TECHNICAL UNIVERSITY BERLIN

MASTER THESIS

Practical Attribute-Based Encryption
For
Secure Cloud Storage Systems

Author:
Marvin PETZOLT

Supervisor:
Prof. Dr. Florian TSCHORSCH

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

FACULTY IV,
Distributed Security Infrastructures

May 14, 2019

Declaration of Authorship

I, Marvin PETZOLT, declare that this thesis, "Practical Attribute-Based Encryption For Secure Cloud Storage Systems", submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Signed: _____

Date: _____

Disclaimer

Parts of Chapter 5 contributed in collaboration with the Hasso-Plattner-Institut Potsdam into the, at this point of writing, unpublished paper *Sukmana et. al.* 2019 "Secure and Scalable Multi-Company Management in Enterprise Cloud Storage Broker System" [47].

TECHNICAL UNIVSERSITY BERLIN

Abstract

FACULTY IV
Distributed Security Infrastructures

Master of Science

Practical Attribute-Based Encryption For Secure Cloud Storage Systems

by Marvin PETZOLT

If a group of users wants to share a file, they usually need to choose a file key under which this file is encrypted. For each entity which shall receive access to the shared file, a copy of this file key has to be re-encrypted using the public key of this entity resulting in many different versions of the same file key. Classical encryption methods reach their storage limits, for a large number of entities. One of the possible solutions to solve this issue is *Attribute-Based Encryption* (ABE), which combines the access rights of different users into one access policy under which a file is secured. We analyze different types of ABE schemes for their practical applicability in the secure cloud storage domain. The implementation of the selected and extended scheme was evaluated for performance, scalability, and security, showing that secure group communication using ABE can achieve sub-linear re-encryption overhead and reduce the number of file keys to a constant amount.

TECHNISCHE UNIVERSITÄT BERLIN

Zusammenfassung

FAKULTÄT IV
Distributed Security Infrastructures

Master of Science

Practical Attribute-Based Encryption For Secure Cloud Storage Systems

von Marvin PETZOLT

Um eine Datei in einer Gruppe von Benutzern zu teilen, erstellt der hochladende Benutzer einen Dateischlüssel mit welchem die Datei verschlüsselt wird. Damit ein neues Mitglied Zugriff auf diese Datei erhält, wird eine individuelle Version des Dateischlüssels mit dem jeweiligen öffentlichen Schlüssel des Benutzers neuverschlüsselt. Diese herkömmliche Verschlüsselungsmethode skaliert, auf Grund der Unmengen von Schlüsseln, nur schlecht mit steigender Benutzeranzahl. Attributenbasierte Verschlüsselung (ABE) kann helfen die Schlüsselmengen zu reduzieren, indem sie mehrere Zugriffsrechte der Benutzer in einer Zugriffsregel kombiniert. In dieser Arbeit werden verschiedene ABE Ansätze auf ihre praktische Anwendbarkeit im Bereich sicherer, cloud-basierter Datenspeicher untersucht. Die Implementierung von dem gewählten und erweiterten Schemata wurde auf Geschwindigkeit, Skalierbarkeit und Sicherheit analysiert mit dem Ergebnis, dass eine sichere Gruppenkommunikation basierend auf ABE einen sublinearen Neuverschlüsselungsaufwand erzielen kann und die Anzahl der Dateischlüssel auf eine konstante Anzahl reduzieren kann.

Acknowledgements

To complete this thesis, I like to thank my external supervisors Muhammad Sukmana (Hasso-Plattner Institut, Potsdam), Kennedy Torkura (Hasso-Plattner Institut, Potsdam) and Hendrik Graupner (Bundesdruckerei GmbH). Further, I want to thank neXenio GmBH to let me use their rooms and Bundesdruckerei GmbH to let me analyze real-world data of the secure cloud storage system Bdrive.

Contents

Chapter 1

Introduction

In the last past years, we witnessed a shift of resources from dedicated end-systems to large-scale transparent server farms: The cloud. This is achieved by using dynamic and scalable resources, in which each user only accesses what he or she needs. One of the earliest adopters of this change were cloud storage systems, whereby storage space was given dynamically to users when needed and upgraded when exceeded.

With growing globalization of businesses, they seek for a secure solution that allows them to access internal data globally and collaborate with people around the world without setting up their own infrastructure. Secure cloud storage systems employ end-to-end encryption to secure the transmission of files between entities, without the possibility of eavesdropping sensible data by third parties. [2] One of the most practiced methods to ensure end-to-end encryption is public-key cryptography.

In public-key cryptography, each user is identified by his unique public and private key pair. The public key, as the name indicates, is made publicly available to that every entity that wants to encrypt content can use the public key of the recipient to do so. The private key, on the other hand, remains securely and offline on the machine. Asymmetric cryptography uses one-way functions so that content which is encrypted by the public key can only be decrypted with the respective private key. The advantage compared to symmetric cryptography, where multiple parties share the same secret key, is that no secret key exchange needs to take place. Since symmetric cryptography is usually less computationally intensive than asymmetric cryptography,[1] a hybrid solution between those two is usually chosen. [44]

Peer-to-peer communication works well with this scheme, but in the secure cloud storage use-case, content needs to be accessed by multiple participants. The data owner has to encrypt the content for each user explicitly. This is done by creating a so-called *file key*. The file key itself is a symmetric key which is used to encrypt the plaintext file. To distribute this file key to each entity that should receive access to the file, it is encrypted asymmetrically with the public key of the respective participant. This results in many different, encrypted versions of the same file key. Assuming that, 100 users, each having an own public-private key pair, share 200 files, $100 * 200 = 20000$ file keys would be created. Even worst, for each additional file, another 100 file keys have to be maintained. In addition, users usually own multiple public-key identities, one for each device. Such schemes do not scale well in the secure cloud storage domain. Here often data owners want to share a large amount of the same content with many different users at the same time. To overcome this obstacle, an encryption scheme needs to be employed which has a constant number of access keys regardless of the number of participants.

[1] Under the assumption that RSA is used to represent asymmetric cryptography and AES representing symmetric cryptography. Each having the same bit-security.

Attribute Based Encryption (ABE), first introduced by Sahai and Waters [36], is a cryptographic encryption scheme which uses attributes of a user as keys for encryption. ABE is originated from the idea of *Identity-Based Encryption* (IBE) [41] which binds identity information, such as an e-mail address, to the public key of the addressed entity and eliminating the need of certificate-chain validations from a public-key infrastructure or the peer public key validations from the web-of-trust. [36] Instead of binding unique user identifiers to the key, ABE binds it to attributes. This enables the data owner to craft a ciphertext over a chosen access policy, combining different attributes with *AND* or *OR* threshold gates to formalize access conditions. This approach is called *Ciphertext-policy Attribute-Based Encryption* (CP-ABE). [4] It is also possible to do it the other way around: Associate the user's key with an access policy, formally known as *Key-Policy Attribute-Based Encryption* (KP-ABE). [16]

Since a single access policy allows multiple entities to decrypt the content, if, and only if, they satisfy the given access condition, it is possible to reduce the number of file keys to one which makes ABE a suitable candidate to solve the scalability problem of secure cloud storage systems. Further, the need for an explicit authorization check before downloading a ciphertext gets eliminated.

ABE uses a so-called *Attribute Authority* (AA) to generate attribute secret keys and distribute them to users. Since users might collude with each other to archive a higher level of decryption power, the AA is in charge of issuing individualized secret keys to each user and thus making it impossible to combine keys. [4]

However, single-authority ABE comes with the disadvantage of global decryption power of the AA. In a secure cloud storage system which only addresses one company, this might not be an issue. Due to possible key loss, encrypted files need to be recoverable by the company administrator anyway. In a multi-company setting, where companies share files between them, an entity with global decryption power is unacceptable. *Multi-Authority Attribute-Based Encryption* (MA-ABE), first proposed by Chase 2007 [8], allows multiple attribute authorities to maintain different attribute domains. This is a much more practical method since it can deescalate the global decryption power of a single, central authority.

The MA-ABE scheme that satisfies the most requirements, shows the best performance and scalability overhead is *Two-Factor Data-Access Control for Multi-Authority Cloud Storage systems* (TFDAC-MACS, [26]). Since it fits not completely the defined requirements, we propose Practically Applied Distributed Two-Factor authentication Data Access Controll for Multi-Authority Cloud Storage systems(PAD-TFDAC-MACS), a fully functional and practical adaptation to TFDAC-MACS that is more suitable for the practical secure cloud storage use-case. In addition to the advantages of TDFAC-MACS, PAD-TFDAC-MACS profits from more complex access formulas that can be present in *Disjunctive Normal Form* (DNF), dynamic secret key issuing and the option to disable two-factor authorization.

PAD-TFDAC-MACS builds upon the security assumptions of TFDAC-MACS which is proven secure against chosen plaintext attacks in the random oracle model. [26] To show that the previous states adaptations do not threaten security we directly deviate PAD-TFDAC-MACS from TFDAC-MACS.

In order to validate that the re-encryption problem can be solved a performance and scalability evaluation of PAD-TFDAC-MACS against the classical public-key cloud storage system is evaluated. It shows that the prototype reduces the file-keys depending on the chosen access policy. In the worst-case scenario (policy containing only OR-gates), PAD-TFDAC-MACS scales with the same overhead of file keys as

the reference system and thus showing no scalability advantages. However, the best-case scenario (policy containing only AND-gates) showed constant overhead of file keys, linear to nearly constant ciphertext size and constant computational overhead. Regarding the number of file keys, PAD-TFDAC-MACS does strictly scale better.

The performance of PAD-TFDAC-MACS shows an additional overhead against the RSA re-encryption technique. By using the best-case scenario as a reference, a certain number of users, which are defined by the same attribute, can be found from which on PAD-TFDAC-MACS performs better than RSA using re-encryption. Where this number is located depends on the underlying system. While our server needs roughly 145 users to be defined by one attribute to show a performance advantage, on mobile devices with limited computing power the number of users will be even higher.

This leads to the conclusion that PAD-TFDAC-MACS should be applied in a secure cloud storage system that deals with a huge amount of users where many of them can be clustered by the same attribute. In systems where a lot of peer-to-peer file-sharing is done the advantage that can be gained by reducing the number of file-keys is not worth the performance trade-off compared to the classical RSA re-encryption technique. In general the performance and storage overhead of PAD-TFDAC-MACS depends on the chosen access policy of the ciphertext.

The rest of this thesis is structured as follows. First, related work will be listed in Chapter 2. Chapter 3 summarizes the basic background of the current implementation of a secure cloud storage system. The re-encryption problem is sketched. In addition to the general and security requirements, a threat model for a distributed system is given. In Chapter 4, different schemes of secure group communication, attribute-based encryption, and multi-authority attribute-based encryption are analyzed and further argumentatively and practically compared against each other.

The most suitable scheme, as defined from the requirements and the threat model, is TFDAC-MACS [26] by satisfying all non-optional requirements. Since TFDAC-MACS does not fit completely into the described use-case, an improved version PAD-TFDAC-MACS is introduced in Chapter 5. A design for a practical applied, distributed system is given. Finally in Chapter 6, the performance and scalability of this scheme will be evaluated by comparing it to the original secure cloud storage system. In the last Chapter 7, an outlook about the future improvements and adaptions of PAD-TFDAC-MACS is given.

Chapter 2

Related Work

CloudRAID [17] [40] [38] [39] is a cloud storage broker system that achieves reliable and secure data storage by encrypting content on the client device and using erasure encoding to split the encrypted file into chunks, distributing each to a different cloud storage provider. *Sukmana et. al.*, 2017 [46] proposed a *Ciphertext-Policy Attribute-Based Encryption* scheme for CloudRAID, namely "CloudRAID for Business". While focusing on the single-authority use-case, CloudRAID for Business showed that ABE is practically applicable to secure cloud storage systems. However, it ignored the need for inter-company file sharing which requires many distinct attribute authorities each administering their own domain and thus deescalating global decryption power. This work focuses heavily on eliminating the global decryption power of a single-authority while further improving the scalability of CloudRAID regarding the number of file keys. The adapted scheme contributed into the (at this point of writing unpublished) follow up paper *Sukmana et. al.*, 2019 "Secure and Scalable Multi-Company Management in Enterprise Cloud Storage Broker System" [47].

2.1 Scalability Of File Keys In Secure Group Communications

A common approach to reducing the number of file keys is to organize file keys in a tree hierarchy. [49] [42] A proposed implementation by *Sandhu* 1988 [37] uses one-way functions which take the parent file key as input and output the child file key. This way a client can directly translate his file system into the file key tree. All files in the same directory are encrypted using the directory file key. To share a folder, the data owner just needs to provide the key for this directory and each recipient can calculate recursively the child file keys. However, due to the hierarchical structure, users who were granted access to a higher level directory would also be able to decrypt the underlying directories, which is not always desirable.

Harney and Muckenhirn 1997, [18] proposed a group key management protocol that creates a new file key for each new group. Another approach utilizes a Key-Aggregation Cryptosystem which combines different access keys into one constant sized key aggregate. [10] [19]

In a similar way as in ABE, in fuzzy *Identity-Based Encryption* [36], a key can be encrypted under multiple identities creating one compart secret key. However, the identities are restricted to be in a close metric space and thus limiting the applicability on an arbitrary field. [10]

In this thesis, we will focus on reducing the storage overhead of file keys by the application of ABE. This is especially promising on the business domain since it is inherently hierarchical and users are already clustered into business groups. Exploiting this assumption, we argue that we can theoretically achieve better scalability then alternative approaches.

2.2 Practical Application Of Attribute-Based Encryption

The topic of ABE enjoyed a lot of research attention in recent years with rapid developments. Since ABE offers a great feature variety (fine-grained access control [16], constant sized ciphertext [26], efficient en-, re- and decryption [54] etc.) schemes are often adapted to better fit the desired use-case. Therefore there are many different schemes that all fit a specific use-case, but a standardized ABE scheme is yet to be found.

ABE's first practical application was the medical sector. By nature, it provides an already existing attribute and user management infrastructure which is required by ABE. Users, as the data owner of their medical record, can define access policies so that only authorized staff can access their medical history. Evaluations for that use-case with focus on mobile end-devices were conducted by *Akinyele et. al.*, 2011 [1]. Similar research focusing on scalability and security by *Li et. al.*, 2013 [24] and later a practical application, evaluation and user case study by *Thatmann et. al.*, 2016 [48] was conducted which analyzed the practicability of ABE in a hospital environment. While these schemes focus on putting the ownership of the medical record back into the patient's hands, this work will focus on the scalability of the number of created file keys.

Cloud storage systems deal with an overhead of authentication and authorization when a cloud storage provider offers a file for download. ABE can eliminate this obstacle by encrypting the data with the access policy which binds the authorization to the decryption process itself. *Wang et. al.*, 2010 [50] and 2011 [51] evaluated a new multi-authority scheme that is based on a hierarchical structure with the application to cloud storage services. Since the root entity (the entity who bootstraps the system) has global decryption power, different other approaches emerged that tried to deescalate the decryption power of the root entity.

"DAC-MACS: Effective data access control for multi-authority cloud storage systems" 2013 [54] [52] [25] [26] created a whole new field of MA-ABE related schemes that improved each other. These schemes focus especially on cloud storage systems and the efficient revocation, encryption, and decryption process without having an entity with global decryption power.

While this work will build up especially on the latest work of *Li et. al.*, 2017 "Two-factor data access control with efficient revocation for multi-authority cloud storage systems" (TFDAC-MACS) [26], it improves this scheme by more fine-grained access control and practical usability by maintaining security and scalability in the enterprise cloud storage scenario. Further, in this thesis, *Cloud Storage Provider* is only used as dummy storage which is not involved in any proxy-re-encryption since the file key is secured by the ABE scheme instead of the file itself.

2.3 Overviews And Comparisons Of Attribute-Based Encryption Schemes

Since the field of ABE rapidly grew researches started to cluster the different subtopics of ABE. *Dubey et. al* 2015 [14] conducted a survey about known ABE schemes. They listed the advantages and disadvantages of those schemes. Since the practical implementations of ABE scheme are very spare, the few known implementations are summarized, clustered, and benchmarked by *Zickau et. al.* 2016 [55].

Other survey papers try to evaluate the current field to a specific topic. Since the field of multi-authority ABE with the application to cloud storage systems grow with

the creation of the DAC-MACS family different survey papers listed techniques, differences, advantages and disadvantages of ABE in cloud environments. [33].

Mostly this work builds up and extends (in Chapter 4) the work of *Lee, Chung and Hwang* 2013 [20]. They clustered the research field of ABE into different sub-topics and extracted requirements for the application of cloud environments.

In comparison to the previously mentioned surveys, this work focuses on the scalability aspect of ABE schemes with respect to the storage consumption of file keys. To do so we perform practical scalability and performance analysis on a scheme-comprehensive level with a focus on the secure cloud storage domain.

Chapter 3

Preliminaries

In the past 20 years, we witnessed a shift from resource consumption on the user side to outsourcing more and more resources into the cloud. In the same way files and data were moved into cloud storage systems. Due to untrusted *cloud storage provider* (CSP), and the raising intransparency the need tor a secure cloud storage solution emerged.

In this thesis, we will focus on the enterprise application of secure cloud storage systems. This is due to two circumstances. First, enterprises usually introduce a large number of users who want to share files between different divisions. This is important since we will also focus on the feature of inter-company sharing. Secondly, enterprise systems introduce a natural hierarchical structure where each user can be identified by attributes. [46] This allows us to also analyze the applicability of Attribute-Based Encryption.

3.1 CloudRAID

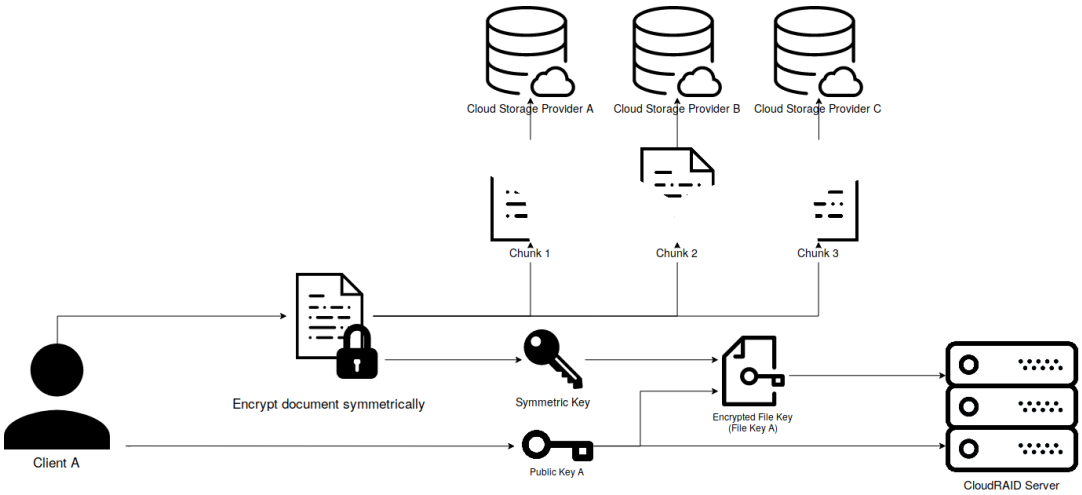

FIGURE 3.1: Device uploads an encrypted file to the CSPs, the file key and public key to the CloudRAID server.

Schnjakin et. al. 2013 proposed CloudRAID [40], a secure cloud storage broker system which uses erasure encodings to splits up files in smaller chunks that are saved separately on different cloud storage providers (CSP). [39] To ensure end-to-end encryption, a CloudRAID client generates a new one-time symmetric key (AES - 256 bits) and uses it to encrypt the plaintext file locally. [40] This key is called a *file key*.

Briefly, the security design of CloudRAID for the business use-case, as described by *Sukamana et. al.* 2018 [46], will be summarized. While the encrypted file is

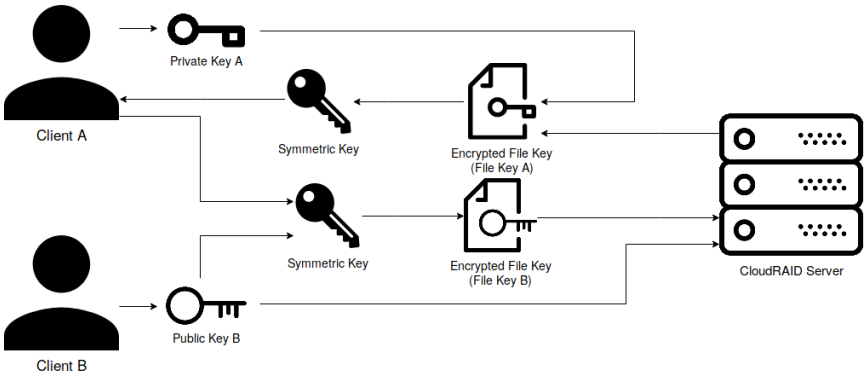

FIGURE 3.2: Device A grants Device B access to the uploaded file by re-encryption the file key.

split into chunks and uploaded to different CSPs, the file key is encrypted under the devices' own public key to formalize the *encrypted file key*. It is uploaded to the CloudRAID server where it is stored securely, as shown in Figure 3.1. The CloudRAID server itself nor any of CSP does have any knowledge about the plaintext content since for decryption the private key of the device is needed.

If the user wants to access a file locally, the devices request the encrypted file key from the CloudRAID server and download the file chunks from the CSPs. Locally, it decrypts the file key with its private key and finally deciphers the assembled file with the recovered file key.

However, this process turns out to be much more computationally complex in a multiple device setting. If a user registers more than one device, the existing data needs to be synchronized to the new device. Since each device has its own private-public key pair (RSA - 4096 bits), an existing device is in charge of making file keys available for a new device. This is done by downloading each encrypted file key for the respective file, receiving the public key of the new device, decrypting the encrypted file key with its own private key, encrypting it again with the public key of the new device and finally, uploading the new encrypted file key to the CloudRAID server. This process is called *re-encryption*, as shown in Figure 3.2.

Usually, in cloud storage systems we also have the concept of *groups* which we will also call *shares*. They describe a collection of clients which share data between them. To express the overhead of joining or leaving a share the following notation will be used: f and n nominate the number of encrypted file keys and the number of devices in a share respectively.

If a new device joins a share in CloudRAID a device needs to make the existing file keys available to the new devices resulting in f additional encryptions, messages, and keys. However, a big advantage of this scheme is that forward secrecy[1] is computationally very cheap. Here, all encrypted file keys belonging to the removed/leaving device are deleted and the other devices do not encrypt further uploaded files for the revoked device anymore. The backward secrecy constraint, while being an imported security feature, will explicitly be broken by clients. This is due to the feature, that data owner can invite a new member into a group and the new member accesses all previously uploaded content. CloudRAID is ensuring this by explicitly re-encrypting all file keys for the new device.

[1]Forward Secrecy: The left entity will have no knowledge about future shared content.

3.2 Problem Description

This current approach of file sharing is not scalable for a large number of users and devices. Each device invited to a share needs to have an own version of the encrypted file key issued by encrypting it with device's public key. This process scales with $O(n * f)$ keys. Further, the same number of messages containing the encrypted file key needs to be sent and each file key needs to be encrypted n times which also results in $O(n * f)$ encryptions in total.

To sketch a practical example, let us assume a manager of a company with 50 employees wants to create a company-wide share. Each of the employees has at least two devices (say one web client and a desktop client). To upload the latest presentation, the device of the manager has to perform $50 * 2 = 100$ encryption operations, resulting in 100 encrypted file keys which need to be distributed to the respective entities causing traffic of $100 - 1$ messages. Even worse, for each new presentation upload, another 100 encrypted file keys need to be maintained. In a large scale company, this overhead becomes unmaintainable when the number of ≈ 200 encrypted file keys per file is exceeded. [2] With increasing computing power this problem can be compensated but not prevented.

Another common business use-case is that companies want to define different access levels for different users or devices. Considering the previous example, the company manager would like to share a confidential file with all full-time employees. In that way, he needs to create a new share and invite all employees that should be granted access. Each new employee needs to be explicitly added to the share manually. Translating the hierarchical structure into authorization policies is not possible with this approach. A practical solution is missing that reduces the number of file keys and allows to dynamically grant entities access to shares if they satisfy a certain qualification goal.

3.3 Threat Model

For the goal of this work, the threat model and different trust level of the entities of the system are defined as follows:

- The **Central Server of CloudRAID** is assumed to try to break into client communication. It might provide false information or wrong states to trick any identity to provide sensitive content. However, the central server will cooperate and will only deviate from the protocol if some advantage could be gained. Simply denying the cooperation - and so disrupting the system - does not belong to the goals of the central server. The central server does not cooperate with other malicious entities.

- **Companies** compete against each other and thus try to eavesdrop content that is encrypted outside of their domain. A company will provide another company with false information to disrupt its service or to achieve an advantage. This distrust can be leveraged by a trust relationship where two companies can explicitly agree to mutually trust each other. This mutual trust relationship indicates that a trusted company is assumed to provide only truthful information to the other trusted company.

[2] Running 'openssl speed rsa4096' on an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz takes on average 0.00499s for encryption. Multiplied by ≈ 200 the second mark is reached.

It is assumed that the company administrator does not betray the company by selling user information or by cooperating with any untrusted company. He/She strictly follows the interest of the company of protecting internal data and user information.

- **Cloud Storage Providers** are simply assumed to be honest-but-curious. They follow the protocol but try to decrypt content if possible. A cloud storage provider is separated from the rest of the system and does not cooperate with any other malicious entity.
- **Users** are untrusted. They try to collude with other users to obtain a higher level of decryption power and thus accessing and decrypting unauthorized files.

  However, it is assumed that they will not sell a decryption black box so that other external, unauthorized or revoked users use the black box to access sensitive content. Revoked or demoted users may use their credentials to access previously accessible data.

3.4 Requirements

Based on the previously introduced entities, the threat model 3.3 and the use-case of a secure enterprise cloud storage system the following requirements can be extracted.

The general requirements (***B***) of this thesis will be summarized by two major points:

B1 **Scalability:** The system should scale, with an increasing number of user and devices, better than the current cloudRAID encryption scheme with respect to the number and storage overhead of file keys.

B2 **Performance:** While improving the scalability the performance impact of that system should be reasonably low. Participating in the system should be possible with low-performance devices (such as mobile phones).

In addition, the following core security requirements (***C****) for such a secure cloud storage system are extracted and extended from the work of *Lee, Chung, and Hwang* 2013 [20] (***B1***, ***C8***, ***C1***, ***C9 O1***, ***O2***).

C1 **Collusion resistance:** For two users it should not be possible to combine their keys to obtain a higher level of decryption power. Collusion resistance need also be ensured on revocation.

C2 **Companies:** Each company is only responsible for its own domain. A company can neither access nor administer users or data from another company. An exception to this constraint, are companies that agreed on a mutual trust-relationships to enable inter-company file sharing. Now they are eligible to also assign keys or attributes to trusted, external users.

C3 **CloudRAID Server:** If a new company is registered the CloudRAID server provides necessary security parameters, while having no (global) decryption power.

C4 **Master Key (if any):** Key recovery requires a secret and securely stored master key. It should solely function in the company domain and not globally.

C5 **Large Universe:** The number of keys and users is not restricted by any bound.

C6 **Bootstrapping New Companies:** It should be possible to add new companies at runtime. Without either shutting down the system or recreating each key.

C7 **Malicious Company:** A corrupt company can only harm its own domain, but can not harm the outside system in any way (except trust relationships). It cannot gain any additional information by action malicious.

C8 **Forward Secrecy:** To employ a practical applicable system, user management must be implemented. This includes revoking a user from the system so that he/she can not access further shared content. If a user gets demoted he/she can not access the previously accessible files anymore.

C9 **Data confidentiality:** Any data in the system is only accessible and decryptable by authorized entities. This includes the data owner, the user who was granted access to the data and the company administrator (if necessary). Any other party, like the CloudRAID server or CSP can not decrypt files in any kind.

C10 : **Access Control:** User could share a file with other users where a file can only be decrypted by its authorized user following defined access conditions.

Other (optional ***O****) requirements are:

O1 **Traitor tracing:** Misbehaving users, who sell their access or file keys to other unauthorized entities to create a decryption black box, should be identifyable [30].

