes Attr generates a decryption key SK_{Attr} for each user using the KeyGen algorithm. Each time a data owner wants to upload a data item M that is associated with the access policy AP, first M is encrypted using the Encrypt algorithm, which outputs a ciphertext C_{AP,M}. The user then stores the ciphertext on the cloud. Each time a data consumer user wants to retrieve data, the corresponding ciphertext is downloaded from the server. The user then checks whether the ciphertext C_{AP,M} can be decrypted using his decryption key SK_{Attr}, where in case of a match between the attributes of the user and the ciphertext policy the data can be decrypted.

In practice, because ABE algorithms are slow compared to symmetric key encryption algorithms, the data is encrypted using a symmetric key encryption algorithm and only the information required to reconstruct the decryption key is encrypted using ABE. In other words, the message in practice M is actually the decryption key of the data. The data consumer also decrypts first M and then using M decrypts data. This way of encryption and decryption allows only the data decryption key which is usually much smaller than the data to be encrypted and decrypted using ABE.

2.4 Multi-Authority Architecture

While traditional architecture of ABE consists of one Key Generation Authority only, there are some schemes that propose Multi-authority key generation, which increases the security of ABE systems. In Multi-authority ABE, several key generation authorities are deployed to generate user decryption keys, where each key generation authority validates specific attributes and then issues a decryption key that is associated with the validated attribute. Having received decryption keys from each authority, the user can then integrate these decryption keys into one in such a way that it is associated with all the attributes. Each key generation authority is in principle independent of other authorities such that if one of the authorities is compromised the risk that the authority generates decryption keys for unauthorized users is also mitigated.
The workflow in this case is almost the same as in the traditional architecture with the exception that the data consumer needs to contact each KGA separately to get the respective secret key share from it.

2.5 Trust Model

The online services that we use nowadays often raise suspicion regarding if and how our privacy is protected. In the domain of user’s health data these privacy concerns are shifting into requiring a different and stricter trust model with respect to the manager of the data. Patients and users as data providers have the right not to fully trust their storage hosts and to reflect that, service providers are increasingly looking for ways to offer more privacy-oriented services.

There are three main trust models described on the basis of “honesty” and “curiosity” of the server. Honesty refers to how the server follows the protocols and policies to control the access of users to the data, and curiosity states that the server might want to learn about the content of the data it hosts. Our assumption for the storage server is “curious and not honest”. Following this trust model we do not trust the data storage services to enforce the access policies correctly. The task of policy enforcement in this case is executed at the data consumer side. Even though it is not yet clear what is entirely meant with “not honest” we assume that the server will at least not delete the data and
provide them on request. In this model the KMS\(^1\) is responsible only for validating users’ attributes, generating user secret keys and revoking them.

Another popular trust model we use is called the “curious-but-honest” model and is often described as “the server will strictly follow the protocols yet it will try to get as much information as possible from the processed data”. This concept is used in the construction of cryptosystems for components like mediators [4] or other outsourced service providers [5] that operate on 3rd party physical facilities (cloud). This trust model is also called as “semi-trusted server”.

2.6 Security Model

Apart from the efficiency aspect of ABE systems, regarded as the complexity of creating the ciphertext, the decryption key, and performing the decryption, there is also the security aspect. The security shows the ability of a scheme to hide the data from adversaries who have access to certain computational resources.

These resources are specified in the security models, where models assuming less restriction on the computational resources of adversaries are more secure. In general, three security models have been used in the ABE schemes: the Generic Group, Selective Security and Full Security models. The constructions provided in this thesis are based on the Generic Group model. Note that security in the generic group does not imply security for any concrete group and it relies on the fact that the attacker does not know the real encoding used for the underlying groups but only has access to oracles performing group operations.

2.7 Conclusion

In this chapter we introduced the background and the main popular architectures of ABE systems and their workflows. We also established the necessary requirements, the security model and the trust model upon which the authorization system works and whereby a trusted policy enforcement server is not necessary.

\(^1\) The Key Generation System comprises all the Key Generation Authorities in the system.
Chapter 3

3. Related Works

3.1 Introduction

In this chapter we review the most influential works in ABE systems and analyze them in terms of efficiency and security. We consider Key/Ciphertext-Policy ABE constructions as well as contributions in ABE architectures where multiple attribute authorities are possible with or without the coordination of a central authority.

3.2 Key-Policy ABE

The first KP-ABE scheme was contributed by Sahai and Waters in [2]. It is the first realization of ABE, which allows specifying only threshold gates over the access policy. This scheme encrypts a message using a number of attributes, in such a way that a decryption key can recover the message if the decryption key is associated with \( k \) out of the total attributes associated with the ciphertext, for some \( k \) which is predefined in the setup phase.

A trade-off here is that even though it is the most efficient ABE scheme, it offers limited expressiveness for specifying policies. To address this problem Cheung and Newport [6] proposed an ABE scheme which allows AND gates in the policy. However, this scheme has a drawback that only fixed number of attributes can be used in the ciphertext and the decryption key, which makes the scheme not suitable for practical situations.

Given the limitation of other ABE schemes to express negative attributes, Ostrovsky et al. present an ABE scheme [7] that allows a user’s private key to be expressed in terms of any access formula over attributes and not only using monotonic access structures. Their construction can handle any access structure that can be represented by a boolean formula involving AND, OR, NOT and threshold operations.

Goyal et al. proposed the first realization of Key-Policy ABE scheme [3], which allows associating an expressive access policy with the decryption key. This scheme is proposed in the selective security model. While the GPSW scheme realizes a KP-ABE which is expressive and achieves acceptable level of security, proposing a CP-ABE scheme was still an open problem.
3.3 Ciphertext-Policy ABE

CP-ABE allows a user to have a list of attributes as credentials and a user encrypting data to specify a policy for who can decrypt. Compared to KP-ABE, CP-ABE is conceptually closer to the traditional access control methods like Role-Based Access Control (RBAC). In [8] Piretti et al. improved the efficiency of the Sahai-Waters (SW) scheme in practice by means of using certain cryptographic optimizations. They implemented the SW scheme using the random oracle model for encryption and key generation and a careful selection of the cryptographic parameters. By these optimizations they increased the efficiency and reduced the cryptographic costs by as much as 98% compared to the SW scheme.

Bethencourt et al. proposed the first CP-ABE scheme which is expressive [9]. This scheme also allows using numerical ranges in the access policy, which allows key revocation with a key expiry date. An important optimization is proposed for the decryption algorithm whereby only a minimal set of attributes that satisfy the access policy are used. The main drawback is that this scheme is proposed in the generic group model which is regarded as a weak model.

In 2008 Goyal et al. proposed the first CP-ABE scheme [10] which is provably secure in the selective security model. This scheme considers a universal access policy which includes all possible attributes and formulas and then maps GPSW KP-ABE scheme into a CP-ABE. While this scheme addresses the security drawback of the BSW [9] scheme, which is security in the generic group model, this scheme suffers from inefficiency disadvantages. The parameters of the ciphertext and the decryption key size scale with $O(n^{3.42})$, where $n$ is the number of the attributes of the policy associated with the ciphertext. The level of complexity limits the usefulness of the scheme in practice.

In [11] Waters presents two CP-ABE constructions where the ciphertext, decryption key and decryption have the complexity linear to the size of the access policy. The schemes proposed in this publication are provably secure in the selective security model. Therefore the level of security is the same as in [10] but with improvements on the efficiency aspect.

3.4 Reference Architectures

The following works do not provide new ABE constructions but rather use them to give new ABE architectures supporting e.g. multiple attribute authorities or introducing a proxy to perform user revocation.
In 2007 Chase provides the first multi-authority ABE architecture [12]. In contrast with traditional ABE architectures where one key management server is responsible of issuing decryption keys for every user, this paper proposes an ABE architecture with multiple key generation authorities. In this architecture, each key generation authority generates parts of the decryption key which is associated with certain attributes. The user should then integrate all the pieces of the decryption key to have a key that includes all the user attributes. The main advantage of this architecture is providing separation of duties which in case one key management server is compromised it cannot generate decryption keys for unauthorized users. A drawback in the architecture is that the central authority needed to coordinate the attribute authorities can decrypt all the messages in the system.

Ibraimi et al. propose a system [4] which allows instantaneous attribute revocation by employing a proxy as a trusted third mediator. This party keeps an Attribute Revocation List related to user identities and revokes an attribute from a single user or from the entire system by affecting only the right users, immediately and without waiting for the attribute expiration date. Revocation using a mediated proxy is faster than using a CRL (Certificate Revocation List) and OCSP (Online Certificate Status Protocol) technologies which are widely used in PKI systems. Through this scheme it is also possible to enforce context attributes such as time or location where request comes from (since proxy is semi-trusted) due to the fact that the context factors need almost always an online entity. However, it is less efficient in decryption because of the operations by the mediator. Also, the requirement for a proxy to be online and generate tokens for each decryption can lead to a possible bottleneck at the proxy.

