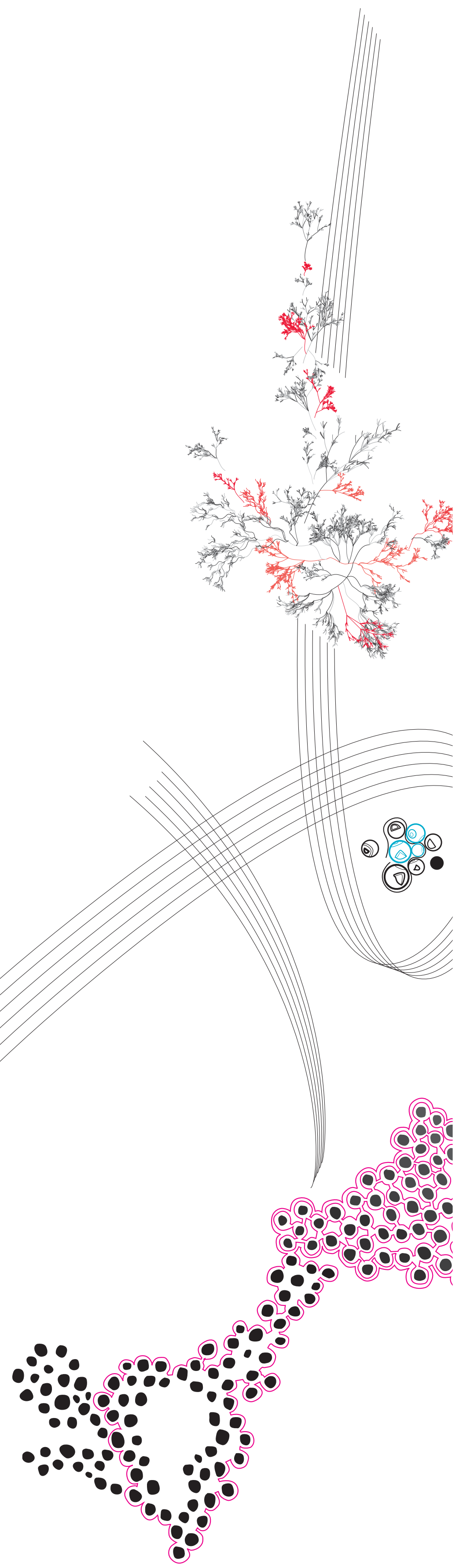
MSc Computer Science
Final Project



Formal Verification of Lightweight Decentralized Attribute-based Encryption

Janneke van Oosterhout

Supervisor: Florian Hahn

June, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Acknowledgements

I would like to acknowledge and give my warmest thanks to my supervisors: Mohammed B.M. Kamel (Eötvös Loránd University) and Sander Dorigo (Fox-IT). Mohammed B.M. Kamel guided me through this process and advised me through all stages of my thesis. His lectures on cryptographic protocols enthused me to learn more about the topic and write my thesis about Attribute-based Encryption (ABE) and formal verification. I would also like to thank Fox-IT and especially Sander Dorigo for giving me the possibility to do my internship and write my thesis at Fox-IT. I had an enjoyable and educational time at Fox, special thanks also to all my colleagues.

Thanks should go to my proofreaders: some colleagues at Fox-IT and Javier Perez y Perez. Your feedback was meaningful and helped me finalize my thesis. I would also like to recognize Florian Hahn for checking my thesis on behalf of the University of Twente.

Lastly, I would like to mention my family, Javier, and friends. Thanks for supporting me through this time!

Abstract

Attribute-based Encryption (ABE) is a public key cryptography scheme that provides one-to-many encryption. Decentralized Attribute-based Encryption (DABE) is a recent extension of ABE that does not have a single central authority (CA). DABE contains some heavy computations in the encryption algorithm, which prevents DABE from being implemented in many resource-constrained devices smoothly. Lightweight DABE versions are designed to perform efficiently on resource-constrained devices. Outsourcing (Outsourced Decentralized Attribute-based Encryption (ODABE)) is an approach for lightweight encryption in DABE. Until now, there did not exist an outsourcing decryption scheme.

Formal verification can be used to show that a cryptographic protocol meets its security properties. Formal verification can be done either by hand or by using an automated tool. Automated formal verification is a kind of Computer-aided Cryptography (CAC) and performs machine-checkable approaches to test the security of a protocol. There are numerous tools that perform automated verification, TAMARIN is such an automated formal verification tool. To formally verify ODABE TAMARIN is used. Security properties are modelled and it is proven that they hold in the model.

This thesis has two goals:

- To design and analyse an outsourcing decryption scheme.
- To formally verify the encryption and the proposed decryption ODABE scheme.

Acronyms

AA attribute authority

ABE Attribute-based Encryption

AC associative-commutative

CA central authority

CAC Computer-aided Cryptography

CP-ABE Ciphertext-Policy Attribute-based Encryption

CT ciphertext

DABE Decentralized Attribute-based Encryption

ECC elliptic curve cryptography

Fuzzy-IBE Fuzzy Identity based Encryption

GID Global Identifier

IBE Identity based Encryption

IoT Internet of Things

KP-ABE Key-Policy Attribute-based Encryption

LSSS Linear Secret Sharing Scheme

MK master key

ODABE Outsourced Decentralized Attribute-based Encryption

PK public key

SDABE Secret Sharing for Lightweight Decentralized Attribute-based Encryption

SK secret key

Contents

1 Introduction **1**

1.1 Overview . . . 1

1.2 Thesis Objectives . . . 3

1.3 Contribution . . . 4

1.4 Organization of the Thesis . . . 4

2 Related Work **5**

2.1 Formal Verification . . . 5

2.2 Attribute-based Encryption . . . 6

2.3 Decentralized Attribute-based Encryption . . . 8

2.4 Lightweight Decentralized Attribute-based Encryption . . . 9

3 Preliminaries **12**

3.1 Secret Sharing . . . 12

3.2 Formal Verification . . . 13

3.3 TAMARIN . . . 13

3.4 Access Policy . . . 14

3.5 Linear Secret Sharing Scheme . . . 15

3.6 Bilinear Map . . . 17

3.7 Attribute-based Encryption . . . 17

3.8 Decentralized Attribute-based Encryption . . . 18

3.9 Lightweight Decentralized Attribute-based Encryption . . . 20

4 Proposed Outsourced Decryption **23**

4.1 Decryption DABE . . . 23

4.1.1 Possible Improvements . . . 23

4.2 Security Definition . . . 24

4.3 Proposed Decryption ODABE . . . 24

4.3.1 Analysis . . . 27

5 Formal Verification **32**

5.1 ABE Model in TAMARIN . . . 32

5.2 DABE Model in TAMARIN . . . 35

5.3 ODABE Model in TAMARIN 38
5.4 ODABE Decryption in TAMARIN 41

6 Conclusion 43
6.1 Summary of the Thesis 43
6.2 Conclusion 43
6.3 Future Research Directions 45

Bibliography 46

A Lemmas TAMARIN models 52
A.1 ABE 52
A.2 DABE 53
A.3 ODABE 57

B Labels TAMARIN models 61
B.1 Facts 61
B.2 Action facts 62
B.3 Trace restrictions 65

List of Figures 66

List of Tables 67

List of Algorithms and Protocols 68

List of Codes 69

Chapter 1

Introduction

This chapter provides an overview of the problem domain, discussing Attribute-based Encryption (ABE) and formal verification. Additionally, the objectives and contributions are discussed. Lastly, the organization of this thesis is described.

1.1 Overview

Most encryption schemes perform one-to-one encryption. In public key cryptography, a user Alice encrypts a message for the receiver Bob using his public key such that nobody else is able to decrypt the message.

However, in many cases, it can be more convenient to encrypt one message for multiple users to decrypt. The reason for this can be that multiple users need to decrypt the message or it is not known yet who needs to decrypt. An example of a case where this can be used is in a municipality. Assume the municipality wants to send an encrypted message to all its residents. Encrypting it individually with their public key is not efficient, instead, they want to send one encrypted message to everyone. Moreover, new residents also need to be able to decrypt a message sent earlier.

ABE provides a one-to-many scheme where the encryptor defines an access policy over attributes and encrypts the message with that policy. Decryptors who possess the attributes that satisfy the access policy are able to decrypt. An attribute is expressed as a string and describes part of the identity of a user [1]. Examples of attributes are the city you live in, your job, study, gender, etc.. The ABE scheme relies on a central authority (CA) that owns a public and private key. The encryptor needs the public key of the CA and an access policy to encrypt a message. A user who wants to decrypt needs to be issued a secret key from the CA. This secret key is associated with a set of attributes that the user possesses. A user can decrypt a ciphertext if his secret key is associated with a set of attributes that satisfies the access policy ascribed to the ciphertext [2].

However, a single CA has a few drawbacks, which might result in potential problems. First of all, the security of the user is at stake since the CA can be malicious. Secondly, the problem of bottleneck can occur when no secret keys can be provided to the users when the CA is malfunctioning [3]. A solution to overcome these problems is to use multiple attribute authorities (AAs).

Decentralized Attribute-based Encryption (DABE) is a version of ABE in which no CA is involved. There are multiple AAs who all contain their own attributes and only distribute secret keys for these attributes. A secret key is only associated with one attribute. In DABE, secret keys can be established without communication between the AAs. To decrypt, a user needs several secret keys that together identify a set of attributes that satisfy the access policy ascribed to the ciphertext. The main challenge in DABE is to make it collusion-resistant; users should not be able to collude together to decrypt a message if they cannot decrypt it on their own. In the DABE decryption process the user's key components are tied together, to make it collusion-resistant [4].

Compared to other public key cryptography protocols, ABE and DABE contain more expensive computations such as exponentiations and multiplications with pairings. Resource-constrained devices in the Internet of Things (IoT) environment cannot perform heavy computations efficiently. Therefore, DABE is not suitable for resource-constrained devices [5].

To use DABE efficiently in the IoT environment, a lightweight DABE variant would have to be used. There are several possibilities to make DABE lightweight, such as outsourcing and secret sharing. Outsourced Decentralized Attribute-based Encryption (ODABE) is a scheme that outsources heavy computations to a computational node. Until now, only an ODABE encryption scheme has been proposed [5]. In the encryption scheme, the encryptor prepares the encryption, sends values to the computational node, which performs the pre-encryption, and the encryptor finalizes the encryption. Since there does not exist an ODABE decryption scheme that outsources to a single computational node, this thesis will contain a proposed decryption scheme that uses outsourcing.

Formal verification is a way to prove whether a cryptographic protocol is secure. This can either be done by hand or digitally. Computer-aided Cryptography (CAC) is the area of research that uses formal, automated checking processes to test the security of a cryptographic protocol [6]. TAMARIN is a formal verification tool released in 2012 and TAMARIN will be used in this thesis. ABE is formally verified using ProVerif, another formal verification tool [7]. The other protocols discussed before, DABE and ODABE, have not been formally verified yet. Before using them in the real world, it is important to verify these protocols. Otherwise, one might wonder, whether this protocol really satisfies the security properties.

1.2 Thesis Objectives

The aim of this thesis is to formally verify an existing lightweight decentralized ABE scheme (ODABE), design an ODABE decryption scheme and formally verify the decryption scheme.

Formal verification The first objective of this thesis is the formal verification of lightweight decentralized ABE (ODABE). The following research questions have to be answered to fulfill this objective:

- How can the security of the existing ODABE approach be formally verified?
- How can the security of the proposed lightweight decryption in DABE be formally verified?

The first step to answer the research questions was to fully understand the ABE, DABE and ODABE protocol. To gain this understanding, several papers on variations of ABE and formal verification were studied in detail. For formal verification of the security of the discussed protocols, the tool Tamarin has been used. Tamarin is well established in the academic community, which is shown by the fact that it has been used to verify for example TLS 1.3, EMV, and 5G-AKA. Hence, the next important step was to understand the workings of this tool. Finally, the models could be implemented in Tamarin. The implementation was performed in three steps: first the ABE scheme was implemented, and this scheme was extended to DABE which was again extended to ODABE. To answer the second research question, the lightweight decryption scheme first had to be designed.

Decryption scheme Another objective of this thesis is to design an outsourcing DABE decryption scheme. The research questions that will be answered are:

- What are the heavy computation components of the decryption algorithm in DABE?
- How can the outsourcing approach in ODABE be implemented in the decryption algorithm of DABE?

After formally proving the DABE and ODABE protocols in Tamarin, the decryption algorithm in DABE was studied and analysed in detail. The heavy computations in the decryption algorithm needed to be identified and the outsourcing approach in ODABE was transformed into an outsourcing decryption scheme. The proposed decryption scheme was analysed on correctness and implemented in Python to test its performance. Finally, the Tamarin protocol verifier has been utilized to formally prove the security of the proposed lightweight decryption scheme.

1.3 Contribution

This section describes the two main contributions: formal verification of ODABE and the outsourcing decryption scheme.

Formal verification Based on the papers written about ABE, DABE, and ODABE, the protocols were formally interpreted and modelled in TAMARIN. Moreover, the papers write about security goals that these protocols should satisfy, which were implemented in the TAMARIN model. Besides these properties, some extra security properties and sanity checks were added. Formally verifying ODABE was the goal, but to do so first ABE needed to be implemented in TAMARIN. The ABE scheme could be extended to the DABE which again could be extended to formally verify ODABE.

Decryption scheme Based on the paper on ODABE [5], the decryption scheme was designed. The first step was to identify the heavy computational parts of the decryption scheme in DABE. The outsourcing approach discussed in the paper [5] was analysed. Combining these two findings, the proposed outsourcing decryption scheme was designed. Additionally, its correctness and performance were discussed. The performance was measured by implementing the proposed decryption scheme in Python and testing this implementation.

1.4 Organization of the Thesis

In Chapter 2, the related work regarding formal verification and different versions of ABE are discussed. Chapter 3 contains preliminaries such as TAMARIN, access policies, and algorithms of the DABE, ODABE schemes. Next, in Chapter 4 the proposed ODABE decryption scheme is described and analysed. In Chapter 5, the formal verification of the schemes in TAMARIN is discussed. Chapter 6 is devoted to the conclusion. At the end of the document, appendices with a more detailed description of the components of the TAMARIN model are added.

Chapter 2

Related Work

This chapter describes the related work regarding formal verification and variants of Attribute-based Encryption. The variants that will be discussed are Attribute-based Encryption (ABE), lightweight ABE, Decentralized Attribute-based Encryption (DABE), and lightweight DABE. Table 2.1 (on page 11) shows an overview of the papers discussed in this section.

2.1 Formal Verification

TAMARIN was first released in 2012 and has already been used to formally verify several protocols, such as TLS 1.3 [8], Identity based Encryption (IBE) [9], the EMV protocol [10], and the 5G authentication key exchange protocol [11]. All papers using TAMARIN are listed on the TAMARIN website[1]. Currently, there is no research paper that models and verifies Decentralized Attribute-based Encryption (DABE) or Outsourced Decentralized Attribute-based Encryption (ODABE) in TAMARIN.

Next to TAMARIN, there are more formal verification tools [6]. Bat-Erdene et al. [12] and Rajeb et al. [13] both modeled ABE using ProVerif [7], an automatic cryptographic protocol verifier. In 2014, it has already been stated that more attention needs to be paid to automated verification to prove that a protocol meets the security goals [14]. However, a paper that models DABE, or ODABE using any formal verification tool is still missing.

[1]`https://tamarin-prover.github.io/`

2.2 Attribute-based Encryption

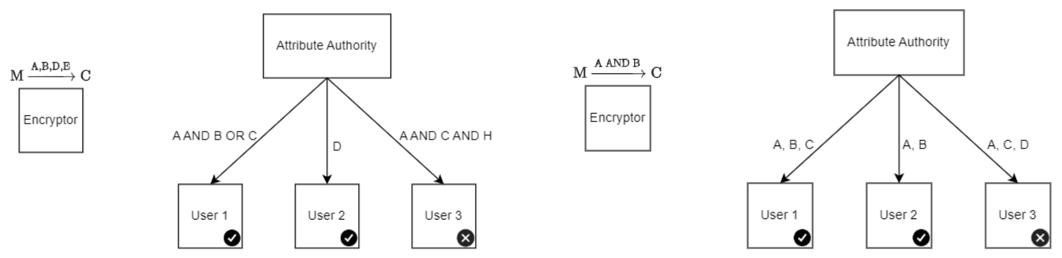
ABE is a form of fine-grained access control that facilitates granting access rights to users [15]. Prior to ABE, there were other systems that tried to manage access control of encrypted data by using secret sharing schemes in combination with IBE. However, these schemes were not collusion-resistant, so parties were able to collude together to gain access [2].