O2 **Fine-grained access control:** The user shall not be bounded on defining fine-grained access policies which requires either an access tree [4] or a linear secret sharing scheme (LSSS) [54] [25] [52] [23] [30].

3.5 Contribution

In this work, different schemes suitable to resolve the re-encryption problem were compared. First, related work in the area of secure group communication is analyzed on a theoretical base. Then it is argued why *Attribute-Based Encryption* (ABE) might scale even better and different subtopics in that research area are compared in a practical evaluation. To do so the charm framework[3] was extended and used as a homogeneous platform to evaluate different benchmarks with respect to the performance and scalability of thous schemes. Those benchmarks are novel in the sense that no other known researchers compared the different ABE topics on a practical level.

Secondly, we propose PAD-TFDAC-MACS (Practically Applied Distributed Two-Factor authentication Data Access Controll for Multi-Authority Cloud Storage systems) a practical and more expressive adaption of the work of *Li et. al.* 2017 "Two-factor data access control with efficient revocation for multi-authority cloud storage systems" (TFDAC-MACS). TFDAC-MACS was extended to make the two-factor authorization optional, the secret key issuing more dynamic and the access policies more fine-grained by introducing the possibility of m-of-n threshold gates on top level of the access tree.

[3] https://github.com/Anroc/charm

Further, system design for a distributed infrastructure of PAD-TFDAC-MACS is given. To show that the proposed prototype is capable of solving the scalability problem of the file keys for a secure cloud storage system, an evaluation is conducted comparing PAD-TFDAC-MACS to the re-encryption scheme of CloudRAID. This evaluation not only shows that our solution is able to reduce the number of file keys to a constant amount, but also that it does this by neither threating security nor heavy performance impacts. However, it must be noticed that the performance and scalability of ABE systems, and especially PAD-TFDAC-MACS , heavily depends on defined access policies and the number of users that can be summarized by a single attribute.

As the final part of the contribution, we give an outlook of the applicability of PAD-TFDAC-MACS in the real world and sketch scenarios when it makes sense to switch from the current security concept of CloudRAID to PAD-TFDAC-MACS .

Chapter 4

Comparison Of Related Schemes

In this chapter, a survey of the related schemes is conducted. They are analyzed based on their applicability to the secure enterprise cloud storage domain and the requirements of Section 3.4. The goal of this chapter is to find the most suitable related work that solves the scalability problem of CloudRAIDs re-encryption scheme. However, since we focus on the application in the business domain, we are dealing with a hierarchical and strictly clustered user structure.

While first comparing schemes on a theoretical level, the more specific the schemes get the more practical our analysis becomes. The final schemes are practically evaluated by measuring performance and scalability in different settings and with an increasing number of users.

4.1 Secure Group Communication

Secure Group Communication (SGC) describes, as summarized in the survey of *Rafaeli et. al.* 2003 [35], schemes that reduce the number of ciphertexts by encrypting all messages with a so-called *group key* (GK) each time the same message is send to multiple devices. Applied to the domain of secure cloud storage systems, SGC can reduce the number of file keys by encrypting all files with a unique GK that is only known to the members in the share. A *Key Distribution Center* (KDC, CloudRAID server) is used to distribute the respective group keys to different members. [18] However, while CloudRAID could manage forward secrecy quite efficiently, these schemes suffer from additional re-encryption overhead if a member leaves the group. [35]

Commonly Secure Group Communications are mentioned together with *multicast messages*. With multicast, the same message is distributed to different devices in one setting. [35] In contrast stands *unicast* where a central server is required that manages authentication and authorization to decide which entity should access which content. With multicast messages, however, content is distributed to each entity regardless if this entity is authorized to access this content. Instead, authorization is handled via encryption of the content. If an entity owns the respective key to decrypt the data it is implicitly granted access to the content as well. Multicast messages are especially handy in group communications since they only need to send one message to distribute the information to all members. As a trade-off the message size usually increases. [35]

While there are many approaches researching secure group communication and making the re-encryption process more efficient, the number of schemes can be reduced to those applicable to secure cloud storage. Here some prerequisites are given that are not generally assumed. Secure cloud storage systems often deploy a public key infrastructure (GK) to bind identities to public keys of clients which means that a key exchange already happened.

In contrast to that are, for example, schemes that extend the 2-parties Diffie-Hellman (DH) key exchange to an n-party scheme, such as described by *Steiner at. al., 1996* [45]. The DH key exchange is designed for unsecured environments to create a unique communication key, known to every member of the group, but not known to a passive attacker. However, since all communication is already disclosed and secured by the GK those schemes can be eliminated. They suffer from additional computation or message exchange overhead compared to schemes that assume an existing GK.

In the following, it is widely assumed that a client is identified uniquely by its public key, which is also known by the cloud server and can be queried by other clients.

4.1.1 Group Key Management Protocol

The Group Key Management Protocol (GKMP) [18] addresses the re-encryption problem in a simple but scalable way. If a device wants to share files with a group of other clients it creates the group key and encrypts all file keys with this group key. This GK needs to be securely distributed to all members which means that the data owner needs to download and store every public key of the group members and encrypt the GK for each of them. Further, the data owner needs to encrypt all shared file keys symmetrically with the chosen group key.

To ensure forward secrecy, the GK needs to be renewed every time a member leaves the group. [35] This overhead is located in the same computational magnitude as bootstrapping a new group.

The big advantage of this scheme, compared to the scheme employed in CloudRAID, is that it reduces the overhead of the number of encryptions, messages, and keys on new file upload dramatically. While the data owner has to create a file key for each member respectively, GKMP reduces this to one encrypted file key per upload. Additionally, since no backward secrecy has to be ensured, the GK only have to be distributed to the new member[1].

4.1.2 Logical Key Hierarchy

Logical Key Hierarchy (LKH) [49] is a SCG scheme that reduces user side storage and re-encryption transmissions by organizing users keys in a tree hierarchy and maximizing the use of multicast messages. The tree is maintained by a central *Key Distribution Center* (KDC). In the work of *Wallner et. al.* [49], which in the following be cited to describe LKH, it is explicitly stated that such trees neither have to be balanced nor binary, but for the sake of simplicity exactly this is assumed. Users (and their respective public keys) are organized at the leaves of the LKH. Each node is composed of a key that is encrypted with its children keys. An encrypted key is known to all leaves related to this node and can, on change of this node, be transmitted in one multicast transmission to all its leaves. The root node summarizes the GK. The GK is used in the same way as in GKMP to secure group communication.

While in GKMP each user needs to store each member's public key to eventually re-encrypt the GK, in LKH each user only needs to store $log(n) + 1$ keys (path from a leaf to root node). Beginning from the bottom, each leaf node could decrypt the next parent node until it arrives at the root node. One other advantage is that many members share the same keys. So it is more efficient to transmit these keys in a multicast setting. LKH needs to do $2log(n)$ multicast transmissions and analogous

[1] With backward secrecy the GK would have been renewed each time a new member joins.

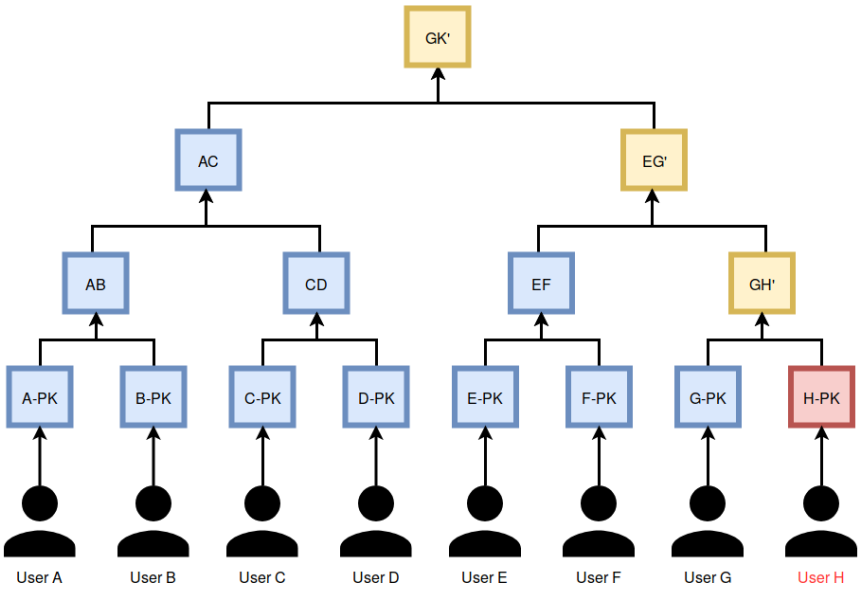

FIGURE 4.1: Balanced binary LKH and user H leaves the group. Node marked in red is removed from the group. Node marked in yellow require an update and need to be stored locally by any leaf of this node. (Graphic adapted from Figure 1 in [35])

$2log(n)$ encryptions on each member leave or join. Here each updated key in the path from a leaf to root node needs to be updated and encrypted two times: One time with the key of right children node and second with the key of the left children node.

In Figure 4.1 user H recently left the group which means, that to ensure forward secrecy, the keys in node GH, EG and GK must be updated. The key server will choose a new GH', EG' and GK', encrypts and distributes them in a bottom-up fashion starting by GH'. GH' is encrypted with user Gs and user Hs public key (G-PK, H-PK) and transmitted to them respectively. Next, user E and F receive in a multicast transmission EG', which is encrypted with EF and user G and H receive in a multicast transmission EG' which is now encrypted with the newly established GH' key. Finally, the GK needs to be updated. For the left subtree GK' is encrypted with AC and transmitted using multicast to the users A, B, C and D. And for the right subtree GK' is encrypted with EG' to transmitted using multicast to user E, F, G, and H. Adding a user to the group follows the same principle. Backward secrecy is implicitly given and not easily removable from this scheme.

4.1.3 One-Way Function Trees

To reduce the transmission and encryption overhead of $2log(n)$ even further to $log(n)$ other schemes use pseudorandom functions [7] or one-way functions [42] to compute the required keys in the path. These schemes are strongly related to *Merkle Trees* since each update of the user set will force an update of the root node: The group key. We will again cite the work of *Rafaeli et. al.* [35], to describe this scheme.

Each user stores a blinded key for each sibling node in the path to the root node. Starting from his individual key, a user can compute the blinded version of his key. To compute the next parent key, a user utilizes his blinded key, together with the siblings blinded key, which is fed into a one-way function to create the key for the

parent node. This node needs to be blinded to serve as the input for the next level. In such a way the user can compute the needed keys up to the GK.

Let us assume the same tree layout as in Figure 4.1 but assume that user H joined the group. Given a cryptographically secure hash function h. First, user H needs to receive the blinded keys, $h(AC)$, $h(EF)$ and $h(G-PK)$ which he stores locally in addition to his own public key $H-PK$. *One-Way Function Trees* (OWFT) have the same storage overhead as LKH with $log(n)+1$. As already known from LKH, GH, EG and GK need to be updated. These nodes are computed from their children blinded values using a merging function m which takes two inputs and produces one output. This could, for example, be the concatenation of both inputs and then applying h to receive a fixed length output value. Starting from the bottom, GH' is computed as $m(h(G-PK),h(G-PK))$, analogous $EG'=m(h(EF),h(GH'))$ and finally the updated group key $GK'=m(h(AG),h(EG')$. The new blinded keys are distributed to their respective users using one multicast transmission each, reducing the transmission and encryption overhead to $log(n)$. If user H leaves the group, user G must come up with a new random secret that is hashed as a placeholder for the previous blinded key of H to compute GH'. Thus forward and backward secrecy is enforced.

4.1.4 Comparison Of Secure Group Communication Schemes

	CloudRAID [40]	**GKMP** [18]	**LKH** [49]	**OWFT** [42]
initial share				
file keys	nf	n	$n-1$	n
messages (unicast)	nf	n	$n(log_2(n)+1)$	$2n(log_2(n)+1)$
messages (multicast)	nf	n	$log_2(n)+1$	$n-2$
encryptions	nf	$f+n$	$f+n-1$	$f+n-1$
member join				
file keys	f	1	$3log_2(n)$	$log_2(n)$
messages (unicast)	f	1	$2(n-1)$	n
messages (multicast)	f	1	$2log_2(n)$	$log_2(n)$
encryptions	f	1	$f+2(log_2(n))$	$f+log_2(n)$
member leave				
file keys	0	n	$3log_2(n)$	$log_2(n)$
messages (unicast)	0	n	$2(n-1)$	0
messages (multicast)	0	n	$2log_2(n)$	$log_2(n)$
encryptions	0	$f+n$	$f+2(log_2(n))$	$f+log_2(n)$
addition of file key				
file keys	n	0	0	0
messages (unicast)	n	0	0	0
messages (multicast)	n	0	0	0
encryptions	n	1	1	1

TABLE 4.1: Comparison of secure group communication schemes. n donates the number of clients and f the number of files. Forward secrecy is ensured in each scheme. Assuming a balanced binary tree as in Figure 4.1.

Table 4.1 compares the discussed schemes against each other regarding the number of keys, messages, and encryptions on initialization of the group, when a member joins or leaves and on the addition of a new file.

It is assumed that clients already downloaded the public keys of their group members. Forward secrecy needs to be ensured every time a member leaves the group. Further, messages in table 4.1 are transmissions that contain a key. There are also meta messages that notify the members about a new file upload or the removal of a member, etc. They are ignored since they add a constant overhead to all schemes. If a message could be processed by multiple communication partners it can be transmitted in multicast transmission. In general, it is assumed that $f > n$ holds true since there are usually more files in a share than devices.

Reviewing the performance comparison, CloudRAIDs scheme is not optimal compared to the different approaches. In fact, CloudRAID has the worst performance on initialization, on addition of an encrypted file key and on average on member join as well. However, CloudRAID shows great performance on member leave since clients simply do not have to encrypt anymore for the left member.

Each of the schemes have their own strength and weaknesses. GKMP has the smallest overhead on member join and OWFT the best re-encryption overhead on member leave. As some could clearly extract from the table that all secure group communication schemes perform better than CloudRAID on file upload. CloudRAID needs to distribute and encrypt a new file key n times for each device while any scheme using a GK only needs one encryption and one multicast transmission.

Under the assumption that secure cloud storage does not need backward secret, GKMP is selected as the most suitable candidate. It has great overhead on member join is easily understandable and profits from the fact that no backward secrecy on member join is needed. LKH and OWFT both have backward secrecy fixed implemented resulting in a slightly performance loss.

4.2 Predicate Encryption

As stated by *Dan, Sahai and Waters* 2011 [6] about classical encryption schemes: *"Access to the encrypted data is all or nothing – one can either decrypt and read the entire plaintext or one learns nothing at all about the plaintext other than its length."* While in the most systems such a scheme might be sufficient, especially in the cloud storage domain there is a need for more flexible computations on ciphertexts. [6]

To achieve a constant overhead regarding the number of file keys, all eligible entities must be granted access to the plaintext via the same ciphertext. An access policy that restricts who is allowed to decrypt a ciphertext would greatly increase the expressiveness of ciphertexts while combining different authorizations into one file key. Such schemes that allow computations on the ciphertext without revealing the underlying plaintext are called *functional encryption* schemes. [6]

With functional encryption, it is possible to reveal partial information about the plaintext, which is not possible in classical encryption. However, a different approach is to formalize conditions over ciphertexts which only reveal the plaintext if the condition is fulfilled. This sub-class of functional encryption is called *predicate encryption*. [6] A closer look at the topic of predicate encryption will be taken since it allows the combination of multiple authorizations into one ciphertext policy, thus reducing the number of file keys to a constant amount.

4.2.1 Identity-Based Encryption

The idea of using a universal identifier instead of public keys to identify individuals goes back until 1984. Shamir proposed the first identity-based encryption scheme [41] which uses a central server to create for each newly registered user a unique universal identifier (e.g. the email address) and the corresponding secret key. Shamir's scheme was the first scheme that did not use classical encryption as known from RSA. Instead, it uses predicate encryption to only allow the individual access to the plaintext which identifier equals the one in the constructed ciphertext. [6]

Without any further third-party interaction, the public key can directly be calculated from the user identifier and thus rendering the use of a certificate to bind the public key to an identity explicitly, as known from the *public key infrastructure* (GK), nugatory. [36] On using the email address as the public key, the sender can be sure that only the identity owning this email address can decipher the email.

Since this scheme could neither be shown to be practically applicable nor to proven secure, it was not until 2001 when *Boneh and Franklin* [5] proposed a new approach to identity-based encryption. The use of the *Weil pairing* revolutionized the field of identity-based encryption.[2]

Bilinear Mappings

Bilinear mappings are a tool for *cryptographic pairings* and define the relationship between two cyclic groups of the same order into a third one. A cyclic group G is defined by a generator g of that form that each $g^n \in G$ with $n \in Z$ completely describes each element in the group G.

A bilinear map function e, also called *pairing* is now defined as the mapping between two groups of the same order G_1 and G_2 into G_t:

$$e : G_1 \times G_2 \rightarrow G_t$$
$$\text{such that for all } g_1 \in G_1, g_2 \in G_2, a, b \in Z$$
$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$$

If G_1 and G_2 describe the same group the mapping is called symmetric. In fact the *decisional bilinear Diffie-Hellman problem* becomes easily computable using bilinear mappings [3].

This construct of pairings is used by many schemes to either distribute a secret or computing the secret without revealing it.

Identity-Based Encryption From The Weil Pairing

To give a short introduction and on how pairings can be used to secure plaintext, the scheme *BasicIdent* of *Boneth and Franklin* 2001 "Identity Based Encryption From the Weil Pairing" [5] will be in the following cited and adapted to match the notation in this work.

As already mentioned is the idea of IBE is to derivate the public key directly from the identifier of a user, say for example his email address. This would eliminate the need for a PKI to validate that the given public key indeed belongs to the recipient's email address.

[2] Curious reader can read up on the Weil pairing here [31] [15] or here https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627

The type of pairing used in BasicIdent is a symmetric pairing.

$$e : G_1 \times G_1 \to G_t$$
$$\text{such that for all } g_1 \in G_1, g_2 \in G_1, a, b \in Z$$
$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$$

On system initialization the central server, for example the email server, chooses the a generator g, a hash function to map an arbitray string into G_1, to map an element in G_T to a fix-length binary stream and the previous defined function e:

$$g \in G_1,$$
$$H_1 : \{0,1\}^* \to G_1,$$
$$H_2 : G_T \to \{0,1\}^n$$
$$e : G_1 \times G_1 \to G_T$$

To register a new user the email server first chooses a random element $s \in Z$ and calculates for the user with the email address ID his secret key $SK = H_1(ID)^s$. In general, $H_1(ID)^s$ is an elliptic curve public key which in this case remains secret on the user side and the private key s which will be the master secret of the email server.

The message M is encrypted by applying *one-time pad* with a computed secret. This secret is computed by taking the recipient ID and for each message a new random element $r \in Z$ as input. The resulting ciphertext is defined as ciphertext $CT = \{C_1, C_2\}$.

$$C_1 = M \oplus H_2(e(H_1(ID), g^s)^r)$$
$$C_2 = g^r$$

The receiving user uses his email address ID and his secret key SK to decrypt the ciphertext CT using the following steps:

$$\begin{aligned}
M &= C_1 \oplus H_2(e(SK, C_2)) \\
&= C_1 \oplus H_2(e(H_1(ID)^s, g^r)) && \text{with } SK = H_1(ID)^s \text{ and } C_2 = g^r \\
&= C_1 \oplus H_2(e(H_1(ID), g)^{sr})) && \text{using } e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} \\
&= C_1 \oplus H_2(e(H_1(ID), g^s)^r) && \text{using } e(g_1, g_2)^{ab} = e(g_1^1, g_2^b)^a = e(g_1^a, g_2^1)^b \\
&= M \oplus H_2(e(H_1(ID), g^s)^r) \oplus H_2(e(H_1(ID), g^s)^r) && \text{with } C_1 = M \oplus H_2(e(H_1(ID), g^s)^r)
\end{aligned}$$

While certainly having the advantage of computing the public key of the recipient directly from his identifier, the central server must be a trusted entity since it computes the master secret s. If it would be malicious it could simply compute any secret keys in the system.

4.2.2 Attribute-Based encryption

Attribute-Based Encryption (ABE) is a predicate encryption scheme, that combines different predicates to formulate an access policy over attributes to secure the ciphertext. [6] In contrast to IBE, which assigned each user a unique identifier, the work of *Sahai and Waters* 2005 [36] describes the approach of assigning each user a descriptive set of attributes. Since many actors share the same attributes plus the group encryption is independent of individuals, it is possible to reduce the number of encrypted file keys to a constant amount.

Take for example Alice who wants to share a file internally in her "Coffee Company". Since each employee received the attribute "Employee of Coffee Company" she can use this attribute to secure her ciphertext.[3] In this participial use-case, only 1 encryption is done, 1 message is distributed using multicast to transmit the same encrypted file to all clients, and 1 file key is created. Please note that his approach scales with a constant amount of encryptions, file key and messages regardless of the number of employees.

In general, an access policy consists of attribute values that are combined using AND and OR-gates to construct a Boolean formula. [4] Some schemes (e.g. [53]) raise expressiveness by further allowing non-monotonic access structures. The use of this access formulas helps to eliminate authorization checks since clients can either decrypt the file and are therefore allowed to access the plaintext or they are not able to decrypt and so do not satisfy the given policy.

Of cause, this advantage does come at some cost. While classical encryption schemes provide complete end-to-end encryption, ABE needs a *attribute authority* (AA), which manages attribute private and public keys, generates user-specific attribute secret keys and serves as a public-key directory. [4] This authority has global decryption power in the administered domain. [9] For that reason, the attributes need to be split up into different domains, each managed by an own AA. In the use-case of cloud storage systems for business, it makes sense to set up an AA for each company.

For the practical applicability ABE, some kind of user management needs to be employed which always indicate a means of revoking users and their attributes. In general two steps need to be made to revoke an attribute from a user. All users, except the revoked user, need to receive a new secret key for this attribute and in addition, each ciphertext encrypted under this revoked attribute needs to be updated to ensure forward secrecy and decryptability for the new attribute secret keys. [26] The revocation process shows a computational overhead while staying constant in space consumption. Other schemes embed a revocation list into the ciphertext, which results in linear space consumption for each additionally revoked user. [30]

The computational overhead and/or space consumption of ABE scales with the number of attributes contained in a ciphertext. [10] Assuming that a group can be described with fewer attributes than there are members in this group, better scalability of ABE compared to the current scheme in CloudRAID can be expected.

ABE comes in many different flavors. The most important for this work are Single-Authority Attribute-Based Encryption (normal ABE) and *Multi-Authority Attribute-Based Encryption* (MA-ABE). The following role definitions are extracted and adopted from *Li et. al.* 2017 [26]. For the single authority use-case three entities can be defined:

[3] This approach is called "Ciphertext-Policy Attribute-Based Encryption" and further described in Section 7.1

1. **Attribute Authority (AA);** An attribute authority issues user their unique identifier (UID), attributes and their respective attribute secret keys. On revocation, the AA will need to update the users' secret keys as well as the ciphertext encrypted with the revoked attribute key. Since an AA always generates the private attribute key to derivate a new user-specific secret key it has global decryption power. That's why in the single-authority use case the AA is assumed to be a trusted entity.

2. **Cloud Storage Provider (CSP);** The cloud storage provider are assumed to be untrusted but they still follow the protocol. That's why they only receive encrypted data. Their only purpose is to store the ciphertext and make them permanently available. Since the authorization to each ciphertext is directly embedded into the ciphertext access policy, the download of the encrypted file does not need any authorization checks.

3. **Users/Data Owners;** Users exist in two groups: Revoked and non-revoked. Non-revoked users try to collude with each other to get a higher level of decryption power. They download the files of the CSP and try to decrypt them. Only if their attribute set matches the policy of the ciphertext they will be able to decrypt the file. Revoked users, on the other hand, try to still decipher ciphertext. In some cases, they try to collude with non-revoked users to intercept the update key to restore their decryption rights. [52] Users are in general untrusted.

   Users (some schemes differentiate between decrypting users and encrypting data owners) want to encrypt content with a specific access policy. To do so they use the publicly available public attribute keys pinned on the bulletin board of the AA. They do not have to know anything about the receiving user or user groups in the system. After encryption, they upload the encrypted content to the CSP.

As already mentioned suffers ABE from additional overhead on revocation. Each ciphertext and secret keys must be renewed and redistributed. To update the ciphertext *Yang et. al.* 2013 [54] introduced a semi-trusted proxy server that helps authorities to update ciphertexts that are affected by the revoked attribute. This technique became known as *proxy re-encryption*. He further proposed to also use the same cloud server to help the user on decryption which further reduced the needed computational resources on the edge device (*proxy decryption*. [54] Multi-Authority ABE (MA-ABE) states the following adaptions and additions:

1. **Attribute Authority (AA);** In the system of MA-ABE there are multiple AAs. Each administrating their own domain. AAs are usually assumed to be honest-but curios. They distrust each other and try to decrypt files of other domains. A big goal for MA-ABE is to achieve inter-authority communication so that ciphertexts can be encrypted using different attributes of different domains. An AA still only administers its users, but can also issue external users attributes.

2. **Central Server (CS);** The purpose of the CS is to issue a global identifier (GID) to individual users. This GID is among the whole user universe unique and makes the user identifiable to prevent collusion. [8] Further, it determines the global public parameter [26] which are needed by any entity that wants to participate in the system. Since it bootstraps the system, in some schemes the CS has global decryption power (e.g. [50], [8]) and thus has to be assumed to

be trusted. If this is not the case the CS remains semi-trusted. The CS can also act as a *Certificate Authority* to certify users as valid participants in the system. [26]

3. **Server;** Some schemes additional introduce a central server. [54] Its task is it to help the user with proxy re- and decryption. If an AA broadcasts a revocation of an attribute, the server downloads all related ciphertexts from the CSP to update them with the new attribute. The threat model for the server is honest-but-curious. [54] If central server and the server have the same trust levels, they are usually merged.

Users and CSP remain unchanged in the MA-ABE case.

Collusion Resistance

Collusion resistance is one of the central requirements of each ABE scheme. [35] Formally, it describes that no two entities are able to combine their keys to achieve a higher level of decryption power. [4] To clarify the importance of collusion resistance a very basic attribute-based encryption scheme is constructed. Assume a distributed crypto-system based on RSA. The role of the AA is to bind attributes to RSA public-private key pairs. The attribute "student" gets bound to $K_{PR(s)}$ for the private key and $K_{PU(s)}$ for the public key. Attribute "works at TU Berlin" gets bound to the key $K_{PR(tu)}$ and $K_{PU(tu)}$ respectively. Now, the very first ABE scheme is created. The AA can pin the public keys of each attributes to its public billboard so that every entity in the system can use the public attribute keys for encryption. Each user who is currently a student receives a copy of the private key $K_{PR(s)}$ and each user who is currently working at TU Berlin receives a copy of $K_{PR(tu)}$.

Alice wants to share content with all students that are also working at TU Berlin. With our basic scheme Alice is able to use *layered encryption*[4] to create an *AND*-policy for her ciphertext. She encrypts the plain text p with both public attribute keys $c = K_{PU(s)}(K_{PU(tu)}(p))$ and publishes the ciphertext c to a public CSP so that everyone can download it. Students that are working at TU Berlin owning both private keys can decipher the ciphertext by applying both private keys in reverse order $K_{PR(tu)}(K_{PR(s)}(c)$.

This ABE scheme is not collusion resistant. On paper collusion resistance it is defined as the impossibility of any two attribute holder to combine their attributes to achieve a higher level of decryption power. [4] Let us assume that Bob is a student and Eve is working at TU-Berlin. Both users received their respective private keys. Now they can simply exchange their attribute keys so that they are both able to decipher the ciphertext even if they separately do not own both attributes.

Collusion resistance is ensured by issuing each user a secret key that is blinded by a random value. [4] This random value is different for each user and will vanish on decryption if and only if the attribute secrets satisfy the given access policy and all used keys are originated from the same user. However, if two users collude they will mix their blinded values resulting in a plaintext that is still blinded by some unknown value. [4] In later schemes the unique user identifier (UID) is used as the blinding value for secret attribute keys. [26] If used with the right attribute secret keys the user is able to use pairings to substitute his blinding value and recover the underlying secret.