Lewko et al. propose an ABE scheme [13] based on the Chase’s scheme which allows multiple key management system to issue decryption keys but without using any central authority. An efficiency disadvantage of this scheme is that it uses bilinear groups of composite order.

### 3.5 Predicate Encryption schemes [14, 15]

Predicate encryption is a class of cryptographically enforcement of policy techniques which in contrast with ABE, hiding the policy in the ciphertext is the main requirement of these techniques. These schemes are appropriate for situations where the policy reveals non-trivial information about the data and should be hidden. However, in general these schemes scarify the expressiveness of the policy and the efficiency of the scheme for achieving such a strong security requirement.
In [14] Lewko et al. present two fully secure functional encryption schemes: One fully secure ABE scheme using composite order bilinear groups where the order is the product of three primes and supports arbitrary monotone access formulas; and a fully secure (attribute-hiding) predicate encryption (PE) scheme for inner-product predicates. Despite of achieving high level of security, using composite order pairing operations for decryption, which are at least 50 times slower than prime order pairings user by prior art, restricts the usage of these schemes in practice.

### 3.6 Analysis of ABE Schemes

Before presenting the table we need to describe more the notation used for the complexity measurements. To provide more clear description on the notation, we use an example of Access Policy Tree (APT) shown in Figure 3, which is a set of attributes with some Boolean formula among them.

![Figure 3 - Access Policy Tree (APT)](image)

In this figure, the value \( \omega \) is the set of attributes \( \{a_1, ..., a_6\} \) used in APT. The number of attributes in APT which is denoted by \(|\omega|\) is six in this example. Some optimized schemes use a smaller set of attributes which are necessary only for the decryption. If e.g. a user has all the attributes of the tree in Figure 3 then the \( \text{Decrypt} \) algorithm will only need \( \{a_3\} \) since it is a set which can satisfy the access tree. We call this minimal set \( \omega' \) and its cardinality \(|\omega'|\). When \( O() \) notation is used as an upper bound complexity measure for the algorithms, a more generic term \( n \) is used to denote the size of the access formula. In some cases, all the nodes in the tree influence the performance so we refer to the total number of nodes as \(|\tau|\) for APT \( \tau \). Some schemes’ efficiency is related to the total number of attributes that may be used to express access policies. This is called the universe of attributes and its cardinality is denoted as \(|U|\). When dealing with non-monotonic access
structures we use $k$ to denote the set of negative or primed attributes and similarly $|k|$ for the cardinality. Negative attributes are attributes that a decryptor must not have in order for the decryption to be successful.

In case the scheme is CP-ABE then the $\omega$ in the encryption efficiency column refers to the number of leaves in the access policy tree (APT) attached to the ciphertext. The same holds for KP-ABE but this time the policy attached to the decryption keys. A resume of the abovementioned notations is given in Table 1 below.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\tau</td>
</tr>
<tr>
<td>$</td>
<td>U</td>
</tr>
<tr>
<td>$n$</td>
<td>size of the access formula</td>
</tr>
<tr>
<td>$</td>
<td>\omega</td>
</tr>
<tr>
<td>$</td>
<td>\omega'</td>
</tr>
<tr>
<td>$</td>
<td>k</td>
</tr>
</tbody>
</table>

Table 1 – Notations

Table 2 provides an evaluation of existing ABE schemes with respect to their efficiency (encryption, key generation and decryption complexity) and security aspects. The naming of the schemes is done by concatenating the first letter of the last names of the authors as they appear on the original paper.
### 3.7 Conclusion

In this chapter we analyzed the state-of-the-art in the field of ABE with a focus on how to leverage it to offer efficient cryptographic access control enforcement. For ease of comparison we put together in a table the most influential works that have contributed to ABE schemes. Either by improving the efficiency on top of some existing scheme or by proposing a different one, these works provide different functionality, efficiency and security.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Complexity</th>
<th>Security Model</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encryption</td>
<td>Key Generation</td>
<td>Decryption</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTMW [8]</td>
<td>Same as SW</td>
<td>Same as SW</td>
<td>Same as SW</td>
</tr>
<tr>
<td>GJPS [10]</td>
<td>O(</td>
<td>U</td>
<td>*n^3.42)</td>
</tr>
</tbody>
</table>

Table 2 – Efficiency-Security comparison table of ABE schemes. In this table, Exp stands for exponentiation and Pairing* denotes composite order pairing.
However there are other aspects to consider when designing an authorization system based on ABE. One important factor is the expressivity of the access policies. Unfortunately most of them lack the desired expressivity or have policy constraints which make the system infeasible. An example is [6] where every ciphertext has to contain a fixed number of attributes pre-determined at setup time. Also in [12] encryptors need to use attributes from every attribute authority in their access policy to produce a ciphertext, which in reality may not be reasonable.
Chapter 4

4. Attribute Revocation

4.1 Introduction

In this section we present an approach that addresses one of the open problems in ABE systems, which is efficiently revoking attributes from decryption keys. In this chapter, we first present a review and analysis of existing attribute revocation methods and then present our solution. Recall that in attribute revocation problem, the user should not be able to use his decryption key to decrypt data that is decryptable using the revoked attributes.

4.2 Intuitions

Here we provide intuition on the techniques used for the construction of ABE schemes.

4.2.1 Access Structures

We refer to the work of Beimel [16] to define what access structures are and how they can be used to share a secret. Basically an access structure is a collection of sets that can reconstruct the secret, in other terms a collection of authorized sets i.e. that can access successfully.

Definition: Let \{P_1 \ldots P_n\} be a collection of parties. An access structure is a monotonic collection \(A\) of non-empty subsets of \{P_1 \ldots P_n\}. It follows that the sets present in \(A\) are the authorized parties and the sets not in \(A\) are called unauthorized.

Monotonic access structures
A monotonic access structure is a collection of sets \(A \subseteq 2^{\{P_1 \ldots P_n\}}\) such that if \(B \in A\) and \(B \subseteq C\), then \(C \in A\). This means that if any set \(B\) part of \(A\) is authorized and \(B\) is a subset of \(C\) then \(C\) is also an authorized set.

Non-monotonic access structures
In non-monotonic access structures, if a set \(B\) of \(A\) is authorized and \(B\) is a sub-set of \(C\) then \(C\) is not necessarily an authorized set. In ABE the concept of non-monotonic access structures is useful to handle negative attributes. In the case when we do not want an at-
tribute to be present e.g. a user must have the attribute “teacher” but not “principal” then
the attribute “principal” is negative. Since in our ABE construction we deal only with pos-
itive attributes, we will use monotonic access structures.

4.2.2 Access Policy Tree

The access structures supported by most of the ABE schemes are monotonic and based on
access policy trees (APT) as showed in Figure 3. APTs are a representation of a logical
formula over attributes and offer a reasonable expressivity compared to the earlier ABE
systems that used simply threshold schemes (users needed to have $n$ out of $k$ attributes in
order to decrypt). The attributes $a_1, a_2 .. a_6$ are in the leaf level of an $n$-ary tree which
needs to be neither complete nor full [17]. The non-leaf nodes are AND, OR gates or
“OUT OF” threshold gates. The tree is evaluated bottom-up and if all the logical gates are
satisfied up to the root then the access policy is satisfied and access is granted (decryption
proceeds correctly). An AND gate is satisfied if all its attribute children are present and all
its logical gates children are satisfied. An OR gate is satisfied if at least one attribute child
is present or if at least one child logical gate is satisfied. “OUT OF” gates represent thresh-
old gates and they are realized using a proper secret sharing scheme for attributes like e.g.
Shamir’s secret sharing [18]. Satisfying sets of attributes for the APT in Figure 3 are {$a_1,$
$a_2$}, {$a_3$}, {$a_4, a_5$} and so on, including {$a_4, a_5, a_6$}.

4.2.3 Secret Sharing Schemes

Secret sharing is a technique that allows sharing a secret among several parties in such a
way that the secret can be recovered if a number of parties, greater than a predefined
threshold, collude. The main usage of secret sharing in cryptography is that a secret can
be shared between the ciphertext and the decryption key $c$ in such a way that the secret
can be recovered if the decryption key matches the ciphertext. Formally a secret sharing
scheme as defined in [16], is a scheme in which a secret $s$ abides to the following two re-
quirements: firstly it is re-constructible by any authorized set of the access structure and
secondly, every unauthorized set of parties cannot reveal partial information about $s$. Se-
cret sharing was first proposed by Shamir. Shamir’s secret sharing scheme [18] works as
threshold method such that a user can split a secret $s$ in $n$ pieces such that $s$ can be re-
constructed from any $k$ pieces, $k <= n$. The secret cannot be reconstructed by less than $k$ pie-
ces. In our case it can be used to make sure that a required threshold number of attributes
is present in the user attribute list or in the access policy.
Shamir’s scheme to share a secret \( s \) among \( n \) users such that only \( k \) out of \( n \) users are needed to reconstruct the secret is as follows:

1. Choose a polynomial of degree \( k-1 \), \( y = f(x) \).
2. Let \( a_0, a_1, \ldots, a_{k-1} \) be the coefficients of \( f(x) \). Set \( a_0 = s \), i.e. \( f(0) = a_0 = s \), and the other coefficients random integers between 0 and \( p-1 \) where \( p > \max(s, n) \).
3. Compute \( s_i = f(i) \mod p \) and send it to user \( p_i \) together with \( i \).
4. Any group of \( k-1 \) users can pool their shares and use Lagrangian interpolation\(^\text{2}\) to find out the coefficients of \( f(x) \) and using them calculate \( f(0) = s \).