In ABE, the ciphertexts are not encrypted for one particular user. Instead, the ciphertexts and users' private keys are associated with either a set of attributes or an access policy. This enables users to decrypt a message when the attributes and access policy match [2]. This construction allows the ABE to provide one-to-many encryption [3].

ABE was first introduced by Sahai and Waters [1], where they described a Fuzzy Identity based Encryption (Fuzzy-IBE) system. Fuzzy-IBE is a type of IBE that views identities as a set of descriptive attributes. Sahai and Waters proposed that the user's keys and ciphertexts are labelled with descriptive attributes, and a user can only decrypt the ciphertext if the attributes match. The two main applications of Fuzzy-IBE are biometric identities and ABE.

Types of ABE There are two types of ABE: Key-Policy Attribute-based Encryption (KP-ABE) and Ciphertext-Policy Attribute-based Encryption (CP-ABE). Goyal et al. [15] defined KP-ABE based on the construction of Fuzzy-IBE. In KP-ABE, the keys of the users are associated with the access policy and the ciphertexts with sets of attributes. This results in the encryptor not having full control on who can decrypt his ciphertext. Take for example a ciphertext with two associated attributes 'Dutch' and 'Student'. Everyone with an access policy that matches the ciphertext can decrypt, such as 'Dutch AND Student', 'Dutch', 'Student', 'Dutch OR Student' etc. The encryptor can thus not restrict the ciphertext to users who are a 'Dutch Student'. Hence, the encryptor does not have the possibility to fully manage the access. In Figure 2.1(a) an example of KP-ABE is shown. The encryptor associates attributes A, B, D, E with the ciphertext. Three users obtain keys from the attribute authority, their keys contain an access policy. Based on the access policies, users 1 and 2 are able to decrypt the ciphertext since the attributes associated with the ciphertext satisfy the access policy. User 3 cannot decrypt.

Based on the KP-ABE construction, Bethencourt, Sahai, and Waters [2] proposed CP-ABE. In CP-ABE, the ciphertexts are associated with access policies and keys with sets of attributes. The encryptor is able to decide who has access to the data, by defining an access policy. Only the users who have a private key that satisfies the access policy associated with the ciphertext can decrypt. A challenge here is



(a) KP-ABE (b) CP-ABE

Figure 2.1: Example of KP-ABE and CP-ABE

to make it collusion-resistant: users should not be able to collude together to decrypt a message. When users decide to collude, they should only be able to decrypt the message when at least one of the users can decrypt it on their own. Figure 2.1(b) shows an example of CP-ABE. The encryptor associates access policy A AND B with the ciphertext. The users obtain a secret key associated with a set of attributes from the attribute authority. The sets of attributes of users 1 and 2 satisfy the access policy, so they are able to decrypt the ciphertext. User 3 does not possess attribute B and is thus not able to decrypt.

Lightweight Attribute-based Encryption The Internet of Things (IoT) describes physical objects (e.g. sensors, actuators) that are connected with other devices over the Internet. Currently, IoT devices are also used to exchange data and the devices may thus contain sensitive information. Therefore, data privacy is crucial in the IoT environment. Implementing a data access control solution, like ABE, can offer the required flexibility and fine-grained access control to manage the protection of the data [16]. However, the ABE schemes described earlier are based on heavy computations, which makes these schemes unsuitable for resource-constrained IoT devices, that have resource limitations in terms of CPU, memory, and battery life [17]. The most costly operations in the CP-ABE schemes are the encryption and the secret key generation algorithms [18]. These algorithms are based on expensive bilinear pairing. Yao et al. [17] introduced a lightweight no-pairing ABE scheme based on elliptic curve cryptography (ECC). ECC is easy to be realized on hardware or a chip. Hence, they replaced the bilinear pairing operations with point scalar multiplications on the elliptic curve. However, Tan et al. [19] stated that the scheme presented by Yao et al. [17] causes a security problem because the decryption keys are the same when the access policy contains a single OR gate. Tan et al. solved the security problem and introduced a lightweight KP-ABE scheme designed for IoT devices. Oualha and Nguyen [16] extended the CP-ABE scheme using effective pre-computation techniques, such that it can also be used on resource-constrained IoT devices.

Another technique that is used to make CP-ABE feasible in an IoT environment, is to delegate the costly operations to a set of assisting unconstrained nodes. Touati et al. [18] introduced this concept to delegate the costly exponentiations in the encryption algorithm to trusted neighbouring nodes. A trusted node is a node that behaves the way it is intended, so it follows the protocol and it does not try to learn more than needed. The costly operations in the key generation algorithm are performed by an attribute authority (AA) with enough computing power to perform the operations without a need for operation delegation. To delegate the computations in Touati et al.'s scheme, it is assumed that each resource-constrained IoT device has at least two trusted unconstrained devices in its neighbourhood and shares pairwise keys with them. Additionally, Nguyen et al. [20] proposed an Outsourcing mechanism for the Encryption of Ciphertext-Policy ABE (OEABE). They want to reduce the costly operations by securely delegating the computations of the encryption algorithm (CP-ABE) to a semi-trusted party. A semi-trusted party follows the defined protocol but it will try to learn as much as possible. Tian et al. [21] proposed an ABE-FPP scheme, a lightweight attribute-based access control scheme with full privacy protection. To achieve lightweight computations, they perform heavy encryption computations offline, which reduces the computation burdens. Moreover, their decryption computations are outsourced.

2.3 Decentralized Attribute-based Encryption

All ABE schemes described earlier rely on a single authority. This means that there is one trusted authority that manages all attributes. However, in multiple scenarios, there is no single authority that maintains and controls all the attributes of the access policy. Chase [22] was the first one to propose an ABE construction in which many different authorities operate simultaneously. Each authority is responsible for a specific set of attributes and the authority hands out secret keys for this set. To achieve this construction two main techniques are used. First, every user gets a Global Identifier (GID), satisfying two properties: no user can claim another user's identifier, and all authorities can verify a user's identifier. Second, there still is a central authority (CA). Each user will send his GID to the CA and receive a corresponding key. A requirement is that the CA needs to be trusted since it holds the master secret for the system. Since the user's secret keys are independent, the scheme is collusion-resistant.

Besides Chase [22], Müller et al. [23] also proposed a multi-authority ABE scheme. However, both the schemes introduced by Chase and Müller et al. still need a CA. Therefore, these schemes still have the key escrow problem, which means there is only one CA who generates and thus knows the secret keys of all users. The key escrow problem causes two risks for the system. Most importantly, the security of the users is at stake since the CA can be malicious. Secondly, the problem of bottleneck can occur when no secret keys can be provided because of a malfunctioning authority [3].

Lin et al. [24] were the first to propose a scheme where the CA is removed. They extended the multi-authority ABE scheme proposed by Chase [22] and Fuzzy-IBE introduced by Sahai and Waters [1]. This scheme presented a threshold multi-authority fuzzy identity-based encryption scheme (MA-FIBE). Their construction is only m-resilient, where m is the number of secret keys that each authority obtains. The security is only guaranteed against a maximum of m colluding users, more than m users can work together to decrypt a ciphertext. Chase and Chow [25] also proposed a multi-authority ABE scheme where the CA is removed. They suggested distributing the functionality of the CA over all AAs. In comparison with the scheme of Lin et al. [24], the construction of Chase and Chow [25] is secure no matter how many users collude. Chase and Chow focus especially on the privacy of the users. Since every user has a GID which needs to be presented when they request a secret attribute key, colluding authorities are able to construct a set of attributes belonging to a specific user. Chase and Chow are using anonymous credentials techniques to guarantee the privacy of the users.

Both the schemes of Lin et al. [24] and Chase and Chow [25] are KP-ABE schemes. Lewko and Waters [4] proposed the first multi-authority CP-ABE system that is proven secure in the random oracle model and does not require any CA. Therefore, they avoid having full trust in one single authority and they solve the bottleneck problem. The only technical difficulty they face is to make the system collusion-resistant. They achieved this by tying the user's key components together. Liu et al. [26] solved two problems of the Lewko and Waters scheme [4]; the construction of a fully secure multi-authority CP-ABE in the standard model, and a CP-ABE scheme which can completely prevent individual authorities to decrypt a message. However, their scheme again uses CAs to distribute keys to the users.

2.4 Lightweight Decentralized Attribute-based Encryption

The effort to propose a lightweight and Decentralized Attribute-based Encryption scheme is a relatively new path of research. One of the methods to transform an ABE scheme to a lightweight scheme is to make sure the heavy computations are not performed by the resource-constrained device in the IoT environment itself. Outsourcing, introduced by Touati et al. [18], is a technique used to delegate heavy computations. Sun et al. [27] presented an outsourced decentralized multi-authority attribute-based signature (ODMA-ABS) scheme. However, as the name indicates, they did not propose the outsourcing decentralized technique for ABE but for Attribute-based Signature (ABS). Chow [28] proposed a framework that outsources the decryption of a multi-authority ABE scheme. A decryption mediator performed by a cloud server performs the heavy computations of the decryption algorithm. Additionally, Tu et al. [29] proposed a multi-authority ABE that outsources the encryption and decryption tasks to a fog server and a cloud service provider. According to Tu et al.: "Fog computing is a revolutionary technology for the next generation to bridge the gap between cloud data centres and end-users. [..] Fog servers are distributed on the network edge and less expensive as compared to cloud servers." [29]. Kamel et al. [5] proposed an Outsourced Decentralized Attribute-based Encryption (ODABE) scheme to outsource the heavy computations during encryption. In comparison with Tu et al. [29], Kamel et al. [5] outsource the heavy computations to a single external node, whereas Tu et al. use several nodes to perform the heavy computations. Therefore, the encryption performed in ODABE contains fewer algorithms and is more efficient. Shao et al. [30] also designed a scheme for mobile cloud computing that outsources encryption and decryption to a single node but an extra node is involved to store the encrypted messages. Moreover, Shao et al.'s scheme contains an encryption proxy, decryption proxy, and the cloud storage server whereas Kamel et al.'s scheme only needs one node for these tasks.

Another technique to create a lightweight DABE is by using a secret sharing scheme. Kamel et al. [31] proposed the Secret Sharing for Decentralized Attribute-based Encryption (Secret Sharing for Lightweight Decentralized Attribute-based Encryption (SDABE)) scheme. By distributing the heavy computations in the encryption algorithm using secret sharing, the user only needs to perform lightweight computations.

Table 2.1 shows the properties of the main proposed ABE schemes. The following properties are checked:

- Decentralized: the scheme does not have a single CA but instead contains multiple authorities.
- Lightweight Encryption: the scheme proposed contains a lightweight encryption algorithm.
- Lightweight Decryption: the scheme proposed contains a lightweight decryption algorithm.
- KP-ABE/CP-ABE: whether the scheme is based on KP-ABE or CP-ABE.

- Single node outsourcing: if the scheme makes use of outsourcing heavy computations, only a single node is involved in the outsourcing process.

As can be seen in Table 2.1, a lightweight DABE scheme does not yet exist that outsources encryption and decryption to a single node. The ODABE scheme proposed by Kamel et al. [5] seems to be the closest to the desired outsourcing approach. However, it covers only the encryption algorithm. Therefore, in this thesis, the ODABE scheme will be extended with an outsourcing decryption scheme.

Paper	**Decentralized**	**Lightweight Encryption**	**Lightweight Decryption**	**KP-ABE/ CP-ABE**	**Single node (outsourcing) involved**
Goyal et al. [15]	✗	✗	✗	KP-ABE	-
Bethencourt et al. [2]	✗	✗	✗	CP-ABE	-
Yao et al. [17]	✗	✓	✓	KP-ABE	-
Tan et al. [19]	✗	✓	✓	KP-ABE	-
Oualha and Nguyen [16]	✗	✓	✗	CP-ABE	-
Nguyen et al. [20]	✗	✓	✗	CP-ABE	✓
Touati et al. [18]	✗	✓	✗	CP-ABE	✗
Tian et al. [21]	✗	✓	✓	CP-ABE	✓
Chase and Chow [25]	✓	✗	✗	KP-ABE	-
Lin et al. [24]	✓	✗	✗	KP-ABE	-
Rouselakis and Waters [32]	✓	✗	✗	CP-ABE	-
Liu et al. [26]	✓	✗	✗	CP-ABE	-
Lewko and Waters [4]	✓	✗	✗	CP-ABE	-
Chow [28]	✓	✗	✓	CP-ABE	✓
Tu et al. [29]	✓	✓	✓	CP-ABE	✗
Shao et al. [30]	✓	✓	✓	CP-ABE	✗
Kamel et al. [5]	✓	✓	✗	CP-ABE	✓
Kamel et al. [31]	✓	✓	✗	CP-ABE	✗
This thesis, extension to Kamel et al. [5]	✓	✓	✓	CP-ABE	✓

Table 2.1: Supported properties ABE schemes

Chapter 3

Preliminaries

This chapter describes the background information for understanding the remainder of this thesis.

3.1 Secret Sharing

Secret sharing refers to dividing a secret s into n pieces, called the shares, in such a way that no information about s is learned by an unauthorized set of parties [33]. Secret sharing schemes have two properties: [34]

- Correctness: the secret can be reconstructed by any authorized set of parties.
- Perfect privacy: any unauthorized set of parties cannot learn anything about the secret.

Shamir [35] introduced the concept of (n, k) threshold scheme, where $k \leq n$, which means that k-out-of-n parties can reconstruct the secret. Shamir's scheme is based on polynomial interpolation.

Shamir's scheme actually has an access structure where any subset of size greater or equal to k is able to reconstruct the secret. Ito, Saito, and Nishizeki [36] proposed a secret sharing scheme realizing any given access structure. A set of parties that satisfies the access structure is able to reconstruct the secret. An example of an access structure can be that two parties p_1 and p_2 are both needed to reconstruct a secret s $(0 \leq s < m)$, then both parties get a share s_1 and s_2 respectively. s_1 and s_2 are computed such that $s = (s_1 + s_2) \bmod m$. Now p_1 as well as p_2 needs the share of the other party to reconstruct s [37].

3.2 Formal Verification

Computer-aided Cryptography (CAC) is a research area used to design, analyse and implement cryptography. It makes use of formal, automated approaches. There are many tools available but they address different parts of the problem space. CAC operates on three levels: design-level, deployment-level, and implementation-level. The focus of this thesis will be on the design-level since the tool used, Tamarin, is a design-level tool [6].

Design-level security At the design-level, CAC can manage the complexity of security proofs and detect possible attacks. There are two sorts of design-level security: symbolic security and computational security. In the symbolic model, messages are represented as terms and cryptographic primitives are black-box functions of terms. An adversary in the symbolic model can only use specified primitives and equational theories. Moreover, the symbolic model contains trace and equivalence properties. Trace properties state that a bad event never occurs on the trace. Equivalence properties state that an adversary cannot distinguish between two protocols [6].

In the computational model, messages are bitstrings and the cryptographic primitives are probabilistic. If part of the bitstring is known in a computational model, less computational resources are needed to decrypt a ciphertext. The security properties in the computational model can be divided into game-based and simulation-based properties. Game-based properties model a probabilistic game between an adversary and a challenger, and a goal that the adversary needs to achieve to win the game and break the scheme. Simulation-based properties model two probabilistic games, the real game that runs under analysis and the ideal game that runs an ideal functionality [6].

Tamarin is an unbounded tool in the symbolic model that uses trace properties. Unbounded here means that the absence of attacks can be proven [6].

3.3 Tamarin

Tamarin is an open-source tool used to verify cryptographic protocols. Tamarin supports falsification as well as unbounded verification in the symbolic model [38]. Moreover, Tamarin supports verification of trace properties and verification of protocols with the global mutable state [6]. The input to Tamarin is equational theories that represent the protocol messages, a model that specifies the cryptographic protocol and the capabilities of the adversary, and some security properties [39].

Users can define equational theories with associative-commutative (AC) axioms. The equational theories can be used to model the properties of functions. An example of an equational theory is `dec(enc(m,k),k) = m` which states that decrypting the ciphertext $enc(m, k)$ with key k gives message m [6].