[4] Layered encryption: encrypting a plain text with multiple keys, forcing the decryptor to own all relevant keys

As showed by *Wu et. al.* 2017 [52], it is important to also ensure collusion resistance on updating a user secret attribute key on attribute revocation. Revoked and non-revoked users would be able to exchange update keys to restore the decryption power of the revoked user, which is clearly not intended. In that way, Wu showed that DAC-MACS [54] was not collusion resistance on revocation. [52]

4.2.3 Comparing Secure Group Communication To Attribute-Based Encryption

Comparing SGC to ABE is non trivial. This is due to a different encryption technique used by ABE namely paring. Pairing scales more or less with the overhead of RSA rather then ECC or block ciphers as stated by *Galbraith et. al.* [15]. But he also mentions that to achieve the same bit security, computations in pairing need to happen in a much bigger bit-field and thus rendering ABE less performant in comparison to RSA. However, in specific scenarios ABE uses fewer keys to set up the group communication. In the following, these scenarios will be described and analyzed to find out where these schemes differ and what use-cases ABE schemes address.

In a RSA sharing scheme, each user has his own public key which needs to be transmitted to a third party to establish a secret connection. Creating a group under this scheme will implicitly force the data owner to retrieve all n public keys of all n group members. The central server needs to provide a GK together with the public keys of each registered user to prove their identity. Thus follows the constraint that the central server needs to be available all the time to provide public keys for newly registered users. Each member of the group receives an encrypted copy of the group key. The number of keys in SGC scales at least linearly with the number of members.

ABE breaks the constraint that the central server has to be available at all time. This is done by using attribute keys on encryption rather than the users' public keys. This reduces the number of keys that need to be maintained to the size of the attribute set describing the group. If a new user is registered in the system no new keys need to be downloaded from the view of the data owner. The registered user retrieves his attribute set and eventually can decrypt the shared files if his attributes satisfy the access policy of the group. Further notable is that the number of keys that are maintained in the group remains constant in 1 since only one access policy is needed to secure the symmetric key under which all files in the group are encrypted.

Given this observation, it can be stated that ABE is adventitious in scenarios where users address an unknown group of individuals. This can be some departments (e.g. attribute: police department of New York), colloquiums (e.g. attribute: chairman of TU-Berlin), or general user groups (attribute issued to all employees working in the security research team).

To better clarify the scalability advantage of ABE, GKMP is compared to the scheme of "Revokable Ciphertext-Policy Attribute-Based Encryption" (R-CP-ABE)[5] [30]. Since R-CP-ABE only supports user revocation for a specific ciphertext, we also take the multi-authority attribute-based encryption scheme "Two-Factor Data Access Control With Efficient Revocation for Multi-Authority Cloud Storage Systems" (TFDAC-MACS) [26] proposed by *Li et. al.* 2017 into the comparison. It supports attribute revocation which allows us two broader represent the overhead of ABE in general.

[5] Section 2 of [30]. The formal definition of this scheme is given in Section 4.2 "Augmented R-CP-ABE".

	CloudRAID [40]	**GKMP [18]**	**R-CP-ABE [30]**	**TFDAC-MACS [26]**
initial share				
file keys	$O(nf)$	$O(n)$	$O(1)$	$O(1)$
messages (unicast)	$O(nf)$	$O(n)$	$O(n)$	$O(n)$
messages (multicast)	$O(nf)$	$O(n)$	$O(1)$	$O(1)$
encryptions	$O(nf)$	$O(f+n)$	$O(f+1)$	$O(f+1)$
member join				
file keys	$O(f)$	$O(1)$	$O(0)$	$O(0)$
messages (unicast)	$O(f)$	$O(1)$	$O(1)$	$O(1)$
messages (multicast)	$O(f)$	$O(1)$	$O(1)$	$O(1)$
encryptions	$O(f)$	$O(1)$	$O(0)$	$O(0)$
member leave				
file keys	$O(0)$	$O(n)$	$O(1)$	$O(F(a^{-1}))$
messages (unicast)	$O(0)$	$O(n)$	$O(n)$	$O(N(a^{-1}))$
messages (multicast)	$O(0)$	$O(n)$	$O(1)$	$O(N(a^{-1}))$
encryptions	$O(0)$	$O(f+n)$	$O(f+1)$	$O(F(a^{-1}))$
addition of file key				
file keys	$O(n)$	$O(0)$	$O(0)$	$O(0)$
messages (unicast)	$O(n)$	$O(0)$	$O(0)$	$O(0)$
messages (multicast)	$O(n)$	$O(0)$	$O(0)$	$O(0)$
encryptions	$O(n)$	$O(1)$	$O(1)$	$O(1)$

TABLE 4.2: Comparison of CloudRAID, GKMP, R-CP-ABE and TFDAC-MACS scheme. n donating the number of members, $N(a^{-1})$ the number of users in the system owning the revoked attribute, f the number of file keys in the group and $F(a^{-1})$ the number of all file keys secured under the revoked attribute. The overhead of creating private/public key pairs or attribute secret keys are ignored. It is assumed that the needed attribute public keys are distributed in advance. On R-CP-ABE the revokation based on a revokation list embedded in the ciphertext is used while in TFDAC-MACS the attribute-level revocation is shown.

We can extract from table 4.2, that R-CP-ABE indeed scales, regarding the number of file keys, better than GKMP and CloudRAID on initialization, member join, and member leave. The main advantage is that no user-specific file keys must be created for each user respectively. Under the assumption that all user already received their respective attribute secret key, the file keys do not have to be re-encrypted for the new user on member join. On member leave, R-CP-ABE only has to create a new file key with the revoked member embedded into the revocation list of the ciphertext. Therefore, the revoked member does not have access to the group anymore. Additionally, all files need to be re-encrypted under the newly created symmetric key to grant new members to access previously shared content.

In contrast to R-CP-ABE, TFDAC-MACS suffers from additional overhead on member leave. This is due to the attribute-level revocation. Each file key that was secured under a policy that utilized the revoked attribute needs to be updated. Further, each user that owned an attribute secret key of the revoked attribute needs to be issued a new secret attribute key.

However, attribute-based encryption tackles the re-encryption problem by focusing on attributes and groups rather than individuals. ABE reduces the number

of keys needed by reusing and combining existing keys. In contrast, stands secure group communication schemes which need to create a new key per each group. Here ABE exploits the fact that groups generally can be described by a unique attribute set. Implicitly it follows that if another group is described by the exact same set of attributes the same keys are used. So the total number of groups is limited to all possible combinations of attributes.

It must be noted, that we focus in table 4.2 purely on the number of file keys. Performance of cryptographic operations and storage overhead of file keys are currently ignored. However, as stated by *Chu et. al.* 2014 [10], ABE is expected to scale with the number of attributes embedded in the access policy. In this work, we will assume a general distribution of $a \leq n$ per group with n users which can be uniquely identified by a attributes. Under this assumption, we can derivate that SGC schemes scale with $O(n)$ (as shown in table 4.2) while ABE scales with $O(a)$ and thus performs better.

In conclusion, is ABE more suitable to make the re-encryption process of CloudRAID more scalable since it can further reduce the number of file keys on initializing a new share and on adding a new member. In addition to secure group communication, ABE does also provides an authorization service. Users are bound to roles and attributes which are tightly interleaved with the encryption scheme. CloudRAIDs target audience is the business domain which by nature have some kind of attribute authority in the form of role management and authorization mechanism embedded. Hence, ABE can enfold its true potential and outperform SGC schemes not only in efficiency but also in additional security features.

4.3 Basic Attribute-Based Encryption

In the following sections, a deeper inside into the different ABE schemes will be conducted. Here, the characteristics of each subtopic of ABE will be extracted, a representative candidate selected and finally, a practical performance comparison will be done to evaluate the scalability. The outcome will give a good indicator in which direction the further evaluation should head.

To select a respective candidate only the requirements ***C1***, ***C3***, ***C5***, ***C8 - C10*** and optional requirements ***O1*** and ***O2*** are taken into account. Some requirements are left out since not all requirements are directly applicable to all ABE schemes. A suitable subset of requirements is mandatory to conduct a reasonable comparison of the different schemes. The outcome of this section will give a good intuition in which direction this analysis will continue. The general requirements ***B1*** and ***B2*** will be evaluated by the practical performance and scalability analysis.

4.3.1 Ciphertext-Based Policy Attribute-Based Encryption (CP-ABE)

Ciphertext-Based Policy Attribute-Based Encryption (CP-ABE) [4] assigns ciphertexts with a policy and user keys with attributes. The big advantage is that the need for authorization checks is obsolete. Each ciphertext could simply be accessed by every user and only those who have the right attribute set present are able to decipher the ciphertext.

To further also revoke access from users, the first proposal of a revocation scheme for CP-ABE was in 2010 [27]. This scheme used a similar approach like LKH to make the revocation process efficient. Further, *Lui et. al.* 2014 proposed also a time-based re-encryption scheme so that entities don't have to be online when the re-encryption is periodically executed. [28] A time-based approach implies indirect revocation

which is not desirable in our use-case as it would allow the revoked user to still access unauthorized files until the period expired.

With R-CP-ABE in 2016 *Lui et. al* [30] proposed a revocable ABE scheme that supports traitor tracing and large attribute universe. To prevent collusion R-CP-ABE assigns each user an index in a user matrix. [30] However, that matrix must be determined from beginning and thus R-CP-ABE suffers from a fixed user universe. As already mentioned R-CP-ABE also uses a revocation list which is attached to each ciphertext and that contains the revoked user indices. This revocation list will grow in size each time a users access to the ciphertext is revoked.

R-CP-ABE uses a *Linear Secret Sharing Scheme* (LSSS) Matrix to represent the access policy of a ciphertext. LSSS produces a matrix where each row correlates to a Boolean variable. An LSSS matrix is just a different and more resource efficient way to display Boolean access trees. [29] An input configuration $\vec{v}$ satisfy the LSSS matrix A if and only if $A\vec{v} = \begin{pmatrix}1 & 0 & \dots & 0\end{pmatrix}^T$. [29]. This behavior can be embedded into the ciphertext to only reveal the secret if A is satisfied. A disadvantage when using LSSS is that the underlying matrix will grow in complexity when defining more and more complex access policies.

For the implementation of the CP-ABE scheme, the paper "Practical attribute-based encryption: traitor tracing, revocation and large universe" [30] (R-CP-ABE) was chosen. Further a reference implementation in Java "Traceable and Revocable Attribute-based Encryption in Java"[6] is given. The goal was to translate this implementation into the charm framework (which is based on Python) to make it comparable with the other schemes. The python implementation will be from now on called LW14.

Since R-CP-ABE uses an LSSS matrix to express its access policies which were not given in the needed representation by the charm framework, it was implemented using the work of *Liu et. al.*, 2010 [29]. For the implementation of LW14, the *Lewko-Waters Algorithm* was implemented which translate an access tree to an LSSS matrix. It must be noted that this algorithm is not optimal regarding space efficiency.

4.3.2 Key-Based Policy Attribute-Based Encryption (KP-ABE)

In contrast to CP-ABE stands *Key-Policy Attribute-Based Encryption* (KP-ABE) [16]. It is a ABE technique that associates ciphertexts with attributes that fulfill a policy embedded in the private key assigned to a user.

The initial paper *Goyal et. al.* 2006 [16] did leak some basic requirements such as a fix-length ciphertext and attribute revocation which were introduced by later schemes (e.g. [21] [43]). Further, the authors stated that users, once issued a policy, are able to delegate a subset of their access tree to other users. Users, themselves could act as a new attribute authority to issue other uses a subset of their decryption power. While this is a nice feature for some use-cases, in a cloud system the administrator would like to restrict this delegation so that no company external users are able to get company secret keys issued.

Initially, KP-ABE was meant as a broadcasting encryption scheme used by radio providers so that they can address any user that bought a paid membership to distribute premium content. [16] However since the access policy was bound to the users key it is not possible to address a ciphertext to a specific user group without knowing each users access policy. Only the central server, which issued the users' private keys in the first place, would know who could decipher the encrypted

[6] https://github.com/TU-Berlin-SNET/jTR-ABE

text. This, in fact, renders KP-ABE impractical for the application in a cloud sharing scheme. The data owner would simply not know which entities are able to decrypt the ciphertext.

The certainly very specific use-case of KP-ABE probably led to the circumstances that not much paper using this technique exists. So few that it was hard to find any paper satisfying all the previously specified requirements. The work of *Lewko, Sahai and Waters* 2010 [21] implements direct revocation using negation of attributes. This would, in turn, lead to the fact that users private key sizes would grow with each revoked attribute. The scheme description of, the in the comparison used schema, can be found in "Revocation systems with very small private keys" [21] Section 3.1, further references as LSW08.

4.3.3 Non-Monotonic Access Structure For Attribute-Based Encryption

While monotonic access structures can describe *AND* or *OR* gates, non-monotonic access structure can also reference *NOT* gates. This adds another layer of fine-granularity into the system. In non-monotonic structures, first introduced by *Ostrovsky et. al.* [34], a user may be excluded from certain topics.

Take for example Alice who is an intern in the Top Secret Company. In monotonic access structure she would receive both attribute private keys: "Intern" and "working in Top Secret Company". By nature of monotonic access structures, she would be able to decipher all content addressed to all employees at the Super Secret Company. However, some content might be so confidential that interns shall not be able to access them. This exclusion would not be possible. With non-monotonic access structure, an administrator may want to encrypt certain information with the policy "working in Top secret Company AND NOT intern".

While being an imported field for Boolean formula and fine-grand access control domain, non-monotonic access structures did not gain the same attention as CP-ABE did. As a candidate which shall represent this ABE topic the method proposed by Yamada *et. al.* [53], Section 7 is used (further referenced as YAHK14). With the negation of attributes, revocation becomes more or less trivial. Each attribute can be versioned and simply excluded on encryption. While this approach is not scalable over time it is simple enough to be implemented in some schemes.

4.3.4 Hirachical Attribute-Based Encryption (HABE

Hierarchical Attribute-Based Encryption (HABE) is based on the idea of key and decryption power delegation. If a private key exists that have a certain access power, it is possible for the key holder to delegate a subset of his access power to a new instance. By nature follows a hierarchical structure where each user could administrate an own subdomain. Many use-cases for cloud computing as well as cloud storage system emerged [50].

A practical approach to implement HABE was given by *Wang, Liu and Wu* 2010 [50] Section 3 (in comparison called WLWG11) which will also be implemented by this work and represent MA-ABE in the following comparison. While this implementation serves with a revocation scheme it depends on an access policy in *disjunctive normal form* (DNF). Any Boolean formula can be transferred into DNF but sometimes this might enforce negation. Since Wang *et. al.* [51] did not provide any negation mechanism, this ABE scheme is limited in expressiveness.

The DNF-access policy is translated into an access matrix where each row related to one AND-condition. This matrix represents for each row an equation system, that

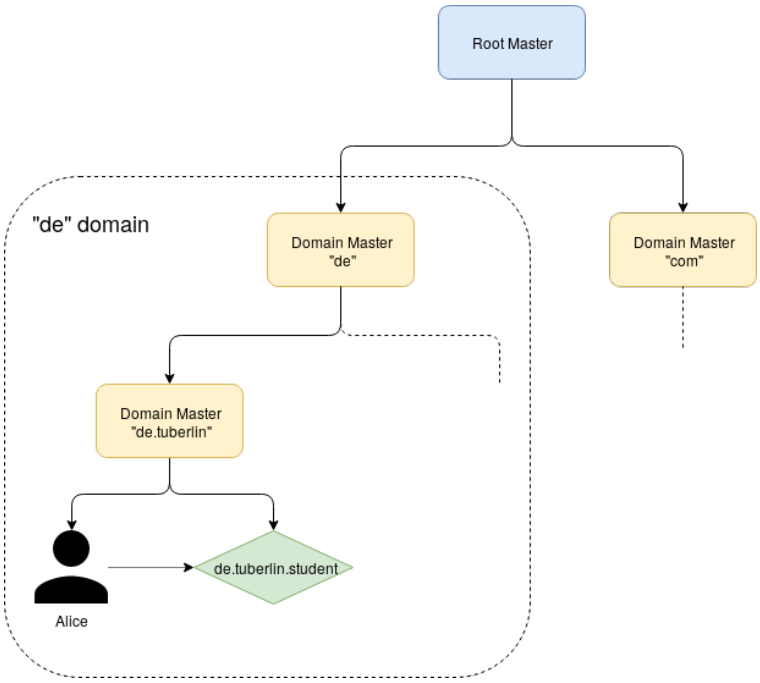

FIGURE 4.2: Structure of hierarchical attribute-based encryption systems. Graphic adapted from [51], Figure 4.

when substituted with the right attribute secret key reveals the encrypted secret. [50] Note that this representation is not an LSSS matrix. It is just a data structure representing the access DNF condition.

The structure can be imaged like the domain name system. At the top level, there is the root master administrating the whole system. As displayed in Figure 4.2, which is based on the approach by Wang *et. al.* [51], the root master summarized the global decryption power of the system and can set up new domain masters (attribute authorities). They, on the other hand, can delegate a subset of their decryption power to a subdomain master. Each domain master can administer users and attributes.

In addition to the theoretical background of HABE by Wang, this work exploits the hierarchical structure to assign each user, attribute and authority a unique identifier that is composed in a top-down directory structure (similar to reverse domain names). So, for example, the attribute "student" of the university TU-Berlin might be displayed as "de.tu-berlin.student". In the same way, also string attributes can be represented e.g. "com.example.mail:alice@exmaple.com". This syntax makes it really easy to extract all domain masters of a given identifier. In the previous example, the domain master identifier would be "com" while the attribute authority id "com.example", the attribute id "com.example.mail" and the attribute value "alice@example.com".

4.3.5 Comparison Of Attribute-Based Encryption Schemes

To implement and compare the selected representatives of the different ABE subtopics the charm framework[7] is chosen. The charm framework is a collection of cartographic prototype algorithm and benchmarks based on Python. It is specifically designed to rapidly implement any cryptographic algorithm. Developers are offered a wide range of mathematical settings which also include bilinear and non-bilinear

[7] http://charm-crypto.io/

elliptic curve groups.[8] Those, in particular, are often the building block for various ABE schemes.

Based on similar implementations of single authority ABE schemes such as LSW08 (KP-ABE [21]) and YAHK14 (non-monotonic CP-ABE [53]), the schemes LW14 (R-CP-ABE [30]) and WLWG11 (HABE [50]) were implemented by this work[9].

Since WLWG11 only supports DNF-Access Policies [50] and YAHK14 was implemented to only support attribute present as numerical strings, each policy was feed into the system as DNF-policy build over numerical strings. Further, the initialization overhead of WLWG11 to set up attribute authorities was ignored to have a baseline for a meaningful comparison.

	Scheme	**Security Scheme**	**Expression of access policy**
LSW08 [21]	KP-ABE	Biliniear maps	LSSS
YAHK14 [53]	Non-Monotone CP-ABE	Binilnear maps	LSSS
LW14 [30]	CP-ABE	Binilnear maps	LSSS
WLWG11 [50]	Hirachical CP-ABE	Binilnear maps	DNF

TABLE 4.3: Scheme descriptions.

	LSW08 [21]	**YAHK14** [53]	**LW14** [30]	**WLWG11** [50]
C1	Yes	Yes	Yes	Yes
C2	No	No	No	No
C3	No	No	No	No
C4	No	No	No	No
C5	Yes	Yes	Yes	Yes
C6	-	-	-	Yes
C7	-	-	-	No
C8	Yes	Yes	Yes	Yes
C9	Yes	Yes	Yes	No
C10	Yes	Yes	Yes	Yes
O1	No	No	Yes	No
O2	Yes+	Yes+	Yes	Yes-

TABLE 4.4: Comparison of schemes against the system requirements explained in Section 3.4.

In table 4.3 the main differences of this schemes are listed. Table 4.4 shows the requirements and whether they are satisfied by each of the schemes 3.4.

Please note that the scheme each address different use-cases and were designed with different assumptions and techniques in mind. So the scalability comparison might not be as verbose as a comparison would be if there was a scheme that exists in all four subtopics of ABE. In this case, the schemes that satisfied most of the requirements for a cloud storage system was selected.

However, from table 4.4 it is clear that no scheme fully fulfill all requirements. They reveal a good indicator in which subtopic the research should be extended. WLWG11 and LW14 would be the best choices, satisfying both 6 of 10 of the requirements.

[8] Extracted from the description of charm `http://charm-crypto.io/`

[9] `https://github.com/Anroc/charm`

C6 and ***C7*** could not be satisfied by LSW08, YAHK14 or LW14 since they are not designed with multi-authority support in mind. Especially ***C7*** cannot be satisfied since each central, single attribute authority has the master key to create a fake identity that owns all attributes, thus can decrypt all ciphertexts. [21] [53] [30] Let us have a look at ***O2***. LSW08 and YAHK14 both support fine grant access control by using a LSSS access structure. Also, both implement non-monotonic access structures which give them the extra plus. [53] [21] LW14, on the other hand, does not support negation of attributes but also uses an LSSS. WLWG11, however, does only support access structures in disjunctive normal form (DNF) which makes this scheme somehow restricted in expressiveness.

Regarding the requirement ***C9***, WLWG11 does not support data confidentiality in this context since the root master can always decrypt any file in the system.

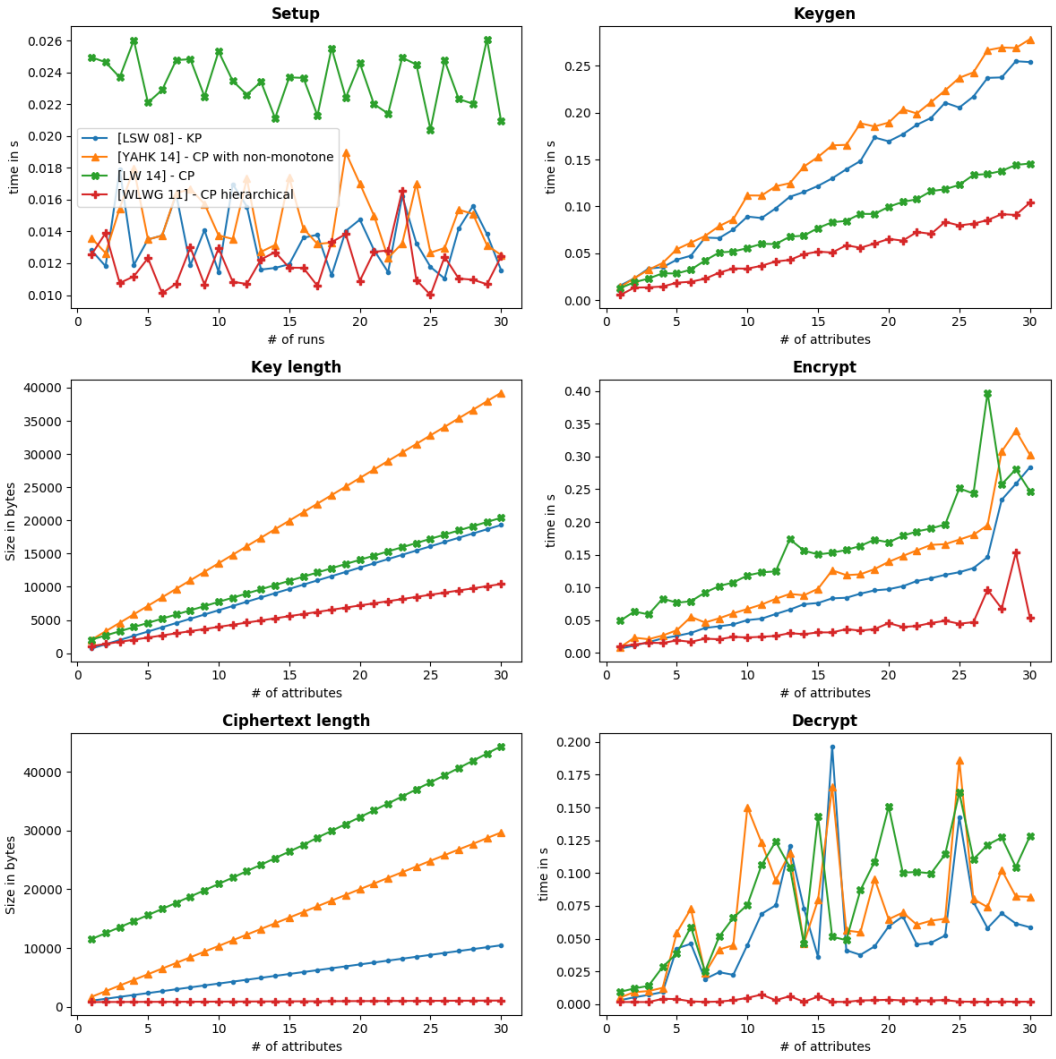

FIGURE 4.3: Performance and scalability comparison between the chosen schemes with increasing number of attributes. The policy for each new attribute a_k with $a_k \in A$ was defined as $\bigwedge_{a \in A}^{a} a$. Key generation relates to the creation of users key pairs given an access policy or attribute set.

The benchmark show in Figure 4.3 is executed with a constant size message

$m \in G_T$ on a Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz with 16 GB RAM running Ubuntu 16.04 and Python 3.

The setup procedure was executed over 30 runs. Here we can see how long the time varies to find suitable curve parameter to initialize the pairing. The other runs are executed with an increasing number of attributes. The policy, that is used to create the ciphertexts and the user keys, is defined as $\bigwedge_{a \in A}^{a} a$ where A defines the set of attributes.

Notable about the comparison in Figure 4.3 is that LSW08 (KP-ABE) was expected to have the biggest overhead in key generation compared to the CP-ABE schemes. We assumed that because KP-ABE has to bind the user secret keys additionally to an access policy while the ciphertext is bound to attributes. However, the benchmark showed different results. It seems more like the time complexity and runtime of the different algorithms surly depends on the design of the scheme. Here we can say that each of the schemes shows a linear overhead in time complexity. While a constant runtime is only achievable by YAHK14 (HABE) regarding the ciphertext length and decryption time. For each other cases, YAHK14 still shows the best overall performance.

The constant ciphertext length is a very important feature in the context of this thesis. Reducing the number of file keys always also means to reduce the storage consumption of file keys. YAHK14 shows a great example of a constant-sized storage overhead regardless of how many attributes are embedded in the access policy while generating only one file key.

Furthermore, due to the great performance and the coverage of most of the requirements of the hierarchical attribute-based encryption technique, the field of multi-authority attribute-based encryption will be evaluated in more depth. Regardless of the great scalability of the HABE, it comes with a non-negotiable disadvantage. The root master and the top level domain master have both global decryption power. For each user regardless of the company affiliation, the administrator of the system will always be able to decipher ciphertexts of each user. This would break end-to-end encryption. So clearly we would favor solutions that support different attribute authorities so that each company can administer their own domain separately from each other, but it is also desirable that the root while being able to bootstrap new attribute authorities, is not able to decipher any ciphertext.

4.4 Multi-Authority Attribute-Based Encryption

Multi-Authority Attribute-Based Encryption (MA-ABE) enjoyed, because of its real-world relation relations, a lot of attention in the research area. In addition to the normal requirements like collusion resistance (Section 4.2.2) and revocation mechanism, MA-ABE, in contrast to HABE (Section 4.3.4), also deals with the question of how to deescalate the global decryption power of the central server (CS) which bootstraps the AAs. Additionally, each AA is only responsible for its own domain but the challenge relies on the fact that users still want to construct policies that embed attributes of different domains into their ciphertexts.

In the following sections, the different subtopics of *Multi-Authority Attribute-Based Encryption* (MA-ABE) will be described, analyzed given all the requirements and finally evaluated based on their performance and scalability.

Hierarchical Attribute-Based Encryption (HABE) uses decryption power deletion to allow multiple authorities to issue attributes. Since HABE was already introduced in Section 4.3.4 we will focus on the other subtopics of MA-ABE. However,

the same algorithm WLWG11 will be used in the practical comparison of MA-ABE schemes.

4.4.1 Introduction Into Multi-Authority Attribute-Based Encryption

2007 was the first year when Chase [8] introduced a working MA-ABE scheme. In her paper, she describes the process of how to derive a multi-authority attribute-based encryption scheme from the single authority scheme. Her proposal focused on interpolation and the fact that no under defined linear equation system could definitely be solved as the main security assumption[10].