The scheme presented in Chapter 4 can also be modified to work with access policy trees supporting thresholds (check [19]) but that would lower the efficiency due to the Lagrangian polynomial interpolation which adds \( O(n^2) \) to the complexity where \( n \) is the degree of the polynomial.

Similarly in ABE constructions secret sharing ensures that all the required attributes of the user are present in order for the user to decrypt a certain resource. ABE schemes can make use of several secret sharing techniques, depending also on the type of access policy. We will use Unanimous Consent Control by Modular Addition (UCCMA) [20] which is a simple shared control scheme. Simple shared control schemes are a subset of threshold schemes like Shamir’s secret sharing scheme.

### 4.2.3.1 Unanimous Consent Control by Modular Addition (UCCMA)

In UCCMA we need unanimous presence of all the secret holders to reconstruct the full secret. To share a secret \( s, 0 \leq s \leq p-1 \) for some integer \( p \), among \( t \) users it one proceeds in the following way:

- To the first \( t-1 \) users distribute a randomly chosen number \( s_i \) such that \( 1 \leq s_i \leq p-1 \) and obviously \( 1 \leq i \leq t-1 \). The last user \( t \) gets \( s_t = s - \sum_{i=1}^{t-1} s_i \mod p \).
- To reconstruct the secret from the \( t \) users calculate \( s = \sum_{i=1}^{t} s_i \). No party has information about \( s \) except the dealer.

The advantage of UCCMA over other secret sharing schemes is that it is efficient and completes in \( O(t) \).

### 4.2.4 Bilinear Maps

\({}^\text{2}http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html\)
The usefulness of bilinear maps or pairings in ABE is that they allow for three-party protocols. In a typical PKI the when a sender $S$ wants to send e.g. an email to receiver $R$, then he has to send it to the central authority (CA) encrypted with its public key and then the CA would fetch R’s public key from a repository and encrypt the message with it to further send it to $R$. This quickly becomes a bottleneck for any messaging system therefore bilinear maps alleviate the need for a central authority to be online all the time. In cryptographic systems they introduce randomness in a computation that cancels out on the decryption side. In the two pictures below we can see the different approaches to secure messaging. Figure 4 illustrates a messaging system with classic PKI.

Figure 4 - PKI approach

Figure 5 below shows an Identity-Based Encryption (IBE) system [21] that uses bilinear maps to enable an efficient cryptosystem for secure messaging.

Figure 5 - IBE approach

The central authority is still a system actor but it is not required to be online all the time, the potential bottleneck is eliminated and the process of encryption/decryption is more
independent. This means that senders already know (easily derive) the receiver’s public key and the receivers can get their private key once, before they need to decrypt. The central authority makes publicly available its Public Key (PK) and senders use it in addition to the public key of the receiver to encrypt the message and deliver it end-to-end. This way the bottleneck in PKI is eliminated.

Definition: Let groups $G_1, G_T$ be both of prime order $p$. Also let $g$ be a generator of $G_1$. A bilinear mapping $\hat{e}: G_1 \times G_2 \to G_T$ is a pairing of two elements in $G_i$ to get an element of $G_T$ and it satisfies the following two properties:

1. **Bilinear:** for all $u, v \in G_1$ and $a, b \in \mathbb{Z}_p^*$, we have $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
2. **Non-degenerate:** $\hat{e}(u, v) \neq 1$ i.e. all the pairings should not lead to the identity element.

The pairing has to be also efficiently computable i.e. there should exist an efficient algorithm to compute $\hat{e}(g_1, g_2)$ for all $g_1, g_2 \in G_1$. The case above is a symmetric pairing since $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab} = \hat{e}(g^b, g^a)$.

4.3 Related Works

In the literature, two approaches for attribute revocation have been proposed. The first one is using a mediator as a semi-trusted proxy server, which enforces the revocation, and the second direction is to use time based decryption keys which expire after certain period of time. Here we review and analyze these solutions.

4.3.1 Proxy Mediator Approach

The mediator approach is based on proxy re-encryption methods which use a proxy to re-encrypt the ciphertexts without revealing the plaintext. This method involves a 3rd semi-trusted party that performs computations under the “honest-but-curious” trust model. In the proxy mediator approach the decryption key is broken into two parts: [4, 22], the proxy receives one part of the decryption key and the user receives the other part. To decrypt data, the user has to first send the ciphertext to the proxy which generates a decryption token using the proxy key. The proxy holds an attribute revocation list which is updated as soon as a user loses some of his attributes. The proxy generates the token only if
user attributes do not occur in the revocation list. Revocation is therefore enforced using the revocation list maintained by the proxy.

The main advantage of the proxy approach is that when an attribute is dropped from a user, the attribute will be immediately revoked from the decryption key of the user. However, this approach has inefficiency drawbacks. The proxy mediator is required to decrypt data partially upon receiving each request to generate a token. This imposes a high load of computations which makes the proxy a bottleneck in the system when a large number of users are involved.

In [22], Xu et al. perform user and attribute revocation with proxy re-encryption using a delegation key. The delegation key has two functions in this scheme: 1. Re-encrypt every ciphertext requested by legitimate users and 2. Get updated to reflect the user attribute revocations. The KMA keeps part of the secret key itself and every time there’s a user revocation the KMA sends a different part of the secret key (delegation key share) to the proxy storage server. When a user wants to decrypt, he makes a request to the storage server which on his side re-encrypts the data with the delegation key. This scheme is different and more efficient compared to Yu et al. [23] where proxy re-encryption is also used but for each revocation, new keys are issued for all non-revoked users and all the affected ciphertexts are re-encrypted.

4.3.2 Key-Expiration Approach

The key-expiring approach uses time based decryption key, in such a way that the key is expired after certain period of time. Bethencourt et al. [9] proposed to attach expiration dates with the attributes and achieve enforcement by doing numerical comparisons in the policy tree. However if the expiration date is to be implemented as a twin attribute of each attribute, this would double the complexity of the ciphertexts.

In practice to implement a numerical comparison using policy trees one needs \( n \) more leaves (attributes) for a \( n \)-bit number. In Figure 6 you can see the example for a variable comparison with a number, \( a<11 \).

![Figure 6 - Policy Tree for a<11. Source: [9]](image-url)
This method also requires that the user encrypting data synchronizes the encryption time with the key generation authority so that both the encryption and decryption use the same time.

However there are two major problems with the key-expiration and periodic key updates approach. Firstly, the attributes cannot be immediately revoked from the key, because the key is valid until certain period of time. Secondly, the complexity of the policy tree grows unjustifiably such that encryption and decryption become infeasible.

4.4 Our Proposed Solution

Our proposed solution uses the proxy mediator approach which allows immediate revocation of attributes. However, our solution is more efficient than existing proxy mediator systems, because the proxy needs to perform less computation.

In our system, the ciphertext is divided into two parts, where one part is stored on the cloud and the other part is stored on the proxy. Our mediator proxy is a semi-trusted entity that holds part of each ciphertext in the system. Each time a user wants to access the data, he needs to download the first part of the ciphertext from the storage cloud and then the second part from the proxy. Before the proxy sends the second part of the ciphertext, it is checked whether the user has any revoked attribute. In case of having any revoked attributes, the proxy will not send the seconds part of the ciphertext to the user, which prevents him to decrypt the data.

Some advantages of this revocation system are that there is no risk of exposing user keys and there is no computational overhead to delay the authorization. The system is also more efficient compared to [4] and the authorization time is reduced proportionally with this increase in efficiency. The revocation proxy is online all the time but unlike in [4] it does not need to compute a token but just issue its part of the ciphertext after the user has been checked for revocations. This way we trade-off computational requirements for space requirements and achieve a faster user authorization. The following Figure 7 illustrates the architecture of such a system.
4.4.1 Construction

- **Setup** – The system is set up by a trusted authority TA which may be the same entity as the Key Generation Authority KGA. Two groups of prime order \( r \) are generated, \( G_i \) and \( G_T \) with a generator \( g \), and an efficient non-degenerate symmetric bilinear map between them \( e: G_1 \times G_T \rightarrow G_T \).