A protocol in Tamarin is defined using multiset rewriting rules. The rules contain multisets of facts, that have terms as arguments. The facts model the system's state [9]. Facts can be either linear, which means that they can only be executed once, or persistent, which means they can be executed multiple times. Persistent facts are indicated with an exclamation mark '!' in front of the fact [8]. The rules define a labelled transition system. A Tamarin rule consists of three parts and is written as: [9]

$$[premises] - [actions] \rightarrow [conclusions]$$

The left part consists of the premises. A rule can only be executed when all the premises are available in the current state. The right-part are the conclusions. When a rule is executed, it will consume the premises, and produce the conclusions. The actions specify events in every trace which are used in the security properties. Actions are optional when defining a rule [8].
An adversary in Tamarin is able to control the entire network, the adversary can thus delete, intercept, modify, delay, inject, and create messages. However, the adversary cannot forge signatures or decrypt messages without knowing the entire key [9].
In Tamarin one can also define trace restrictions. A restriction is a logical formula that constrains the application of protocol rules [40].

The security properties, also called lemmas, in Tamarin are written in first-order logic and modelled as trace properties. They are checked against the traces of the label transition system. Tamarin tries to find a counterexample for the security properties. If Tamarin finds one, the security property does not hold since it can be attacked by an adversary. The interactive Tamarin mode can be used to see how the adversary is able to attack. When no counterexample exists, the security property holds and the proof is shown in the interactive mode [41].

3.4 Access Policy

An access policy is used to make sure that only people with the right set of attributes are able to decrypt a message. The access policy is a boolean formula consisting only of AND and OR operators [2].

Access tree In ABE, an access policy has to be turned into an access tree [2]. In an access tree, the leaves describe the attributes of the access policy. The nodes of the access tree are threshold gates, where the AND gates can be described as n-of-n threshold gates and the OR gates as 1-of-n threshold gates [15].

To check if a set of attributes y satisfies the access policy, the subtrees are checked recursively. Let's take $child_x$ as the number of children of node x and t as the threshold value, $0 < t \leq child_x$. If the threshold gate is an OR gate then $t = 1$, for an AND gate $t = child_x$. For every node x, the following must be checked:

- If x is a non-leaf node, check all children of node x. Return 1 if and only if at least t children return 1. Otherwise, return 0 and stop checking the tree.
- If x is a leaf node, check if the attribute denoted with node x is in y. If this is the case, return 1, otherwise, return 0.

If 0 has been returned when x is a non-leaf node, it can be concluded that y does not satisfy the access tree. When all nodes are checked and all non-leaf nodes returned 1, y satisfies the access tree [15].

3.5 Linear Secret Sharing Scheme

A Linear Secret Sharing Scheme (LSSS) is a secret sharing scheme where the relations between all shares are linear. A LSSS matrix $\mathcal{M}$ is used to specify the access policy Γ of the ciphertext. Matrix $\mathcal{M}(\Gamma)$ contains r rows, where r is the number of leaves in the access tree. The number of columns, c, is the depth of the access tree [42].

Definition 1. (Linear Secret Sharing Scheme (LSSS) [43]) A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear if

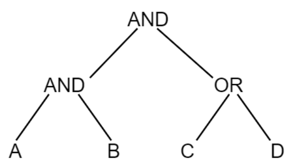
1. the shares for each party form a vector over $\mathbb{Z}_p$;
2. there exists a matrix $\mathcal{M}$ with r rows and c columns. For all $k = 1, .., r$, the k^{th} row $\mathcal{M}_k$ of $\mathcal{M}$ belongs to party ρ. Given that the column vector $v = (s, r_2, ..., r_c)$ of matrix $\mathcal{M}$ contains the shared secret s and randomly chosen $r_2, .., r_c$, we have a vector γ of r shares of the secret s according to matrix Π. Share γ_k is the share belonging to party k.

LSSS conversion To convert an access tree to a LSSS, algorithm 1 can be used.

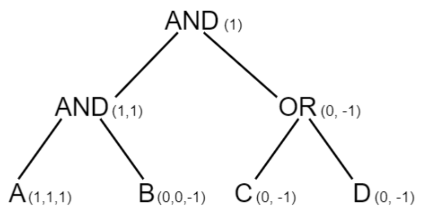
Algorithm 1 LSSS Conversion algorithm [42]

```
Initialize counter $c$ to 1
Label the root node of the tree with vector (1)
while There are still nodes without a vector do
    if Node is a parent node and an OR gate labelled with vector $v$ then
        Its children are labelled by $v$
    else if Node is a parent node and an AND gate labelled with vector $v$ then
        Pad $v$ with 0's at the end to make it of length $c$ (if necessary)
        Label one of its children with vector $v|1$
        Label the other child with vector $(0,..,0)|-1$, the zero vector has length $c$
        Increment $c$ by 1
    end if
end while
```

Example access policy to LSSS matrix Assume the access policy is 'A AND B AND (C OR D)'. The access tree will look as shown in Figure 3.1(a). To transform the access tree to a LSSS matrix, algorithm 1 is executed. First, a counter c is initialized to 1, and the root node is labelled with vector (1). The left child of the root node gets labelled with vector $(1, 1)$ and the right child of the root node with $(0, -1)$. The counter is incremented by 1, so $c = 2$ now. The AND gate labelled with vector $(1, 1)$ now labels its children with vector $(1, 1, 1)$ and $(0, 0, -1)$ respectively. The counter c is increased to 3. The OR gate labels both its children with vector $(0, -1)$. Now all nodes are labelled, and the labelled result can be seen in Figure 3.1(b).



(a) Access tree



(b) Labelled nodes after LSSS conversion algorithm

Figure 3.1: Conversion from access tree to LSSS matrix for access policy 'A AND B AND (C OR D)'

From the labelled access tree, a LSSS matrix can be created where each row is an attribute (leaf) of the tree, so the first row is attribute A, the second row is attribute B, etc.. If a vector is not of length c, it is appended with 0's to make it of length c:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

If this matrix is multiplied with a vector that has as first entry secret s and the other entries have random values, it can be seen that indeed both attribute A, B and either C or D are needed to obtain s.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} s \\ r1 \\ r2 \end{bmatrix} = \begin{bmatrix} s + r1 + r2 \\ -r2 \\ -r1 \\ -r1 \end{bmatrix}$$

This matrix multiplication gives $A \leftarrow s{+}r1{+}r2$, $B \leftarrow -r2$, $C \leftarrow -r1$ and $D \leftarrow -r1$. There are now several ways to check if a set of attributes satisfy the access policy. One way is to check if you can recover the secret s based on the calculations above. Take for example the set of attributes A, B, C then $A{+}B{+}C$ gives $s{+}r1{+}r2{-}r2{-}r1 = s$, so this set of attributes satisfies the access policy. Another way is to use the way written in section 3.4. Take for example the set of attributes: A, C. The OR gate gets a threshold $t = 1$ and both AND gates gets a threshold $t = 2$ since they both have two children. Leaf nodes A and C return 1, and both B and D return 0 since they are not part of the attribute set. Now the OR gate returns 1, because one of its children (C) returns 1 which equals the threshold. However, the left AND gate returns 0 since only one child returns 1 where the threshold is 2. Hence, this attribute set does not satisfy the access policy.

3.6 Bilinear Map

Let G and G_T be finite cyclic groups of prime order p, and let g be a generator of group G. A bilinear map $e : G \times G \to G_T$, has the following two properties [1]:

- Bilinearity: $e(aP, bQ) = e(P, Q)^{ab}$ for all $a, b \in \mathbb{Z}_p$ and $P, Q \in G$.
- Non-degeneracy: $e(g, g) \neq 1$, a bilinear mapping of generator g with itself, gives that $e(g, g)$ is a generator in G_T.

3.7 Attribute-based Encryption

Cipher-Policy Attribute-based Encryption The CP-ABE scheme consists of four algorithms: [2]

- `Setup`: takes as input the implicit security parameter and outputs public parameters public key (PK) and master key (MK).
- `Encrypt(PK,` M`,` Γ`)`: takes as input the public parameters PK, message M, and access policy Γ. The algorithm encrypts M in such a way that the cipher-

text (CT) can only be decrypted by users that possess the set of attributes that satisfy the access policy.

- `Key Generation(MK,` S`)`: takes as input MK and a set of attributes S that describe the generated key. The output of the algorithm is a secret key (SK).
- `Decrypt(PK, CT, SK)`: takes as input the public parameter PK, ciphertext CT, and secret key SK for a set of attributes S. If the set S of attributes satisfies the access policy Γ, the secret key SK can be used to successfully decrypt ciphertext CT associated with access policy Γ. The algorithm will return the message M if the ciphertext is successfully decrypted.

The KP-ABE scheme contains the same four (`Setup, Encrypt, Key Generation, Decrypt`) algorithms. However, where CP-ABE takes the access policy, KP-ABE uses the set of attributes and where CP-ABE takes the set of attributes, KP-ABE takes the access policy [15]. The dependency of the original ABE scheme [2] and some of its variants [15], [16], [17], [18], [19], [20], [21] on a centralized trusted entity causes its implementation in scenarios with a huge number of participants to be less efficient and suffer from security issues. If the central authority, for example, gets compromised, the entire system cannot function properly anymore.

3.8 Decentralized Attribute-based Encryption

Lewko and Waters' Decentralized Attribute-based Encryption system Lewko and Water proposed a DABE system that is secure in the random oracle model. The construction of their multi-authority CP-ABE scheme consists of the following algorithms: [4]

- `Globalsetup(`λ`)` $\rightarrow$ GP. It takes as input the security parameter λ. A bilinear group G of order $N = p_1 p_2 p_3$, and generator g_1 of G_{p_1} are created. A hash function $H : \{0,1\}^* \rightarrow G$ that maps global identities GID to elements of G is published. The output is the global parameter GP that consists of N and g_1.
- `AuthoritySetup`(GP) $\rightarrow$ SK, PK. This algorithm is run by an AA. It takes as input the global parameters GP. For each attribute i belonging to the AA, the AA chooses two random exponents $\alpha_i, \beta_i \in \mathbb{Z}_N$. The algorithm outputs public key PK $= \{e(g_1, g_1)^{\alpha_i}, g_i^{\beta_i} \forall i\}$ and the secret key SK $\{\alpha_i, \beta_i \forall i\}$ is kept private by the AA.
- `Encrypt`$(M, (A, \rho)$, GP, $\{$PK$\})$ $\rightarrow$ CT. The algorithm takes as input message M, an access matrix (A, ρ), the set of public keys for relevant authorities, and GP. The algorithm creates two vectors v and w. The first entry of vector v is a

random $s \in \mathbb{Z}_N$ and the first entry of w is 0, the other entries are random values in $\mathbb{Z}_N$. λ_x denotes the share of s corresponding to row x, $A_x \cdot v$. w_x denotes $A_x \cdot w$. For each row x of A, it chooses a random $r_x \in \mathbb{Z}_N$. The output of the algorithm is ciphertext $\mathrm{CT} = \{C_0, \forall x : C_{1,x}, C_{2,x}, C_{3,x}\}$, which is computed as shown in equations 3.1, 3.2, 3.3, and 3.4.

$$C_0 = Me(g_1, g_1)^s \tag{3.1}$$

$$C_{1,x} = e(g_1, g_1)^{\lambda_x} e(g_1, g_1)^{\alpha_{\rho(x)} r_x} \quad \forall x \tag{3.2}$$

$$C_{2,x} = g_1^{r_x} \quad \forall x \tag{3.3}$$

$$C_{3,x} = g_1^{\beta_{\rho(x)} r_x} g_1^{\omega_x} \quad \forall x \tag{3.4}$$

- `KeyGen`(GID, GP, i, SK) $\rightarrow$ $\mathrm{K}_{i,GID}$. This algorithm is called by an attribute authority to create a key for a user. It takes as input an identity GID, GP, attribute i belonging to an authority, and the authority's secret key SK. It creates and outputs a key $\mathrm{K}_{i,GID}$ for this attribute identity pair as shown in equation 3.5.
$$K_{i,GID} = g_1^{\alpha_i} H(GID)^{\beta_i} \tag{3.5}$$

- `Decrypt`(CT, GP, $\{\mathrm{K}_{i,GID}\}$ $\rightarrow M$. The decryption algorithm takes as input the ciphertext CT, GP, and a collection of keys corresponding to attribute, identity pairs for one fixed GID. The decryptor computes H(GID) from the random oracle. If the decryptor received the secret keys that are needed to satisfy the access policy, he uses equation 3.6 to compute for each attribute x that is part of the access policy. Otherwise, the decryption fails.

$$C_{1,x} \cdot e(H(GID), C_{3,x}) / e(\mathrm{K}_{\rho(x),GID}, C_{2,x}) = e(g_1, g_1)^{\lambda_x} e(H(GID), g_1)^{\omega_x} \tag{3.6}$$

The decryptor chooses constants $c_x \in \mathbb{Z}_N$ such that $\sum_x c_x A_x = (1, 0, .., 0)$ and computes (equation 3.7):

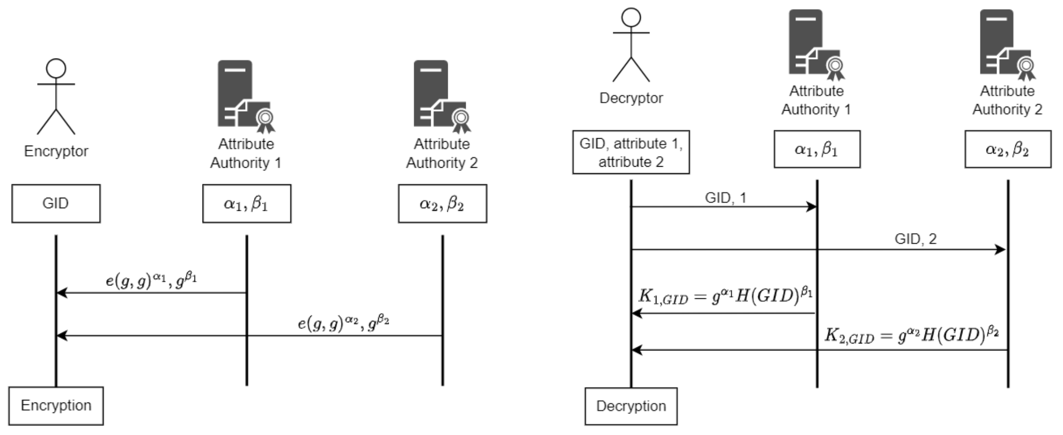
$$\Pi_x (e(g_1, g_1)^{\lambda_x} e(H(GID), g_1)^{\omega_x})^{c_x} = e(g_1, g_1)^s \tag{3.7}$$

The message can be obtained as:

$$M = C_0 / e(g_1, g_1)^s \tag{3.8}$$

Figure 3.2 contains an overview of DABE encryption and decryption.
The challenging part of the scheme is to be collusion-resistant. Lewko and Waters [4] developed a technique for tying users' key components together. ω_x is used which is a share of 0 in the exponent with base $e(H(GID), g_1)$. This structure is used to

unblind the secret value s only if a user has the correct set of attributes. Two users with different GIDs, cannot collude to reconstruct the secret value s, since their ω_x's have a different base.



(a) Encryption overview (b) Decryption overview

Figure 3.2: DABE overview

3.9 Lightweight Decentralized Attribute-based Encryption

The proposed DABE scheme contains some heavy computations. Therefore, it cannot be used on resource-constrained devices in the IoT environment. A solution is to create a lightweight version of DABE. An example of a lightweight DABE model is described below, namely Outsourced Decentralized Attribute-based Encryption (ODABE).