To ensure system-wide collusion resistance, each user usually gets a unique user identifier (UID) assigned. [8] This UID is issued by the CS which is the only entity having an overview of the whole system. Each user's secret keys are blinded with their identifier so that when used on decryption the plaintext is still blinded with the UID. Using pairings (bilinear maps, Section 4.2.1) this identifier can be substituted and the plaintext revealed.

Chases scheme had two major disadvantages which were not addressed in her initial scheme. First, the CS had global decryption power. That was due to the bootstrapping of the different AAs and users. The CS had to give the right seeds and global parameter to each AA to make sure that each decryption of a ciphertext results in a blinded plaintext that can only be decrypted using attribute keys of the same user. If this would not be the case users could easily collude.

Chase improved her scheme in 2009 [9] to deescalate the global decryption power of the CS. Now all AAs will do an n-party key exchange to agree on n secret seeds from which the personal initial seed can be calculated. This results in the fact that no AA know the whole master secret but only its required secret seeds. Agreeing on a well-known value the CS was no longer required and as long $N - 2$ AAs are not colluding with each other the master secret remains secure.

The other disadvantage, which is also still present in the updated scheme, is that no new AA could be added after system initialization since it would trigger a new system bootstrap and distribution of the master secret.

The lack of adding new attributes and the missing revocation scheme makes Chase's scheme impractical for further evaluations but gives a good introduction to the challenges of the MA-ABE schemes.

4.4.2 Decentralized Attribute-Based Encryption

That a system without a central server that bootstraps the authorities is possible, showed *decentralized attribute-based encryption* (CP-ABE), first formalized by *Lewko and Waters* 2011 [22], which is structured around the idea of having a peer network of authorities and users. No entity has a global view of the system and no entity is in charge of bootstrapping AAs. This ecosystem is self-sustaining so that each entity can become an AA if needed and start issuing new attributes to users.

However, some form of centralization must still be present. The global parameters need to be known to each new entity and every user still needs to get a unique global identifier assigned to prevent collusion. [22] Also, attributes need to be synchronized since the set of attribute identifier need to be non-intersecting across different domains. While not being a limitation it is rather a challenge to enable global communication and achieve global synchronization across the network.

[10] LSSS matrix is based on the same assumption.

Revocation in a distributed setting remains an open issue. Since no central server has a global view over the users, no authority can be in charge of revoking them. An authority could revoke its issued attributes but only in an indirect fashion since decentralized systems always have to take into account that nodes do not have to be online all the time. [11] If an authority would go offline as soon at it received the revocation request the revocation procedure would never trigger.

Cui and Deng showed in [11] that a DABE system with indirect revocation could exist. Each key and ciphertext get a liveness assigned which make them valid for a certain time period. After this time period expired all keys need to be reissued by the respective AA. Indirect revocation happens by simply excluding the user from reissuing certain attribute keys. This system is implemented by this work for the comparison in Section 4.4.4 and will from now on be called CD16.

CD16 again uses a LSSS matrix, where we use the self-implemented LSSS construction as described in Section 7.1. The scheme of CD16 is simple and the reader is highly encouraged to have a look at the scheme details in [11] Section 4.

As described in CD16 [11], the time period is a string that is concatenated to the UID of the user which will produce a hash in the elliptic curve field G_1. While this hash helps on preventing collusion, it also enables the possibility to make the key only valid at the defined date. Bigger or smaller time periods can be chosen for example by only including the year as the timestamp (to extend the time period over the whole year) or the date by including the full qualified timestamp down to the current day.

However, the question remains if such a system would be practically applicable in the real world since each ciphertext needs to be reuploaded and each attribute keys need to be redistributed in every time period. Another issue is how to deal with international time zones. An expired key that was issued in certain time-zone can still be valid in a different time-zone, even if it would have been expired in the original time-zone. Until now no direct revocation DABE was developed.

4.4.3 Efficient Data Access Control For Multi-Authority Cloud Storage

The most explored scheme family in MA-ABE is the *Efficient Data Access Control for Multi-Authority Cloud Storage Systems* (DAC-MACS)). First introduced by Yang *et. al.* 2013 [54] it describes an efficient, revocable MA-ABE scheme based on CP-ABE.

To summarize DAC-MACS we will in the following cite [54]. On system initialization, the central server (CS) setups up the security parameter and the master secret key pairs. Users register directly to the CS and retrieve a unique UID as well as a global public and secret key. This key is already bound to the given user identifier. AAs register to the CS and retrieve in exchange their unique authority identifier (AID). Taking the security parameter as input, an AA autonomously generates its secret key and a public key as well as a version (public) attribute key and attribute secret key for each administered attribute. To register a user given an attribute set, the AA computes his respective secret key. For encryption, the data owner collects the needed attribute version keys from the respective AAs and encrypts his message M using an access matrix.

To decrypt a ciphertext DAC-MACS uses proxy de-/reencryption method, as demonstrated in the literature by [54], [52], [26] and [51]. It is a technique where the server (not the CS[11]) helps the user on decryption to reduce the computationally intensive work. It is motivated by the fact that mobile devices often don't have much

[11] If the CS would help the user it could decrypt the ciphertext since it issued the user his/her secret key.

computing resources. The user provides his secret attribute keys and public key to the server. The server downloads the respective ciphertext and does a preprocessing based on the secret keys on the ciphertext to produce the so-called decryption token. The server has no knowledge about the plaintext because to recover the plaintext from the decryption token the user secret key is needed.

Finally, the client can decrypt the decryption token using its secret key and revoker the plaintext. DAC-MACS also provides a mechanism for efficient revocation which is not described in this work. For any more scheme details, the reader is referred to the paper DAC-MACS [54] Section 3.

In contrast to Chases scheme, DAC-MACS eliminates the need for the global decryption power of the CS by issuing k file keys: One per AA. [12] It does not require any coordination between authorities which enables to add new AAs at runtime without recreating the user keys. This scheme also includes features for efficient revocation while it claims to maintain forward and backward secrecy. In sort DAC-MACS satisfy all non-optional requirements. For the comparison, we will use the charm implementation of DAC-MACS [54] which from now on will be called YJ14.

DAC-MACS is not collusion resistance on attribute revocation under the active attack model. The scheme NEDAC-MACS (New-Extended DAC-MACS) shows and solves this vulnerability [52]. Recent studies introduce a more efficient, scalable and secure approaches such as MAACS [25] and TFDAC-MACS (Two-Factor DAC-MACS [26].

TFDAC-MACS [26] is the latest DAC-MACS scheme providing non-global decryption power, secure revocation channels, and both backward and forward secrecy. In addition, TFDAC-MACS introduces the two-factor authentication. The data owners issue and revoke *authentication keys* to and from other users. This adds an additional layer of security. In total TFDAC-MACS achieves a better performance than the other DAC-MACS schemes providing constant decryption and encryption overhead. [26] In comparison to DAC-MACS, TFDAC-MACS removes the proxy decryption part as well as the public and secret user keys. It still maintains the proxy re-encryption for ciphertext updates on revoking attribute or two-factor keys.

While TFDAC-MACS manages to reduce the number of created file keys down to one, it suffers from less expressiveness since it only supports n-of-n threshold policies. The term m-of-n threshold gates defines that at least m conditions of n must be fulfilled until the gate returns true ($m \leq n$). The term n-of-n threshold gate simply defines an AND-gate while a 1-of-n threshold gate defines an OR-gate.

To compare the TFDAC-MACS to the other schemes we will implement TFDAC-MACS from scratch. We will refer to this implementation with LTXWC16. The complete scheme is defined in Chapter 5.1 of this work.

4.4.4 Comparison Of Multi-Authority Attribute-Based Encryption Schemes

In general each MA-ABE scheme is compomsed of six steps.

1. **Setup:** The global setup phase where the public and security parameters are determined.

2. **Authority Setup:** AAs can register itself to the central server (if any) and compute their secret keys. Usually, the attribute secret keys are generated.

3. **Register User:** User register themselves to a central server to retrieve their UID.

[12] If the ciphertext does not require any attributes of a specific authority it does not have to create a ciphertext for this domain.

	LTXWC16 [26] (TFDAC-MACS)	**YJ14 [54] (DAC-MACS)**	**LW14 [51] (HABE)**	**CD16 [11] (DABE)**
Revocation	Direct	Direct	Direct	Indirect
Security scheme	Bilinear maps	Bilinear maps	Bilinear maps	Bilinear maps
Expression of access policy	n-of-n threshold	LSSS	DNF	LSSS

TABLE 4.5: Scheme descriptions.

	LTXWC16 [26] (TFDAC-MACS)	**YJ14 [54] (DAC-MACS)**	**LW14 [54] (HABE)**	**CD16 [11] (DABE)**
C1	Yes	No	Yes	Yes
C2	Yes	Yes	Yes	Yes
C3	Yes	Yes	No	Yes
C4	Yes	Yes	No	Yes
C5	Yes	Yes	Yes	Yes
C6	Yes	Yes	Yes	Yes
C7	Yes	Yes	No	Yes
C8	Yes	Yes	Yes	Yes-
C9	Yes	Yes	Yes	No
C10	Yes	Yes	Yes	Yes
O1	No	No	No	No
O2	No	Yes	Yes	Yes

TABLE 4.6: Requirements comparison of the implemented schemes.

4. **Key generation:** Attributes are assigned to users and the respective secret keys are generated.

5. **Encrypt:** The ciphertext is encrypted by the data owner under an access policy.

6. **Decrypt:** The user decrypts the ciphertext and recovers the secret message.

To compare the subtopics of MA-ABE it is necessary to extend the charm framework with the implementations of CD16 (CP-ABE), WLWG11 (HABE, as already done in the previous section), and LTXWC16 (TFDAC-MACS). YJ14 (DAC-MACS), on the other hand, is already present the framework.

These steps are shown and compared in 4.4. It can be argued that YJ14 (DAC-MACS) shows the worst performs, then CD16 CP-ABE and LTXWC16 (TFDAC-MACS) and WLWG11 (HABE) show roughly equal overhead. This can be shown by comparing the most common use-cases: key generation, encryption, and decryption. While WLWG11 and LTXWC16 both perform roughly equally, WLWG11 has linear overhead on encryption where LTXWC16 shows a constant one. On the other hand, on user key generation LTXWC16 does not perform as good as WLWG11. The other schemes show worse performance on encryption and decryption then LTXWC16 and WLWG11.

However, for practical usage especially the encryption and decryption performance is important. Here only LTXWC16 has a constant overhead while all other schemes show a linear overhead. Taking also the complete requirements of table

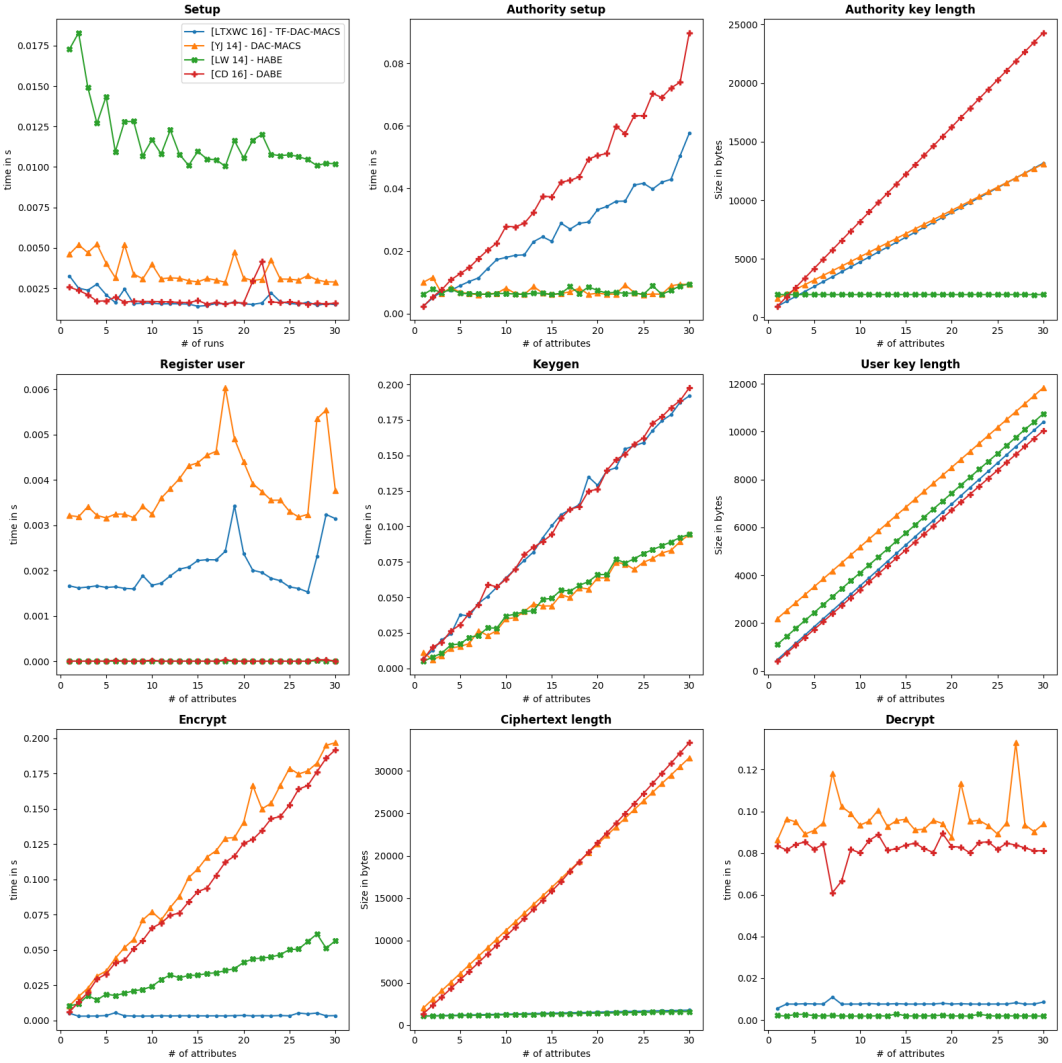

FIGURE 4.4: Performance and scalability comparison between the chosen schemes with increasing number of attributes. The policy for each new attribute a_k with $a_k \in A$ was defined as $\bigwedge_{a \in A}^{a} a$. Key generation relates to the creation of users key pairs given an access policy or attribute set.

4.6 into account, it can be shown that only two schemes satisfy all of the requirements: LTXWC16 and CD16. CD16 profits also from the fact that it has a more fine grant access control than LTXWC16 (TFDAC-MACS only allows AND-gates, as can be seen in table 4.5). However, the indirect revocation scheme of CD16 is the disadvantage that leaves us with LTXWC16 as our final candidate. As mentioned in Section 4.4.2 the implemented scheme uses indirect time-based revocation, which forces data owner to periodically re-encrypt and re-upload their content. As this is not shown in the comparisons it would render CD16 practically not applicable.

The comparison of LTXWC16 with other schemes of the DAC-MACS family was left out in this work since it was already done in the TFDAC-MACS paper [26]. Here some could easily see that TFDAC-MACS is currently the most performant scheme in the DAC-MACS family.

4.5 Summary

In the past sections, the path from the classical re-encryption scheme to multi-authority attribute-based encryption was evaluated. Different topics in secure group communication were reviewed and the direction of why attribute-based encryption might be a more suitable scheme was motivated. Finally, it was shown that multi-authority attribute-based encryption is more appropriate for the secure cloud storage systems domain regarding the number of file keys. In that research area, TFDAC-MACS revealed as the most fitting candidate which finally satisfies all the requirements and provides acceptable scalability. However, it still leaks fine-granularity for access policies.

In comparison to the classical re-encryption scheme, better performance on encryption and on adding new members into a group is expected when scaling up the number of clients. It further has to be noticed that the performance improvements of ABE in comparison to CloudRAID depends surely on the selected use-case. If a group will be formed that can be described completely by a low number of attributes ABE will be more efficient. However, if groups are created that can not easily be described by a low number of attributes, ABE will perform with the same computational overhead class as CloudRAID.

Whether this assumption holds true in practice and if the day-to-day use-cases of the user are sufficient to provide the promised performance boost will be evaluated in the following sections.

Chapter 5

PAD-TFDAC-MACS

As evaluated in the last sections this work will proceed to implement TFDAC-MACS [26] with small adaptions for a practical secure cloud storage system. PAD-TFDAC-MACS (Practically Applied Distributed Two-Factor authentication Data Access Controll for Multi-Authority Cloud Storage systems)[1] describes the practical application and adaption of TFDAC-MACS for a distributed system for the use-case of secure cloud storage systems. To do that TFDAC-MACS will be briefly sketched as it is described in the work of *Li et. al.* 2017, [26].

5.1 TFDAC-MACS

"Two-factor data access control with efficient revocation for multi-authority cloud storage systems" (TFDAC-MACS) claims to provide content sized ciphertexts and low computational cost.[2] A big advantage of this scheme is the two-factor authorization. With this additional security layer, data-owners are able to end-to-end encrypt data so that even the AA can not decrypt the ciphertext.

It must be noted that TFDAC-MACS uses the notation of "data owner" and "decrypting user". Both entities aren't necessarily disjunct. Users can adapt any or even both roles to decrypt and encrypt content. Further, the certificate and certificate creation mentioned in TFDAC-MACS and which is bound to every user in the system is neglected in this description since it is not imported to understand the scheme in general.

In the following Sections the scheme of TFDAC-MACS [26], mainly Chapter 4 "Our Contribution", will be cited. For more in-depth details we refer to the paper TFDAC-MACS.

Let the universe of attributes be defined as $U = u_1, \ldots, u_n$ and corresponding to that $U_{aid} = \{u_{aid_1}, \ldots, u_{aid_{n_{aid}}}\}$ donate the attribute domain for AA_{aid} where n_{aid} is the total number of attributes managed by AA_{aid}.[3] Further, set $S_{aid,i} = \{v_{aid_i,1}, \ldots, v_{aid_i,n_i}\}$ as the multi-value set for u_{aid_i}.

TFDAC-MACS system design is split into six phases: Setup, key generation, data encryption, data decryption, attribute revocation and user revocation. This phases will be cryptographically defined.

5.1.1 System Initialization

The system initialization phase creates all the needed keys and parameters for the subsequent phases.

[1] https://github.com/Anroc/PAD-TFDAC-MACS

[2] In Section 5.2.1 it is stressed that TFDAC-MACS really provides constant-sized ciphertexts.

[3] Note that TFDAC-MACS is not limited in the number of attributes as described in Section 5.3.2.

- *CSSetup*: The system is initialized by choosing the implicit input security parameter K. The CS chooses two multiplicative groups G and G_T with the same prime order p. Let g be the generator of G and $e : G \times G \rightarrow G_T$ be a bilinear map. The CS also chooses a hash function $G : \{0,1\}^* \rightarrow G$. The global public parameter (GPP) are defined as $GPP = (p, g, G, G_T, e, H)$.

- *UserSetup*: For each user, the CS assigns a global unique identifier uid and a pair of public/secret keys (pk_{uid}, sk_{uid}). [4]

- *AASetup*: Each AA takes the GPP and its managed attribute domain U_{aid} as the input of this algorithm to generate the public key and its corresponding master secret key.

  The AA_{aid} first generates the AA master key by randomly choosing $x_{aid} \in Z_p^*$ and computing $e(g,g)^{x_{aid}}$. Secondly, the attribute value key pairs are generated by choosing $y_{aid_i,j} \in Z_p^*$ and computing $g^{y_{aid_i,j}}$ for each attribute value $v_{aid_i,j} \in S_{aid,i}$. The output is defined as the public key $PK_{aid} = (APK_{aid} = e(g,g)^{x_{aid}}, \{UPK_{aid_i,j} = g^{y_{aid_i,j}} | u_{aid_i} \in U_{aid} \wedge v_{aid_i,j} \in S_{aid,i}\})$ and its corresponding master secret key $SK_{aid} = (ASK_{aid} = x_{aid}, \{USK_{aid_i,j} = y_{aid_i,j} | u_{aid_i} \in U_{aid} \wedge v_{aid_i,j} \in S_{aid,i}\})$.

  This step will be made more dynamic by Section 5.3.2.

- *DOSetup*: The data owner DO_{oid} is identified by the identifier oid. He takes the GPP as the input of this algorithm to generate a pair of public/secret key used for authorization.

  The data owner DO_{oid} randomly chooses $\alpha \in Z_p^*$ as the authorization secret key OSK_{oid} and computes its corresponding public key $OPK_{oid} = g^{\alpha}$.

5.1.2 Secret Key And Authorization Key Generation

The AA and the data owner can issue secret keys to the users independently. To prevent collusion the hash $H(uid)$ is included as the basis of the secret component thus rendering it impossible to combine different keys with each other to achieve a higher level of decryption power.

- *AAKeyGen*: The AA_{aid} assigns an attribute list $L_{uid,aid}$ to the decrypting user DC_{uid} according to his/her role or identity. The AA_{aid} runs this algorithm with taking the global public parameter GPP and the master secret key SK_{aid} as input to create a set of attribute secret keys $SK_{uid,aid}$ for the user DC_{uid}.

  Suppose $v_{aid_i,j} \in L_{uid,aid}$ the AA_{aid} computes $SK_{v_{aid_i,j}} = g^{x_{aid_i}} H(uid)^{y_{aid_i,j}}$. Thus, the corresponding attribute secret keys is $SK_{uid,aid} = \{SK_{v_{aid_i,j}} | v_{aid_i,j} \in L_{uid,aid}\}$.

- *Auth*: When a user DC_{uid} submits an authorization request to the data owner DO_{oid}, the data owner runs this algorithm with taking the authorization secret key OSK_{oid} as input. It outputs an authorization key $SK_{uid,oid}$ for the user DC_{uid}.

  The data owner DO_{oid} computes the authorization key $SK_{uid,oid} = H(uid)^{\alpha}$ and delivers it to the user DC_{uid}.

[4] It must be noticed that the keys (pk_{uid}, sk_{uid}) are not used in the scheme at all. See Section 5.2.2.

5.1.3 Data Encryption

The data owner DO_{oid} runs this algorithm to encrypt a data m under the access policy W with taking the public keys $\left(\bigcup_{aid \in I_W^A} APK_{aid}, \bigcup_{v_{aid_i,j} \in W} UPK_{aid_i,j}\right)$ and the authorization secret key OSK_{oid} as input, where I_W^A donates as the index set of involved AAs. Let n_w^{aid} donate the number of involved attributes that are managed by AA_{aid} in W.

The data owner DO_{oid} randomly chooses $s \in Z_p^*$ and sets $CT_W = (W, C_1, C_2, C_3)$, where

$$C_1 = m * \left(\prod_{aid \in I_W^A} e(g,g)^{x_{aid} n_w^{aid}} \right)^s \tag{5.1}$$

$$C_2 = g^s \tag{5.2}$$

$$C_3 = \left(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}} \right)^{s+\alpha} \tag{5.3}$$

As the final step, the data owner selects a unique label ID_W for this data and uploads the (oid, ID_W, CT_W) onto the CSP. [5]

5.1.4 Data Decryption

Upon receiving the ciphertext CT_W from the CSP, user DC_{uid} first checks whether $\bigcup L_{uid,aid} \models W$. If this is true, the user DC_{uid} runs this algorithm with taking his/her global unique identifier uid, the attribute secret key $\bigcup SK_{uid,aid}$ and the authorization key $SK_{uid,oid}$ to decrypt the data m.

Based on the attribute values in W, the user DC_{uid} aggregates the attribute secret keys to generate $SK_W = \prod_{v_{aid_i,j} \in W} SK_{v_{aid_i,j}}$. Then, the user computes $UPK_W = \prod_{v_{aid_i,j} \in W} UPK_{aid_i,j}$. The data m is recovered as:

$$m = \frac{C_1 \cdot e(H(uid), C_3)}{e(C_2, SK_W) e(SK_{uid,oid}, UPK_W)} \tag{5.4}$$

5.1.5 Attribute-Level Revocation

Lets assume that an attribute of user DC_{uid} is revoked from AA_{aid} and the revoked attribute value is $v_{aid_i,j}$. The involved AA_{aid} first queries the CSP for ciphertext components $\bigcup(oid, ID_W, C_2)$ where $v_{aid_i,j} \in W$. Then, the AA_{aid} generates a ciphertext update component $CUK_{v_{aid_i,j}}^{ID_W}$ for each (oid, ID_W, C_2). Moreover, the AA_{aid} needs to compute an attribute update key $KUK_{v_{aid_i,j}}^{uid'}$ for each non-revoked user $DC_{uid'}$. This phase contains the following three algorithms:

- *KeyUpdate*: The AA_{aid} runs this algorithm with taking a non-revoked user list NRU, the secret key ASK_{aid}, the master secret key $USK_{aid_i,j}$, the public key OPK_{oid} and the ciphertext components $\bigcup(oid, ID_W, C_2)$ as input. It outputs a new $UPK_{aid_i,j}$ for $v_{aid_i,j}$ attribute update keys $\bigcup KUK_{v_{aid_i,j}}^{uid'}$ and ciphertext update components $\bigcup CUK_{v_{aid_i,j}}^{ID_W}$.

[5] In our proposed scheme the encrypted file will be uploaded to the CSP. The ciphertext, as described by TFDAC-MACS, will be uploaded to the central server.

The AA_{aid} randomly chooses $y'_{aid_i,j} \in Z_p^*$ as the new master secret key for the attribute value $v_{aid_i,j}$. Then, the AA_{aid} computes $UPK_{aid_i,j} = g^{y'_{aid_i,j}}$.

For each non-revoked user $DC_{uid'} \in NRU$, the AA_{aid} generates an attribute update key $KUK^{uid'}_{v_{aid_i,j}} = H(uid')^{y'_{aid_i,j} - y_{aid_i,j}}$.

For each (oid, ID_W, C_2) the AA_{aid} computes the ciphertext update component $CUK^{ID_W}_{v_{aid_i,j}} = (g^s \cdot g^\alpha)^{y'_{aid_i,j} - y_{aid_i,j}}$. The AA_{aid} sends $KUK^{uid'}_{v_{aid_i,j}}$ and $\bigcup(oid, ID_W, CUK^{ID_W}_{v_{aid_i,j}})$ to each non-revoked user $DC_{uid'}$ and the CSP respectively.

- *SKUpdate*: Upon receiving the attribute update key $KUK^{uid'}_{v_{aid_i,j}}$, the user $DC_{uid'}$ runs this algorithm to update his/her attribute secret key as

$$\begin{aligned} SK'_{v_{aid_i,j}} &= SK_{v_{aid_i,j}} \cdot KUK^{uid'}_{v_{aid_i,j}} \\ &= g^{x_{aid}} \cdot H(uid)^{y_{aid_i,j}} \cdot H(uid)^{y'_{aid_i,j} - y_{aid_i,j}} \\ &= g^{x_{aid}} \cdot H(uid)^{y'_{aid_i,j}} \end{aligned} \tag{5.5}$$

- *CTAUpdate*: The CSP runs this algorithm to update the involved ciphertexts when receiving $\bigcup(oid, ID_W, CUK^{ID_W}_{v_{aid_i,j}})$. According to the oid and ID_W the CSP first retrieves the involved ciphertexts components (C_1, C_2, C_3). Then the CSP randomly chooses $r \in Z_p^*$ and computes:

$$\begin{aligned} C'_1 &= C_1 \cdot \left(\prod_{aid \in I^A_W} e(g,g)^{x_{aid} n^{aid}_W} \right)^r \\ &= m \cdot \left(\prod_{aid \in I^A_W} e(g,g)^{x_{aid} n^{aid}_W} \right)^{s+r} \end{aligned} \tag{5.6}$$

$$C'_2 = C_2 \cdot g^r = g^{s+r} \tag{5.7}$$

$$\begin{aligned} C'_3 &= C_3 \cdot CUK^{ID_W}_{v_{aid_i,j}} \cdot \left(\prod_{v_{aid_t,j} \in W, v_{aid_t,j} \neq v_{aid_i,j}} g^{y_{aid_t,j}} \right)^r \cdot g^{y'_{aid_i,j} r} \\ &= \left(\prod_{v_{aid_t,j} \in W, v_{aid_t,j} \neq v_{aid_i,j}} g^{y_{aid_t,j}} \right)^{(s+\alpha+r)} \cdot \left(g^{y'_{aid_i,j}} \right)^{(s+\alpha+r)} \end{aligned} \tag{5.8}$$

5.1.6 User-Level Revocation

Suppose that the data owner DO_{oid} wants to revoke the access privilege of user DC_{uid}. The data owner DO_{oid} first chooses a new authorization secret key and computes its corresponding public key. Then, the data owner DO_{oid} generates a new authorization update key for each non-revoked users and a set of ciphertext update components for ciphertext update, This phase contains the following three algorithms:

- *DAAuthUpdate*: The data owner DO_{oid} runs this algorithm by taking the old authorization secret key OSK_{oid}, the non-revoked user list NRU' and the public parameters from each AA_{aid} as inputs.