For each attribute \( \{a_j\}_{j=1}^{n} \) for some integer \( n \) in the system a random element is chosen \( t_j \in \mathbb{Z}_p^* \). Also let \( y = \hat{e}(g, g)^\alpha \) where \( \alpha \in \mathbb{Z}_p^* \). The public key is set as:

\[
PK = (g, y, \{T_j = g^{t_j}\}_{j=1}^{n})
\]

and the master secret key is set as:

\[
MSK = (\alpha, \{t_j\}_{j=1}^{n})
\]

- **KeyGen** – Generates secret keys for users after verifying possession of attributes. Firstly the base component of the key is generated as \( d_0 = g^{\alpha - r} \) where \( r \in \mathbb{Z}_p^* \). Then the attribute component of the users’ secret key is computed as \( d_j = g^{t_j} \).

- **Encrypt** – Runs by the user encrypting who chooses a random element \( s \) randomly from \( \mathbb{Z}_r \) such that \( s = s_1 + s_2 \). The secret \( s \) is distributed using Unanimous Consent Control by Modular Addition throughout the nodes of the APT. \( s_1 \) represents the share
of the secret that pertains to attribute with index \( i \). This is how secret sharing is done in the access tree:

We start at the root \( r \) of the APT with \( s \). As we go down the APT recursively, if \( r \) is an \( OR \) gate then the children of \( r \) get also the secret share of the parent \( r \), otherwise if it is an \( AND \) gate the children of \( r \) get their secret share assigned by using UCCMA with \( r \)’s secret share.

![Figure 8 – UCCMA Secret Sharing in the APT](image)

The ciphertext is then produced using the attributes \( a_j \) in the leaf level of the APT as:

\[
CT_1 = \{ τ, c_{01} = g^{s_1}, c_1 = m \cdot \hat{e}(g^α, g^s), \forall a_j \in τ: c_{j,i} = c_{j} \} \quad CT_2 = \{ τ, c_{02} = g^{s_2} \}
\]

\( CT_1 \) is delivered to the storage server while \( CT_2 \) to the revocation proxy. When a user makes a request for a resource he downloads \( CT_1 \) and then when he requests \( CT_2 \) the revocation proxy issues it to him if he has no revoked attributes.

- **Decrypt** - To decrypt the ciphertext users need to have both \( CT_1 \) and \( CT_2 \). Then they compute:

  From \( CT_1 \):
  \[
c_1' = \hat{e}(c_{01}, d_0) = \hat{e}(g^{s_1}, g^{α-r})
  \]

  From \( CT_2 \) (proxy):
  \[
c_2' = \hat{e}(c_{02}, d_0) = \hat{e}(g^{s_2}, g^{α-r})
  \]

  For each attribute \( a_j \in ω' \):
  \[
c'' = \prod \hat{e}(T^s_{i,j}, d_j) = \prod \hat{e}(g, g)^{T^s_{i,j}}
  \]

  and if the user has the correct key for each \( a_j \in ω' \) then \( c'' = \hat{e}(g, g)^{sr} \).

  To recover the plaintext \( m \), finally computes:

  \[
m = \frac{c_1}{c_1' c_2'} = \frac{m \cdot \hat{e}(g, g)^{as}}{\hat{e}(g, g)^{s_1(α-r)} \cdot \hat{e}(g, g)^{s_2(α-r)} \cdot \hat{e}(g, g)^{sr}}
  \]
and since \( s = s_1 + s_2 \) it follows that

\[
m = \frac{m * \hat{e}(g, g)^{αs}}{\hat{e}(g, g)^{s(α−r)} * \hat{e}(g, g)^{sr}}
\]

### 4.4.2 Efficiency Analysis

Note that the revocation proxy obtains no information of the plaintext whatsoever from his share of the ciphertext and his space requirements is one element of \( G_1 \) for each ciphertext. In Table 3 below we compare the efficiency with that of [4].

<table>
<thead>
<tr>
<th>Revocation Proxy Scheme</th>
<th>The ITHJ Scheme [4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp. (( G_i ))</td>
<td>Exp. (( G_f ))</td>
</tr>
<tr>
<td>Encrypt</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>τ</td>
</tr>
<tr>
<td>KeyGen</td>
<td>(</td>
</tr>
<tr>
<td>Decrypt*</td>
<td>-</td>
</tr>
</tbody>
</table>

**Notation:**
- \( τ \) – access policy tree
- \( ω \) – the set of user attributes
- \( ω' \) – minimal set of satisfying attributes for \( τ \), \( ω' \) \( ω \)
- * for [4] includes the operations at the mediator proxy

Table 3 - Comparison with ITHJ scheme

We can say that our revocation proxy is more efficient by 50% in the decryption phase which translates into a faster authorization. Decrypting users also do not need to send the ciphertext to the proxy like in [4] which reduces transmission costs. Compared to the scheme presented below in Chapter 5 here we have 1 more pairing in decryption and 1 more exponentiation in encryption.

When the proxy gets a data retrieval request it needs to also find the correct part of ciphertext. Although we leave that problem to a database system or something similar, with an efficient searching algorithm this takes logarithmic time with respect to the number of ciphertexts stored.

### 4.4.3 Security Analysis

Further we provide a discussion on some of the security requirements that are fulfilled in the system.

- Users who do not satisfy the access policy cannot obtain the plaintext.
To get the plaintext users need to reconstruct $\hat{e}(g, g)^{\alpha s}$. To do so they need to have $\hat{e}(g, g)^{s'}$ which can be computed by pairing parts of the ciphertext $T_j^{s_i}$ with the user’s secret key shares for each attribute $d_j$, $\prod \hat{e}(T_j^{s_i}, d_j)$. Obviously the reconstruction of $s$ in the exponent will fail if the right key shares are not available.

- **Revoked users cannot obtain the plaintext.**

  When a user is revoked he does not receive the part of ciphertext stored by the proxy i.e $g^{s_2}$. This way the revoked user can still reconstruct $\hat{e}(g, g)^{s'}$ but he only has $c_{01}$ with which he can get $\hat{e}(g, g)^{s_i(\alpha - r)}$ and this way it is impossible to get $\hat{e}(g, g)^{\alpha s}$ and thus $m$.

- **If a user compromises the proxy he cannot obtain the plaintext.**

  If an unauthorized user is able to receive the proxy part of the ciphertext he still needs the correct decryption key parts to be able to decrypt the ciphertext.

- **If the proxy is compromised and its data lost, the KGA can recover the ciphertexts.**

  Notice that in absence of the ciphertext part of the proxy, the KGA can use its master secret key $MSK = (\alpha, \{t_j\}_{j=1}^n)$ to recover the plaintexts.

  Using the ciphertext available by the storage server: Take for each attribute $a_j \in \omega'$: $T_j^{s_i} = g^{t_je_i}$. Use the appropriate secret keys $t_j$ for the attributes $a_j \in \omega'$ to get

  \[ \prod_{a_j \in \omega'} (T_j^{s_i})^{1/t_j} = (g^{t_je_i})^{1/t_j} = g^s. \]

  Use $\alpha$ to get $g^{\alpha}$ and then pair it with $g^s$ to obtain $\hat{e}(g, g)^{\alpha s}$ and then the plaintext $m$.

- **Users cannot collude with each other.**

  Users’ secret keys have a random element $r$ embedded in their secret key in such a way that when two users try to combine their secret key shares to obtain $\hat{e}(g, g)^{sr}$ it will be impossible since they have different random elements $r$.

- **Forward secrecy**

  Enforcing proper user attribute revocation in ABE systems is directly related to forward secrecy. The definition in [24] states that: “Forward secrecy (in ABE) means that any user who drops an attribute should be prevented from accessing the plaintext of the subsequent data exchanged/created after he drops the attribute, unless the other valid attributes that he is holding satisfy the access policy.” It is obvious that by properly enforcing user attribute revocation we will also have forward secrecy enforced in our system.
4.5 Conclusion

In this chapter we have presented a CP-ABE scheme that achieves immediate user revocation by using a revocation proxy. The proxy holds part of the ciphertext which is necessary to the user for decryption. After revocation the proxy stops serving the revoked users.

This approach eliminates the risk of key exposure and has \( O(n) \) space requirements for the proxy where \( n \) is the amount of ciphertext parts stored. In case the contents of the proxy are lost, ciphertexts in the storage server can be decrypted by the KGA.
Chapter 5

5. Multi-Authority CP-ABE Scheme using Access Policy Trees

5.1 Introduction

In the previous chapter we presented a data protection system that allows attribute revocation from the decryption key. In this chapter we extend key management functionalities of our system to a Multi-Authority key management system which can enhance the security of our system.

Multi-Authority Attribute Based Encryption is a technique that aims at enhancing the security of data protection systems that use ABE. While in traditional ABE schemes, there is one authority that generates decryption keys for the user, in Multi-Authority ABE, several key management systems are deployed for the key generation. Each key management system issues a decryption key that is associated with specific attributes. Here, for a user to receive a decryption key, each key management system generates part of a decryption key for the user where each key part is associated with certain attributes. Having received all the decryption key parts, the user can integrate them into a decryption key that is associated with all the attributes. The main advantage of using Multi-Authority ABE is that if a key management is compromised, it can issue decryption keys with limited attributes for unauthorized users.