Outsourced Decentralized Attribute-based Encryption (ODABE) The ODABE scheme proposed by Kamel et al. [5] implements the same algorithms as used in the DABE scheme [4]. In addition, a set P exists which contains the computational nodes that have high computation power. Moreover, the `Encrypt()` algorithm changed compared to Lewko and Water's scheme [4]. Before encrypting, the secret parameter generation is executed. A user u chooses $x_i, y_i, z_i \in \mathbb{Z}_p$ for attribute i. The attribute authority that contains attribute i, a_i receives these values from the users and sends the following back to the user:

$$g^{y_i}, g^{z_i}, g^{\beta_i z_i}, e(g,g)^{x_i}, e(g,g)^{\alpha_i z_i}$$

Encryption in the proposed ODABE scheme works as follows: [5]

- User u randomly chooses $s \in \mathbb{Z}_p$ and generates vectors $\gamma = (s, ...)$ and $\omega = (0, ...)$
- For every row i in $\mathcal{M}(\Gamma)$, a random value $r_i \in \mathbb{Z}_p$ is created and $\gamma_i = \mathcal{M}(\Gamma)_i \cdot \gamma$ and $\omega_i = \mathcal{M}(\Gamma)_i \cdot \omega$ are computed.
- A computational node is chosen $P_0 \in P$ and the user u sends:

$$\Gamma, \forall i : \gamma'_i \equiv \gamma_i - x_i \bmod p, \omega'_i \equiv \omega_i - y_i \bmod p, r'_i \equiv r_i - z_i \bmod p$$

- For every leaf i in Γ, the computational node P_0 computes $\mathcal{E}_{i1}$ (Equation 3.9), $\mathcal{E}_{i2}$ (Equation 3.10), $\mathcal{E}_{i3}$ (Equation 3.11) and sends the result ($\forall i : \{\mathcal{E}_{i1}, \mathcal{E}_{i2}, \mathcal{E}_{i3}\}$) back to the user u.

$$\mathcal{E}_{i1} = e(g,g)^{\gamma'_i} e(g,g)^{\alpha_i r'_i} \tag{3.9}$$

$$\mathcal{E}_{i2} = g^{r'_i} \tag{3.10}$$

$$\mathcal{E}_{i3} = g^{\beta_i r'_i} g^{\omega'_i} \tag{3.11}$$

- User u calculates the final ciphertext of message m using equations 3.12, 3.13, 3.14, and 3.15.

$$C_0 = m \cdot e(g,g)^s \tag{3.12}$$

$$C_{1,i} = \mathcal{E}_{i1} e(g,g)^{x_i} e(g,g)^{\alpha_i z_i} \quad \forall i \tag{3.13}$$

$$C_{2,i} = \mathcal{E}_{i2} g^{z_i} \quad \forall i \tag{3.14}$$

$$C_{3,i} = \mathcal{E}_{i3} g^{y_i} g^{\beta_i z_i} \quad \forall i \tag{3.15}$$

Figure 3.3 shows an overview of ODABE encryption.

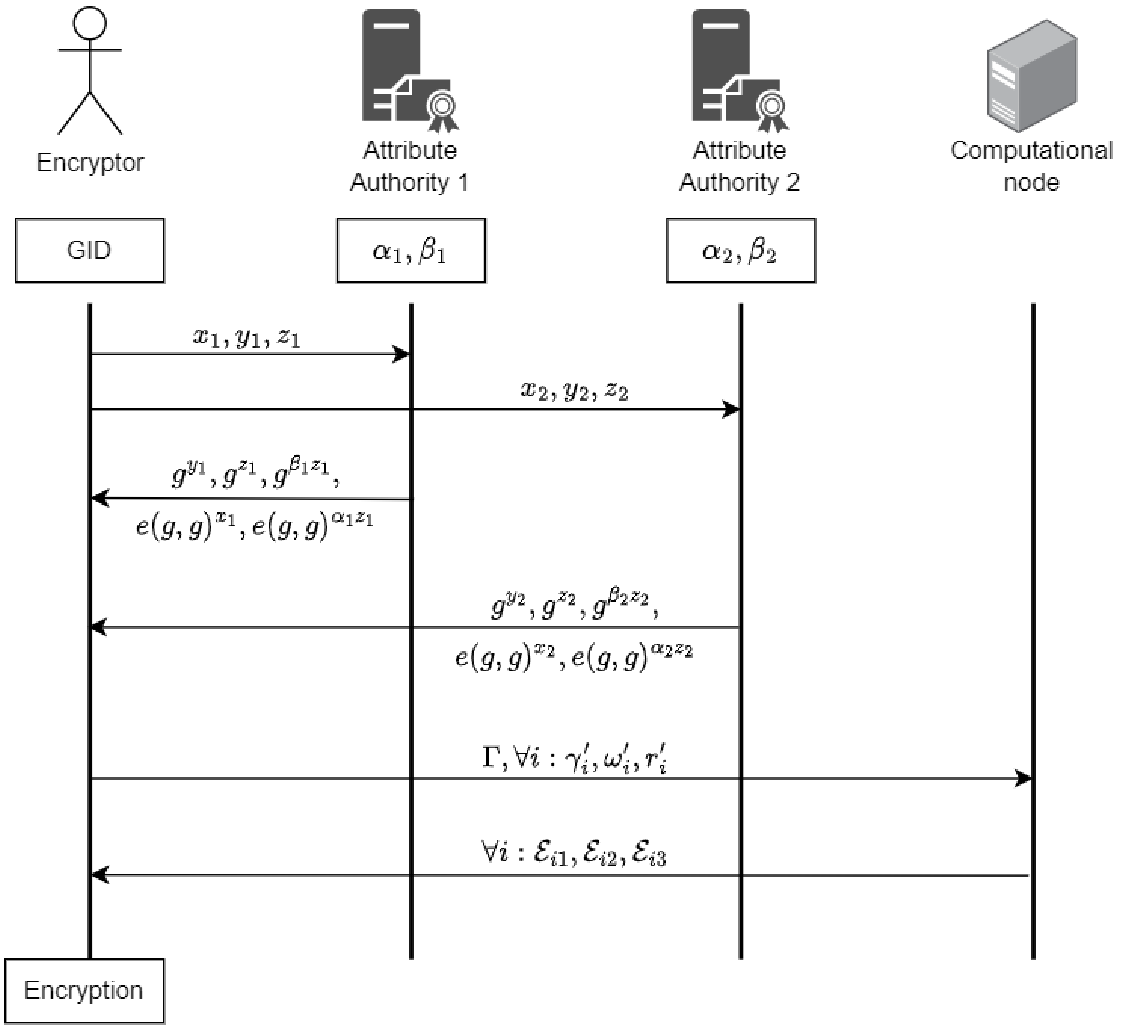

Encryptor
Attribute Authority 1
Attribute Authority 2
Computational node
GID
α_1, β_1
α_2, β_2
x_1, y_1, z_1
x_2, y_2, z_2
$g^{y_1}, g^{z_1}, g^{\beta_1 z_1}, e(g,g)^{x_1}, e(g,g)^{\alpha_1 z_1}$
$g^{y_2}, g^{z_2}, g^{\beta_2 z_2}, e(g,g)^{x_2}, e(g,g)^{\alpha_2 z_2}$
$\Gamma, \forall i : \gamma'_i, \omega'_i, r'_i$
$\forall i : \mathcal{E}_{i1}, \mathcal{E}_{i2}, \mathcal{E}_{i3}$
Encryption

Figure 3.3: ODABE Encryption overview

Chapter 4

Proposed Outsourced Decryption

This chapter contains the proposed Outsourced Decentralized Attribute-based Encryption (ODABE) decryption scheme. Moreover, the proposed scheme will be analysed.

4.1 Decryption DABE

The Decentralized Attribute-based Encryption (DABE) decryption scheme is not optimized yet for the Internet of Things (IoT) environment. The scheme contains some heavy computations. As can be seen in equation 3.6, a multiplication and division with pairs must be done for every attribute x that a user needs for decryption. This is a relatively costly computation in terms of CPU and memory usage. Moreover, as can be seen in equation 3.7, for every attribute x an exponentiation has to be calculated. Lastly, the decryptor needs to divide C_0 by $e(g,g)^s$. Hence, the heavy computations of the DABE scheme are the exponentiations, multiplications and divisions for every attribute x that is part of the access policy and the set of attributes a user possesses. Table 4.1 shows the number of exponentiations, divisions, and multiplications in the target group.

4.1.1 Possible Improvements

In an ideal process, the decryptor only needs to compute $C_0/e(g,g)^s$ (Equation 3.8) to obtain the message and all other computations are outsourced to a computational node. The computational node can have access to the public keys of the attribute authorities (AAs), the ciphertext, and some other values the decryptor can send to the computational node. However, the secret keys of the decryptor ($K_{i,GID}$) should not be sent to the computational node directly. Moreover, if the computational node performs all computations except the division, then the computational node can also obtain the message by also doing the division. Therefore, it is not possible to let a computational node perform the computations of equation 3.6 and 3.7 with the same values as in the original decryption method. The decryptor can send the computational node the secret key and hash of his Global Identifier (GID) ($g^{\alpha_i} H(GID)^{\beta_i}$ and $H(GID)$). However, these values cannot be sent like this to the computational node. They need to be modified such that the original value cannot be obtained.

To solve the problem of not being able to send the secret user key, this key can be raised to a power ρ where $\rho \in_R \mathbb{Z}$. Now the secret key can be sent to the computational node since the computational node cannot recover the real secret key. If only the secret key is raised to the power ρ and the computational node will perform the same computations 3.6, the equation cannot be simplified. Therefore, it is necessary to also raise $H(GID)$ and $C_{1,x}$ for all attributes x, which are needed to decrypt the ciphertext, to the power ρ.

4.2 Security Definition

Definition 2 (ODABE Correctness)**.** The protocol is correct if $\forall u \in \mathcal{U}$ with attributes T_u that satisfy access policy Γ during encryption by the `ODABE_encrypt()` algorithm, u can decrypt the ciphertext using the `ODABE_decrypt`$(C, sk_{(*,u)}, P_0 \in \mathcal{P}, GP)$ algorithm in ODABE, and get the plaintext M, such that:

$$\texttt{ODABE\_decrypt}(\texttt{ODABE\_encrypt}(M, \Gamma, GP, P_0 \in \mathcal{P}), sk_{(*,u)}, P_0 \in \mathcal{P}, GP) = M$$

4.3 Proposed Decryption ODABE

As described in the paper written by Kamel et al. [5], the ODABE model consists of three disjoint sets. The set of users is indicated with $\mathcal{U}$ and contains the users that can encrypt and decrypt. The attribute authorities are part of set $\mathcal{A}$. Lastly, set $\mathcal{P}$ contains the nodes that have high computational power and can execute heavy computations.

The proposed ODABE decryption scheme is written in Protocols 2 and 3 below. Protocol 2 describes the parameter generation which needs to be done before executing the outsourced decryption written in Protocol 3. It is assumed that the computational node is semi-honest, which means the computational node follows the protocol honestly but tries to learn from the messages it receives. The communication between the decryptor and the computational node is encrypted. Moreover, the decryptor can only decrypt the ciphertext if the decryptor contains a set of attributes Λ that satisfies the access policy Γ.

Protocol 2 ODABE Secret decryption parameter generation

A user $u \in \mathcal{U}$ randomly chooses a number $\rho \in \mathbb{Z}_p$. ρ is sent to every attribute authority $a_i \in \mathcal{A}$ and the attribute authorities sends the following back to user u (if they already generated a secret key of attribute i for user u):

$$K^{\rho}_{i,GID} = g^{\alpha_i \rho} H(GID)^{\beta_i \rho}, H(GID)^{\rho}$$

Protocol 3 ODABE Decryption

The proposed outsourced decryption scheme works as follows:

- **Step 1:** The decryptor $u \in \mathcal{U}$ has ciphertext $C = \{C_0, \forall i : C_{1,i}, C_{2,i}, C_{3,i}\}$, $H(GID)^{\rho}, K^{\rho}_{i,GID}$, and creates a subset of attributes Λ such that $\sum_j^{|\Lambda|} A_j = (1, 0, ..., 0)$.
- **Step 2:** The decryptor computes $\forall j \in \Lambda : C^{\rho}_{1,j}$.
- **Step 3:** The decryptor chooses a computational node $P_0 \in \mathcal{P}$ and sends:

$$\forall j \in \Lambda : C^{\rho}_{1,j}, C_{2,j}, C_{3,j}, K^{\rho}_{j,GID}, H(GID)^{\rho}$$

- **Step 4:** The computational node P_0 computes for each leaf j in Λ:

$$\mathcal{E}_j = C^{\rho}_{1,j} \cdot e(H(GID)^{\rho}, C_{3,j}) / e(K^{\rho}_{j,GID}, C_{2,j}) \tag{4.1}$$

- **Step 5:** The computational node P_0 computes the following and sends it back to the decryptor:

$$\mathcal{E} = \prod_j^{|\Lambda|} \mathcal{E}_j \tag{4.2}$$

- **Step 6:** The decryptor computes the following to obtain the message M:

$$M = C_0 / (\mathcal{E}^{1/\rho}) \tag{4.3}$$

An overview of the proposed outsourcing decryption scheme can be seen in Figure 4.1.

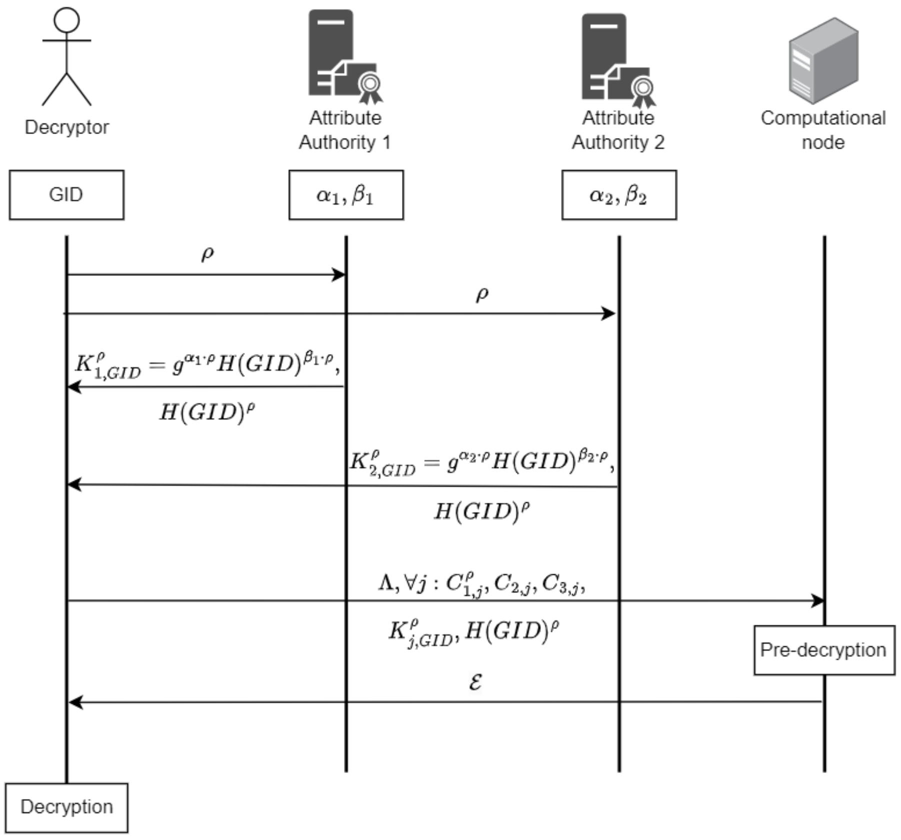

Decryptor
Attribute Authority 1
Attribute Authority 2
Computational node
GID
α_1, β_1
α_2, β_2
ρ
ρ
$K^{\rho}_{1,GID} = g^{\alpha_1 \cdot \rho} H(GID)^{\beta_1 \cdot \rho}$, $H(GID)^{\rho}$
$K^{\rho}_{2,GID} = g^{\alpha_2 \cdot \rho} H(GID)^{\beta_2 \cdot \rho}$, $H(GID)^{\rho}$
$\Lambda, \forall j : C^{\rho}_{1,j}, C_{2,j}, C_{3,j}$, $K^{\rho}_{j,GID}, H(GID)^{\rho}$
Pre-decryption
$\mathcal{E}$
Decryption

Figure 4.1: ODABE Decryption overview

4.3.1 Analysis

In this subsection, it is shown that the proposed ODABE decryption scheme is correct, and performs better than the DABE decryption scheme.