  The data owner DO_{oid} randomly chooses $\beta \in Z_p^*$ as the new authorization secret OSK'_{oid} and computes the corresponding public key $OPK'_{oid} = g^{\beta}$.

  For each non-revoked user $DC_{uid'} \in NRU'$ the data owner DO_{oid} generates an authorization update key $AUK_{uid',aid} = H(uid')^{\beta-\alpha}$.

  For each attribute value $v_{aid_i,j}$ the data owner computes the ciphertext update component $UAU_{aid_i,j} = (UPK_{aid_i,j})^{\beta-\alpha} = (g^{y_{aid_i,j}})^{\beta-\alpha}$.

  Finally, the data owner sends $AUK_{uid',aid}$ and $(oid, \bigcup UAU_{aid_i,j})$ to each non-revoked user and the CSP respectively.

- *AuthUpdate*: Upon receiving the new authorization update key, each non-revoked user $DC_{uid'}$ runs this algorithm to update his/her authorization key as $SK'_{uid',oid} = SK_{uid',oid} \cdot AUK_{uid',aid} = H(uid')^{\beta}$.

- *CTOUpdate*: The CSP runs this algorithm to update all the outsourced ciphertexts of data owner DO_{oid} when receiving $(oid, \bigcup UAU_{aid_i,j})$.

  For each retrieved ciphertext (oid, ID_W, CT_W) the CSP fist randomly chooses $r' \in Z_p^*$ and computes:

$$C'_1 = C_1 \cdot \left(\prod_{aid \in I_W^A} e(g,g)^{x_{aid} n_W^{aid}} \right)^{r'}$$

$$= m \cdot \left(\prod_{aid \in I_W^A} e(g,g)^{x_{aid} n_W^{aid}} \right)^{s+r'} \tag{5.9}$$

$$C'_2 = C_2 \cdot g^{r'} = g^{s+r'} \tag{5.10}$$

$$C'_3 = C_3 \cdot \prod_{v_{aid_i,j} \in W} UAU_{aid_i,j} \cdot \left(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}} \right)^{r'}$$

$$= \left(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}} \right)^{(s+\beta+r')} \tag{5.11}$$

5.2 Critics On TFDAC-MACS

While analyzing and designing PAD-TFDAC-MACS some critics on TFDAC-MACS were noticed.

5.2.1 Stressing The Constant Sized Ciphertext Claim

TFDAC-MACS claims to provide constant sized ciphertexts. This is true under the assumption that C_1, C_2 and C_3 are the only components of the ciphertext. However, from a practical view the access policy W is a non-neglectable component and should count into the ciphertext size computation as well. Without knowing W the user

would be unable to decrypt the ciphertext since he would not know which secret attribute keys to use. Taking W into account TFDAC-MACS shows a linear increasing ciphertext size correlating with the rising number of attributes in the policy.

Some could argue that W can be provided in a constant-sized form if the attribute universe was finite. In that way, a binary array could be provided with the size of the attribute universe. When an attribute is used in the policy the bit at the index of this attribute is set. While this approach is constant in size, it is not space efficient. Considering an infinite attribute universe, the binary array would be infinite in size as well.

So the conclusion is made that, if the access policy W would be displayed in a space efficient manner, the size of the ciphertext increases with each additional attribute embedded in the access policy. Moreover, does TFDAC-MACS define the ciphertext as $CT_W = (W, C_1, C_2, C_3)$, which clearly includes the attribute policy W as an element of the ciphertext.

5.2.2 Private And Secret-Keys Of The User Setup Phase Are Not Used

It is stressed that the public/secret keys (pk_{uid}, sk_{uid}) that are created by the CS on user setup (Section 5.1.1) are never applied in the described scheme. Also, no reason is seen why the CS needs to create those key-pairs in the first place.

5.3 Adaptions And Improvements

Although TFDAC-MACS satisfies all our requirements mentioned in Section 3.4, it is still not fully suitable for the system design in order to provide flexible file sharing. Therefore, we propose several adaptations and improvements to TFDAC-MAC scheme to make it more suitable with our system without compromising its security.

5.3.1 Removing The Fix Two-Factor Constrain

To make TFDAC-MACS more practically applicable the fixed two-factor constrain was removed from encryption, decryption and the ciphertext update phase. The secret two-factor authorization key α is used by the data owner to restrict the access to the content for certain users. This makes sense since only if the two-factor authorization is in place end-to-end encryption can be guaranteed. Users are in charge of generating these two-factor keys, while the attribute keys are created by the AA which has the decryption power of all attributes it created.

This leads to the fact that the underlying ABE scheme loses some of its expressiveness. The zero knowledge of the data owner on which an individual is able to decrypt the ciphertext is broken with the two factor part. Here each user, who wants to decrypt the encrypted content, needs to make an *authorization request* to the data owner to receive the corresponding decryption key. To restore the possibility to let an unknown user group decrypt the ciphertext, the two factor part was made optional. To do so the encryption, decryption and the ciphertext update process was adopted. The authorization key update will be ignored since it makes no sense to apply it on a non-existing authorization key.

- **Encryption:** Only the C_3 part of the ciphertext needs to be updated since it is the only one containing the two factor component α.

The original C_3:

$$C_3 = \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}} \Big)^{s+\alpha}$$

is adapted to:

$$\widehat{C}_3 = \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}} \Big)^{s} \tag{5.12}$$

In addition the *oid* can be removed from the ciphertext description since it referrers to the data owner ID which is only needed on two-factor authorization key update.

- **Decryption:** $SK_W = \prod_{v_{aid_i,j} \in W} SK_{v_{aid_i,j}}$ and $UPK_W = \prod_{v_{aid_i,j} \in W} UPK_{v_{aid_i,j}}$ remain defined in the same was as defined in the paper.

  On decryption the user does not need to generate UPK_W and $SK_{uid,oid}$ anymore. The decryption equation is updated to:

$$m = \frac{C_1 \cdot e(H(uid), \widehat{C}_3)}{e(C_2, SK_W)}$$

Note that the original decryption equation results in the above equation when the two factor part is deducted.

$$\begin{aligned}
m &= \frac{C_1 \cdot e(H(uid), C_3)}{e(C_2, SK_W) e(SK_{uid,oid}, UPK_W)} \\
&= \frac{C_1 \cdot e\Big(H(uid), \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}}\Big)^{s+\alpha}\Big)}{e(C_2, SK_W) e(H(uid)^{\alpha}, \prod_{v_{aid_i,j} \in W} UPK_{v_{aid_i,j}})} \\
&= \frac{C_1 \cdot e\Big(H(uid), \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}}\Big)^{s+\alpha}\Big)}{e(C_2, SK_W) e(H(uid)^{\alpha}, \prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}})} \\
&= \frac{C_1 \cdot e\Big(H(uid), \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}}\Big)\Big)^{s+\alpha}}{e(C_2, SK_W) e(H(uid), \prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}})^{\alpha}} \\
&= \frac{C_1 \cdot e\Big(H(uid), \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}}\Big)\Big)^{s}}{e(C_2, SK_W)} \\
&= \frac{C_1 \cdot e\Big(H(uid), \Big(\prod_{v_{aid_i,j} \in W} g^{y_{aid_i,j}}\Big)^{s}\Big)}{e(C_2, SK_W)} \\
&= \frac{C_1 \cdot e(H(uid), \widehat{C}_3)}{e(C_2, SK_W)}
\end{aligned} \tag{5.13}$$

As shown, no security is threatened since we end up at the same equation as we would do if we had the two factor part included.

- **Attribute revocation:** The ciphertext update key is adapted from

$$CUK^{ID_W}_{v_{aid_i,j}} = (g^s \cdot g^{\alpha})^{y'_{aid_i,j} - y_{aid_i,j}}$$

to

$$\widehat{CUK}_{v_{aid_i,j}}^{ID_W} = (g^s)^{y'_{aid_i,j} - y_{aid_i,j}}$$

$\widehat{C}'_3$ now computes as

$$\begin{aligned} \widehat{C}'_3 &= \widehat{C}_3 \cdot \widehat{CUK}_{v_{aid_i,j}}^{ID_W} \cdot \Big(\prod_{v_{aid_t,j} \in W, v_{aid_t,j} \neq v_{aid_i,j}} g^{y_{aid_i,j}} \Big)^r \cdot (g^{y'_{aid_i,j}})^r \\ &= \Big(\prod_{v_{aid_t,j} \in W, v_{aid_t,j} \neq v_{aid_i,j}} g^{y_{aid_i,j}} \Big)^{s+r} \cdot (g^{y'_{aid_i,j}})^{s+r} \end{aligned} \tag{5.14}$$

It can be shown that C'_3 computes to the message m in the same way as shown in equation 5.13.

- **Authorization update:** No changes are required since the ciphertext does not contain authorization components.

5.3.2 Dynamic Secret Key Generation

As shown in the setup phase 5.1.1 of TFDAC-MACS the attribute universe must be predefined. It was noticed that this necessarily does not have to be the case. Since no other setup mechanism depends on these parameters, they can as well be created on each user's key generation.

If a user key generation is requested for an attribute that is not yet known, this attribute will be created by first choosing new random values $y_{aid_i,j} \in Z_p^*$ for all unknown attribute values $v_{aid_i,j} \in S'_{aid,i}$ and then computing for each new value the public attribute value key $\{UPK'_{aid_i,j} = g^{y_{aid_i,j}} | u_{aid_i} \in U_{aid} \wedge v_{aid_i,j} \in S'_{aid,i}\}$. The original attribute universe set of $S_{aid,i}$ finally redefined as $S_{aid,i} = S_{aid,i} \cup S'_{aid,i}$ in the same way the public attribute value keys are redefined to $UPK_{aid_i,j} = UPK_{aid_i,j} \cup UPK'_{aid_i,j}$ and analogous the attribute secret value keys to $USK_{aid_i,j} = USK_{aid_i,j} \cup \{USK_{aid_i,j} = y_{aid_i,j} | u_{aid_i} \in U_{aid} \wedge v_{aid_i,j} \in S'_{aid,i}\}$. This reduces the universe of possible attribute values to thous who are actually needed.

While this approach makes our scheme more dynamic, it also helps to reduce the communication between the CS and AA. If the AA only publishes its attribute information to the CS if this attribute is also used in the system, unnecessary information and state must not be preserved by the CS.

5.3.3 Extension To DNF-Policy

One big advantage of TFDAC-MACS is its constant-sized ciphertext components (C_1, C_2, C_3). It achieves this by restricting the access policy to n-of-n threshold gates. This means that a data owner is only allowed to create *AND*-gate policies to encrypt his content. However, extending this scheme to an m-of-n threshold gate policy is not trivial. To not break any security constraints, it was decided to just upload multiple versions of the same ciphertext encrypted under different policies. Now, a user only needs to decipher one version of the ciphertext to recover the encrypted content.

To mathematically enable this, the random message $m \in G_T$, which will be the AES-key in the later process, needs to be the same across the different ciphertexts.

Creating different ciphertexts over the same secret does not threat security since an attacker still needs to find the blinding factor s first to decipher any message. Since this s is chosen for each ciphertext randomly and independently of the secret m it can be assumed that encrypting the same secret with different policies does not result in any security violations.

Using this approach, access policies in disjunctive normal form (DNF) are enabled for the trade-off of linear ciphertext-size overhead and linear overhead on ciphertext updates (both scaling with the number of disjunctions). Formally spoken, each Boolean formula can be represented which satisfy the following grammar displayed in Listing 5.1.

```
root:               or_expression
or_expression:      (or_expression (OR or_expression)* | and_expression
and_expression:     and_expression (AND and_expression)* | attribute_value
attribute_value:    AID.ATTR_NAME:ATTR_VALUE
```

LISTING 5.1: Phseudo gramma for creating a DNF formula

So for example "(aa.tu-berlin.de.role:student or (aa.tu-berlin.de.role:professor and aa.tuberlin.de.institution:dsi))" can be represented by creating ciphertext policies "(aa.tu-berlin.de.role:student)" and "(aa.tu-berlin.de.role:professor
and aa.tu-berlin.de.
institution:dsi)" respectively. However, without reordering the conditions, we are not able to represent the following example: "(aa.tu-berlin.de.institution:dsi and (aa.tu-berlin.de.role:professor or aa.tu-berlin.de.institution:student))".

It must be noted that every Boolean formula can be translated into DNF representation. [12] However, it comes with the disadvantage that each formula, when converted to DNF, will grow exponentially in length. So a formula with n variables will result in 2^n Boolean terms. However, the proposed DNF-access policy does not support negations and thus can only display monotonic-access structures.

5.3.4 Numerical Attributes

As described by Bethencourt *et. al.* 2007 [4] we could display numeric values in binary. Each number x is composed of $\lceil log_2(x) \rceil$ attributes. Each of this attributes relate to either a 1 or 0 in one position in the binary number representation of x. So for example the number 5 in binary would be displayed as 0101 and its attribute would be: $x : 0 * **$, $x : *1 * *$, $x : * * 0*$ and $x : * * *1$.

If a user now wants to challenge a number x to be greater or equal to 3, he creates a policy: "$(x : 1 * * * or(x : *1 * *or(x : * * 1 * andx : * * *1))$". Analogously, the policy for x smaller than 4: "$(x : 0 * * * and(x : *0 * *or(x : *1 * *andx : * * 0 * andx : * * *0))$".

A disadvantage of this representation is that it is limited in space. To display a 32-bit number we must issue 64 attribute values and maintain 64 attribute value keys. It also has to be kept in mind that this formula needs to be converted to DNF representation, resulting in a very large overhead of storage and computation.

5.4 System Design

In the following the system design of PAD-TFDAC-MACS will, with focus on a distributed and secure system, be sketched. The system life cycle can be described in the phases: Setup, encryption, decryption, attribute revocation and two-factor-key

revocation. In each of these life cycles the interaction of the following five entities will be described:

- **Central Server (CS)** does the initial setup of the global public parameter used in the later process. It publishes this information on a public bulletin board. Further, the CS is the central entry point to trigger and permit AA creations. It audits the *Authority Identifier* (AID) provided by the AA which should be unique in this system.

  Further, the CS functions as a *Certificate Authority*. It issues a certificate for the *user identifier* (UID) and RSA keys of the user. The *user identifier* (UID), which is given by the administering AA, will be checked for uniqueness. This certificate can be revoked so that the user is not able to obtain new secret keys from the AA or other data owners.

  The CS acts as a central key repository for storing signed versions of attribute/ authority public keys as well as signed RSA public keys of the respective AAs.

  Finally, the CS also functions as a central key escrow service to store encrypted file keys (ciphertexts) and encrypted user secret keys. If a ciphertext update is needed the CS acts as a proxy re-encryption service to help to update the respective file keys.

  In our use-case, the CS will be materialized by the CloudRAID server.

- **Attribute Authority (AA)** creates the attribute public and master key as well as the user-specific attribute secret keys. Further, it generates its own authority secret and public key and analogously its RSA key-pair. Each AA service is controlled by a company and represents the company-wide role and user management system.

  Attributes are prefixed with the AID to ensure uniqueness among the attribute universe. The AA has decryption power for each ciphertext that is solely encrypted using attributes of its own domain. Further, it is in charge of revoking its issued attributes. The system is designed to support multiple AAs.

- **Cloud Storage Provider (CSP)** provides storage to save the encrypted files. It does not do any authorization checks to the downloading client.

- **Users** download and decipher ciphertexts. They receive attribute secret keys and their UID from their AA, certificates from the central server and two-factor keys from the data owners.

- **Data owner** are specifications of users who hold the ownership to a file in the system. Therefore they can issue two-factor authorization keys to other users to enable end-to-end encryption. The so-called *Two-Factor Authorization Key* (2FA-Key), adds an additional security layer to the ciphertext. It restricts access to the encrypted plaintext to an even smaller selected user group.

5.4.1 Setup

The setup phase combines the steps described in TFDAC-MACS [26] such as setup, user registration, data owner registration, authority setup, key generation and authorization requests.

Generate Global Public Parameter

The very first step is to create global public parameters (GPP) on the CS. Those parameters are exposed on a public bulletin board and available for each entity in the system. Since they are required in every step it is assumed for each of the following steps that they are downloaded in advance.

Setup Attribute Authority

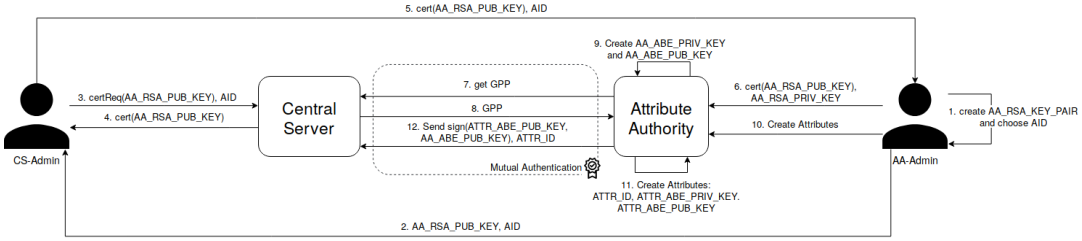

FIGURE 5.1: Attribute Authority setup.

An AA is registered, as shown in Figure 5.1, with the help of the central server administrator (CS-Admin) and the attribute authority administrator (AA-Admin). First, the AA-Admin creates a new RSA key-pair and chooses a new authority ID (AID) (Step 1). The AID should be globally unique among this ecosystem so that authority domains can be clearly differentiated. To enable that AIDs can be for example the domain name of this institution. In Step 2., the RSA public key and the AID are sent to the CS-Admin. Now the CS-Admin creates a new certificate signing request embedding the RSA public key of the new AA (Step 3). The central server checks that the given AID is unique and signs the certificate request (Step 4). The resulting certificate is passed back to the AA (Step 5) where it, together with the RSA private key, can be used for mutual certificate authentication to the CS.

The AA-Admin uploads the received certificate and the private key to the AA service (Step 6) so that it can communicate confidently and authenticated with the CS. For each ABE related key generation, the GPP is needed, which can now be retrieved over a secured channel from the CS (Step 7 and Step 8). Based on the GPP the authority ABE private and public key can be created (Step 9).

Now the AA-Admin specifies the attributes by providing the attribute name and values to the service (Step 10). Internally, the attribute value keys are generated (Step 11). For each attribute value, an ABE key-pair is generated. To ensure uniqueness of the attribute identifier, it is composed using the following syntax: "<AID>.attr.<Attribute_name>:<Attribute_value>". For example, the major computer science would be displayed as "aa.tu-berlin.de.attr.major:computer_science". In the final step, a signed version of the public attribute value keys, a signed version of the authority public key and the attribute identifier is uploaded to the CS (Step 12).

User Registration

Figure 5.2 shows the user registration flow. Users are registered by the AA-Admin. He chooses a user identifier (UID) and attributes for the new user (Step 1). The AA now checks if all attributes and attribute values are already known or if a new attribute value key pair needs to be created (Step 2, formally described in Section 5.3.2). If new attribute values are created, their public components are to be signed

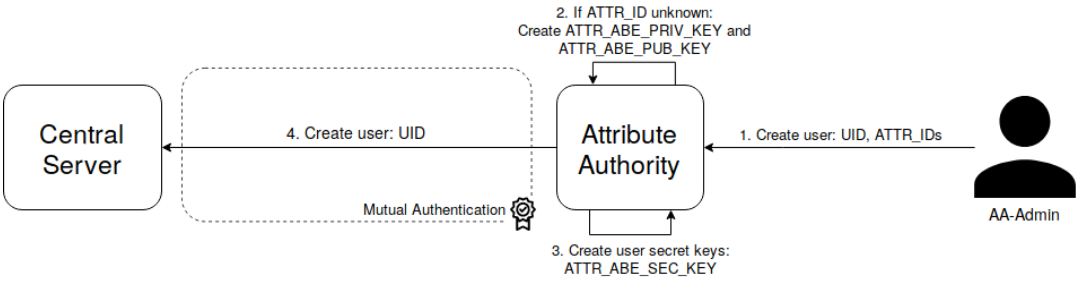

FIGURE 5.2: Register a user.

and published to the CS. Given the set of attribute values that this user should receive and the respective private keys (including the authority master key), the user-specific attribute secret keys are created (Step 3). Those keys contain the hash of the UID embedded and bound into them so that on colluding with other users this hash would mesh and no successful decryption would be possible (see Section 5.1.2). Finally, the central server is notified about this new user (Step 4). It checks that the provided UID is indeed unique among the known universe and saves the user object for future reference.

Device Registration

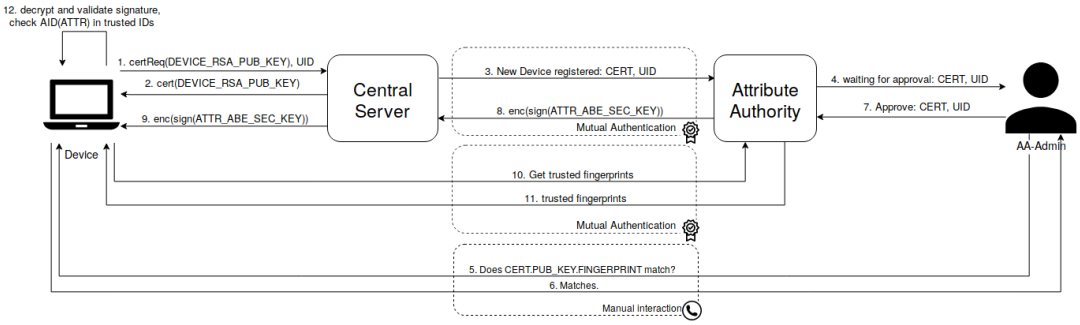

FIGURE 5.3: Register a device for a user.

A device is the end-system that is legally allowed to encrypt and decrypt files in the system. Figure 5.3 shows the rather complex flow for registering a new device. A device is bootstrapped with the certificate of the AA it belongs to saved in its trusted storage. This method is called *certificate-pinning* and provides the possibility for the device to authenticate each response of its AA. [32] After the device is bootstrapped, a new RSA key pair is created on the device. It is later used to securely transfer the secret attribute keys to the client.

The device creates a new certificate signing request and embedding its public key as the subject. This certificate signing request is posted as a pending request under a user object (Step 1). The CS processes the certificate signing request, validates that the user with the desired UID exists and returns an X.509 certificate embedding the devices public key and UID (Step 2). In addition, the CS notifies the AA that a new device for the user UID was initialized by handing the newly created certificate over (Step 3). The AA-Admin gets notified for this new device and retrieves the certificate (Step 4). He then contacts the addressed user via a separate secure channel (e.g. phone or in person) and asks him if he registered a new device by comparing the fingerprints of the public key (Step 5 and Step 6). If they match the AA-Admin

approves the device (Step 7). The AA encrypts the attribute secret keys of the user with the RSA public key embedded in the approved certificate, signs the content, using the AA private key, and publishes this information to the central server (Step 8).

Sanity check between AA-Admin and user is important because the CS, due to the threat of man-in-the-middle-attacks, might have generated an own RSA key pair, self-issued a the valid certificate for the forged keys and secretly added it as a fake device to the user. If the AA would just send the attribute secret keys along without double checking with the user, the CS would retrieve the attribute secret keys for this user.

Next, the device gets notified that it has been approved and receives the encrypted and signed attribute secret keys (Step 9). In a later stage, external AA might issue the user attributes as well. To retrieve and authenticate them, the device needs a set of trusted AA fingerprints. This set is retrieved from the AA directly in step 10 and 11. The device will only trust attribute secret and public keys coming from a trusted AA. If it retrieves attributes of one external AA, it will request the certificates of this AA. The certificate fingerprint will be computed and checked against the stored trusted fingerprints. Only if the certificate fingerprint matches one of the trusted fingerprints, the authenticity of the public and secret attribute keys can be guaranteed.

In the final step, the device decrypts the attribute secret keys using its RSA private key and validating the signature using the pinned-certificate or the retrieved certificate of an external AA (Step 12). Some might have noticed that the device could just retrieve its attribute secret keys directly from the AA. However, form the view of designing a practical system it makes sense to only have one entry point to retrieve attributes from instead of crawling all trusted AAs periodically.[6]

Register User And Device As An Extern Attribute Authority

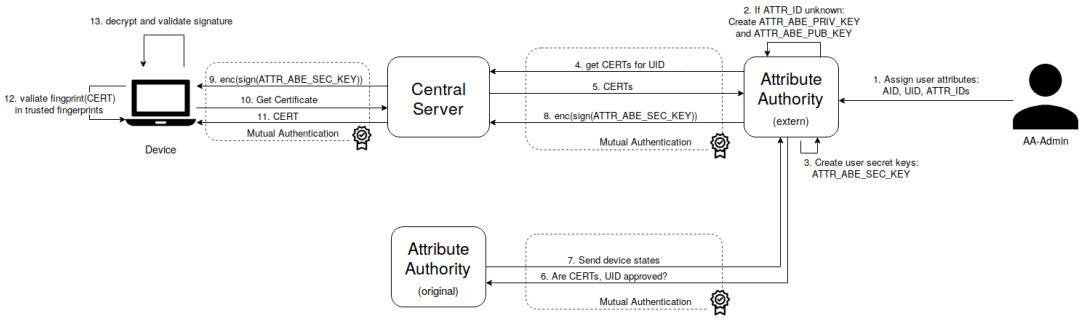

FIGURE 5.4: Register a device for a user as an extern AA. Since the new device is located outside of the externs AA domain, it needs to request the original AA to validate this device.

Since PAD-TFDAC-MACS is developed for a multi-authority setting, the feature of issuing user attributes, that are not located in the same domain as the AA must be enabled. In its core this flow is a merge of the processes shown in Figure 5.2 and in Figure 5.3. To assign an external user attributes, the original AA and the external AA must be in a trust relationship. They exchanged the fingerprints of their certificates

[6]In the future we also want to implement the AA as a client application (Section 7.1). This would increase security and reduce infrastructure setup overhead.

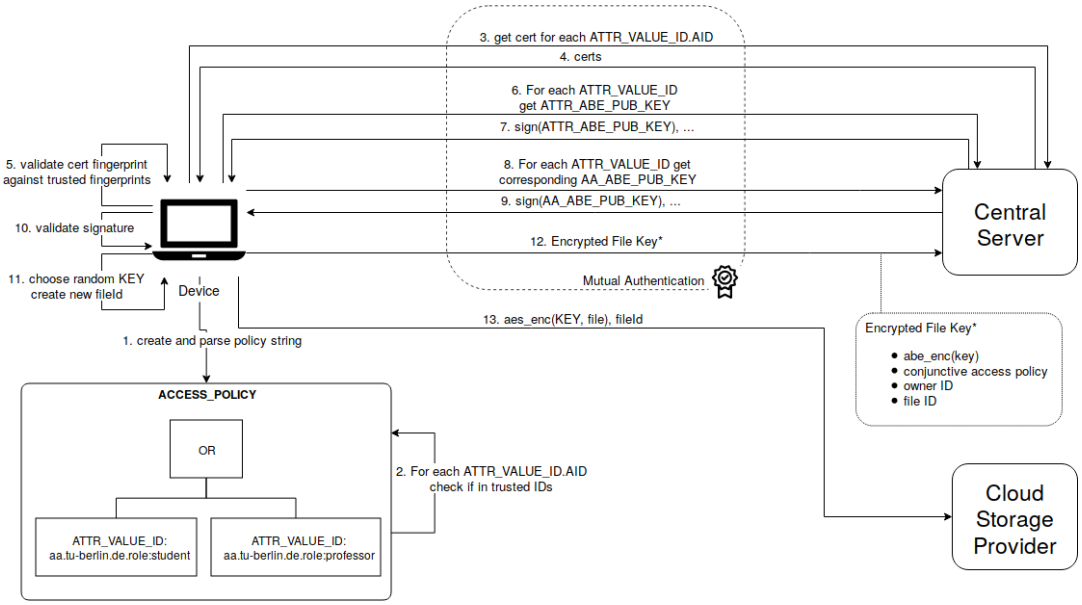

FIGURE 5.5: Encryption phase.

and added them to their trusted set of fingerprints that are downloaded by clients on setup.