In this chapter we present the ABE construction that forms the basis of our implementation of eAuthorization system. The construction is secure in the generic group model and it provides encryption efficiency that scales perfectly linearly with the number of attributes used.

In the typical case one single Key Generation Authority distributes decryption keys to the data consumers. We see this in the early ABE schemes [9, 3, 7] and their shortcoming is for situations where more than one authority is responsible for assigning attributes to users. An example is a university student who gets the “Student” attribute from his university but gets his “FirstAid” certificate and attribute from a health center. Few Multi-Authority schemes are presented in the current literature [12, 13] but they lack the functionality desired to have expressive access policies like access trees and at the same time being efficient enough for deployment. The main challenge in our scheme is to en-
sure that two colluding users cannot obtain keys from different authorities and decrypt a message that they are not supposed to.

To enable user eAuthorization with ABE we use a Multi-Authority Ciphertext-Policy ABE (CP-ABE) construction that requires a central authority. The scheme supports multiple KGAs and it operates in feasible efficiency compared to other schemes in the generic group model. Here users can encrypt choosing any set of attributes from any KGA with no restriction on the number of attributes used. The access policy is represented as an $n$-ary tree which needs to be satisfied in order for authorization to be successful.

This chapter is organized as follows: in section 5.2 we give an overview of the existing Multi-Authority schemes and reason why they are not practical for our application. In section 5.3 and 5.4 we present our solution and its formal construction. In section 5.5 is an efficiency evaluation of the scheme and finally in section 5.6 we provide a way to deal with attributes in a cross-domain environment.

5.2 Related Works

There are few works in the state of the art literature about ABE that enable multiple attribute authorities distribute parts of the secret key such that the user can combine them and decrypt the messages. Chase’s scheme [12] allows users to achieve decryption if they possess a threshold of $d$ attributes from each of the $K$ attribute authorities. Furthermore the central authority is a trusted entity which can decrypt every ciphertext with its information. The issue with this scheme is that the attribute threshold requirement in decryption doesn’t allow for complex access structures like access trees. Also it implies the unrealistic requirement that users must have attributes from each and every authority, which is often not the case in practice. Lewko and Waters later proposed a system [13] which securely coordinates the authorities to work without a central authority. However their construction uses bilinear groups of composite order which severely hurt the efficiency of the scheme.

5.3 Scheme Description

The multi-authority CP-ABE scheme is comprised of five algorithms which are run at different components:

- **Setup(s)** – the setup algorithm is run by the Central Authority and it generates the public system parameters and a master secret key.
• **KeyGen** – secret key generation for the users is done partially by the CA and partially by each KGA they receive attributes from.
  
  — **KeyGenCA**(MSK, Iu) - runs at the CA and takes the master key and a verified user identity to generate the base component of the user secret key.
  
  — **KeyGenKGA**(ωu, Iu) runs at the respective KGA and takes the set of user attributes ωu and his verified identity Iu to generate the attribute component of the user secret key.

• **Encrypt**(τ, M, PK) – encryption is done by the data provider and takes an access policy tree τ, the plaintext m and the respective public key PK from the KGAs to generate the ciphertext CT.

• **Decrypt**(CT, SKu, Iu) – decryption is run by the data consumer and it takes the ciphertext, the user’s secret key and his identity to produce the original plaintext.

Note that depending on the system architecture it is possible to put the central authority in one trusted KGA. In practice each KGA maintains its list of attributes and may dynamically add new attributes.

### 5.4 Construction

• **Setup** – The system has to be set up in a trusted environment. Two groups of prime order r are generated, G1 and Gt with a generator g, and an efficient non-degenerate symmetric bilinear map between them e: G1 x Gt \( \rightarrow \) Gt.

  — **Central Authority** chooses a master secret key MSK = α ∈ Zr and sets the system public key PKs = e(g, g)α.

  — **Key Generation Authority** l chooses random elements tli from Zr for each attribute a(l) that it manages and sets its public key PK = {Tli = g(tli)}

• **Encrypt** produces the ciphertext which is composed of the policy τ, c0, c1, and one element TjSj in Gt for each attribute used in τ. A random element s is chosen randomly from Zr and it is distributed using Unanimous Consent Control by Modular Addition throughout the nodes of the APT as explained in Chapter 4. The ciphertext is then produced using the attributes aj in the leaf level of the APT as:

\[
CT = \{τ, c_0 = g^s, c_1 = m \ast e(g^{α}, g^s), \forall a_j \in τ: c_{j,i} = T_j^{s_i} = g^{t_{j,i}s_i}\}
\]
• KeyGen generates decryption keys for the end-users after proving possession of their attributes.
  
  — Central Authority generates the base component of the users’ secret key \( d_0 = g^{\alpha - u_{id}} \). For each user, a unique random element \( u_{id} \) which is bound to their identity \( I_u \) is generated such that collusion between users becomes impossible.
  
  — Key Generation Authority \( l \) generates the attribute component of the users’ secret key \( d_{lj} = g^{\gamma_l,j} \).
  
• Decrypt To decrypt users compute

\[
c’ = \hat{e}(c_0, d_0), \ c'' = \prod a_j \in \omega': \hat{e}(r''_j, d_{lj})
\]

\[
m = \frac{m * \hat{e}(g, g)^{\alpha s}}{\hat{e}(g, g)^{\alpha (\alpha - u_{id})} * \hat{e}(g, g)^{2u_{id}}} = \frac{m * \hat{e}(g, g)^{\alpha s}}{\hat{e}(g, g)^{\alpha s}} = m
\]

5.5 Efficiency Analysis

<table>
<thead>
<tr>
<th></th>
<th>Exp. (( G_1 ))</th>
<th>Exp. (( G_T ))</th>
<th>Pairings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>(</td>
<td>r</td>
<td>+ 1)</td>
</tr>
<tr>
<td>KeyGen</td>
<td>(</td>
<td>\omega</td>
<td>+ 1)</td>
</tr>
<tr>
<td>Decrypt</td>
<td>-</td>
<td>-</td>
<td>(</td>
</tr>
</tbody>
</table>

**Notation:**

- \(|r|\) – access policy tree
- \(\omega\) – the set of user attributes
- \(\omega'\) – minimal set of satisfying attributes for \(r, \omega'\)

| Table 4 - MA CP-ABE Scheme Efficiency |

The above scheme achieves encryption with exponentiations that scale linearly with the number of attributes \(|\omega|\) used in the encryption access tree \(r\). The key generation includes exponentiations in \(G_1\) that scale with the number of attributes that the user has \(|\omega_u|\). Decryption is the most efficient one as it includes pairings that scale linearly with the number of satisfying attributes \(|\omega'\) for the access tree \(r\). Note that there might be more than one set of such satisfying sets. For an example taking a look back at Figure 3 of an APT: In this case the possible sets of satisfying attribute sets are \(\{(a_1, a_2) \cap (a_3, a_4) \cap (a_5, a_6) \cap (a_7, a_8)\}\). One possible \(\omega'\) set is \(\{a_1, a_2\}\). If decryption fails with this set (e.g. because \(a_1\) is just recently revoked) then the user may try the other possible \(\omega'\) sets.
5.6 Attribute Management with a Shared Ontology

In order to enable attribute-based authorization in a cross-domain environment there should be a way to resolve cross-domain attribute mappings. For instance when a “Radiology Doctor” from Domain A requests a resource of Domain B which is available to “All Doctors”, it is reasonable that he should get access to it. However in ABE this is not possible without first establishing a relationship between the attributes. Therefore a common ontology needs to be defined.

Ontologies represent a vocabulary defined to describe concepts or entities and their relationships in a certain domain [25]. A common ontology is usually a hierarchical representation of the superset of attributes of all domains involved. The common ontology of attributes is a tree/hierarchy of “IS A” relationships between the attributes like in Figure 9.

![Attributes' Hierarchy](image)

If e.g. a user has the attribute “Doctor” and the ciphertext is available to “Medical Staff” then the user should be able to decrypt. However this is because a “doctor” IS A “medical staff” thus we can endorse the user with the attribute “Medical Staff” also. However when he tries to decrypt some ciphertext that is available to “Nurses” he will not be able to decrypt because the “Nurse” IS A “Medical Staff” but not vice versa. So when the user has an attribute that IS A sub-attribute of the required attributes of the access policy, then we can assign that attribute to the user and issue him the proper key to decrypt. This means that users will be issued all the attributes in the ontology tree that are in the path from the root to the lowest attribute in the tree that the user possesses. On revocation all the attributes along that path should be revoked also.
5.7 Conclusion

In this chapter we put forward a construction for a Multi-Authority CP-ABE scheme with the purpose of representing different organizations as different security domains defining their own attributes. The main advantage of such a system is its improved security with respect to single authority ABE systems. In single authority ABE the system is compromised if the only KGA is compromised i.e. the attacker would be able to decrypt all the data. On the other hand this risk is alleviated by having more than one KGA in the system.