Correctness

Theorem 1. *The proposed ODABE scheme with outsourced decryption is correct.*

Proof. First, the computational node receives from the decryptor:

$$\forall j \in \Lambda : C_{1,j}^{\rho}, C_{2,j}, C_{3,j}, K_{j,GID}^{\rho}, H(GID)^{\rho}$$

It computes, $\forall j \in \Lambda$, using equation 4.1:

$$\begin{aligned}
\mathcal{E}_j &= \frac{C_{1,j}^{\rho} \cdot e(H(GID)^{\rho}, C_{3,j})}{e(K_{j,GID}^{\rho}, C_{2,j})} \\
&= \frac{e(g,g)^{\gamma_j \rho} e(g,g)^{\alpha_j r_j \rho} e(H(GID)^{\rho}, g^{\beta_j r_j} g^{\omega_j})}{e(g^{\alpha_j \rho} H(GID)^{\beta_j \rho}, g^{r_j})} \\
&= \frac{e(g,g)^{\gamma_j \rho} e(g,g)^{\alpha_j r_j \rho} e(H(GID), g)^{\beta_j r_j \rho + \omega_j \rho}}{e(g,g)^{\alpha_j r_j \rho} e(H(GID), g)^{\beta_j r_j \rho}} \\
&= \frac{e(g,g)^{\gamma_j \rho} e(H(GID), g)^{\beta_j r_j \rho + \omega_j \rho}}{e(H(GID), g)^{\beta_j r_j \rho}} \\
&= e(g,g)^{\gamma_j \rho} e(H(GID), g)^{\omega_j \rho}
\end{aligned}$$

The computational node then computes the following using equation 4.2:

$$\begin{aligned}
\mathcal{E} &= \prod_{j}^{|\Lambda|} \mathcal{E}_j \\
&= \prod_{j}^{|\Lambda|} e(g,g)^{\gamma_j \rho} e(H(GID), g)^{\omega_j \rho} \\
&= e(g,g)^{\sum_j^{|\Lambda|} \gamma_j \rho} e(H(GID), g)^{\sum_j^{|\Lambda|} \omega_j \rho} \\
&\text{since} \sum_{j}^{|\Lambda|} \gamma_j = \sum_{j}^{|\Lambda|} A_j v = v \cdot (1, 0, ..., 0) = s \\
&\text{and} \sum_{j}^{|\Lambda|} \omega_j = \sum_{j}^{|\Lambda|} A_j w = w \cdot (1, 0, ..., 0) = 0 \\
&= e(g,g)^{s\rho}
\end{aligned}$$

Lastly, the decryptor finalizes the decryption by computing (equation 4.3):

$$\begin{aligned}\frac{C_0}{\mathcal{E}^{1/\rho}} &= \frac{Me(g,g)^s}{(e(g,g)^{s\rho})^{1/\rho}} \\ &= \frac{Me(g,g)^s}{e(g,g)^s} \\ &= M\end{aligned}$$

The decryptor obtains message M successfully. As written in Definition 2, recovering message M successfully implies that the ODABE scheme is correct. □

Collusion-resistance

The most challenging aspect of any Attribute-based Encryption (ABE) scheme is to make it collusion-resistant. If two users both have a part of the attribute set needed to decrypt the ciphertext, they cannot collude to decrypt the ciphertext together.

Theorem 2. *The proposed ODABE scheme with outsourced decryption is collusion-resistant.*

Proof. As written in the paper by Lewko and Waters [4], hash functions on the GID of the users are used to manage collusion-resistance. It has not changed in comparison to the DABE scheme. When users collude to try to decrypt a ciphertext, the GID's do not match. The share ω is used to blind the real secret, as can be seen in equation 3.7. When a user possesses a set of attributes that satisfy the access policy, the last part of the equation with ω will cancel out. If there are multiple GID's, the cancellation will not work since the ω shares have different bases. Therefore, two users who try to collude and are not able to decrypt the ciphertext by themselves. Hence, they will not obtain the message. □

Performance

Heavy computations In Table 4.1, the number of exponentiations, divisions, and multiplications in the target group are shown for the ABE, DABE, and ODABE decryption schemes. There are no computations in the source group in the decryption algorithms, therefore the source group is left out. i is the number of attributes that are needed to decrypt the ciphertext, so set Λ from Protocol 3. x is the number of nodes of an access tree $\mathcal{T}$ of the policy used to encrypt.

- *ABE*: In the paper by Bethencourt et al. [2], the ABE decryption model is described. For every node i of set Λ they perform a division. Moreover, it is written that: "The decryption algorithm could require two pairings for every leaf of the access tree that is matched by a private key attribute and (at most) one exponentiation for each node along a path from such a leaf to the root." [2]. It performs at most one exponentiation for every node in the access tree except the root. The last computation that needs to be performed to obtain the message contains two divisions in the target group.

- *DABE*: As can be seen in equation 3.6, a multiplication and division in the target group is executed for every attribute i that is part of set Λ. Equation 3.7 shows a product of i terms, so $i - 1$ multiplications, and every term has two exponentiations to the power c_i. Lastly, the division to obtain the message is shown in equation 3.8.
- *ODABE*: In step 2 of protocol 3, the decryptor computes $C_{1,i}^{\rho}$, so it includes i exponentiations. The other steps are executed by a computational node, except for the last step. In step 6, the decryptor performs another exponentiation ($\mathcal{E}^{1/\rho}$) and a division.

As can be seen in Table 4.1, the proposed ODABE scheme is much more efficient than the DABE decryption scheme because it contains fewer exponentiations, divisions, and multiplications in the target group.

	Exponentiations in target group	Division in target group	Multiplications in target group
ABE	$x - 1$	$2 + i$	0
DABE	$2i$	$1 + i$	$2i - 1$
ODABE	$1 + i$	1	0

Table 4.1: Number of heavy computations in decryption schemes, $i = |\Lambda|$ and $x = |\mathcal{T}|$

Execution time In order to be able to analyze the proposed ODABE decryption scheme, the proposed protocol has been implemented and validated in Python. Kamel [44] already implemented the encryption ODABE scheme in Python. His code is extended with the ODABE decryption scheme. Moreover, code was added to also time the decryption schemes. On GitHub[1] the Python code can be found (`odabe.py`).

To test the performance of the original DABE decryption scheme and the proposed ODABE decryption scheme, executions with a certain access policy are executed 5 times, and their mean value is registered in Table 4.2. The proposed decryption scheme has been validated in a setup including a Raspberry Pi zero w (1GHz

[1]https://github.com/jannekevano/MScThesisLightweightDABE

single-core CPU with 512MB RAM) as the decryptor. Figure 4.2 shows an overview of the test setup.

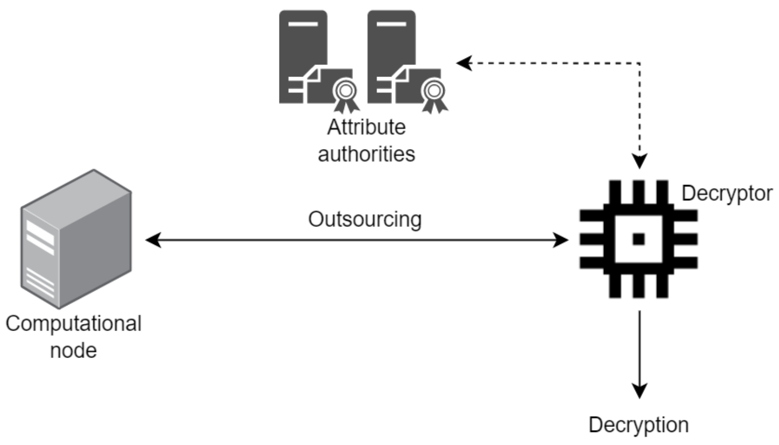


Figure 4.2: ODABE Decryption setup overview

Every time messages are sent in parallel, 20 ms delay is added as communication time. Take for example the proposed decryption scheme in Figure 4.1. The decryptor sends ρ in parallel to the two attribute authorities. The decryptor then receives the key and hash of the GID raised to the power ρ back in parallel. Lastly, the decryptor sends and receives a message to the computational node. In total, there are four moments of communication and therefore, $4 \cdot 20 = 80$ ms is added as an extra communication time.

Table 4.2 shows the execution times of the decryptor in DABE and ODABE decryption based on the number of attributes in the access policy, assuming they are coupled with AND operators. Figure 4.3 shows a graph of the decryption execution times. As can be seen, the decryption time of the decryptor is way less in the proposed ODABE scheme compared to the DABE scheme proposed by Lewko and Waters [4]. Therefore, it can be concluded that the performance of the proposed ODABE scheme has improved compared to the DABE decryption scheme.

Number of attributes	**DABE Decryptor (s)**	**ODABE Decryptor (s)**	**DABE & ODABE Communication (s)**
1	0,300267	0,126761	0,08
2	0,403176	0,152812	0,08
3	0,502324	0,185990	0,08
4	0,651917	0,182681	0,08
5	0,705068	0,194130	0,08

Table 4.2: Decryptor execution time of decryption schemes

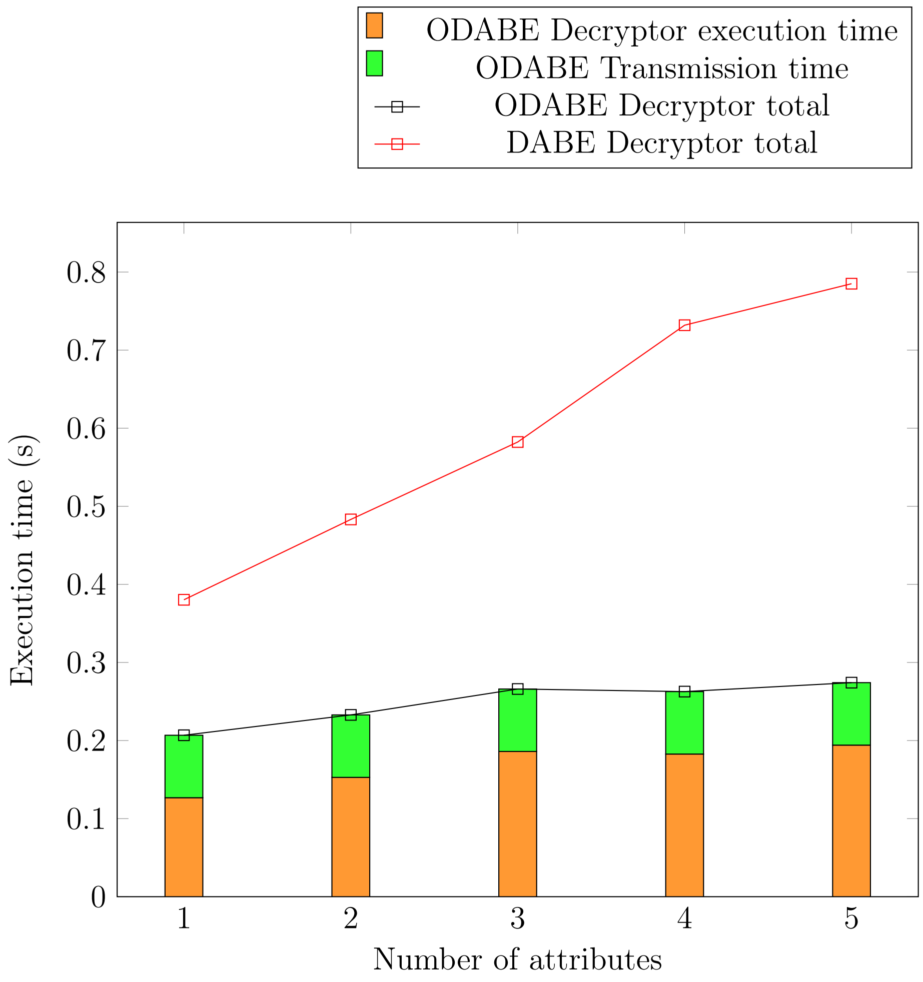


Figure 4.3: DABE and ODABE decryptor execution time

Chapter 5

Formal Verification

In this chapter, the formal verification of the Attribute-based Encryption (ABE), Decentralized Attribute-based Encryption (DABE), and Outsourced Decentralized Attribute-based Encryption (ODABE) models in TAMARIN are discussed.

TAMARIN is used to formally verify the ODABE protocol. To verify the ODABE protocol, the ABE and DABE protocol are formally verified first. This helps to create a model for the ODABE protocol; improvements can be made to the ABE model to create the DABE model, and again to the DABE model to create the ODABE model.

5.1 ABE Model in TAMARIN

The implementation of the ABE model in TAMARIN is based on the paper written by Bethencourt, Sahai and Waters [2]. The ABE TAMARIN model can be found on GitHub[1] (`abe.spthy`).

The built-ins used in the model are `diffie-hellman` for the exponentiation (g^x where g is a generator and x a value) and `bilinear-pairing` for the bilinear pairings where functions `em` and `pmult` are used. [38]
When implementing the model in TAMARIN, it is assumed the access policy consists of two attributes which are coupled via an OR operator. Hence, only one of the two attributes is needed to decrypt the message. This assumption is used to simplify the model. Some future research can be done to include other access policies as well.
The functions in the ABE TAMARIN model are:

- `generator/0` is the generator that is used.

[1]https://github.com/jannekevano/MScThesisLightweightDABE

- `encABE/3` is used for encryption and takes as input the public key of the attribute authority (AA), an access policy consisting of two attributes, and a message m. It outputs the ciphertext ciphertext (CT).

- `decABE/2`, the decryption algorithm, takes as input the ciphertext CT, and a secret key belonging to one of the attributes in the access policy (`skaABE`). The output is message m.

- `skaABE/4` is used as secret key generation algorithm. It generates a secret key for the combination of a user with Global Identifier (GID) and an attribute *att*, assuming the user contains this attribute. `skaABE` takes as input the secret key of the AA, the GID of a user, attribute *att*, and attribute authority AA.

The equation that is used in the ABE model is defined as follows:

$$\texttt{decABE}(\texttt{encABE}(pk, < att, attX >, m), \texttt{skaABE}(sk, gid, att, AA)) = m$$

This equation states that encryption can be done by taking the public key, two attributes, and a message. Decryption takes the result of encryption together with a secret key, which is created by taking the secret key of the AA, GID, attribute *att* and AA. Decryption `decABE` returns the message m.

To model the protocol, several rules are implemented in TAMARIN:

- `attribute_authority_setup` is used to setup an authority. It creates a fresh α and β and generates the secret key secret key (SK) (g^{α}, β) and public key $(e(g,g)^{\alpha}, g^{\beta})$. This rule is only executed once, which is indicated by the `Once` trace restriction. Two other actions that are included in the rule are `AA_Setup`(AA) and `SecretKey_AA`(AA, SK). In the conclusion, the public key of the AA is sent over a public channel and available to everyone. Additionally, two states are stored, one stores the secret key together with the AA entity (`AA_SecretKey`) and one the public key together with the AA entity (`AA_publicKey`).

- `reveal_AA_key` models the compromise of the secret key of the attribute authority AA. The rule reads the secret key database entry and sends it on the public channel such that it is also accessible for an adversary. The action `RevealAA`(AA) states that the secret key of AA was compromised.

- `user_create` generates a fresh GID. A GID should be unique which is checked by trace restriction `Once`. This rule stores the GID together with agent A in a `UserID` state.

- `create_keys_users` is a rule to create a secret key for a user for a specific attribute. The AA is the one creating this key, so the `AA_SecretKey` state is taken as input, together with the `UserID` state that contains the GID, and a fresh attribute *attr* is created. The secret key for a user with GID and attribute *attr*, assuming the user owns this attribute, is called keyGID and the function `skaABE` is used to create this. The observable actions `CreateKey` and `SecretKey` are included in the rule. Two states are stored, of which the first one is `UserKeyCombi` which contains the GID, entity A, keyGID, and attribute *attr*, the other state is `AttributeAuthority` that couples the attribute *attr* to authority AA.

- `create_AP` is used to create an access policy based on two attributes. As stated before, in the ABE Tamarin model it is assumed that the access policy consists of two attributes coupled via an OR operator. This rule consumes as input two `AttributeAuthorities` states containing attributes $attr1$ and $attr2$. An access policy is created by concatenating these two attributes. An access policy can only be created once (checked by using the `Once` trace restriction) and action fact `Create_AP` is included. The rule stores a state `APState` which contains the attributes $attr1$, $attr2$, and the access policy.

- `sender_encrypt` is the rule where the encryptor creates the ciphertext and sends it on a public channel. It takes the `APState` and `AA_publicKey` states and creates a fresh message m. The ciphertext CT is created using the `encABE` function. Two actions facts are performed, namely `Encrypt`(AA, m) and `Secret_m`(m). The ciphertext CT is sent on a public channel.

- `receiver_decrypt` is the rule where the receiver decrypts the ciphertext, assuming he has the right attribute. As input, it takes the ciphertext which is sent on a public channel and the `UserKeyCombi` state for the attribute that is part of the access policy. It reconstructs the message by using the `decABE` function, which takes the ciphertext and keyGID as input. Only one keyGID is needed since the access policy contains an OR operator. An action fact checks if the message that is the result of decryption equals the message that was encrypted, using the `Equality` trace restriction. Additionally, `Decrypt`(message) is an action used later in the security properties.