The complete flow of adding an external user is shown in Figure 5.4. The Step 1-3 are the same as if a new user is normally registered (see Figure 5.2). The external AA retrieves for each device of the targeted user their respective X.509 certificates (Step 4 and Step 5). Since the external AA must verify these certificates before using them it establishes a second channel to the original AA. It sends all the fingerprints to the original AA and validates that all devices are approved and non-revoked (Step 6 and Step 7). If this check succeeds the external AA will use those, in the certificate embedded, RSA public keys as the base-line to encrypt and signs its generated attribute secret key for each device respectively (Step 8).

Each device will continue, as already known from the previous section, with retrieving the encrypted attribute secret keys (Step 9). Since the signature is known, the device will ask the CS for the certificate of the external AA (Step 10 and Step 11). Due to the fact that the response might be tampered by the CS, the device must consolidate its set of trusted fingerprints to check whether the retrieved certificate matches one of the fingerprints (Step 12). After this check returned success the device decrypts and validate the signature of the retrieved secret attribute keys.

5.4.2 Encryption

The encryption process is displayed in Figure 5.5. In the very first step, a user given policy is parsed to create a so-called *parsing tree* (Step 1). Having this parsing tree grantees that no syntax error are present in the parsed policy. For the semantic validation, additional Steps 2-10 need to be made. For each attribute value identifier (ATTR_VALUE_ID) we first extract the attribute authority ID. This is done by simply exploiting the syntactical structure of an attribute value.

Recapitulate that an attribute value identifier has the following syntax: ATTR_VALUE_ID = <AID>.<ATTR_NAME>:<ATTR_VALUE>. Knowing this, the authority ID can be extracted from the given attribute value. The next steps are done for each attribute value respectively.

In Step 2, it is validated that the given attribute values and their authorities are a subset of the trusted authorities retrieved in the register device process. This is necessary since the CS can be malicious and might provide tampered attribute public keys which will later be used for encryption resulting in insecure encryption. To countermeasure this attack, the device only accepts attribute public keys signed by a trusted authority. The certificate of this authority can be retrieved safely (Step 3, Step 4) since the device can validate the fingerprint of this certificate using the given trusted authority fingerprints provided by his originated authority (Step 5).

Step 6-9 retrieve the public attribute and authority keys needed for encryption under the given policy. For each request, a signed ABE public key is returned. In Step 10 these signatures are validated given the certificates of Step 4 which are validated by the trusted fingerprint checks of Step 5.

As described in Section 5.3.3 an extension to DNF-access policies was introduced. If the access policy contains OR-gates, the policy will be split into sub-policies each referencing a sub-condition of an OR-gate. This process is recursively repeated until each sub-condition only contains AND-gates which can be encrypted by TFDAC-MACS. For each sub-policy, a new ciphertext is created encrypting the same secret K. In that way, when the decrypting client satisfies one AND-access policy he can recover K and thus decrypt the encrypted file.

As can be seen in Figure 5.5 the example access policy is composed of two attribute values combined with an OR-gate. In that case, the client would produce one encrypted file key under the policy "(aa.tu-berlin.de.role:student)" and another for the policy "(aa.tu-berlin.de.role:professor)". As already mentioned, this comes with a performance trade-off in storage and computing power. Ciphertext updates and ciphertext length scale now with a linear overhead for each OR-gate, rather than constant as in the original scheme.

To relate the encrypted file and encrypted file keys a new random file ID is chosen and attached to both payloads. In Step 12 the encrypted file keys (ciphertexts) are uploaded to the CS so that they can be queried and discovered by other clients. A ciphertext, as described by TFDAC-MACS [26], is composed of an encrypted part (namely $C1$, $C2$ and $C3$), the data owner Id, which is only present if two-factor authorization is used (see Section 5.4.2), an conjunctive access policy (set of attribute value IDs) and in addition PAD-TFDAC-MACS adds the file ID which references the actual encrypted content. This last one is uploaded and stored permanently on the CSP (Step 13).

Encryption Utilizing Two-Factor Keys

In order to also secure the ciphertext utilizing two-factor keys, additional preparation needs to be done. As displayed in Figure 5.6, first a new two-factor private and public authorization key is created (Step 1). The private key is kept secretly on the device. It will be used later to issue more two-factor authorization keys to other users or to revoke them.

For each user that shall receive a two-factor authorization key, the device needs to create a user-specific secret two-factor authorization key. To prevent collusion it is made unique for each user using the hash of the UID as the uniqueness source. Before uploading these keys to the CS, the device needs to encrypt them for each user respectively.

This process is quite complex since the CS cannot be trusted to provide the right user device certificates. It can spoof them to retrieve and decrypt the two-factor keys. To circumvent this, the two-factor issuer asks the CS for the authority ID of

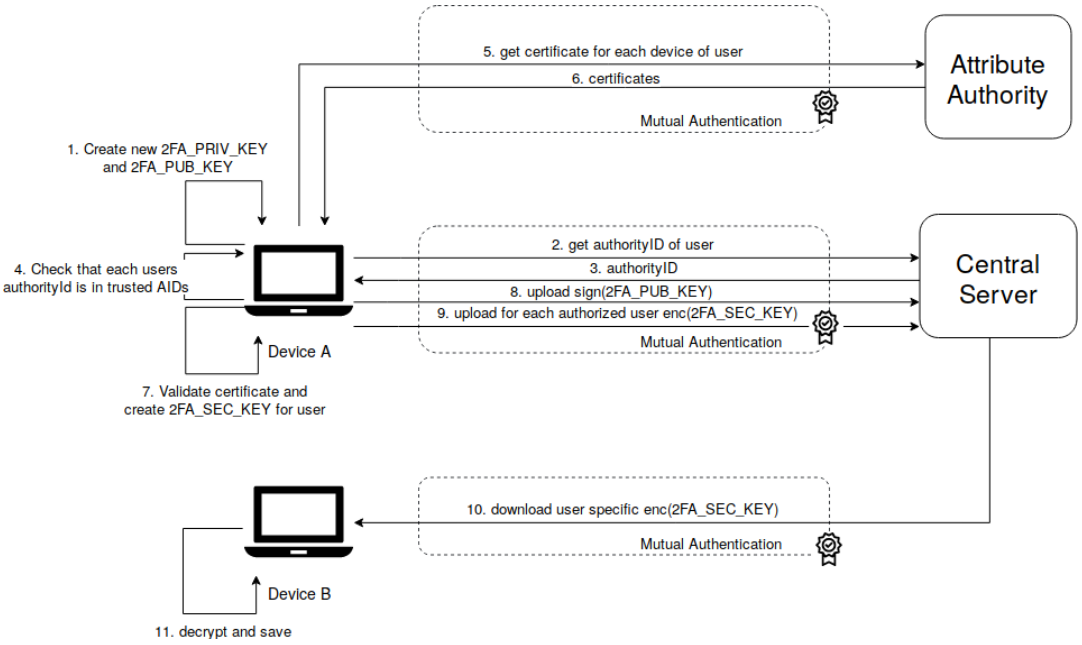

FIGURE 5.6: Encryption using two-factor authorization keys.

the target user (Step 2 and Step 3). It is validated, that the retrieved authority ID is an element in the trusted authority ID set (Step 4). If this check returns success the respective authority is asked directly for the certificate of the devices of the target user (Step 5 and Step 6). The certificates are validated and finally, the user secret two-factor authorization key is created (Step 7). The public two-factor authorization key will be signed and uploaded to the CS (Step 8). It serves as input for the attribute revocation process initialized an AA. Finally, the user secret two-factor authorization keys are encrypted using the RSA public keys of the retrieved X.509 certificates and uploaded to the CS (Step 9). The targeted users, will download the encrypted two-factor secret keys (Step 10) and decrypt them using their private key (Step 11).

Please note, that the secret keys must not necessarily be signed by the data owner. If they are tampered or replaced by the CS, decryption would be broken which is not the goal of the CS.

The encryption process is then triggered by the two-factor issuer. He can use his private two-factor key to additionally secure the prepared file key. Only users who own the correct two-factor secret key and fulfill the access policy can decrypt the encrypted file key.

5.4.3 Decryption

Compared to encryption, decryption is rather straight forward. As Figure 5.7 shows, periodically the device checks if new satisfiable encrypted file keys are present at the CS. This is done by sending all attribute value IDs for that the devices own a secret ABE key in a query to the CS (Step 1). The CS, on the other hand, checks if any ciphertext can be satisfied by any subset of this given attribute set. All ciphertexts are queried if the access policy is completely contained in the given attribute ID set. All ciphertext that satisfies this constrains are returned (Step 2).

The device proceeds to extract the file ID from the given ciphertext and request this encrypted file from the CSP (Step 3 and Step 4). Locally, the encrypted file key is decrypted using the user-specific ABE secret keys issued on device register (see

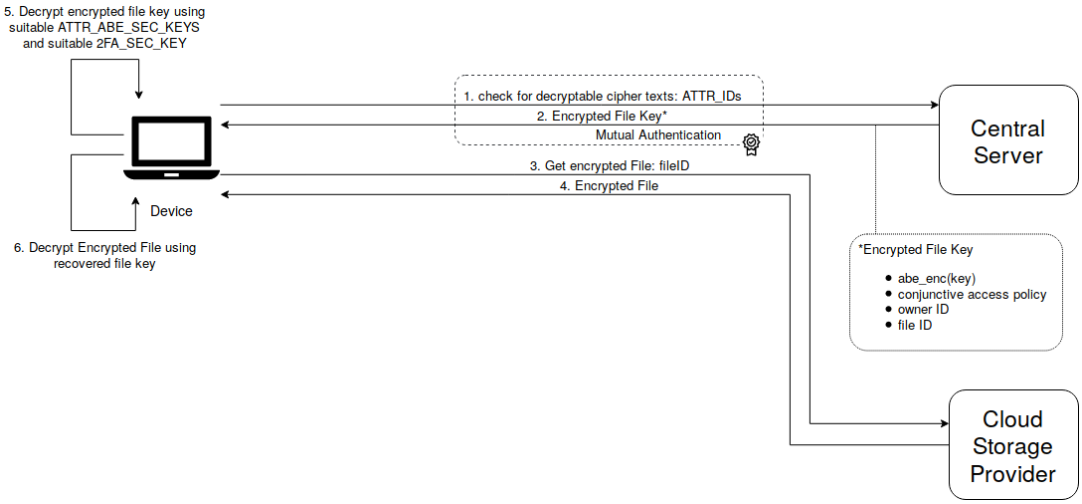

FIGURE 5.7: Decryption phase.

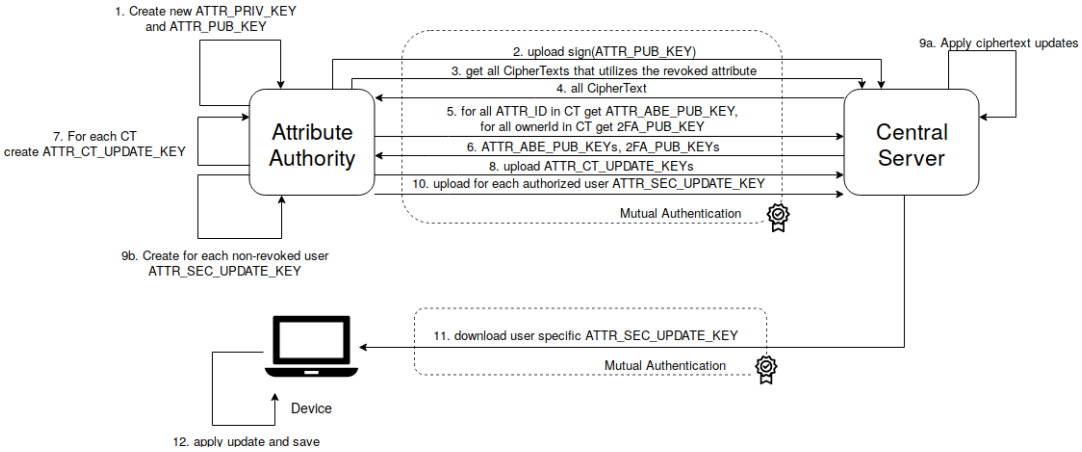

FIGURE 5.8: Revocing an attribute.

Section 5.4.1) revealing the shared secret $K \in G_T$ (Step 5). Using K the plaintext can be recovered from the encrypted file (Step 6).

Decryption Utilizing Two-Factor Keys

If a ciphertext is additionally secured by a two-factor authorization the user needs to use the suitable two-factor secret key to decrypt the encrypted file key. Using a wrong or even no two-factor secret key at all will result in a wrong decryption result.

5.4.4 Attribute Revocation

To complete a revocation process the AA needs to create a new attribute key pair, upload the new attribute public key to the CS where it can be used for additional encryptions, update all encrypted file keys that reference the revoked attribute and finally update the all non-revoked secret attribute user keys accordingly.

As displayed in Figure 5.8, a revocation of an attribute key is always initiated by the AA that issued this attribute in the first place. It creates new private and public attribute components (Step 1) and uploads a signed version of the attribute public key to the CS so that every ciphertext that is created from now on uses the new attribute public key (Step 2). The public attribute key needs to be tamper proof

because it is used on encryption. The CS could act as an own AA to secretly replace the attribute public key with a spoofed one, breaking the encryption process due to global decryption power.

For the ciphertext and user secret key update, a delta between the new attribute private key and the revoked attribute private key is calculated. Since the delta contains no secret information the update key does not have to be encrypted.

Next, the AA needs to update all ciphertexts that have this revoked attribute present. The CS is requested for this affected ciphertexts (Step 3 and Step 4). To construct a valid ciphertext update key, the public attribute keys and the public two-factor keys for each ciphertext are needed. They are requested in Step 5 and retrieved in Step 6. The signature of each of the public keys does not necessarily have to be checked since the threat model of the CS only states that it will tamper values if it results gives a significant decryption advantage. Nevertheless, tampering public attribute keys in that step will result in invalid update keys which will render the encrypted file keys unusable.

Given the public attribute keys, the public two-factor authorization keys and the previously computed delta the AA can create the ciphertext update key (ATTR _CT_UPDATE_KEY) for each ciphertext (Step 7). Those ciphertext update keys are uploaded to the CS (Step 8) which starts updating the corresponding ciphertexts (Step 9a).

While the CS updates the ciphertexts, the AA proceeds by creating an attribute update key for each user's secret key (ATTR_SEC_UPDATE_KEY, Step 9b). Their keys are again user specific to prevent collusion. The actual revocation process is simply to not create an update key for the revoked user. The created secret user attribute update keys are uploaded to the CS where they can be discovered by the respective user (Step 10). Again, this update keys do not have to be signed since they do not contain tamper worthy material, nor do they need to be encrypted since they contain only the delta to the old version. If the old attribute secret key is not known, the delta is useless.

In the last steps, the new update key is downloaded for each user respectively (Step 11) and multiplied to the existing attribute secret key of the user to derive the new attribute secret key (Step 12).

To delete a user all attributes need to be revoked from him. So the sketched process will be triggered for each attribute the deleted user owned.

5.4.5 Two-Factor Key Revokation

The revocation process of revoking a two-factor authorization key (Figure 5.9) from a user follows the same guidelines as the revocation of an attribute. The only difference is that the revocation process is triggered by the user that issued the two-factor authorization key in the first place. Again a new two-factor authorization key pair needs to be created, the public two-factor key needs to be uploaded to the CS, the ciphertexts and the user secret two-factor authorization keys must be updated.

The data owner first creates a new two-factor authorization public and private key pair (Step 1) and uploads the signed public two-factor key to the CS (Step 2) so that an AA can use it for ciphertext updates. As already known from the previous section the delta to the old version two-factor authorization private key is calculated which is used as the baseline for the update key creations.

To create the ciphertext update key, all ciphertexts need to be collected from the CSP that were encrypted using two-factor authorization by this user (Step 3 and Step 4). For each ciphertext, the attribute policy is extracted and added into a set

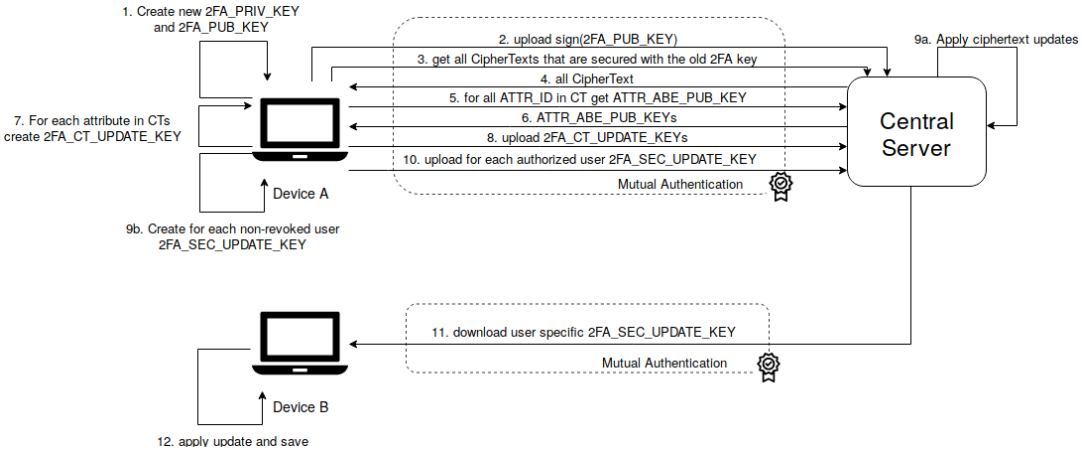

FIGURE 5.9: Revoking two-factor authorization keys.

of attribute IDs. This set of accumulated unique attribute IDs will be sent to the CS (Step 5) to receive the corresponding attribute public keys in return (Step 6).

Taking the previously computed delta and the attribute public keys as an input, the client calculates for each attribute public key a two-factor authorization ciphertext update key (2FA_CT_UPDATE_KEY, 7.). In Step 8, each of the update keys are sent to the CS where they will be applied to update the old two-factor authorizations in the ciphertexts (Step 9a).

While the CS applies the update, the client continues with creating for each non-revoked user a user-specific two-factor authorization update key (Step 9b). Those update keys are uploaded to the CS so that they can be discovered by the respective user, downloaded (Step 11) and applied to the local two-factor authorization secret (Step 12).

5.5 Technologies

The designed system was implemented in a distributed micro-service infrastructure called PAD-TFDAC-MACS . [7] It utilizes the following cryptographic libraries:

- **Bouncy Castle**[8]**:** Bouncy Castle is a cryptographic library that provides many different cartographic settings. In this work, it is many used for 1024-bit RSA encryptions and 256-bit symmetric AES cryptography. Further, it is used to create X.509 certificates and to process certificate signing requests.
- **jPBC**[9]**:** *Pairing-Based Cryptography* is a library written in C that provides the interface to pairing operations. *jPBC* is the Java implementation of this library. Using the native extension a speed-up of pairing operations in the magnitude of 10 to 25 milliseconds can be achieved. [13]
- **Spring Security**[10] Spring Security is a module of the Spring framework and provides a collection of high-level authentication mechanism. In this work, it is used for its mutual certificate authentication.

[7] https://github.com/Anroc/PAD-TFDAC-MACS
[8] https://www.bouncycastle.org/java.html
[9] http://gas.dia.unisa.it/projects/jpbc/
[10] https://spring.io/projects/spring-security

5.6 Implementation And Design Challanges

While implementing and sketching the system design of PAD-TFDAC-MACS different problems were faced. The two main problems are stated in the following.

5.6.1 En- and Decrypting Arbitrary Data

Due to the nature of TFDAC-MACS, PAD-TFDAC-MACS can only encrypt a message K that is an element in the curve field G_T ($K \in G_T$).

However, the jPBC library did not provide any means of mapping arbitrary data to a certain point in G_T. It only provided us with a function to retrieve the binary representation from a point in G_T.

To overcome this obstacle, the algorithm first chooses a random $K \in G_T$ and encrypts K in the ciphertext. K is translated to a binary stream which is then hashed into a byte buffer using SHA-256. In the next step, we will use AES-256 to encrypt our arbitrary data with the previously computed hash.

On decryption, K is recovered from the ciphertext, again transformed into a binary stream, and then the SHA-256 hash of K is calculated to reconstruct the AES-256 secret key. This key can finally be used to decrypt the encrypted file.

5.6.2 Distributed Infrastructure And Parallelism

There are two updates in TFDAC-MACS: The attribute revocation update and the two-factor key update. If an administrator removes a user from the system all attributes of this user need to be renewed. To make ciphertexts decryptable for these new attributes they must be updated as well. For that, the AA issues an update key which is used by the CS to calculate the new updated version of the ciphertext.

On the other hand, there is the two-factor revocation. A user wants to revoke the two-factor key access from a specific user. To do so he issues update keys to each non-revoked user and a ciphertext update key to the CS to also update the respective ciphertexts.

While both operations are by design associative, they require transactional execution of the revocation process. But this constraint cannot be provided in a distributed infrastructure.

For a two-factor revocation, all relevant attribute public keys are needed. If at the same time a public attribute key is updated by an AA a malfunctioning two-factor update key will be calculated. For example, assume the following execution order:

1. The Data Owner downloads the attribute public key in version 1 and computes the update key 2FA-Update.
2. AA issues an attribute public keys in version 2 and uploads this one to the CS and computes the update key ATTR-Update.
3. AA sends the ATTR-Update key to the CS which triggers the update process for all ciphertexts which are encrypted using this given attribute.
4. In parallel, the data owner sends the 2FA-Update key to the CS which starts the update process of all two-factor secured ciphertexts that are secured using the old two-factor authorization key
5. Critical is now this step where the CS updates a ciphertext using the 2FA-Update key which is already updated using the ATTR-Update key rendering the ciphertext unusable

$$C_3'' = \left(\prod_{v_{aid_t,j} \in W, v_{aid_t,j} \neq v_{aid_i,j}} g^{y'_{aid_t,j}} \right)^{(s+\beta+r')} \cdot \underbrace{\left(g^{y'_{aid_i,j}}\right)^{(s+\alpha+r')} \cdot \left(g^{y_{aid_i,j}}\right)^{(\beta-\alpha)}}_{\text{Due to different basis terms can not be aggregated.}} \tag{5.15}$$

In equation 5.15, $g^{y'_{aid_i,j}}$ references the revoked attribute and $\left(g^{y_{aid_i,j}}\right)^{(\beta-\alpha)}$ the two factor update key $UAU_{aid_i,j}$ for this revoked attribute. In that case the data owner used the old attribute public key $g^{y_{aid_i,j}}$ to compute $UAU_{aid_i,j}$. In contract to the attribute-level revocation process of Section 5.1.5 and the user-level revokation process of Section 5.1.6, the bases are now different which does not allow us to combine them, rendering the ciphertext unusable.

The very first step to countermeasure this kind of problem is to detect this issue. This is done by assigning each key a version. If a private ABE key gets created it gets the initial version 0 assigned. Each other key that is derivated from this private key ends up with the same version. In the same way, the user attribute secret keys and two-factor secret keys are created. Both of them need to be updated if a revocation takes place.

On the other hand, ciphertexts contain for each attribute used in its policy the version of the used attribute public key as well as the version of the public two-factor authorization key that was used to additionally secure it. Only keys that are present in the same version as the used public encryption keys can be used to decrypt this ciphertext. Each other key will be rejected.

Update keys contain a target version and increment the version of the applied key. If applied onto a ciphertext it will increment the version of the targeted two-factor component or targeted attribute component.

Having this versioning in place, the CS can now detect any wrong issued updates to a ciphertext. It will check that the given update is applicable to the given ciphertext and only if this check returns true, it will proceed to update the ciphertext. Next steps will be to notify the update issuer about the conflicting versioning and request an updated update key.

Versioning of the keys also helps the user on decryption. If he downloads a ciphertext that is secured using old or too new public keys the client can try to check for new updates of his secret keys or a new version of the ciphertext and then continue the decryption. Since updates on keys need to be incremental and strictly sequential, the CS or user need to wait until a suitable update was issued for him.

Chapter 6

Evaluation

The main issue of the current implementation in CloudRAID is the poor scalability of the number of file keys. For each new device, a new file key needs to be created and maintained. The proposed prototype serves as the proof-of-concept that using multi-authority attribute-based encryption the number of file keys and ciphertext size can be reduced while the practical applicability is still preserved.

To evaluate this thesis, different benchmarks will be conducted to compare the proposed prototype to a similar environment such as CloudRAID. The goal of this section will be to show that the following assumption holds true:

The ABE-based solution of the in Section 5 proposed system to solve the scalability problem of a secure cloud storage system, scales at least as good as the RSA-based approach, described in Section 3.1, regarding the number of file keys that need to be maintained.

While this assumption should hold true for all group operations the performance of those operations needs to be taken into account as well. The goal is to find the number of attributes per user, at which PAD-TFDAC-MACS performs at least as well as the classical encryption scheme. It is expected that such a number can be found for encryption and member join operations. But when it comes to member leave operations, it is expected that there is no configuration in which the proposed prototype performances better than the RSA solution, due to additional shared-key management effort and complex revocation mechanism.

In Section 5.3.1 we proposed an adaptation to TFDAC-MACS making the two-factor authorization key optional. The following benchmarks will be conducted without the two-factor authorization key mainly because TFDAC-MACS [26] already conducted a performance analysis comparing its scheme against related once and because the two-factor key only adds a small constant overhead.

6.1 Classical RSA-Based Re-Encryption Scheme

The implemented cryptographic reference system creates a new file key per encrypted file and accessing entity. To encrypt a file, a symmetric 256-bit key (AES-256 bit) is generated. To grant other entities access, this symmetric key is encrypted asymmetrically (RSA-1024 bit) using the public key of each member respectively.

Since this results in many encrypted file keys for one file, the file keys size (ciphertext size) is defined as the aggregate sum of the storage consumption of all encrypted file key for this file. The performance to share a file is measured over the whole encryption process until for each member a respective file key is created and the file is encrypted accordingly.

On decryption, one member is picked at random who decrypts his encrypted file key using his RSA private key to recover the symmetric key. This key is used to recover the encrypted file.

Inviting a new member is defined as the process of re-encrypting all previous shared encrypted file keys for the new member. This process is done by one member that decrypts all of his encrypted file keys using his RSA private key. The resulting symmetric keys are then encrypted again using the new members RSA public key. The symmetric file keys do not change and the encrypted file remains untouched.

Finally, if a member leaves or gets removed from a group, all of his encrypted file keys are deleted. The underlying symmetric keys remain unchanged.

For the cryptographic operations of RSA and AES the library *bouncy castle 1.60* is used while the previously mentioned group sharing scheme is implemented by this work. All computations are done locally on one machine without sending anything over the wire.

6.2 Upper-Bound And Worst-Case Scenario

In this section, it will theoretically be argued why PAD-TFDAC-MACS scales at least as good as the current solution for secure cloud storage systems. The number of file keys scale in a ratio of one-to-one with the number of OR-gates $+1$[1] in the access policy as described in section 5.3.3. The maximal number of OR-gates in an access policy would be to mention each user in the group explicitly. Such a policy is displayed as an example in Listing 6.1.

```
(alice@example.com OR bob@example.com OR ... OR zara@example.com)
```

LISTING 6.1: Example worst-case access policy. The used email address is a unique attribute per user.

It does not make sense to include more OR-gates into this policy since it would represent redundant information. The number of file keys per n users scale with $O(n)$ (same way as CloudRAID) in PAD-TFDAC-MACS if and only if, no suitable subset of distinct attributes of any two users can be found so that the n users can be described with $o < n - 1$ OR-gates o. That means if at least k-users, with $k > 1$ can be completely described by an AND-policy, the number of file keys needed will shrink to $n - k + 1$ file keys. Each OR-gate that can be substituted by AND-gates reduces the number of file keys. If that is not the case, say no users share the same attribute, then a unique attribute for each user has to be used (see Listing 6.1).