We implemented the construction and present the results of the measurements in Chapter 6. The results showed a perfectly linear scaling in encryption time with the number of attributes in the ciphertext. We also explained how attributes can be managed in a distributed environment using a common ontology. Note that modifying the scheme in this chapter to include the revocation technique mentioned in Chapter 4 is quite straightforward.
Chapter 6

6. Implementation

6.1 Introduction

In this chapter we present and analyze performance measurements of our cryptosystem proposed in Chapter 5, which uses a Multi-authority CP-ABE scheme with a central authority. The implementation is done in a Java environment using the JPBC library [26]. We also used type A symmetric pairing for best efficiency. In the following section we show in details the concrete workflow for two main operations of our authorization system, which are ABE data encryption to associate an access policy to the data, and ABE data decryption to enforce the policy. We assume that communication of secret keys between entities is secure at least in the transportation layer using SSL or similar technology.

6.2 Implementation for ABE Data Encryption

6.2.1 Generate an APT

To test our authorization system we generated random access policy trees with a predefined number of attributes in the leaves. First we generated access policy trees as complete binary trees [17] and we randomly assigned logical gates as internal nodes like in Figure 10.

![Figure 10 – Randomly generated APT with 5 attributes. Attributes in the leaf level are written as [KeyGenerationAuthority:Name:Value](image)](image)

Then we recursively normalized each node of the randomly generated binary APT starting from its root into the following form in Figure 11.
Normalization of a node $N$ means to take $N$’s non-leaf children with the same gate as $N$, (call them the set $M$) and make their children ($N$’s grandchildren) $N$’s children and eliminate $M$. We also eliminate parent nodes with only one child by attaching the child to the grand-parent.

We used the normalized $n$-ary APT to perform ABE encryption. By normalizing the APT we reduce the time needed to generate satisfying sets $\omega'$ that will be attached to the ciphertext.

6.2.2 Generate $\omega'$

The set $\omega'$ is the set of authorized sets of the APT. As it can be seen in APT of Figure 11, there are several authorized attribute sets e.g. $\{\text{doctor5, doctor3, doctor4}\}$, $\{\text{doctor6, doctor3, doctor4}\}$ etc. To save authorization time we generate and store these sets in encryption phase instead of during decryption. Note that our approach is to generate all the possible satisfying sets and their number can grow significantly with a large APT.

6.2.3 ABE encrypt

We generate a random element of the underlying group $G_T$ to serve as the plaintext in ABE and perform AES key encapsulation with it to derive an AES key. The data is encrypted using the derived AES key in GCM mode [27].

Key encapsulation

Encryption of large amounts of data using ABE is not practical due to the time required. As we will see from the ABE benchmark measurements, about 11ms is required for one type A bilinear mapping performed on a high-end workstation. The overhead from exponentiations for an access policy with 32 attributes is ~500ms to encrypt a single element of the finite field $G_T$.

To achieve a better performance from ABE we will use a technique called “Key Encapsulation” which allows us to encrypt our original plaintext data using symmetric cryptography and derive the symmetric key using ABE. Key encapsulation is used analo-
gously in public key cryptography to encapsulate a symmetric key using public key pairs. Figure 12 illustrates the flow of this process.

![Figure 12 - AES key encapsulation in ABE](image)

First we generate a random element $e$ of group $G_T$. Then we derive the symmetric key by hashing this element with a cryptographic hash function. We continue by encrypting $e$ using ABE. We apply the same hash function to the decryption of $e$ on the decryption side to reconstruct the symmetric key and use it to finally decrypt the data.

### 6.3 Implementation of ABE Data Decryption

When a user downloads data from the server with no restriction he has to use his attributes to authorize himself against the encrypted data. Depending on the attribute revocation technique he might need to receive also a decryption token from a revocation proxy. Further he chooses the smallest set of attributes in $\omega'$ such that he has all the attributes in it and uses the proper part of his secret key to decrypt the ABE element. Finally he derives the AES key with the above mentioned method to decrypt the original data.

### 6.4 Evaluation

We measured the performance of encryption and decryption of one message in ABE under policies with varying number of attributes in the access tree. Figure 13 and Figure 14 show the results of the measurements. It is clear that the exponentiations in encryption time scale linearly with the number of attributes used in the ciphertext.
Decryption on the other hand consists of pairing operations that depend directly on the minimal number of attributes that satisfy the access tree.

We performed decryptions for varying number of attributes in the APT. However for each APT there may be many satisfying sets $\omega'$. The blue line represents the average number of attributes in the satisfying sets $\omega'$ while the red line corresponds to the average time needed to decrypt using the sets in $\omega'$. E.g. using 50 attributes in the APT, sets in $\omega'$ contained on average 4 attributes and decryption took 51ms. Note that the distribution of the gates in the APT is completely random and the average cardinality of the sets in $\omega'$ is variable. If there are only AND gates in the tree than $|\omega'|=|\omega|$ i.e. the user needs to use all the
attributes present in the ciphertext (if he has them) to decrypt, while in some other cases just one attribute is enough. You can find documentation and more details on the implementation in Appendix A.

6.5 Benchmarking

6.5.1 JPBC Library

Here we present the results of benchmarking the basic operations in the JPBC library. We measured the time taken in milliseconds to perform a type $A$ pairing in $G_1$, exponentiation, multiplication and setting to random element in both groups $G_1$ and $G_T$. The exact testbed specifications can be found in Appendix A.

<table>
<thead>
<tr>
<th>Operation</th>
<th>$G_1$</th>
<th>$G_T$</th>
<th>$Z_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element#pow(BigInteger)</td>
<td>0.57</td>
<td>0.08</td>
<td>0.03</td>
</tr>
<tr>
<td>Element#powZn(Element)</td>
<td>14.01</td>
<td>1.21</td>
<td>0.33</td>
</tr>
<tr>
<td>Element#setToRandom()</td>
<td>2.07</td>
<td>1.83</td>
<td>0.02</td>
</tr>
<tr>
<td>Element#mul(BigInteger)</td>
<td>0.5</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>Pairing#pairing(e1, e2)</td>
<td>10.68</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5 - Basic operations in bilinear groups of type $A$ pairing

*powZn – exponentiation with an integer mod ring (i.e. $Z_r$ for some $r$)

6.5.2 Multi-Authority CP-ABE Cryptosystem

Using the JPBC library to develop the cryptosystem we measured its performance of operations performed for encrypting data. We logged the time needed for these operations including ABE and AES encrypting, generation of $\omega'$ sets, and details like the composition of the APT. Due to limitations mentioned in Appendix A we could not consider transmission costs. In Table 6 we present the results.
Recall that for decryption the only changing factor is the “ABE decrypt” time which can be derived from Figure 14.

### 6.6 Extension: XACML Policy Attributes to APT

Due to the fact that many legacy systems have already XACML-defined policies in place we developed a component that extracts user attributes from the existing XACML policies. This offers a convenience when porting one’s authorization system to use ABE access policies.

Note that this is not a policy translation from XACML to ABE access policies. After some research we concluded that such a translation is almost impossible due to the vast expressivity limitations of APTs compared to XACML policies. We tried to safely map XACML policies into APTs in the sense that the authorization decisions for both the original and the mapped policy would be the same for the same requests on a particular resource. However the set of safely mapped XACML policies would be too small and thus not practical. As mentioned before the reason for this is the high complexity of XACML policy language model compared to the APTs. Therefore we consider this extension as merely a help to the security officers that define the access policies in an organization.

#### 6.6.1 Introduction to XACML
XACML defines a policy language, a request/response scheme as well as an architecture for authorization. As it is not the purpose of this thesis to explain in detail XACML, you can refer to the official specification here [28].

From the XACML policy language model we know that each PolicySet and Policy has a <Target> element and each rule may optionally have at most one. Through the <Target> element which contains Environments, Resources, Actions and Subjects fields, the XACML engine determines whether a certain policy is applicable to a request before evaluating the policy itself. Since the only applicable action in ABE is “read” and the policy is attached to the resource we just need to deal with the Subject attributes of the Target.