The two trace restrictions that are added to the model are `Once`, and `Equality`. Their names already explain what they do. `Once`(x) makes sure the entry x is unique and only created once. `Equality`(x, y) checks that two entries x and y are equal.

Security Properties In the ABE TAMARIN model, the confidentiality of the message m and the keys is proven. Therefore, several security properties (lemmas) are added to the TAMARIN model:

- `executable` and `executable_without_decrypt` are sanity checks to see if the protocol is able to be executed entirely.
- `secret_message` states that for all messages if the message is encrypted then the adversary cannot learn message m at any point in time. This lemma proves the confidentiality of the message.
- `secret_user_key` proves that an adversary cannot learn the secret keys of the user.
- `secret_AA_key` states that an adversary cannot learn the secret key of an attribute authority AA unless the adversary performed a long-term secret key reveal on this key.

Using the interactive mode in TAMARIN, it can be seen that all security properties are successfully verified. Appendix A elaborates more on the security properties.

5.2 DABE Model in TAMARIN

The implementation of the DABE model in TAMARIN is based on the paper written by Lewko and Waters [4]. The DABE TAMARIN model can be found on GitHub[2] (`dabe.spthy`).

The DABE model in TAMARIN is an extension of the ABE model.
First of all, the functions are called `gen/0`, `dabeEnc/3`, `dabeDec/2`, and `skaDABE/4`. These functions are the same as the functions in the ABE model. The equation looks different since in the DABE TAMARIN model it is assumed that the attributes are coupled via an AND operator, so both attributes of the access policy are needed to decrypt a ciphertext. Again, this assumption is used to simplify the model and future research can be performed to include other access policies as well.
There is not one single attribute authority which results in needing multiple public keys for encryption. Hence, the equation looks as follows:

$$\texttt{dabeDec}(\texttt{dabeEnc}(m, < att, att2 >, < pk1, pk2 >),$$
$$< \texttt{skaDABE}(sk1, gid, att1, AA1), \texttt{skaDABE}(sk2, gid, att2, AA2) >) = m$$

[2]https://github.com/jannekevano/MScThesisLightweightDABE

`dabeDec` takes the result of the encryption algorithm `dabeEnc` and a concatenation of two secret keys belonging to the attributes and user. The encryption algorithm `dabeEnc` needs a message m, two attributes and two public keys since it is assumed the attributes belong to two different attribute authorities. The concatenation of secret keys consists of two `skaDABE` functions which take the secret key of an attribute authority AA, the GID of the user, the attribute belonging to the AA, and the AA entity itself. `dabeDec` returns the message m.

The rules are mostly the same. However, there are multiple attribute authorities which slightly change some of the rules. The following rules are part of the DABE model:

- `authority_setup` this rule is similar to the `attribute_authority_setup` in ABE. However, there can be multiple attribute authorities now. A fresh α and β are created, such as the public key public key (PK) $(e(g,g)^{\alpha}, g^{\beta})$ and the secret key SK (α, β). Since there can be multiple attribute authorities, the trace restrictions `Once`(α) and `Once`(β) are added to make sure that two attribute authorities cannot have the same secret key. The other actions that are included in the rule are `AA_Setup`(AA), `Public_key_AA`(AA, PK) and `SecretKey_AA_key`(AA, SK). The public key of the AA is sent over a public channel and available to everyone. Additionally, three states are stored, one stores the secret key together with the AA entity (`AA_SecretKey`), one the public key together with the AA entity (`AA_publicKey`), and the last one both the public and secret key together with the AA entity (`AA_keys`).

- `reveal_AA_key` is not changed compared to the ABE Tamarin model. This rule models the compromise of the secret key of an attribute authority AA.

- `user_create` has slightly changed. The only difference is that an extra action is added which will be used when defining the security properties (`UserGID`).

- `create_keys_users` is a rule to create a secret key for a user for a specific attribute. The rule is similar to this rule in the ABE model. `AA_keys`, both public and secret key, is taken as input in comparison to only the secret key in the ABE model. KeyGID is created with the `skaDABE` function. The observable actions `CreateKey`, `UserGID`, `AttrAA` and `Secret_key` are included in the rule. Moreover, trace restriction `Once` makes sure the attribute is only used once, and the keyGID is only created once. Same as in the ABE model, two states are stored: `UserKeyCombi` and `AttributeAuthority`. However, `AttributeAuthority` also contains the public key of the attribute authority AA.

- `create_AP` works the same as in the ABE model. The only differences are the action facts. Three extra trace restrictions are checked in the actions. First, `Once` to make sure an access policy is only created once. Additionally, `Attribute` is added to check the attributes are indeed part of the access policy. And lastly `Inequality` makes sure the attributes of the access policy are not the same attributes.

- `encryptor` is the rule where the encryptor creates the ciphertext and sends it on a public channel. It takes the `APState`, `AttributeAuthority` and `AA_publicKey` states and creates a fresh message m. The ciphertext CT is created using the `dabeEnc` function. Several action facts are performed: `Encrypt`, `Secret_m`, `EncryptAtt`, `OutCT` and `EncryptCheck`. The trace restriction `Inequality` is used to check the attributes are not the same. In the conclusion, the ciphertext CT is sent on a public channel.

- `receiver` is the rule where the receiver decrypts the ciphertext, assuming he has the right attributes. As input it takes the ciphertext which is sent on a public channel and the `UserKeyCombi` states for the attributes that are part of the access policy. It reconstructs the message by using the `dabeDec` function, that takes the ciphertext and keyGIDs as input. An action fact checks if the message that is the result of decryption equals the message that was encrypted, using the `Equality` trace restriction. Moreover, it is checked with the `Attribute` trace restrictions if the attributes are actually part of the access policy used to encrypt the message. Additionally, `Decrypt`, `DecryptAttUser`, `InCT`, and `UserGID` are actions used later in the security properties.

As can be seen in the rules, in addition to the `Equality` and `Once` trace restrictions (from the ABE Tamarin model), the DABE model also has an `Inequality` and `Attribute` trace restriction. The `Inequality`(x, y) restriction states that x is not equal to y. The `Attribute`(x, y, ap) restriction makes sure that the access policy ap equals $< x, y >$ or $< y, x >$, so it ensures the attributes are part of the access policy.

Security Properties The paper written by Lewko and Waters [4] includes some security properties the DABE model should satisfy. Based on this, the following security properties are checked in the DABE Tamarin model:

- `executable`, there are several executable lemmas in the model to check that the entire protocol is executable.

- `secret_message` checks if the message that is encrypted is kept secret to the adversary.

- `collusion_resistant` states that users cannot collude to decrypt a ciphertext. If two users each have one attribute of the access policy, they cannot combine their keys to decrypt the ciphertext. Hence, neither of the two users can decrypt the ciphertext if they only have one of the two attributes of the access policy.
- `only_decrypt_with_right_attributes` checks that a user can only decrypt the ciphertext if it has the attributes according to the access policy.
- `secret_user_key` proves that an adversary cannot learn any of the secret keys of the user.
- `secret_AA_key` states that an adversary cannot learn the secret key of an attribute authority AA unless the adversary performed a long-term secret key reveal on this key.
- `not_two_AA_same_attribute` proves that an attribute is only associated with one attribute authority AA.
- `sameCT` states that the ciphertext that is decrypted in the protocol is also created before.
- `gid_hiding` states that an adversary does not know the GID of a user.
- `check_correct_AAs_encrypt` checks if the attribute authorities that are used for encryption are the correct ones, in other words, the public keys and attributes belong to the AAs.

All security properties are successfully verified in the Tamarin interactive mode. Appendix A elaborates more on the security properties.

5.3 ODABE Model in Tamarin

The implementation of the ODABE model in Tamarin is based on the paper written by Kamel, Ligeti and Reich [5]. The ODABE Tamarin model can be found on GitHub[3] (`odabe.spthy`).

The ODABE model in Tamarin is based on the DABE model described above. The differences have to do with outsourcing.

The functions used are `gen/0` for the generator, `odabeEnc/4` for encryption, `odabeDec/2` for decryption, `skaODABE/4` for secret key generation for the user, and

[3]https://github.com/jannekevano/MScThesisLightweightDABE

`compNode/7` used for the computation of the computational node. The equation now looks as follows:

$$\texttt{odabeDec}(\texttt{odabeEnc}(m, < att, att2 >, < varGen1, varGen2 >,$$
$$\texttt{compNode}(x, y, z, nodeID, gidEnc, pubk1, pubk2)),$$
$$< \texttt{skaODABE}(sk1, gid, att1, AA1), \texttt{skaODABE}(sk2, gid, att2, AA2) >) = m$$

The encryption algorithm `odabeEnc` takes a message m, two attributes, two sets of generated variables needed for encryption, and `compNode` which performs the computations of the computational node. `compNode` takes x, y, z, the $nodeID$, the GID of the encryptor, and the public keys of the attribute authorities that possess the attributes of the access policy. The decryption algorithm `odabeDec` takes the output of the encryption algorithm (ciphertext CT) and a concatenation of two secret keys which are generated with the `skaODABE` function. It returns the message m.

The following rules are the same as in the DABE Tamarin model: `authority_setup`, `reveal_AA_key`, `create_keys_users`, `create_AP` and `receiver`. The other rules in the ODABE Tamarin model either differ slightly from the rules in the DABE model or are completely new rules to cover the computational node:

- `user_create` still generates a fresh GID for a user. However, it now also generates values x, y and z. These values should be kept secret which will be checked in the security properties with action fact `SecretVar`. Besides the `UserID` state also a `Vars` state is stored which will be used to send the values x, y, z and the GID to an attribute authority AA.

- `authority_varGen` receives the values x, y, z and generates variables to send back to the user. It takes as input the `Vars` state which contains the variables x, y, z and the GID of a user, and the `AA_keys` state which takes the public and secret key of an attribute authority AA. The variables that are generated (`varGen`) are: $(g^y, g^z, g^{\beta z}, e(g, g)^x, e(g, g)^{\alpha z})$. The action facts used to check security properties are `AA_Setup_var`(AA) and `SecretVar(varGen)`. The state that is stored is `UserValues` which contains the `varGen`, GID, AA, and the public key of the AA.

- `comp_node_create` is used to create a computational node. A random fresh $nodeID$ is generated. This rule can only be executed once, since there is one computational node, and this is restricted with the `Once` trace restriction. The state `CPNode` is stored that contains the $nodeID$.

- `encrypt_prepare` is the preparation phase of the encryption. The x, y, z variables are sent by the encryptor to the computational node. As input the states `Vars` and `CPNode` are taken. The action fact `StartEncrypt` is used to define security properties. The state that is stored is `CP_Encrypt` which contains x, y, z, the $nodeID$, GID, and two attribute authorities AA1 and AA2. These attribute authorities should contain the attributes that are part of the access policy the user wants to use.

- `comp_node_encrypt` performs the pre-encryption that is done by the computational node. The input that is taken are the `CP_Encrypt` and two `AA_pubkey` states for the attribute authorities that are also part of the `CP_Encrypt` state. `preComp` is the result of the function `compNode`, which contains $x, y, z, nodeID$, GID, and the public keys of the attribute authorities, $pk1$ and $pk2$. The action facts used here are `CPPreEncrypt` and `SecretVar`. In the conclusion, `PreEncrypt` is the state that contains `preComp`, $nodeID$, GID, AA1, and AA2.

- `encryptor` differs from the DABE Tamarin model in a way that it takes more states as input. The extra input states are `PreEncrypt`, `UserID` and `UserValues` twice for each attribute authority. The function used to create the ciphertext is `odabeEnc`.

The decryption in this ODABE model does not use outsourcing. Therefore, the decryption works the same as the decryption in the DABE model.
The same trace restrictions are used as in the DABE model.

Security Properties The security properties in the ODABE Tamarin model are the same as the security properties in the DABE model. They are based on the papers written by Lewko and Waters [4] and Kamel et al. [5]. Three security properties are added:

- `encrypt_cpnode` checks that the user that sends values to the computational node is also the user that encrypts the message after receiving values back from the computational node.

- `secret_variable` proves that the x, y, z variables and the variables that are generated by the computational node are not known to the attacker.

- `authority_setup_correct` checks that the setup phase of public and secret key for an attribute authority is performed successfully before the setup of the other values occurs.

The security properties are successfully verified, which can be seen in the Tamarin interactive mode. Appendix A elaborates more on the security properties.

5.4 ODABE Decryption in Tamarin

The implementation of the ODABE decryption model is based on the ODABE decryption scheme proposed in Chapter 4. The Tamarin ODABE decryption Tamarin model can be found on GitHub[4] (`odabe_dec.spthy`).

The ODABE Tamarin model described above is extended with the decryption scheme. Therefore, besides the decryption everything else stays the same compared to the ODABE Tamarin model above.
Two functions are added, which are the `compNodeDec/3` and `raisedTo/2` functions. `compNodeDec` is used by the computational node to perform the pre-decrypt computations. It takes two secret keys raised to the power p and the hash of the GID raised to the power p. `raisedTo` is a function that takes two parameters and raises the first parameter to the second parameter. This function is added because it has to be used in the equation, and the built-in for the hat-symbol ˆ from `diffie-hellman` cannot be used in the equation. The equation now looks as follows:

$$\texttt{odabeDec}(\texttt{odabeEnc}(m, < att, att2 >, < varGen1, varGen2 >,$$
$$compNode(x, y, z, nodeID, gidEnc, pubk1, pubk2)),$$
$$\texttt{compNodeDec}(\texttt{raisedTo}(\texttt{skaODABE}(sk1, gid, att1, AA1), p),$$
$$\texttt{raisedTo}(\texttt{skaODABE}(sk2, gid, att2, AA2), p), \texttt{raisedTo}(h(gid), p))) = m$$

The equation works the same as the ODABE equation above. The difference is that `odabeDec` takes `compNodeDec` as input instead of a concatenation of the secret keys. `compNodeDec` contains the secret keys (`skaODABE`) but they are `raisedTo` the power p, it also includes the hash of the GID `raisedTo` the power p.

The extra rules added for outsourcing the decryption are:

- `decrypt_prepare` is the preparation phase of the decryption. The secret keys of the user of the attributes needed to decrypt are raised to the power p, which is a fresh value, and also the hash of the GID is raised to the power p. As input the ciphertext, `APState`, `UserKeyCombis`, and `CPNode` are taken. The action fact `startDecrypt` is used to define the security properties. Moreover, action fact `SecretVar` will be used to check that p stays secret. The conclusion contains two states. First of all, `CP_Decrypt` which contains the ciphertext, $nodeID$, GID, the secret keys raised to the power p, and the hash of the GID raised to the power p. The other state that is stored is `DecryptorPrep` that contains the ciphertext, the attributes, GID, user entity, two attribute

[4]https://github.com/jannekevano/MScThesisLightweightDABE

authorities AA1 and AA2, and the access policy. This state is stored to make sure the `decryptor` rule has the correct inputs.

- `comp_node_decrypt` performs the pre-decryption that is done by the computational node. The input that is taken is the `CP_Decrypt` state. `preCompDec` is the result of the function `compNodeDec`, which contains the secret keys raised to the power p and the hash of the GID raised to the power p. The action facts used here are `CPPreDecrypt` and `SecretVar`. In the conclusion, `PreDecrypt` is the state that contains `preCompDec`, the *nodeID* and the GID.
- `decryptor` is the final step of the outsourced decryption. It differs from decryption in the previous ODABE Tamarin model in a way that it takes `CPPreDecrypt` and `DecryptorPrep` as inputs. The rest stays the same.

The same trace restrictions are used compared to the ODABE Tamarin model with DABE decryption.

Security Properties The security properties in this ODABE Tamarin model are the same as in the ODABE model above. One property changed slightly, which is the `executable` property. The decryption performed by the computational node is added here: `StartDecrypt` and `CPPreDecrypt`.

Moreover, an extra security property is added, `decrypt_cpnode`, that checks if the user that sends the values to the computational node is also the user that decrypts the message eventually.

Again, all security properties are successfully verified and Appendix A elaborates more on them.

Appendix B shows a list of the facts, action facts and trace restrictions used in the Tamarin models with a short description.

Chapter 6

Conclusion

This chapter summarizes the achieved results achieved in this thesis. Additionally, the conclusion is stated and the open problems for future research directions are discussed.