Analogously, the best access policy that can be constructed would be one using only AND-gates resulting in only one file key regardless of the number of embedded attributes. This behavior is inherited from TFDAC-MACS where it is possible to embed any number of attributes in a disjunctive condition into one file key.

6.3 Performance And Scalability Analysis

To evaluate the performance and scalability of PAD-TFDAC-MACS against the RSA-based implementation different benchmarks are conducted. While the main focus remains on reducing the number of file keys for group operations, it is also important to measure the performance of setup, encryption, decryption, member join, and member leave actions. In the following sections, each of the five different actions will be evaluated and an expectation about the experiment outcome will be given.

[1] A file key needs to be created for the left and right sub-tree respectively. Having an access policy with one OR-gate will cause the creation of two file keys.

The benchmark evaluation is executed on a Server machine utilizing 6 vCPU cores (Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 30MB cache size), 12 GB RAM and running on Ubuntu 16.04 LTS 64bit. For the evaluation, OpenJDK 1.8.0_191 is used. Each of the benchmarks does not spawn any threads or uses parallelism. [2] For pairing operations, the native PBC extension is used. [3]

In the following sections, the performance of scalability of the number of size of file keys will be measured. As already mentioned in Section 6.1, the file key size (ciphertext size) of the reference system will be calculated as the aggregate of all file key sizes for a file. For any group operations, the performance will be measured in terms of how long the execution took to complete. A group operation is finished if all non-revoked members can access and decrypt the encrypted file. The content that is encrypted and shared is always the simple ASCII encoded string "Sample text that needs to be encrypted.".

Each benchmark is averaged over many executions. Still, artifacts and noise may occur causing spikes or sudden outliers. Before each benchmark, a warm-up phase is executed to fill caches and optimize branch predictions.

6.3.1 Setup

In the setup phase the initial global public system parameter (GPP), authority key, attribute value key, attribute user secret key, two-factor key and the two-factor secret key is created. The generation of these keys has a constant overhead. The fact that a system based on RSA only needs to create the respective RSA key pair can be taken as a disadvantage of ABE.

Expectation

It is expected that all of the key generations terminate rapidly. The GPP creation will take a bit longer depending on how fast suitable random parameter can be gathered.

Benchmark: Setup

In Figure 6.1 the box plot over the different setup procedures is displayed. It displays the median of the data (green line in the box) and the dentist area where the 50 percentile of the data is located in (the box). The so-called whiskers (horizontal lines) indicate where the missing 48.6 percentile of the data is located. The bubbles show the missing 1.4 percentile of outliers.

RSA 1024 bit key generation takes on average $\sim$ 7.5ms. This is certainly due to finding two random large prime numbers. This process might take longer depending on how fast these numbers can be found.

Since the generation of Two-Factor, authority and attribute key are computationally equal intensive (see Section 5.1.1) they also show the same overhead in Figure 6.1. The box plot displays very less measurement variety and these operations usually terminate within $\sim 1.8ms$. These operations need to be done once per entity. More complex operations need to be done for the secret key generations of the two-factor key and the attribute value key (see Section 5.1.2). Here an additional hash operation in combination with the respective private key result in an additional overhead of $\sim 6ms$ per user.

[2] Single thread assumption is based on the implementation of PAD-TFDAC-MACS . This assumption must not hold true for libraries that are used under the hood for RSA and AES cryptography (bouncy castle 1.60) or pairing-based cryptography (jPBC 2.0.0).

[3] https://crypto.stanford.edu/pbc/

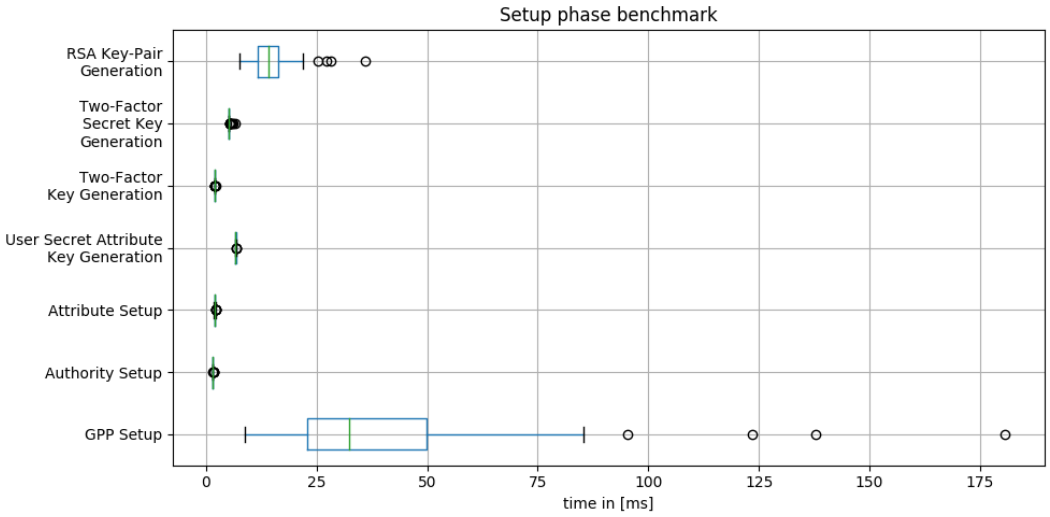

FIGURE 6.1: Different setup mechanism evaluated over 50 executions.

As expected, the longest setup operation is the creation of the GPP. Here a suitable elliptic curve must be found that is pairing compatible in the desired field. The reason for the big execution variance from 15ms to 85ms is rooted in the problem of finding a suitable curve parameter as well as a valid generator for this curve. However, since the GPP only need to be determined once by the central server it is no issue that this operation consumes time.

6.3.2 Encryption

Encryption is the most interesting topic to evaluate since it scales using RSA with the number of users and using ABE with the number of attributes. Having the assumption of Section 4.2.3 in mind, that states that the number of attributes is smaller or equal to the number of users in the shared group, ABE should be able to reduce the number of file keys.

Expectation

Classical encryption that uses RSA to encrypt the file key asymmetrically scales with the number of key-identities, in the following entitled users. For each new file in the group, a suitable file key for each user must be created. The resulting assumption is that the number of file keys in the RSA based approach scales linearly with an increasing number of users. Same holds true for the ciphertext size which describes the aggregated sum of all file keys for one file. For the RSA-based approach, the file keys should scale in the ratio of 1-to-1 with the number of users and therefore show a linearly increasing overhead.

In contrast to that stands PAD-TFDAC-MACS . It scales with the number of attributes per ciphertext rather than the number of users. Given this assumption, an intersecting point of "attributes per user" can be calculated where PAD-TFDAC-MACS is expected to perform better than the RSA approach. Where this intersecting point is located depends on the number of attributes, number of users, the chosen policy and the computing power of the executing machine. At some point, the access policy can get too large so that the prototype solution will have no advantage regarding the computation time.

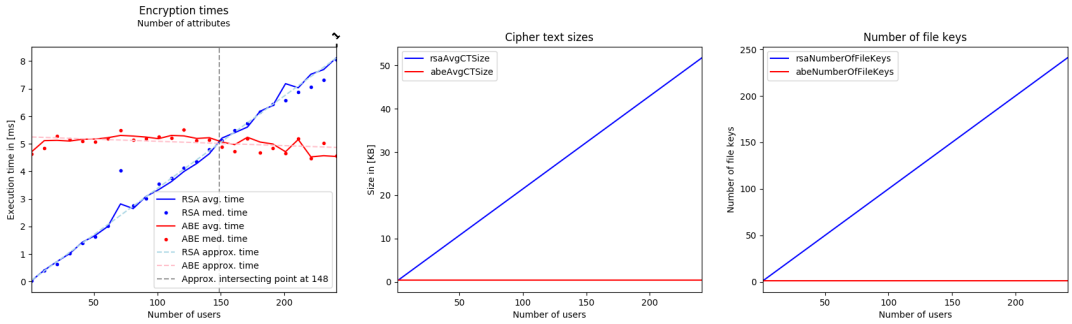

FIGURE 6.2: 1-for-∞ configuration using AND-gate policies.

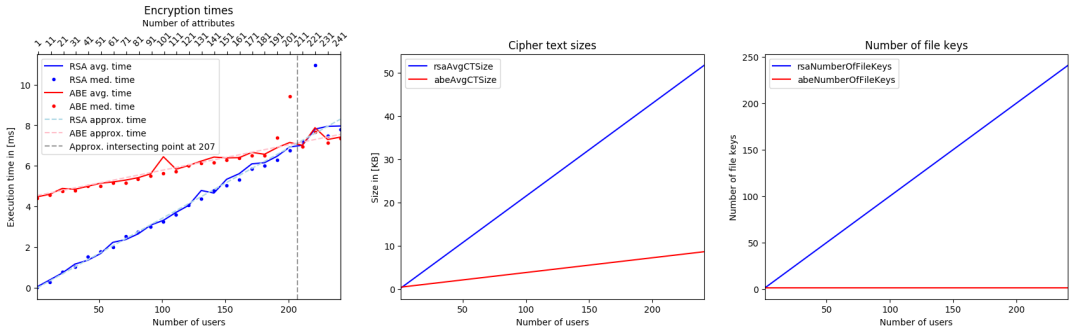

FIGURE 6.3: 1-for-1 configuration using AND-gate policies.

While the previous assumption holds true for only AND-gate policies we already stated in Section 5.3.3 that the number of file keys for policies utilizing OR-gates scale with the number of OR-gates +1. Since for each conditional subtree of such an OR-gate a completely new ciphertext is created, the performance will decrease as well.

Benchmark: AND-Gate Policy

PAD-TFDAC-MACS provides the user with the option to use logical AND-gates or OR-gates to describe his access policy. The AND-policies (containing the best-case) and then the OR-policies (containing the worst-case) are analyzed independently from each other for computational performance and scalability of the number and size of the file keys. The benchmarks are performed over an increasing number of users. For each n users a new attribute is introduced. The notation of a "a-for-n configuration" defines a as the number of attributes *for* n users. The benchmark is evaluated for the configurations 1-for-∞ (Figure 6.2), 1-for-1 (Figure 6.3), 1-for-2, 1-for-4, 1-for-5 (Figure 6.4). Given the number of users the attributes can be calculated in the following way: $\lfloor \frac{n}{a} \rfloor$.

The configuration of 1-for-∞, as shown in Figure 6.2, describes the best-case scenario. Here a group can be completely described by only one attribute regardless of the number of users. This can, for example, be the case when an employee wants to share a file with the whole company. In this setting he only uses the attribute "works in company X" to encrypt the file. PAD-TFDAC-MACS has a constant overhead since the number of attributes remains constant in one as well. Interestingly, the first graph in Figure 6.2 shows a linear decreasing overhead. Such behavior can only be traced back to optimized caches and correct branch predictions and less left-over artifacts from the initialization process. For that best-case scenario, the intersecting

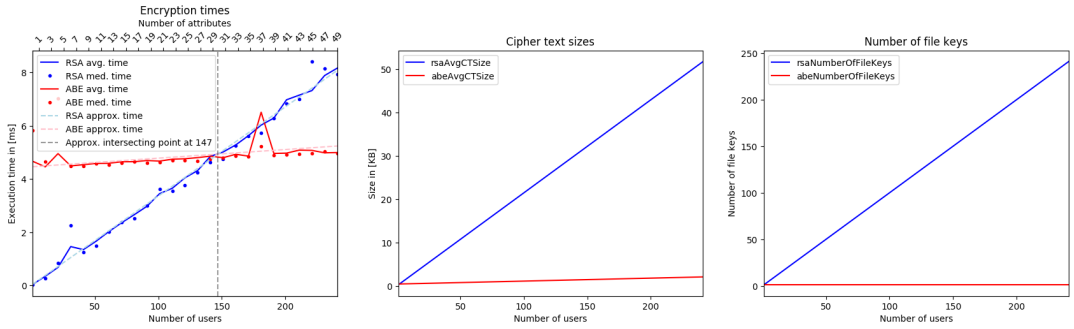

FIGURE 6.4: 1-for-5 configuration using AND-gate policies.

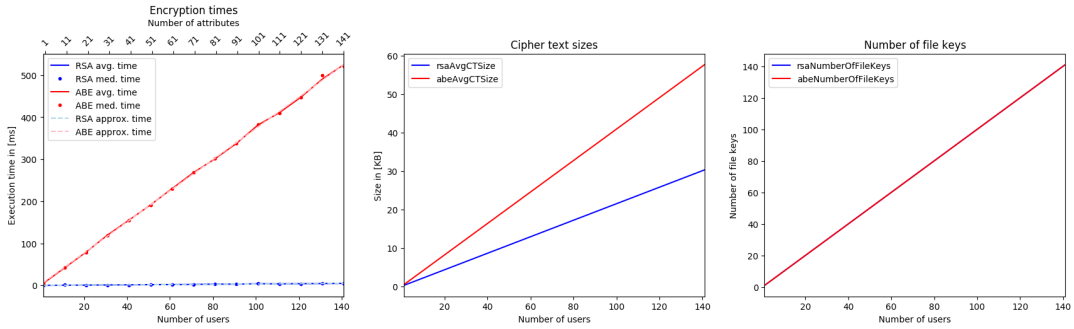

FIGURE 6.5: 1-for-1 configuration using OR-policy.

point where PAD-TFDAC-MACS performs better than the RSA-based approach can be approximated at 145 users.

With an increasing number of attributes per user, the overhead of PAD-TFDAC-MACS becomes greater. In the 1-for-5 configuration in Figure 6.4 the overhead of the prototype has become linear. This is due to additional computations for the increasing number of attributes in the ciphertext. In the configuration of 1-for-1 (Figure 6.3) where the number of attributes is the same of the number of users the intersecting point gets pushed back to roughly 200 users.

Comparing the number of file keys over the increasing number of users, it is observable that the number of file keys indeed scale truly linearly to the number of users in the RSA-based solution. On the other hand, remain the number of file keys constant at 1 for PAD-TFDAC-MACS when using only AND-gates. Since it is not enough to simply decrease the number of file keys the ciphertext sizes must be considered as well. We can also show that in the prototype implementation the ciphertext size grows linearly with an increasing number of attributes that need to be embedded into this policy, therefore, invalidating the claim of TFDAC-MACS to have constant sized ciphertexts (Section 5.2.1).

Benchmark: OR-Gate Policy

A completely different picture is drawn for the benchmark of OR-policies. Since for each OR a new ciphertext needs to be created PAD-TFDAC-MACS scales with the number of OR-gates present in the access policy. This affects execution time, the number of file keys and ciphertext size.

Please note that the RSA-based function is the same in every figure. The scale and the ABE-plot differ for each figure.

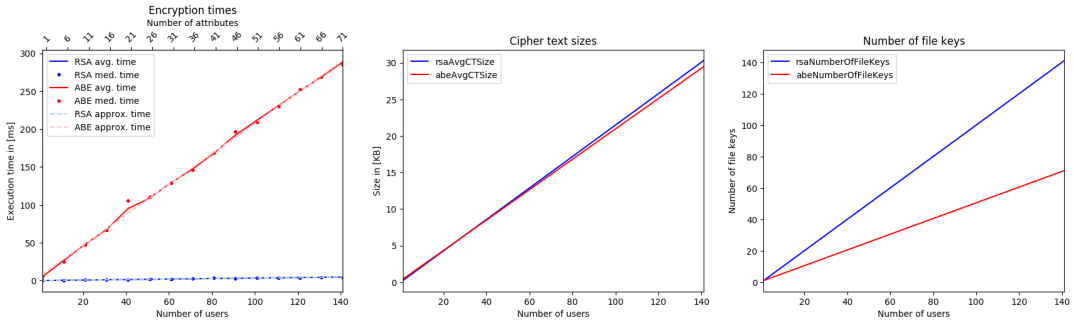

FIGURE 6.6: 1-for-2 configuration using OR-policy.

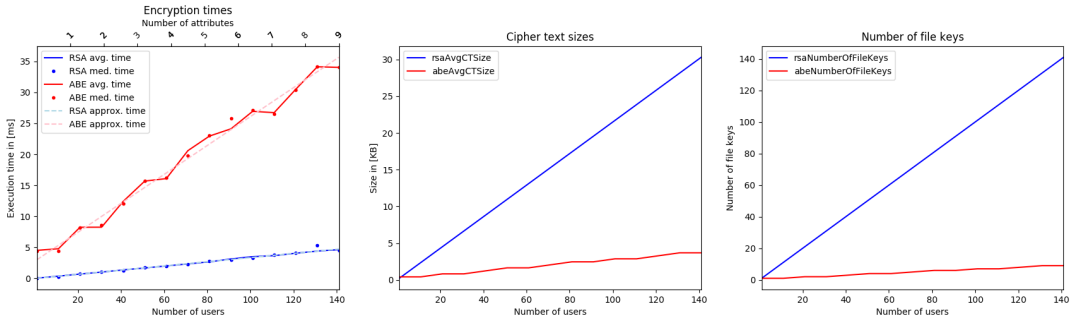

FIGURE 6.7: 1-for-16 configuration using OR-policy.

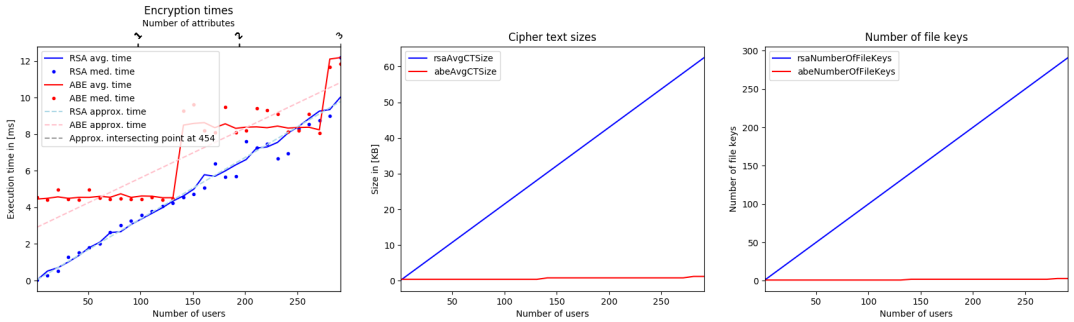

FIGURE 6.8: 1-for-140 configuration using OR-policy.

The worst-case scenario, as introduced in Section 6.2, is defined as the scenario were the number of users is equal the number of unique attributes combined in an OR-gate policy. This case is shown in Figure 6.5. Here the encryption time of PAD-TFDAC-MACS is two magnitudes higher ($\sim$ 0ms $-$ 550ms) than the computation time of the RSA-approach ($\sim$ 0ms $-$ 10ms). Even the ciphertext size is substantially larger than the ciphertext size per file key. This results from the fact that now, PAD-TFDAC-MACS scales with the same number of file keys as the RSA approach. No advantage can be gained by the prototype in the worst-case scenario. In general, it holds true that the regarding the number of file keys PAD-TFDAC-MACS scales better if a group of user can be described by $o < n - 1$ OR conditions o and n as the number of users.

As expected, if a configuration of one attribute per two users is used (Figure 6.6) the file keys scale twice as well and the ciphertext size scales rather similar to the size of the RSA based file keys. Further, the ciphertext size is now again below the ciphertext size of the RSA approach.

Reducing the overhead to a 1-for-16 configuration using OR-policies, as shown

in Figure 6.7, a stepwise, increasing function is visible in each of the plotted graphs. Each step is directly correlated to another attribute addition in the access policy. Since each addition of an attribute into the OR-gate policy triggers the creation of a new ciphertext, the computational overhead increases stepwise.

As we extracted from the plot in Figure 6.2, at least $\sim$ 145 users need to be described by one attribute to scale better than the classical encryption scheme. Using this result as a baseline, the configuration 1-for-140 was chosen, where every 140 users a new attribute is combined using an OR-gate with the existing policy. The result was a stepwise increasing function as displayed in 6.8. Each step correlates again to the addition of a new attribute, triggering the creation of a complete new ciphertext. The same conclusion can be made for the ciphertext size and the number of file keys. Based on the shown approximation line, the assumption can be extracted that PAD-TFDAC-MACS performs better regarding computation time if only every 140 users a new OR-gate is introduced in the group access policy.

6.3.3 Decryption

The decryption process in RSA is a constant overhead process. Simply the file key needs to be decrypted. In PAD-TFDAC-MACS on the other hand, scales with the number of attributes embedded in the given ciphertext. All by the policy required attribute secret keys need to be multiplied together before the underlying secret can be recovered. This operation has only little overhead.

Since PAD-TFDAC-MACS generates for each OR-gate a new ciphertext, the access policy in the real file keys only contains AND-gates. A decrypting user only needs to decrypt one ciphertext to retrieve access to the encrypted file. Therefore, it makes no difference for decryption performance whether the data owner chose to embed OR-gates into his policy or only uses AND-gates.

Expectation

PAD-TFDAC-MACS will show a bigger overhead as the RSA approach due to more expensive pairing based operations.

Benchmark: Decryption

In this benchmark, only one attribute was used by the data owner to secure the ciphertext. Each decryption process was measured over 50 runs, including the decryption of the encrypted file.

Figure 6.9 mirrors the expectation. The proposed prototype indeed shows a clearly visible overhead on decryption compared to the RSA-approach. The overhead of the additional two-factor authorization is located in the order of $\sim 1.2ms$. The RSA-based approach shows an impressive speed in the order of less than one millisecond.

6.3.4 Member Join

If a new member is invited to a group, in RSA each file key needs to be re-encrypted for the new member. That means that each file key needs to be decrypted using the private key of an existing member and encrypted again with the new members public key.

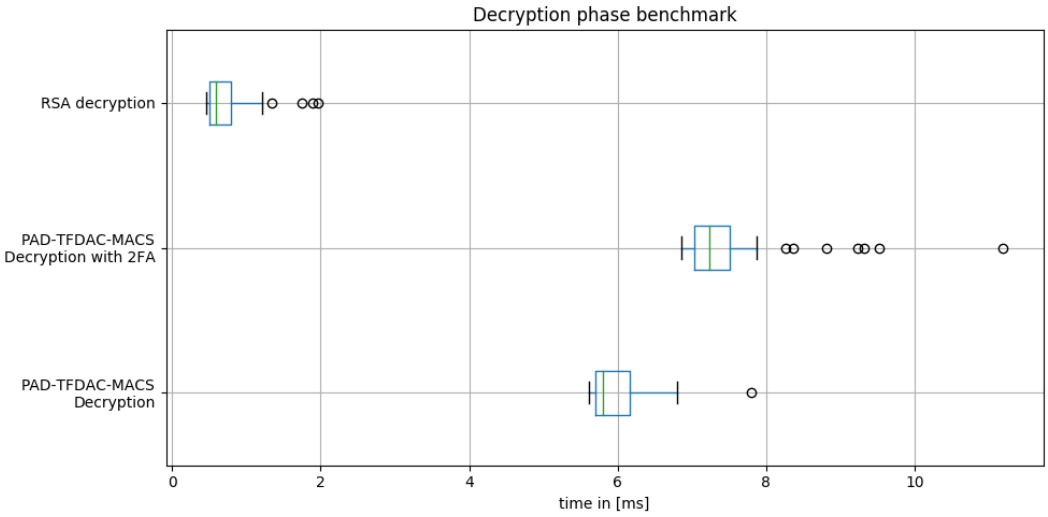

FIGURE 6.9: Box plot of the decryption process with and without two-factor authorization.

For PAD-TFDAC-MACS , the AA just issues the respective attribute secret keys to the member which is a process with constant overhead. Automatically, he is granted access to the encrypted file keys in the group.

Expectation

Based on those facts, it can be assumed that PAD-TFDAC-MACS scales at a constant overhead while the RSA approach scales linearly with an increasing number of ciphertexts. While the number of file keys will stay the same for PAD-TFDAC-MACS , for the reference system they will scale linearly depending on the number of already present file keys.

Benchmark: Member Join

Three group configurations were evaluated: A group having 1, 2 and 4 attributes respectively. In the benchmark, it is assumed that a user needs to be issued all of the 1, 2 or 4 attribute secret keys that describe the group to become part of it.

As shown in Figure 6.10 the previously stated assumptions can be backed. Each of the different configurations show a constant overhead. Depending on the number of attributes the intersecting points of 20, 40 and 90 ciphertexts can be identified. After this amount of ciphertexts in the group, the prototype scales better. Of cause, these numbers depend on the underlying hardware and might be different on other systems. However, the overhead should still scale with the same ratio.

6.3.5 Member Leave

If a member leaves or is removed from a group, all of the group attributes need to be revoked from him. This comes in PAD-TFDAC-MACS with additional overhead since for each attribute in the group a new attribute private and public key need to be generated and for each non-revoked user, a secret update key must be created. These secret update keys need to be applied to the secret attribute keys of the non-revoked users respectively. And finally, a ciphertext update key needs to be calculated and

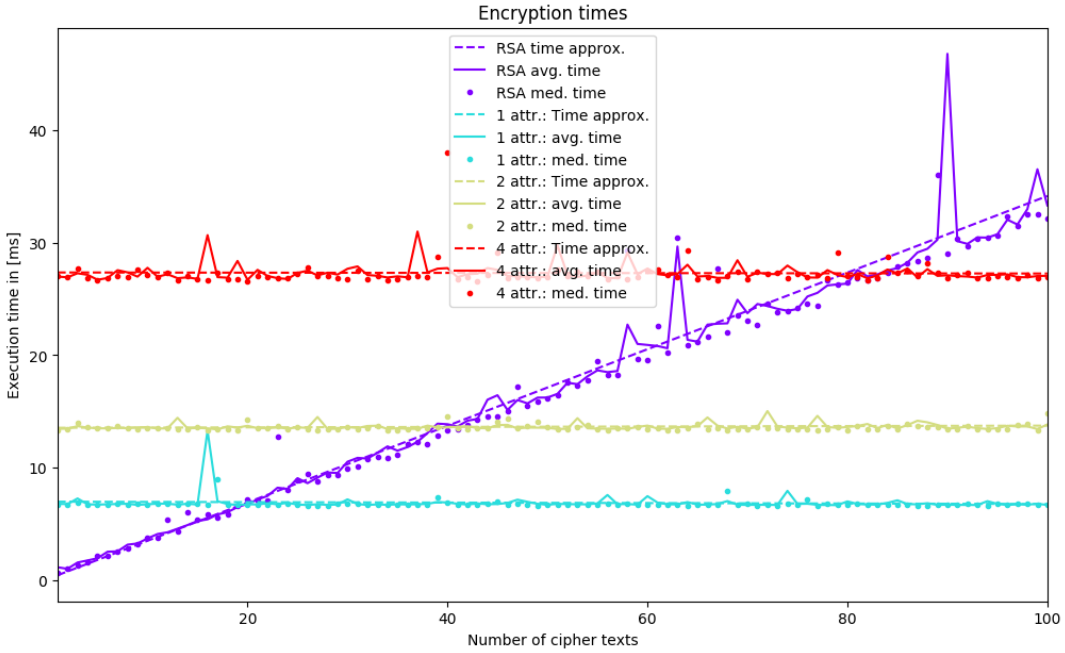

FIGURE 6.10: Re-encryption of file keys for one new member. Scaled over the number of attributes.

applied to all ciphertexts that contain the revoked attribute to make them accessible to the new attribute key.

The RSA-based approach has a big advantage here: Due to the fact that each file key serves as a unique entry point for a specific user to decrypt the encrypted file, to revoke the user form the group this file key just have to be deleted.

Expectation

The assumption is simply stated that PAD-TFDAC-MACS does not perform better than the RSA-based approach. Since it scales with the number of attributes that need to be revoked a_{rev}, the number of other non-revoked users $u - 1$ and the number of ciphertexts c we end up with an overhead of $O(a_{rev}(u-1)c)$ which equals a cubic overhead. The reference system is expected to just scale linearly with the number of ciphertexts.

Benchmark: Member Leave

Figure 6.11 shows the exact expected behavior. For already one ciphertext the gap between the different number of attributes of the ABE implementation and the RSA-based approach is already clearly notable. PAD-TFDAC-MACS is significantly slower than the RSA-based approach. The RSA member leave operation does also scale linearly. It is just not visible since the overhead is neglectable small.