We look specifically the target elements of the rules and the policies as specified in XACML 2.0 specification [28]. In the policy target and rule targets we extract the subject attributes as disjunctions of conjunctions and construct an APT with them. Take the following example from a XACML policy rule:

```
</Rule>
</RuleEffect>"Deny" RuleId="R2">
</Target>
</Subjects>
</Subject>
</Target>
</Rule>

Figure 15 - XACML target subject conjunction example

The rule above states that subjects must have role “clerk” and “manager” at the same time. In XACML 2.0 specification the <Subject> element is mandatory and there can be many of them nested in the <Subjects> element forming a disjunctive set. The <Subjects> element inside the <Target> element is optional and if it is missing it implies no restriction on the subject. Following the specifications of XACML 2.0 we have the following configuration of disjunctive and conjunctive sets inside the <Target> element.
You can see in Figure 16 above the possible structure of a `<Target>` element inside a `<Rule>` or `<Policy>` (pseudo-syntax is used) along with the cardinality requirements from the specification.

![Figure 17 - APT mapping of a `<Target>` element.](image)

In figure Figure 17 is the target subjects of Figure 16 mapped into an APT. To derive the resultant target of e.g. a PolicySet from the targets of its Policies we can follow the examples given in the XACML specification.

One way is to process the resultant as the union of all the target elements of its children (Policies, Rules) and another way is as an intersection of those target elements. In case we make a union of all the target elements of the children of e.g. a policy, this means that we will have to merge those targets into an OR gate with each other, i.e. create a disjunctive set. Equivalently making an intersection set is not recommended unless there is a relatively simple target data-model in place. We follow the first example by computing a union of all the target elements.


6.7 Conclusion

In this chapter we presented the implementation details of the authorization system components. We included the measurements performed on every step of the authorization workflow starting with encryption of data, key encapsulation and including decryption and authorization using access policies with different complexity. Due to limitations in the cryptography library JPBC used we could not perform measurements for transmission costs between entities. We also provided an extension as a way to extract user attributes from XACML defined policies into APTs.
Chapter 7

7. Conclusions and Future Work

In this chapter we present the conclusions of this thesis. We also present recommendations for future work.

7.1 Conclusions

In this thesis we answered the research question “How can we efficiently achieve reliable access control enforcement for shared data stored on the cloud, under the assumption that the storage cloud server is not trusted for the enforcement?” In particular, the purpose of this thesis was to evaluate the existing cryptographically enforced access control methods focused on Attribute-Based Encryption, and to propose a new system that addresses the open problems identified in the evaluation. Furthermore we focused on developing and implementing a system suitable for a secure collaborative setting between different organizations.

In this thesis we proposed an immediate attribute revocation from an ABE decryption key using a semi-trusted mediated proxy. Furthermore, for an improved security we provided an ABE system with multiple Key Generation Authorities. The main advantage coming from multiple KGAs is that they eliminate the need to fully trust one KGA and thus avoiding a single point of compromise. Note that still in case any of the KGAs is compromised, a decryption key that is associated with attributes that the user does not possess can be generated.

We implemented the multi-authority CP-ABE system and performed feasibility measurements. Following the practical efficiency analysis we conclude that the authorization system based on multi-authority CP-ABE is feasible compared to the state-of-the-art systems and thus applicable in practice.

Considering negative attributes for APTs on the other hand is still challenging. We agree with the recommendations given in the XACML 2.0 specification according to which the negative attributes can lead to policy violations in two cases if not used with care. First case is when an attribute is intentionally or unintentionally suppressed by the user using e.g. privacy controls such that the attribute is not published or simply not known by the authorities. The other issue comes from a distributed environment setting
where two organizations merge and the system may not know that two similar attributes refer to the same thing. For ABE the suppressing of attributes is the major problem.

7.2 Future Work

In this section we put forward some possible improvements to the ABE authorization system that may e.g. theoretically improve the expressivity of the access policies or practically increase the portability of the system.

• **Negative attributes (i.e. non-monotonic attributes) in access policies** – Even though our recommendation is to avoid negative attributes, it may be that an application requires support for them and therefore they have to be integrated in the ABE authorization system. Currently there is only one scheme in the state of the art which achieves non-monotonic structure [7]. However it puts a lot of restrictions on the other aspects of the scheme like e.g. in case of CP-ABE every ciphertext has to have a fixed number of attributes.

• **Updating access policy** – There are generally two solutions to this problem. One is to outsource the updating to a central server performing updates for users and the other is to decentralize it such that each user updates his access policies himself. To do this the user needs to always be able to decrypt what he himself encrypted. This requires something an identity attribute attached to the APT during encryption even though it may seem counter-intuitive to involve identities in ABE. Client-side policy updating relieves the server of a high load of computations but the drawback is that the client has to be online on that time.

• **Mobile devices** – One important practical issue related to the emerging of mobile technologies is how to make our authorization system work on mobile devices with relatively constrained resources. Some existing works have dealt partially with this problem. Examples are Grewal et al. who proposed an efficient implementation of bilinear pairings on ARM processors [29] or, in the same direction, Sanchez et al. [30]. However the implementations are far from being practically feasible e.g. the benchmarks performed with the JPBC library on a HTC Desire mobile device\(^3\) show that one bilinear operation takes 491ms while one exponentiation in the finite group \(G_1\) takes 1.2 seconds.

---

\(^3\) URL: http://gas.dia.unisa.it/projects/jpbc/benchmark.html#.U-OGFPmSy4I – Testbed 3
• Persistence layer - A practical issue which one could deal with is the persistence layer of our implementation. Since JPBC library does not support serialization we could not store and transmit keys or other elements of the cryptosystem. Because of this our measurements don’t take into account the communication overhead e.g. in key delivery. Therefore a project to make JPBC library data structures serializable would solve this problem and help for more accurate measurements for the authorization system.
Appendix A - Implementation Documentation

Here we put the details of the implementation of the application which is a demo of the new eAuthorization system developed by using multi-authority CP-ABE as explained in Chapter 5. In this appendix we include the user and developer documentation for usage and future development purposes.

A.1 User Documentation

In the user documentation we show the necessary information to run and use the application from its development environment. The scope of the developed ABE eAuthorization system is to provide a simulation of user authorization enforcement. Since it deals only with the enforcement aspect of authorization it does not provide means of managing or distributing attributes to users, which can be done using other tools like Shibboleth. Also it is assumed that the authorization decisions are based on top of a successful authentication which is also not the scope of this application or thesis.

Running the application

Before running the application inside the development environment you need to successfully complete the following three steps:

1. **Import Maven project in Eclipse**

First copy the project’s folder into the workspace of Eclipse. Then open Eclipse and go to “File – Import” select “Maven – Existing Maven Projects”. Localize the root directory of the project (which should be in the Eclipse workspace) and continue with importing. After importing you will notice the errors and warnings since the external library dependencies are not configured.

2. **Import External Dependencies**

To successfully build the project you need to go to the project’s Build Path in Eclipse and select all the jars in the folder “Dependencies/jars” folder and add them as external jars by clicking “Add External JARs”. In Table 7 you can see the external open-source libraries used in the development.
Table 7 - External open-source library dependencies

<table>
<thead>
<tr>
<th>Name</th>
<th>Ver.</th>
<th>Description</th>
<th>License</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPBC (Java Pairing Based Cryptography (JPBC))</td>
<td>2.0.0</td>
<td>Performs the mathematical operations underlying pairing-based cryptosystems directly in Java.</td>
<td>GNU LESSER GPL Version 3, 29 June 2007</td>
<td><a href="http://goo.gl/xTGNls">http://goo.gl/xTGNls</a></td>
</tr>
<tr>
<td>Sun’s XACML Implementation</td>
<td>1.2</td>
<td>Sun’s XACML Implementation is an access control policy evaluation engine written in the Java programming language. It supports XACML 1.x and 2.0 core features.</td>
<td>BSD 2 – Clause License</td>
<td><a href="http://goo.gl/huKM">http://goo.gl/huKM</a></td>
</tr>
<tr>
<td>LibStructure-Graphic</td>
<td>1.0</td>
<td>Allows visualization of any data structure in Java that can be represented as a Directed Tree Graph.</td>
<td>GNU GPL V3, 29 June 2007</td>
<td><a href="http://goo.gl/83K6iv">http://goo.gl/83K6iv</a></td>
</tr>
<tr>
<td>Bouncy Castle</td>
<td>1.50</td>
<td>A Java implementation of cryptographic algorithms containing a light-weight API suitable for use in any environment.</td>
<td>MIT X Consortium License</td>
<td>License URL: <a href="http://goo.gl/P0T2Z4">http://goo.gl/P0T2Z4</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Library URL: <a href="http://goo.gl/NDxqwY">http://goo.gl/NDxqwY</a></td>
<td></td>
</tr>
</tbody>
</table>

Table 7 - External open-source library dependencies

3. Create Resource Folders

The resource folders hold the resources that are to be encrypted and decrypted in the application. They are stored respectively inside the folder “Resources” under sub-folders named “plaintext”, “encrypted” and “decrypted”. If one of these folders is missing you need to create them.

Now you can start the application by opening Eclipse browsing to the gui.java class and run it using the “gui” run configuration. The graphical user interface is developed using WindowBuilder\(^4\) for Eclipse.

Graphical User Interface

The graphical user interface (GUI) is composed of several tabbed panels. These panels are like perspectives of the system from different actors in it. However some perspectives like

\(^4\) http://www.eclipse.org/windowbuilder/
the Users tab can be shared from more than one actor. From each perspective you can perform different operations. The following illustrations will serve as a means to show what can be achieved using ABE eAuthorization demo software.