6.1 Summary of the Thesis

The goals of this thesis are the formal verification of Outsourced Decentralized Attribute-based Encryption (ODABE), and designing and formally verifying an outsourcing decryption scheme. These are the goals since ODABE has not been formally verified before and an outsourcing (to a single node) decryption scheme does not yet exist.

Utilizing TAMARIN helped to achieve these goals. ODABE has been formally verified in TAMARIN, together with Attribute-based Encryption (ABE) and Decentralized Attribute-based Encryption (DABE). Based on the existing ODABE encryption scheme [5], an outsourcing decryption scheme was proposed. The ODABE decryption scheme was checked on correctness before utilizing TAMARIN to formally verify the scheme. Additionally, the performance of the ODABE decryption scheme was tested in the Internet of Things (IoT) environment.

As part of this thesis, the analysis of outsourced encryption in DABE has been accepted in the 19th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2023) to be presented in Polytechnique Montreal in Canada. Furthermore, other achieved results aimed to be published.

6.2 Conclusion

To conclude the research described and performed in this thesis, the research questions are answered.

How can the security of the existing ODABE approach be formally verified?
The ODABE scheme has been formally verified by using the formal verification tool Tamarin. The implementation of the model first required implementing the ABE and DABE models in Tamarin. The models are created by studying the schemes in detail and interpreting them in Tamarin. Based on the papers on these schemes, the security properties were defined that the model should satisfy. Examples of security properties that the ODABE scheme satisfies are collusion-resistance, the secrecy of the keys, and the secrecy of the message. The ODABE Tamarin model satisfies all security properties that are tested in the model. Hence, it can be concluded that the ODABE model was formally verified against these security properties.

What are the heavy computation components of the decryption algorithm in DABE?
After analyzing the DABE scheme proposed by Lewko and Waters [4], the number of exponentiations, divisions, and multiplications of pairs are the heavy computations in the decryption algorithm. Table 4.1 shows that in the decryption algorithm of DABE $2i$ exponentiations, $1 + i$ divisions, and $2i - 1$ multiplications are executed, where i is the number of attributes needed to decrypt. These numbers of exponentiations, divisions, and multiplications can be improved.

How can the outsourcing approach in ODABE be implemented in the decryption algorithm of DABE?
The approach that can be used to create an outsourcing decryption algorithm can be generally described as follows; first, the decryptor chooses a random number ρ which he sends to the attribute authorities to receive some variables back. The decryptor will send these variables to the computational node. The computational node performs the pre-decryption and sends the result back to the decryptor. Lastly, the decryptor finalizes the decryption to obtain the message. A detailed description is given in Chapter 4.

In general, the computational node is used to perform the heavy computations of the DABE decryption scheme. This results in a more efficient decryption algorithm. The number of exponentiations, divisions, and multiplications performed by the decryptor decreased significantly.

How can the security of the proposed lightweight decryption in DABE be formally verified?
Again, Tamarin was used for formal verification. Using the ODABE Tamarin

model created before, the decryption scheme is implemented. The security properties of this model are almost the same as in the ODABE TAMARIN model, some properties on the outsourcing decryption are added. The TAMARIN model satisfies all security properties that are tested. Hence, it can be concluded that the ODABE decryption model was formally verified against these security properties.

6.3 Future Research Directions

There are multiple ways in which the work presented in this thesis can be extended.

Semi-honest setup to malicious assumption The computational node in the outsourcing decryption scheme is assumed to be semi-honest. Semi-honest means that the node follows the protocol as specified but tries to learn from the messages it receives. Instead, it can be assumed that the computational node is malicious. A malicious node would, depending on the attack model, be able to change, withhold or replay messages back to the decryptor. In this case, the proposed scheme should be improved to make sure it is secure against this malicious node.

Possible errors during communication In the proposed outsourcing decryption scheme, it is assumed there are no errors in the communication between the decryptor/computational node or decryptor/attribute authority. It can, of course, happen in real life that communication is intercepted or communication errors occur etc. An extension would thus be to check and prevent the possible errors during communication. This ties in with the option to research malicious computational nodes.

Access policies in TAMARIN models As written in the formal verification chapter (Chapter 5), to simplify the implementation of the ABE, DABE, and ODABE schemes in TAMARIN it is assumed the access policy consists of two attributes. In the ABE scheme, it is assumed the attributes are coupled via an OR operator. In both the DABE and ODABE schemes, the assumption is made that the attributes are coupled via an AND operator. As an extension more access policies could be added or it could be changed such that the access policy is not fixed anymore and more extensive policies can be tested as well.

Bibliography

[1] Amit Sahai and Brent Waters. “Fuzzy Identity-Based Encryption”. In: *Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science*. Ed. by R. Cramer. Vol. 3494. Berlin, Heidelberg: Springer, 2005, pp. 457–473. URL: https://doi.org/10.1007/11426639_27.

[2] John Bethencourt, Amit Sahai, and Brent Waters. “Ciphertext-Policy Attribute-Based Encryption”. In: *2007 IEEE Symposium on Security and Privacy (SP ’07)*. Berkeley, France, 2007, pp. 321–334. DOI: 10.1109/SP.2007.11.

[3] Q. M. Malluhi et al. “Decentralized ciphertext-policy attribute-based encryption schemes for lightweight devices”. In: *Computer Communications* 145 (Sept. 2019), pp. 113–125. ISSN: 1873703X. DOI: 10.1016/j.comcom.2019.06.008.

[4] Allison Lewko and Brent Waters. “Decentralizing Attribute-Based Encryption”. In: *Advances in Cryptology – EUROCRYPT 2011. EUROCRYPT 2011. Lecture Notes in Computer Science*. Ed. by K.G. Paterson. Vol. 6632. Berlin, Heidelberg: Springer, 2011, pp. 568–588. ISBN: 978-3-642-20465-4. URL: https://doi.org/10.1007/978-3-642-20465-4_31.

[5] Mohammed B.M. Kamel, Peter Ligeti, and Christoph Reich. “Poster: ODABE: Outsourced Decentralized CP-ABE in Internet of Things”. In: *Applied Cryptography and Network Security Workshops. ACNS 2022. Lecture Notes in Computer Science*. Vol. 13285. Springer, 2022. URL: https://doi.org/10.1007/978-3-031-16815-4_35.

[6] Manuel Barbosa et al. “SoK: Computer-aided cryptography”. In: *Proceedings - IEEE Symposium on Security and Privacy*. Vol. 2021-May. Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 777–795. ISBN: 9781728189345. DOI: 10.1109/SP40001.2021.00008.

[7] Bruno Blanchet and Vincent Cheval. *ProVerif: Cryptographic protocol verifier in the formal model.* URL: `https://bblanche.gitlabpages.inria.fr/proverif/`.

[8] Cas Cremers et al. "A comprehensive symbolic analysis of TLS 1.3". In: *Proceedings of the ACM Conference on Computer and Communications Security.* Association for Computing Machinery, Oct. 2017, pp. 1773–1788. ISBN: 9781450349468. DOI: `10.1145/3133956.3134063`.

[9] David Basin, Lucca Hirschi, and Ralf Sasse. "Symbolic Analysis of Identity-Based Protocols". In: *Foundations of Security, Protocols, and Equational Reasoning.* Ed. by J.D. Guttman et al. Vol. 11565. Springer, 2019, pp. 112–134. DOI: `10.1007/978-3-030-19052-1`. URL: `https://doi.org/10.1007/978-3-030-19052-1_9`.

[10] David Basin, Ralf Sasse, and Jorge Toro-Pozo. "The EMV standard: Break, fix, verify". In: *Proceedings - IEEE Symposium on Security and Privacy.* Vol. 2021-May. Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 1766–1781. ISBN: 9781728189345. DOI: `10.1109/SP40001.2021.00037`.

[11] David Basin et al. "A formal analysis of 5g authentication". In: *Proceedings of the ACM Conference on Computer and Communications Security.* Association for Computing Machinery, Oct. 2018, pp. 1383–1396. ISBN: 9781450356930. DOI: `10.1145/3243734.3243846`.

[12] Baasansuren Bat-Erdene et al. "Security Verification of Key Exchange in Ciphertext-Policy Attribute Based Encryption". In: *2022 7th International Conference on Signal and Image Processing, ICSIP 2022.* Institute of Electrical and Electronics Engineers Inc., 2022, pp. 377–381. ISBN: 9781665495639. DOI: `10.1109/ICSIP55141.2022.9887218`.

[13] Narjes Ben Rajeb et al. *Formal Analyze of a Private Access Control Protocol to a Cloud Storage.* Tech. rep. 2017, pp. 495–500. DOI: `10.5220/0006461604950500`.

[14] Nigel P. Smart et al. *Study on cryptographic protocols.* Tech. rep. European Union Agency for Network and Information Security, 2014. DOI: `10.2824/3739`.

[15] Vipul Goyal et al. "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data". In: *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security.* 2006, pp. 89–98. DOI: `10.1145/1180405.1180418`. URL: `https://doi.org/10.1145/1180405.1180418`.

[16] Nouha Oualha and Kim Thuat Nguyen. “Lightweight attribute-based encryption for the internet of things”. In: *2016 25th International Conference on Computer Communications and Networks, ICCCN 2016*. Institute of Electrical and Electronics Engineers Inc., Sept. 2016. ISBN: 9781509022793. DOI: `10.1109/ICCCN.2016.7568538`.

[17] Xuanxia Yao, Zhi Chen, and Ye Tian. “A lightweight attribute-based encryption scheme for the Internet of Things”. In: *Future Generation Computer Systems* 49 (Aug. 2015), pp. 104–112. ISSN: 0167739X. DOI: `10.1016/j.future.2014.10.010`.

[18] Lyes Touati, Yacine Challal, and Abdelmadjid Bouabdallah. “C-CP-ABE: Cooperative ciphertext policy attribute-based encryption for the internet of things”. In: *Proceedings - 2014 International Conference on Advanced Networking Distributed Systems and Applications, INDS 2014*. Institute of Electrical and Electronics Engineers Inc., Nov. 2014, pp. 64–69. ISBN: 9781479951789. DOI: `10.1109/INDS.2014.19`.

[19] Syh Yuan Tan, Kin Woon Yeow, and Seong Oun Hwang. “Enhancement of a Lightweight Attribute-Based Encryption Scheme for the Internet of Things”. In: *IEEE Internet of Things Journal* 6.4 (Aug. 2019), pp. 6384–6395. ISSN: 2327-4662. DOI: `10.1109/JIOT.2019.2900631`.

[20] Kim Thuat Nguyen, Nouha Oualha, and Maryline Laurent. “Securely outsourcing the ciphertext-policy attribute-based encryption”. In: *World Wide Web* 21.1 (Jan. 2018), pp. 169–183. ISSN: 1386145X. DOI: `10.1007/s11280-017-0473-x`.

[21] Hui Tian et al. “A Lightweight Attribute-Based Access Control Scheme for Intelligent Transportation System with Full Privacy Protection”. In: *IEEE Sensors Journal* 21.14 (July 2021), pp. 15793–15806. ISSN: 15581748. DOI: `10.1109/JSEN.2020.3030688`.

[22] Melissa Chase. “Multi-authority Attribute Based Encryption”. In: *Theory of Cryptography. TCC 2007. Lecture Notes in Computer Science*. Ed. by S.P. Vadhan. Vol. 4392. Berlin, Heidelberg: Springer, pp. 515–534. URL: `https://doi.org/10.1007/978-3-540-70936-7_28`.

[23] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert. “Distributed Attribute-Based Encryption”. In: *Information Security and Cryptology – ICISC 2008. ICISC 2008. Lecture Notes in Computer Science*. Ed. by P.J. Lee and J.H. Cheon. Vol. 5461. Berlin, Heidelberg: Springer, 2009, pp. 20–36. URL: `https://doi.org/10.1007/978-3-642-00730-9_2`.

[24] Huang Lin et al. “Secure Threshold Multi Authority Attribute Based Encryption without a Central Authority”. In: *Progress in Cryptology - INDOCRYPT 2008. INDOCRYPT 2008. Lecture Notes in Computer Science.* Ed. by D.R. Chowdhury, V. Rijmen, and A. Das. Vol. 5365. Berlin, Heidelberg: Springer, 2008, pp. 426–436. ISBN: 978-3-540-89754-5. URL: `https://doi.org/10.1007/978-3-540-89754-5_33`.

[25] Melissa Chase and Sherman S.M. Chow. “Improving Privacy and Security in Multi-Authority Attribute-Based Encryption”. In: *CCS ’09: Proceedings of the 16th ACM conference on Computer and communications security.* Association for Computing Machinery, 2009, pp. 121–130. ISBN: 9781605583525. URL: `https://doi.org/10.1145/1653662.1653678`.

[26] Zhen Liu et al. “Fully Secure Multi-authority Ciphertext-Policy Attribute-Based Encryption without Random Oracles”. In: *Computer Security – ESORICS 2011. Lecture Notes in Computer Science.* Ed. by Vijay Atluri and Claudia Diaz. Vol. 6879. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 278–297. ISBN: 978-3-642-23821-5. URL: `https://doi.org/10.1007/978-3-642-23822-2_16`.

[27] Jiameng Sun, Jing Qin, and Jixin Ma. “Securely Outsourcing Decentralized Multi-authority Attribute Based Signature”. In: *Cyberspace Safety and Security. CSS 2017. Lecture Notes in Computer Science().* Ed. by Sheng Wen, Wei Wu, and Aniello Castiglione. Vol. 10581. Springer, 2017, pp. 86–102. ISBN: 978-3-319-69470-2. URL: `https://doi.org/10.1007/978-3-319-69471-9_7`.

[28] Sherman S.M. Chow. “A framework of multi-Authority attribute-based encryption with outsourcing and revocation”. In: *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT.* Vol. 06-08-June-2016. Association for Computing Machinery, June 2016, pp. 215–226. ISBN: 9781450338028. DOI: `10.1145/2914642.2914659`.

[29] Shanshan Tu et al. “A revocable and outsourced multi-authority attribute-based encryption scheme in fog computing”. In: *Computer Networks* 195 (Aug. 2021). ISSN: 1389-1286. DOI: `10.1016/j.comnet.2021.108196`.

[30] Jiaye Shao, Yanqin Zhu, and Qijin Ji. “Efficient decentralized attribute-based encryption with outsourced computation for mobile cloud computing”. In: *Proceedings - 15th IEEE International Symposium on Parallel and Distributed Processing with Applications and 16th IEEE International Conference on Ubiquitous Computing and Communications, ISPA/IUCC 2017.* Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 417–422. ISBN: 9781538637906. DOI: `10.1109/ISPA/IUCC.2017.00067`.

[31] Mohammed B.M. Kamel, Peter Ligeti, and Christoph Reich. “SDABE: Efficient Encryption in Decentralized CP-ABE using Secret Sharing”. In: *International Conference on Electrical, Computer, and Energy Technologies, ICECET 2022*. Prague, Czech Republic: Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1–6. ISBN: 9781665470872. DOI: `10.1109/ICECET55527.2022.9872711`.

[32] Yannis Rouselakis and Brent Waters. “Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption”. In: *Financial Cryptography and Data Security. FC 2015. Lecture Notes in Computer Science*. Ed. by R. Böhme and T. Okamoto. Vol. 8975. Berlin, Heidelberg: Springer, 2015, pp. 315–332. ISBN: 978-3-662-47853-0. URL: `https://doi.org/10.1007/978-3-662-47854-7_19`.

[33] Ehud D. Karnin, Jonathan W. Greene, and Martin E. Hellman. “On Secret Sharing Systems”. In: *IEEE Transactions on Information Theory* 29.1 (1983), pp. 35–41. ISSN: 15579654. DOI: `10.1109/TIT.1983.1056621`.

[34] Amos Beimel. “Secret-Sharing Schemes: A Survey”. In: *Coding and Cryptology. IWCC 2011. Lecture Notes in Computer Science*. Ed. by Yeow Meng Chee et al. Vol. 6639. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46. ISBN: 978-3-642-20900-0. DOI: `10.1007/978-3-642-20901-7`. URL: `https://doi.org/10.1007/978-3-642-20901-7_2`.

[35] Adi Shamir. “How to Share a Secret”. In: *Communications of the ACM*. Vol. 22(11). 1979, pp. 612–613. URL: `https://doi.org/10.1145/359168.359176`.