In a real-world system, the ciphertext and the user secret key update would happen on the server and client device side respectively. Further, the application of the update key to the secret keys or ciphertexts can happen in parallel to further speed-up this process. In the end, PAD-TFDAC-MACS scales much worse since it comes with unneglectable additional overhead.

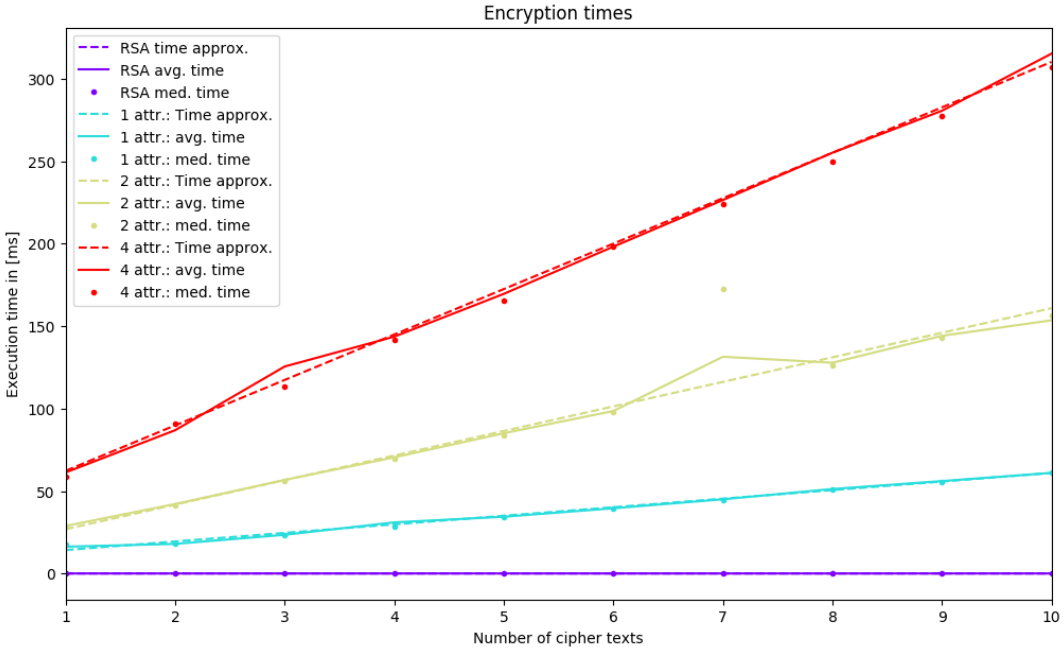

FIGURE 6.11: One member leaves the share. Scaled over the number of attributes.

6.4 Practical Applicability

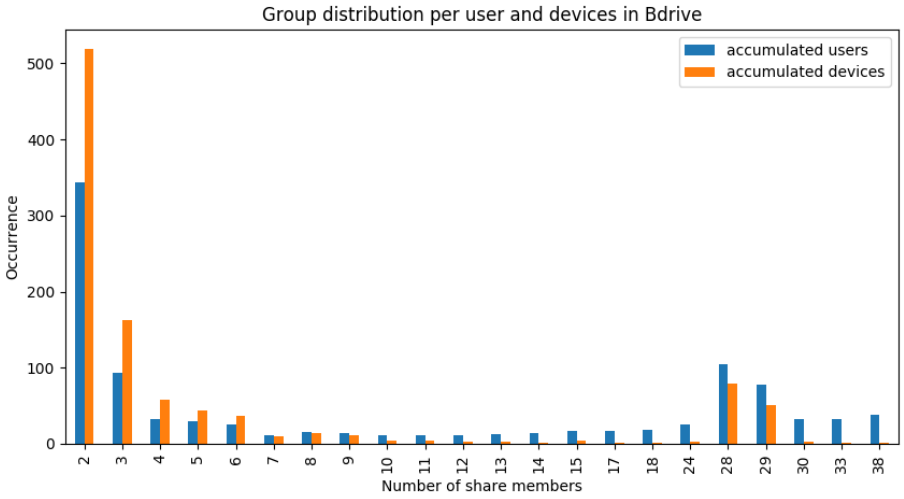

FIGURE 6.12: The distribution of shared folders per users and their aggregated devices in Bdrive.

To evaluate whether PAD-TFDAC-MACS would be practically applicable in the real world, we can use the conclusion of the best-case scenario as a baseline. It was stated that PAD-TFDAC-MACS performs and scales better than the CloudRAID if at least 145 users can be described by the same attribute. To extrapolate this conclusion onto real-world data, a look at the secure cloud storage system Bdrive[4] was conducted.

[4] https://bdrive.cloud

Bdrive is the practical implementation of that in CloudRAID and its proposed erasure encoding. It uses the classical re-encryption scheme based on RSA and AES to secure and share a file in a group. Bdrives target audiences are businesses which make it perfectly fitting for the use-case of PAD-TFDAC-MACS .

The distribution of groups per members and devices of Bdrive is listed in Figure 6.12. As it can be extracted from the figure, many users share file only in a peer-to-peer fashion. In that case, PAD-TFDAC-MACS would provide no performance advantage. However, really interesting for the ABE use-case is the second spike at 28 to 30 members. Since Bdrive is addressed to business it can be assumed that this second spike defines the intra-company-wide shared folders.

In intra-company shared folders, all employees are members of the same share. Therefore, PAD-TFDAC-MACS can enfold its true potential since all members can be described by only one common attribute: "Employee in Company X". In large scale companies, the number of users that share the same attributes might even rise and thus exceed the 145 threshold limit where PAD-TFDAC-MACS can scale better than the classical approach.

However, it must also be noted that PAD-TFDAC-MACS only provides an advantage if the cloud storage system deals with companies that have thousands of users. Even in the use-case of an intra-company shared folder, it only provides a performance advantage on single file encryption, if the group as more then 145 members.

Further, PAD-TFDAC-MACS shows significant worst performance on peer-to-peer file sharing since this scenario would indicate many policies build over unique user attributes combing them with logical OR-gates. As an implication, PAD-TFDAC-MACS creates for each peer a file key and thus, as described in Section 6.2 and validated in Figure 6.5, comes with much more overhead than RSA would produce.

In conclusion, PAD-TFDAC-MACS can reduce the number of file keys dramatically if combined with the right access policy. In the end, it comes down to the company administrator to assign each user suitable attributes so that fine-grained access formulas without using many OR-gates can be constructed.

6.5 Summary

The performance and scalability analysis performed in this chapter, strengthen the thesis that PAD-TFDAC-MACS scales better than the reference implementation of the classical RSA-based sharing scheme regarding the number of file keys and the ciphertext size.

It was shown that in the worst-case scenario the same number of file keys where produced and in the best-case scenario only one file key need to be maintained by CloudRAID. While the number of file keys is certainly an issue, the size of the resulting file keys needs to be considered as well. The proposed ABE-based solution consumes much more storage per file key due to additional meta information, such as the access policy and data owner references. Taking this into account PAD-TFDAC-MACS achieves its goal of consuming less file key storage if a group can be described by $n/2$ conditions combined using OR-gates (see Figure 6.6 for reference).

The provided benchmarks were conducted with enabled two-factor authentication as well but showed a neglectable overhead in performance. There are no changes in the number of file keys.

In the end, it boils down to the access policy a user chooses. If this policy is constructed in an excluding fashion (using only or mostly AND-gates) PAD-TFDAC-MACS performs better in the long-term. But if too many attributes are used in the access policy many ciphertexts need to be updated each time a user or attribute gets revoked from the system.

Chapter 7

Conclusion

In this thesis, a solution to the scalability problem of classical RSA-based group sharing scheme was presented. Careful respect was put into not violating any of the in Section 3.4 introduced security requirements. The classical systems suffer from an unmaintainable number of file keys that increases on each file upload and on adding a new member into the group. Existing members need to make sure that the newly joined member can access all previously shared files by decrypting all file keys and encrypting them again with the new members public key, creating own file keys for the new member. If a new file is updated, it needs to be made accessible to all members of the group. A file needs to be encrypted symmetrically creating a file key, which will then be encrypted asymmetrically for each member in this group.

Different approaches are analyzed and argumentatively compared. The well-known concept of secure group communications was analyzed for applicability. Such systems introduce a shared secret key to encrypt the shared files. While greatly improving the file key overhead to the current reference implementation, we can do even better. A closer look at ABE based cryptography was taken. The advantage of ABE in comparison to secure group sharing scheme relies on the fact that with ABE the amount of file keys scale with the number of attributes rather than the number of users involved in the sharing. Under the assumption that a system will have fewer attributes then there are users, ABE can achieve a better file key overhead. Since ABE encrypts under attributes rather than users, it comes additionally with the advantage that users are implicitly granted access to groups when they get a new attribute issued. In that way, no initial re-encryption of the group key is needed.

MA-ABE is the practical adaptation of ABE introducing multiple authorities and thus deescalating the global decryption power of a centralized attribute authority. ABE uses attributes to create an access policy under which a file is secured. This access policy acts like a shared group key that is only accessible to the members that satisfy this access policy. Each scheme using a group key only needs to encrypt the file once using the group key resulting in one shared file key. In general, does ABE and SGC schemes come always with additional overhead when it comes to forward secrecy. The group key must change and so must all previously encrypted file be re-encrypted with the updated group key.

From the current field of MA-ABE schemes, TFDAC-MACS [26] was the candidate that showed great scalability and performance, no global decryption power of the CS and a mechanism to revoke users from the system. It further managed to reduce the number of file keys to one regardless of the number of accessing users. Since it does not support 1-of-n threshold gate policies (OR-gates), a dynamic attribute universe nor was the two-factor part optionally, an improved version of TFDAC-MACS, namely PAD-TFDAC-MACS (Practically Applied Distributed Two-Factor authentication Data Access Controll for Multi-Authority Cloud Storage systems), was constructed and evaluated.

To evaluate whether PAD-TFDAC-MACS indeed scales better than the classical re-encryption scheme different benchmarks were performed against a reference implementation of the classical RSA-based re-encryption scheme. The outcome showed that the classical system scales with the same overhead as the worst-case scenario of PAD-TFDAC-MACS . In each other cases, PAD-TFDAC-MACS showed scalability improvements regarding the file key size and the number of file keys.

Performance-wise the proposed implementation can achieve a better computational overhead depending on which access policy is chosen by the user. To achieve a better single encryption performance than the classical system at least 145 users need to be describable with a single AND-policy.

The resulting implication is that PAD-TFDAC-MACS benefits from large scale systems with a lot of users that share a lot of attributes. Small groups that are identified by peer-to-peer sharing suffer more from the additional overhead. Here PAD-TFDAC-MACS can only be beneficial if storage and not computing power is the bottleneck.

7.1 Future Work

As shown in this work, ABE has the potential to improve today's applied cryptography. However, several improvements can be conducted to make PAD-TFDAC-MACS better practically applicable.

Handle Attributes On Device Level

In this prototype, attributes are handled on user-level. Intuitively, that makes sense. Users are assigned to attributes and not something else. But a much better approach would be to handle attributes on device level. In that way, a user can define more fine-grained which device should be able to decrypt which data. For example, very sensitive information should not leave the company building and in turn not be decryptable by mobile devices.

Multiple Two-Factor Keys

Another missed opportunity to make the scheme more expressive was to be able to create different two-factor private keys. In the current implementation, two-factor private keys are identified and bound by the user ID who created the ciphertext. That behavior was defined by TFDAC-MACS and directly applied to PAD-TFDAC-MACS . Since there always can just be one two-factor key per user, the number of possible two-factor keys are limited to the universe of users.

A more scalable way of defining two-factor keys would be to let each user create as many two-factor private keys as he desires. In theory, this should be simply possible by providing each ciphertext a different two-factor private key on encryption. Users that want to decrypt this ciphertext extract from the ciphertext the ID and version of the used two-factor key, check whether they have a suitable secret key and decrypt the ciphertext normally.

AA As A Client-Service

To eliminate the need of having the AA always available it is possible to implement it as a client. In that way, it would reduce the security risk of being exposed to the outer world. Such a system can be designed if the central server is used as a message

broker. Since the threat model that the central server is untrusted but follows the protocol, each message that is sent to the central server needs to be authenticated by a signature.

Using the technique of *certificate pinning*, a set of trusted certificates can be installed at the client on bootstrapping it in a trusted environment. In that way, it is ensured that only authentic signatures are accepted. This procedure comes with additional overhead on revoking one of these certificates.

To prevent replay attacks clients (devices) can additionally also be bootstrapped with a random seed. This seed is only known to the client and the owning AA. Each seed for each device should be different. This seed can be feed into a pseudo-random function which produces a stream of random numbers. Each number will be included in the next message that will be exchanged and signed by the sender. In that way replay attacks of the central server can be mitigated, since the central server never knows which nonce will come next. The owning AA uses its known seeds and the public key of each device to distribute new seeds and onboard new AA that are trusted as well.

More Expressiv Fine-Grant Access Control

In this work, a very easy adaptation to DNF-access policies was given. Different access policies are built over the same file key so that if a user can decrypt one of the ciphertexts he can decrypt the file. PAD-TFDAC-MACS is able to support DNF-access formulas with the trade-off for performance and storage consumption.

A much better way would be to embed the access formula directly into the ciphertext as it is also done with AND-gates. TFDAC-MACS uses multiplicative terms that substitute each other if combined with the right secret keys to reveal the underlying secret. Other approaches in the field of MA-ABE support fine-grant access formulas already such as [30] [54] [22]. They utilize a technique called *Linear Secret Sharing Scheme* (LSSS, more information in).

Having fine-grained access control implemented in the system would greatly improve the value of such system. Arbitrary monotone-access structures would enable to include numerical attributes into the system. In addition, the number of file keys can be reduced to one file key regardless of the access policy. Here additional evaluation needs to be conducted to analyze whether the trade-off for ciphertext size is negligible.

Automatically Compose Access Policies

To be fully practically applicable an algorithm must be developed that automatically chooses the access policy with the least overhead, given a user group. For the normal user, only very simple access policies are sizable. Such as "Works in company X" or "Is full-time employed". But sometimes users want to share files with specific users. They face the problem of not knowing which attributes their communication partner own and who shares the same attributes and so will also be accidentally included in the group without them knowing. Most likely user will fall back to use explicit OR-policies to prevent this case from happening. This behavior would result in the creation of many worst-case access policies.

An algorithm can help here. Such an algorithm can analyzed the given user group and find the smallest intersecting subset of attributes that are needed to completely define this group and without including any unwanted members. If no such

excluding policy can be constructed the algorithm could either create a group defining attribute (same overhead as secure group communications) or fall back to the worst-case access policy (same overhead as the classical re-encryption scheme).

Hybrid Scheme

As evaluated in practical applicability Section 6.4, PAD-TFDAC-MACS would show a significant overhead on peer-to-peer file sharing. To overcome this obstacle a hybrid approach is proposed. For simple peer-to-peer file sharing, the classical encryption scheme of CloudRAID can be used since it shows a much better overhead in those scenarios than PAD-TFDAC-MACS . For each other groups with many members, PAD-TFDAC-MACS could be applied to reduce the number of file keys.

In that way, the advantages of both schemes can be combined to achieve good overall performance while still maintaining a scalable system regarding the number of file keys and ciphertext size.

7.2 Summary

In this thesis, a new MA-ABE scheme was presented that used TFDAC-MACS as a baseline. It was shown that such a secure cloud storage system utilizing PAD-TFDAC-MACS can exist in practice. The proposed scheme is able to reduce the number of file keys, depending on the chosen access policy, to a minimum. Such a system comes with the trade-off of performance and a bigger storage overhead per file key. Different evaluations are conducted to find the point where PAD-TFDAC-MACS scales better than the current implemented solution of a secure cloud storage system. This point can be found for the most encryption and member-join operations, while this system comes with an additional overhead on member-leave and decryption operations. The member-leave operation can be parallelized over different systems which leads to the conclusion that PAD-TFDAC-MACS is ready to be practically evaluated in a real-world system.

Here an open question remains: Since ABE schemes include the user level attributes into the cryptography, administrator and users need to adapt in the way they handle attributes and address other users. Administrators need to define attributes in the way that groups are usually describable by one attribute to use ABE efficiently. Users, on the other hand, need to understand that they address encrypted files to an attribute policy rather than to individuals. This last point can be obfuscated by an algorithm that automatically calculates the most efficient access policy. But the fact remains that an ABE system can only be used effectively and efficiently if all participants interact not individually, but on a descriptive attribute-level with each other.

Acronyms

2FA Two-Factor Authorization

AA Attribute Authority

AA-Admin Attribute-Authority Administrator

ABE Attribute-Based Encryption

AES Advanced Encryption Standarc

AID Authority Identifier

CP-ABE Ciphertext policy Attribute-Based Encryption

CS Central Server

CS-Admin Central-Authority Administrator

CP-ABE Decentralized Attribute-Based Encryption

DAC-MACS Effective Data Access Control for Multi-Authority Cloud Storage Systems

DH Diffie-Hellmann

DNF Disjunctive Normal Form

ECC Ellipic-Cruve Cryptography

GID Global identifier

GK Group Key

GKMP Group Key Management Protocol

GPP Global Public Paramter

HABE Hirachical Attribute-Based Encryption

IBE Identity-Based Encryption

KDC Key Destribution Center

KP-ABE Key policy Attribute-Based Encryption

LSSS Linear Secret Sharing Scheme

MA-ABE Multi-Authority Attribute-Based Encryption

MAACS Multi-Authority Access Control System

NEDAC-MACS New Extended Effective Data Access Control for Multi-Authority Cloud Storage Systems

OWFT One-Way Function Tree

PAD-TFDAC-MACS Practically Applied Distributed Two-Factor authentication Data Access Controll for Multi-Authority Cloud Storage systems

GK Public-Key Infrastructure

R-CP-ABE Revocable Ciphertext policy Attribute-Based Encryption

RSA Rivest-Shamir-Adleman (public/private key cryptosystem)

SGC Secure Group Communication

SHA Secure Hash Algorithm

TFDAC-MACS Two-Factor Effective Data Access Control for Multi-Authority Cloud Storage Systems

TU Technical University

UID User Identifier

Bibliography

[1] Joseph A Akinyele et al. "Securing electronic medical records using attribute-based encryption on mobile devices". In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM. 2011, pp. 75–86.

[2] Nathalie Baracaldo et al. "Reconciling end-to-end confidentiality and data reduction in cloud storage". In: *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*. ACM. 2014, pp. 21–32.

[3] John Bethencourt. "Intro into Bilinear Maps". https://people.csail.mit.edu/alinush/6.857-spring-2015/papers/bilinear-maps.pdf. 2015.

[4] John Bethencourt, Amit Sahai, and Brent Waters. "Ciphertext-policy attribute-based encryption". In: *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE. 2007, pp. 321–334.

[5] Dan Boneh and Matt Franklin. "Identity-based encryption from the Weil pairing". In: *Annual international cryptology conference*. Springer. 2001, pp. 213–229.

[6] Dan Boneh, Amit Sahai, and Brent Waters. "Functional encryption: Definitions and challenges". In: *Theory of Cryptography Conference*. Springer. 2011, pp. 253–273.

[7] Ran Canetti et al. "Multicast security: A taxonomy and some efficient constructions". In: *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE. 1999, pp. 708–716.

[8] Melissa Chase. "Multi-authority attribute based encryption". In: *Theory of Cryptography Conference*. Springer. 2007, pp. 515–534.

[9] Melissa Chase and Sherman SM Chow. "Improving privacy and security in multi-authority attribute-based encryption". In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 121–130.

[10] Cheng-Kang Chu et al. "Key-aggregate cryptosystem for scalable data sharing in cloud storage". In: *IEEE transactions on parallel and distributed systems* 25.2 (2014), pp. 468–477.

[11] Hui Cui and Robert H Deng. "Revocable and decentralized attribute-based encryption". In: *The Computer Journal* 59.8 (2016), pp. 1220–1235.

[12] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. eng. 2. ed., reprint. Cambdridge, 2005. ISBN: 0521784514.

[13] Angelo De Caro and Vincenzo Iovino. "jPBC: Java pairing based cryptography". In: *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*. IEEE, 2011, pp. 850–855. URL: http://gas.dia.unisa.it/projects/jpbc/.

[14] Pooja Dubey, Amit Saxena, and Manish Manoria. "A Survey on Various Attribute based Public Key Cryptography". In: *International Journal of Computer Science and Information Technologies* 6.4 (2015), pp. 3791–3794.

[15] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. "Pairings for cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121.

[16] Vipul Goyal et al. "Attribute-based encryption for fine-grained access control of encrypted data". In: *Proceedings of the 13th ACM conference on Computer and communications security*. Acm. 2006, pp. 89–98.

[17] Hendrik Graupner et al. "Secure access control for multi-cloud resources". In: *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*. IEEE. 2015, pp. 722–729.

[18] Hugh Harney and Carl Muckenhirn. *Group key management protocol (GKMP) architecture*. Tech. rep. SPARTA, Inc., 1997.

[19] Komal Kate and S. D. Potdukhe. "Data sharing in cloud storage with key-aggregate cryptosystem." In: *International JournalofEngineering Research and General Science* 2 (2014), pp. 882–886.

[20] Cheng-Chi Lee, Pei-Shan Chung, and Min-Shiang Hwang. "A Survey on Attribute-based Encryption Schemes of Access Control in Cloud Environments." In: *IJ Network Security* 15.4 (2013), pp. 231–240.

[21] Allison Lewko, Amit Sahai, and Brent Waters. "Revocation systems with very small private keys". In: *2010 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2010, pp. 273–285.

[22] Allison Lewko and Brent Waters. "Decentralizing attribute-based encryption". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2011, pp. 568–588.

[23] Keying Li. "Matrix access structure policy used in attribute-based proxy re-encryption". In: *arXiv preprint arXiv:1302.6428* (2013).

[24] Ming Li et al. "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption". In: *IEEE transactions on parallel and distributed systems* 24.1 (2013), pp. 131–143.

[25] Qi Li et al. "Secure, efficient and revocable multi-authority access control system in cloud storage". In: *Computers & Security* 59 (2016), pp. 45–59.

[26] Xiaoyu Li et al. "Two-factor data access control with efficient revocation for multi-authority cloud storage systems". In: *IEEE Access* 5 (2017), pp. 393–405.

[27] Xiaohui Liang et al. "Ciphertext policy attribute based encryption with efficient revocation". In: (2010).

[28] Qin Liu, Guojun Wang, and Jie Wu. "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment". In: *Information sciences* 258 (2014), pp. 355–370.

[29] Zhen Liu, Zhenfu Cao, and Duncan S Wong. "Efficient generation of linear secret sharing scheme matrices from threshold access trees". In: *Cryptology ePrint Archive: Listing* (2010).

[30] Zhen Liu and Duncan S Wong. "Practical attribute-based encryption: traitor tracing, revocation and large universe". In: *The Computer Journal* 59.7 (2016), pp. 983–1004.

[31] Victor S. Miller. "The Weil Pairing, and Its Efficient Calculation". In: *Journal of Cryptology* 17.4 (2004), pp. 235–261. ISSN: 1432-1378. DOI: 10.1007/s00145-004-0315-8. URL: https://doi.org/10.1007/s00145-004-0315-8.

[32] Veelasha Moonsamy and LM Batten. "Mitigating man-in-the-middle attacks on smartphones-a discussion of ssl pinning and dnssec". In: *Proceedings of the 12th Australian Information Security Management Conference*. Edith Cowan University. 2014, pp. 5–13.

[33] R Nagarajan and G Maria Priscilla. "An Overview of Data Access Control in Security for Multi authority Cloud Storage Systems". In: *IJSRCSEIT - International Journal of Scientific Research in Computer Science* 3.3 (2018), pp. 1901–1910.

[34] Rafail Ostrovsky, Amit Sahai, and Brent Waters. "Attribute-based Encryption with Non-monotonic Access Structures". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. Alexandria, Virginia, USA: ACM, 2007, pp. 195–203. ISBN: 978-1-59593-703-2. DOI: 10.1145/1315245.1315270. URL: http://doi.acm.org/10.1145/1315245.1315270.

[35] Sandro Rafaeli and David Hutchison. "A survey of key management for secure group communication". In: *ACM Computing Surveys (CSUR)* 35.3 (2003), pp. 309–329.

[36] Amit Sahai and Brent Waters. "Fuzzy identity-based encryption". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2005, pp. 457–473.

[37] Ravinderpal S Sandhu. "Cryptographic implementation of a tree hierarchy for access control". In: *Information Processing Letters* 27.2 (1988), pp. 95–98.

[38] Maxim Schnjakin and Christoph Meinel. "Taming the Uncertainty of Public Clouds". In: *International Journal of Cloud Computing (IJCC)* 1.2 (2013), pp. 28–47.

[39] Maxim Schnjakin, Tobias Metzke, and Christoph Meinel. "Applying erasure codes for fault tolerance in cloud-raid". In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. IEEE. 2013, pp. 66–75.

[40] Maxim Schnjakin et al. "Implementation of a secure and reliable storage above the untrusted clouds". In: *2013 8th International Conference on Computer Science & Education*. IEEE. 2013, pp. 347–353.

[41] Adi Shamir. "Identity-based cryptosystems and signature schemes". In: *Workshop on the theory and application of cryptographic techniques*. Springer. 1984, pp. 47–53.

[42] Alan T Sherman and David A McGrew. "Key establishment in large dynamic groups using one-way function trees". In: *IEEE transactions on Software Engineering* 29.5 (2003), pp. 444–458.

[43] Yanfeng Shi et al. "Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation". In: *Information Sciences* 295 (2015), pp. 221–231.

[44] Gurpreet Singh. "A study of encryption algorithms (RSA, DES, 3DES and AES) for information security". In: *International Journal of Computer Applications* 67.19 (2013).

[45] Michael Steiner, Gene Tsudik, and Michael Waidner. "Diffie-Hellman key distribution extended to group communication". In: *Proceedings of the 3rd ACM conference on Computer and communications security*. ACM. 1996, pp. 31–37.

[46] Muhammad IH Sukmana et al. “Redesign cloudraid for flexible and secure enterprise file sharing over public cloud storage”. In: *Proceedings of the 10th International Conference on Security of Information and Networks*. ACM. 2017, pp. 3–10.

[47] Muhammad I.H. Sukmana et al. “Secure and Scalable Multi-Company Management in Enterprise Cloud Storage Broker System”. unpublished paper, submitted to SecureComm at 16.04.2019. 2019.

[48] Dirk Thatmann, Philip Raschke, and Axel Küpper. “" Please, No More GUIs!": A User Study, Prototype Development and Evaluation on the Integration of Attribute-Based Encryption in a Hospital Environment”. In: *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*. Vol. 2. IEEE. 2016, pp. 496–502.

[49] Debby Wallner, Eric Harder, and Ryan Agee. *Key management for multicast: Issues and architectures*. Tech. rep. National Security Agency, 1999.

[50] Guojun Wang, Qin Liu, and Jie Wu. “Hierarchical Attribute-based Encryption for Fine-grained Access Control in Cloud Storage Services”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. Chicago, Illinois, USA: ACM, 2010, pp. 735–737. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866414. URL: http://doi.acm.org/10.1145/1866307.1866414.

[51] Guojun Wang et al. “Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers”. In: *computers & security* 30.5 (2011), pp. 320–331.

[52] Xianglong Wu, Rui Jiang, and Bharat Bhargava. “On the security of data access control for multiauthority cloud storage systems”. In: *IEEE Transactions on Services Computing* 10.2 (2017), pp. 258–272.

[53] Shota Yamada et al. “A Framework and Compact Constructions for Non-monotonic Attribute-Based Encryption”. In: *Public-Key Cryptography – PKC 2014*. Ed. by Hugo Krawczyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 275–292. ISBN: 978-3-642-54631-0.

[54] Kan Yang et al. “DAC-MACS: Effective data access control for multiauthority cloud storage systems”. In: *IEEE Transactions on Information Forensics and Security* 8.11 (2013), pp. 1790–1801.

[55] Sebastian Zickau et al. “Applied attribute-based encryption schemes”. In: *19th International ICIN Conference-Innovations in Clouds, Internet and Networks-March*. 2016, pp. 1–3.