- **Central Authority**
  - Setup the system
    By pressing the button "SETUP ABE" you initiate an ABE system, basically run the ABE Setup algorithm, initialize a user attributes authority named AA_KMS1 with 100 test attributes with name “role” and value “doctorN” N = {1..100}. Also 10 users are initially set up as part of the system and 10 attributes from the 100 attributes of AA_KMS1 are assigned to each of them (without overlaps).

![Central Authority perspective](image)

- View existing KGAs and add KGA to the system and set issuing attributes for it.
  Click on “Show Attribute Authorities” to see a list of KGAs in the system. If you need to add a KGA then:
1. Press “Clear” to make the table ready to enter the new attributes for the new KGA.
2. Add the attributes names and value.
3. Provide a name for the KGA.
4. Press “Create”.

- **Attribute Authority (i.e. KGA)**

![Attribute Authority screenshot](image)

**Figure 19 – Key Generation Authorities perspective**

From this perspective you can:

- View existing KGAs and their related user groups (users who share the same attribute)
- Add attribute to user (i.e. add user to “User group”)
  1. Select the KGA that issues the attribute
  2. Select the attribute in the left side table to get a list of users that possess it on the right side also called “User group”.
  3. Click on “Add user” button and select the user. After clicking “OK” the user will have the selected attribute in the left.
- Remove attribute from user (revoke attribute)
  1. Select the KGA that issues the attribute
2. Select the attribute in the left side table to get a list of users that possess it on the right side also called "User group". Note that a proper attribute revocation technique needs to be implemented behind this functionality. Also for each revocation the CA needs to be notified.

- Users

![Figure 20 - Users tab](image)

From the users’ perspective you can:

- View existing users’ details.
- View users’ valid and revoked attributes.
- Add new user to the system.

- Encrypt Data
To encrypt data in the system as a particular user:

1. Select a user on behalf of which to encrypt.
2. Select a resource to encrypt.
3. Generate random APT using $n$ attributes from AA_KMS1
   a) Select the number of attributes to include in the APT and press “Generate”.
   b) Press “Encrypt”, wait for encryption process to finish and see the progress in the status box.

- **Decrypt Data**

From this view you can select a certain user perspective and try to decrypt an encrypted resource using that user’s attributes. A revocation method needs to be implemented here also to correctly enforce the policy for revoked user attributes.
A.2 Developer Documentation

In this part of the Appendix we include information necessary for those who want to develop further the eAuthorization system e.g. by extending it with a policy updating method. Here we provide parts of the software documentation and a description of the development environment. Additionally we put together a list API calls for common tasks in the system and a description of the main classes and their relationship.

Setup

Before starting to use the MA CP-ABE cryptosystem one needs to set up a development environment with the proper settings. Here we give a brief “How-to” manual about the setting up the development environment, testing the implementation and further developing it. The implementation is completely written in Java™ on Eclipse Kepler (4.3.2) SR 2.0.

Testbed

The measurements are performed on an Intel Core i7 machine with a 2.2GHz x64-bit processor, 8GB RAM and running Windows 8.1.
**Persistence layer**

In the demo application there is no persistent data left after one execution i.e. setting up the cryptosystem should be done in each run and so does distributing user keys. The reason for this is that the JPBC library offers no possibility to serialize and thus persist its member fields on disk. To make this possible one should modify the JPBC such that its classes follow the serialization rules and thus become serializable. This inconvenience makes it impossible to perform measurements on the transmission costs of the authorization system or having exact measurements on the space requirements of it. We leave this possible improvement as part of future work on the project. The only persistent part of the application is the resources stored in the “Resources” folder inside the project folder. There are three sub-folders inside the “Resources” folder.

**API**

The following classes are developed to enable the setting up and using the MA CP-ABE cryptosystem. Each of the classes contains methods that developers can use to create secret and public keys

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Handles symmetric encryption and decryption for different AES keys, using GCM (Galois Counter Mode). AES GCM offers encryption authenticity (integrity) and confidentiality.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Represents a user attribute in the form (AttributeAuthority:Name:Value).</td>
</tr>
<tr>
<td>AttributeAuthority</td>
<td>Represents the entity of Key Generation Authority with a name, its attributes, and public and secret keys for the attributes.</td>
</tr>
<tr>
<td>CentralAuthority</td>
<td>Represents the Central Authority in ABE system; Responsible for: - setting up the system by choosing 'alpha' as master secret key - computing and publishing (g,g)^alpha as system's PK;</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Provides functionality to encrypt data using ABE; Use the main method encrypt(d,p) to encrypt an element d of Gt with access policy tree (APT) p.</td>
</tr>
<tr>
<td>KeyGen</td>
<td>Provides functionality to generate and return the secret key for user 'u' with an attribute set w and identifier Iu.</td>
</tr>
<tr>
<td>Node</td>
<td>Represents the structure of an Access Policy Tree (APT) in the MA CP-ABE scheme which is an irregular tree made of logical gates in the non-leaf nodes and of attributes nodes in the leaf level.</td>
</tr>
<tr>
<td>Pair&lt;first,second&gt;</td>
<td>Represents a generic pair of objects of any kind; Used to set individual secret key components of users.</td>
</tr>
<tr>
<td>Policy</td>
<td>Represents an access policy with a name and a tree which is a Node with possibly children Nodes.</td>
</tr>
<tr>
<td>Resource</td>
<td>Implements the structure of an encrypted Health Record in the system, supported by metadata necessary for the ABE eAuthoriza-</td>
</tr>
</tbody>
</table>
To use the ABE scheme you need to first initialize the CentralAuthority by running `setupABE()`. In the following table we provide the necessary steps in the code to perform basic operations of the authorization cryptosystem. A more complete API documentation will be provided separately.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CentralAuthority.setupABE()</code></td>
<td>Instantiates the ABE system parameters, <code>PK</code> and <code>MSK</code> using pre-generated system parameters which are in the project build directory in file “a.properties”. “a” here stands for type A bilinear pairings.</td>
</tr>
</tbody>
</table>
| `Encrypt.encrypt(plaintext, policy)` | Use to encrypt `plaintext` using `policy` with public key parameters.  
@plaintext - Plaintext of type Element in GT.  
@policy  - APT used to encrypt the data.  
@return - Ciphertext |
| `KeyGen.getSecretKey(User)` | @User – The user for which to get secret key.  
@return - The secret key components for each attribute of `User` |
| `uDecrypt.uDecrypt(User, ciphertext, minAttributeList)` | @User - User performing the decryption.  
@ciphertext – Ciphertext to decrypt.  
@minAttributeList - A list attributes needed to satisfy the APT  
@return - ABE plaintext associated with ciphertext (Element in G). |
List of Figures

Figure 1 - ABE Traditional Architecture and Workflow .......................................................... 6
Figure 2 - Multiple Key Generation Authorities distribute user secret keys ......................... 8
Figure 3 - Access Policy Tree (APT) ......................................................................................... 14
Figure 4 - PKI approach .............................................................................................................. 22
Figure 5 - IBE approach .............................................................................................................. 22
Figure 6 - Policy Tree for a<11. Source: [9] .............................................................................. 24
Figure 7 - Revocation Proxy Architecture ............................................................................... 26
Figure 8 – UCCMA Secret Sharing in the APT ....................................................................... 27
Figure 9 - Attributes' Hierarchy ................................................................................................. 35
Figure 10 – Randomly generated APT with 5 attributes ........................................................ 37
Figure 11 - Normalized APT ..................................................................................................... 38
Figure 12 - AES key encapsulation in ABE ............................................................................. 39
Figure 13 - Encryption performance ........................................................................................ 40
Figure 14 - Decryption performance ........................................................................................ 40
Figure 15 - XACML target subject conjunction example ....................................................... 43
Figure 16 – Simplified XACML structure of a <Target> element ........................................... 44
Figure 17 - APT mapping of a <Target> element ..................................................................... 44
Figure 18 - Central Authority perspective ................................................................................ 52
Figure 19 – Key Generation Authorities perspective .............................................................. 53
Figure 20 - Users tab .................................................................................................................. 54
Figure 21 - Data upload tab ....................................................................................................... 55
Figure 22 - Data download tab ................................................................................................. 56
List of Tables

Table 1 – Notations.............................................................................................................................................. 15
Table 2 – Efficiency-Security comparison table of ABE schemes................................................................................. 16
Table 3 - Comparison with ITHJ scheme............................................................................................................... 28
Table 4 - MA CP-ABE Scheme Efficiency............................................................................................................... 34
Table 5 - Basic operations in bilinear groups of type A pairing....................................................................................... 41
Table 6 - Performance measurements for data upload.............................................................................................. 42
Table 7 - External open-source library dependencies............................................................................................. 51
Table 8 - API classes.................................................................................................................................................. 58
Table 9 – API.............................................................................................................................................................. 58
Bibliography


2011, pp. 1214-1221.