[36] Mitsuru Ito, Akira Saito, and Takao Nishizeki. “Secret sharing scheme realizing general access structure”. In: *Proc. of the IEEE Global Telecommunication Conference, Globecom 1987*. 1987, pp. 99–102. URL: `https://doi.org/10.1002/ecjc.4430720906`.

[37] Josh Benaloh and Jerry Leichter. “Generalized Secret Sharing and Monotone Functions”. In: *Advances in Cryptology — CRYPTO’ 88. CRYPTO 1988. Lecture Notes in Computer Science*. Ed. by S. (Shafi) Goldwasser. Vol. 403. New York, NY: Springer, 1990, pp. 27–35. ISBN: 9780387971964. URL: `https://doi.org/10.1007/0-387-34799-2_3`.

[38] The Tamarin Team. *Tamarin-Prover Manual Security Protocol Analysis in the Symbolic Model*. Tech. rep. 2022. URL: `https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf`.

[39] Simon Meier et al. *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*. Tech. rep. DOI: 10.1007/978-3-642-39799-8{_}48.

[40] Sevdenur Baloglu et al. "Provably Improving Election Verifiability in Belenios". In: *Electronic Voting*. Ed. by Robert Krimmer et al. Vol. 12900. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 1–16. DOI: 10.1007/978-3-030-86942-7. URL: https://doi.org/10.1007/978-3-030-86942-7_1.

[41] David Basin et al. "Tamarin: Verification of Large-Scale, Real World, Cryptographic Protocols". In: (2022). DOI: 10.1109/msec.2022.3154689. URL: https://hal.archives-ouvertes.fr/hal-03586826.

[42] Zhen Liu et al. *Efficient Generation of Linear Secret Sharing Scheme Matrices from Threshold Access Trees*. Tech. rep. 2014.

[43] Brent Waters. "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization". In: *Public Key Cryptography – PKC 2011. PKC 2011. Lecture Notes in Computer Science*. Vol. 6571. Berlin, Heidelberg: Springer, 2011, pp. 53–70. URL: https://doi.org/10.1007/978-3-642-19379-8_4.

[44] Mohammed Kamel. *ODABE*. 2022. URL: https://github.com/mohammed-kamel/odabe.

Appendix A

Lemmas Tamarin models

A.1 ABE

This section contains the lemmas of the ABE Tamarin model.

executable_without_decrypt Sanity check to see if the protocol is executable until decryption. This lemma checks if there exists a trace where the setup of attribute authority attribute authority (AA) was successful and that this attribute authority is also used for encryption and secret key generation. This lemma is used in the developing phase.

```
lemma executable_without_decrypt:
  exists-trace
    "Ex A AA m gid att1 #i #j #k. AA_Setup(AA)@i & Encrypt(AA, m)
       @j & CreateKey(A, AA, gid, att1) @k"
```

Code A.1: executable_without_decrypt lemma ABE

executable Sanity check to see if the protocol is executable. This lemma checks if there exists a trace where the setup for attribute authority AA was successful and that this attribute authority is also used for encryption. Moreover, message m is the message that is encrypted and also the result of decryption.

```
lemma executable:
  exists-trace
    "Ex AA m #i #j #k. AA_Setup(AA)@i & Encrypt(AA, m) @j & Decrypt
       (m) @k"
```

Code A.2: executable lemma ABE

secret_message This lemma proves the confidentiality of the message. It states that for all messages, if the message m is encrypted then the adversary cannot learn message m at any point in time.

```
lemma secret_message:
    "All m AA #i #j. Encrypt(AA, m)@i & Secret_m(m)@j
    ==> not(Ex #k. K(m)@k)"
```

Code A.3: secret_message lemma ABE

secret_user_key This lemma proves the confidentiality of the secret keys of the user. It states that for all secret keys k of the user, the adversary cannot learn this secret key k at any point in time.

```
lemma secret_user_key:
    "All k #i. SecretKey(k)@i
    ==> ( not (Ex #k. K(k)@k))"
```

Code A.4: secret_user_key lemma ABE

secret_AA_key This lemma proves the confidentiality of the secret key of the attribute authority AA. It states that for all secret keys k of attribute authority AA, the adversary cannot learn the secret key k unless the adversary performed a long-term secret key reveal on the attribute authority AA.

```
lemma secret_AA_key:
    "All AA k #i. SecretKey_AA(AA, k)@i ==> ( not (Ex #k. K(k)@k) |
        (Ex #l. RevealAA(AA)@l))"
```

Code A.5: secret_AA_key lemma ABE

A.2 DABE

In this section, the security properties of the DABE TAMARIN model are described.

executable_setup_encrypt Sanity check to see if the setup and encryption phase are executable. The lemma checks if there exists a trace where both attribute authorities $AA1$ and $AA2$ performed the setup correctly and both authorities are used to encrypt a message. This lemma is used in the developing phase.

```
lemma executable_setup_encrypt:
  exists-trace
    "Ex AA1 AA2 #i #j #k. AA_Setup(AA1)@i & AA_Setup(AA2)@j &
       Encrypt(AA1, AA2)@k"
```

Code A.6: executable_setup_encrypt lemma DABE

executable_encrypt_decrypt Sanity check to see if the encryption and decryption are executable. The lemma checks if there exists a trace where both attribute authorities $AA1$ and $AA2$ are used for encryption and decryption. This lemma is used in the developing phase.

```
lemma executable_encrypt_decrypt:
  exists-trace
    "Ex AA1 AA2 #i #j. Encrypt(AA1, AA2)@i & Decrypt(AA1, AA2)@j"
```

Code A.7: executable_encrypt_decrypt lemma DABE

executable Sanity check to see if the protocol is executable. This lemma checks if there exists a trace where the attribute authorities $AA1$ and $AA2$ are correctly setup. Besides that, two keys are created for user u for attributes $att1$ and $att2$ and the attribute authorities $AA1$ and $AA2$ possess the attributes $att1$ and $att2$ respectively. Moreover, the access policy on the ciphertext contains the attributes $att1$ and $att2$ and these attributes are also used in the encryption phase. Lastly, that user u decrypts the ciphertext with access policy $att1$ AND $att2$. This lemma checks the entire functioning of the protocol.

```
lemma executable:
  exists-trace
    "Ex AA1 AA2 u att1 att2 #i #j #k #l #m #n #o #p #q. AA_Setup(
        AA1)@i & AA_Setup(AA2)@j &
          CreateKey(u, att1)@k & CreateKey(u, att2)@l & AttrAA(att1,
              AA1)@m & AttrAA(att2, AA2)@n & Create_AP(att1, att2)@o
              &
          EncryptAtt(att1, att2)@p & DecryptAttUser(att1, att2, u)@q
            "
```

Code A.8: executable lemma DABE

secret_message This lemma proves the confidentiality of the message. It states that for all messages, if the message m is encrypted with an access policy that contains attributes from attribute authorities $AA1$ and $AA2$, then the adversary cannot learn message m unless the adversary performed a long-term key reveal on the attribute authorities $AA1$ and $AA2$.

```
lemma secret_message:
    "All m AA1 AA2 #i #j. Encrypt(AA1, AA2)@i & Secret_m(m)@j
    ==> ( not (Ex #k. K(m)@k)
      | ((Ex #l. RevealAA(AA1)@l) & (Ex #l. RevealAA(AA2)@l)))"
```

Code A.9: secret_message lemma DABE

collusion_resistant The lemma checks that the protocol is collusionresistance, which means that users cannot collude to decrypt a ciphertext. This lemma checks for all traces that if we have an access policy consisting of attributes $att1$ and $att2$ and these attributes are also used for encryption, and user $u1$ only has attribute $att1$ and thus a key for this attribute and user $u2$ only has attribute $att2$ and the secret key, then these two users both cannot decrypt the ciphertext, so they are not able to collude together.

```
lemma collusion_resistant:
    "All u1 u2 att1 att2 #i #j #k #l.
    Create_AP(att1, att2)@i & EncryptAtt(att1, att2)@j &
    CreateKey(u1, att1)@k & CreateKey(u2, att2)@l & not(Ex #p.
       CreateKey(u1, att2)@p) & not(Ex #q. CreateKey(u2, att1)@q)
    ==> not( (Ex #m. DecryptAttUser(att1, att2, u1)@m) | (Ex #n.
       DecryptAttUser(att1, att2, u2)@n))"
```

Code A.10: collusion_resistant lemma DABE

only_decrypt_with_right_attributes The lemma checks that a user can only decrypt the ciphertext if it has the attributes according to the access policy. It checks that for all traces where the encryption is done with an access policy containing attributes $att1$ and $att2$ and decryption is done by user u, it means that user u has a secret key for attributes $att1$ and $att2$.

```
lemma only_decrypt_with_right_attributes:
    "All u att1 att2 #i #j.
    EncryptAtt(att1, att2)@i & DecryptAttUser(att1, att2, u)@j
    ==> ((Ex #k. CreateKey(u, att1)@k) & (Ex #l. CreateKey(u, att2)
       @l))"
```

Code A.11: only_decrypt_with_right_attributes lemma DABE

secret_user_key This lemma is exactly the same as in the ABE TAMARIN model, lemma A.4.

not_two_AA_same_attribute This lemma proves that an attribute is only associated to one attribute authority. It states that for all attribute authority $AA1$ which possesses attribute att, there does not exists an attribute authority $AA2$ who also possesses attribute att, where $AA1$ and $AA2$ are not the same.

```
lemma not_two_AA_same_attribute:
    "All AA1 att #i. AttrAA(att, AA1)@i ==> not(Ex AA2 #j. AttrAA(
       att, AA2)@j & not(AA1 = AA2))"
```

Code A.12: not_two_AA_same_attribute lemma DABE

secret_AA_key This lemma is exactly the same as in the ABE TAMARIN model, lemma A.5.

sameCT The lemma proves that the ciphertext that is decrypted in the protocol is also encrypted before. So for all ciphertext that is used as an input in the decryption algorithm, there exists a trace where this ciphertext was the output of the encryption algorithm.

```
lemma sameCT:
    "All ct #i. InCT(ct)@i ==> (Ex #j. OutCT(ct)@j)"
```

Code A.13: sameCT lemma DABE

gid_hiding This lemma proves the confidentiality of the Global Identifier (GID) of a user. It states that for all user GIDs *gid*, the adversary cannot learn the GID *gid*.

```
lemma gid_hiding:
    "All gid #i. UserGID(gid)@i
        ==> not (Ex #j. K(gid)@j)"
```

Code A.14: gid_hiding lemma DABE

check_correct_AAs_encrypt The lemma checks if the attribute authorities that are used for encryption are the correct ones, so the public keys and attributes belong to the AAs. It states that for all encryption phases with attribute authorities $AA1$ and $AA2$, attributes $att1$ and $att2$, and public keys $pk1$ and $pk2$, there exists traces where the public key $pk1$ belongs to attribute authority $AA1$, the public key $pk2$ belongs to attribute authority $AA2$, attribute $att1$ is associated to attribute authority $AA1$, and attribute $att2$ is associated to attribute authority $AA2$. This lemma is used in the developing phase.

```
lemma check_correct_AAs_encrypt:
    "All AA1 AA2 att1 att2 pk1 pk2 #i. EncryptCheck(AA1, AA2, att1,
        att2, pk1, pk2)@i
        ==> (Ex #j #k #l #m. Public_key_AA(AA1, pk1)@j &
            Public_key_AA(AA2, pk2)@k & AttrAA(att1, AA1)@l & AttrAA
            (att2, AA2)@m)"
```

Code A.15: check_correct_AAs_encrypt lemma DABE

A.3 ODABE

This section contains the lemmas of the full ODABE scheme in TAMARIN, so including the proposed ODABE decryption scheme. The lemmas are similar to the lemmas in the ODABE scheme without the proposed ODABE decryption scheme, except for the computational node decryption lemmas.

executable_setup_encrypt Sanity check to see if the setup and encryption phase are executable. The lemma check if there exists a trace where both attribute authorities $AA1$ and $AA2$ performed both setups, the public/private key setup and variable generation setup, correctly and both authorities are used to encrypt a message. This lemma is used in the developing phase.

```
lemma executable_setup_encrypt:
  exists-trace
    "Ex AA1 AA2 #i #j #k #l #m. AA_Setup(AA1)@i & AA_Setup(AA2)@j &
        AA_Setup_var(AA1)@l & AA_Setup_var(AA2)@m & Encrypt(AA1,
      AA2)@k"
```

Code A.16: executable_setup_encrypt lemma ODABE

executable_encrypt_decrypt This lemma is exactly the same as in the DABE TAMARIN model, lemma A.7.

executable Sanity check to see if the protocol is executable. This lemma checks if there exists a trace where the attribute authorities $AA1$ and $AA2$ are correctly setup, both with normal setup and variable setup. Besides that, two keys are created for user u with GID $gid2$ for attributes $att1$ and $att2$ and the attribute authorities $AA1$ and $AA2$ possess the attributes $att1$ and $att2$ respectively. Moreover, the access policy on the ciphertext contains the attributes $att1$ and $att2$ and these attributes are also used in the encryption phase. The encryptor has GID gid and prepares the encryption to send to the computational node cp, who performs pre-encryption. Lastly, that user u with GID $gid2$ decrypts the ciphertext with access policy $att1$ AND $att2$. He also prepares decryption for the computational node cp to perform the pre-decryption. This lemma checks the entire functioning of the protocol.

```
lemma executable:
  exists-trace
```

```
    "Ex AA1 AA2 u att1 att2 cp gid gid2 #i #j #k #l #m #n #o #p #q
       #s #t #u #v #w #x. AA_Setup(AA1)@i & AA_Setup(AA2)@j &
       AA_Setup_var(AA1)@u & AA_Setup_var(AA2)@v &
        CreateKey(u, att1, gid2)@k & CreateKey(u, att2, gid2)@l &
          AttrAA(att1, AA1)@m & AttrAA(att2, AA2)@n & Create_AP(
          att1, att2)@o &
        EncryptAtt(att1, att2)@p & DecryptAttUser(att1, att2, u)@q
           & StartEncrypt(cp, gid)@s & CPPreEncrypt(cp, gid)@t &
          StartDecrypt(cp, gid2)@w & CPPreDecrypt(cp, gid2)@x"
```

Code A.17: executable lemma ODABE

authority_setup_correct Sanity check to see if the setup of variables for attribute authority AA is done after setting up the attribute authority AA (public/private key). The lemma states that when the variable setup of attribute authority AA is performed, there exists a trace that the normal setup of attribute authority AA is performed before the variable setup. This lemma is used in the developing phase.

```
lemma authority_setup_correct:
    "All AA #i. AA_Setup_var(AA)@i ==> (Ex #j. AA_Setup(AA)@j & j <
       i)"
```

Code A.18: authority_setup_correct lemma ODABE

secret_message This lemma is exactly the same as in the DABE TAMARIN model, lemma A.9.

collusion_resistant The lemma checks that the protocol is collusion resistance, which means that users cannot collude to decrypt a ciphertext. This lemma checks for all traces that if we have an access policy consisting of attributes $att1$ and $att2$ and these attributes are also used for encryption, and user $u1$ with GID $gid1$ only has attribute $att1$ and thus a key for this attribute and user $u2$ with GID $gid2$ only has attribute $att2$ and the secret key, then these two users both cannot decrypt the ciphertext, so they are not able to collude together.

```
lemma collusion_resistant:
    "All u1 u2 att1 att2 gid1 gid2 #i #j #k #l.
    Create_AP(att1, att2)@i & EncryptAtt(att1, att2)@j &
    CreateKey(u1, att1, gid1)@k & CreateKey(u2, att2, gid2)@l & not
       (Ex #p. CreateKey(u1, att2, gid1)@p) & not(Ex #q. CreateKey(
       u2, att1, gid2)@q)
    ==> not( (Ex #m. DecryptAttUser(att1, att2, u1)@m) | (Ex #n.
       DecryptAttUser(att1, att2, u2)@n))"
```

Code A.19: collusion_resistant lemma ODABE

only_decrypt_with_right_attributes The lemma checks that a user can only decrypt the ciphertext if it has the attributes according to the access policy. It checks that for all traces where the encryption is done with an access policy containing attributes *att*1 and *att*2 and decryption is done by user *u* with GID *gid*, it means that user *u* has a secret key for attributes *att*1 and *att*2.

```
lemma only_decrypt_with_right_attributes:
    "All u att1 att2 #i #j.
    EncryptAtt(att1, att2)@i & DecryptAttUser(att1, att2, u)@j
    ==> ((Ex gid #k. CreateKey(u, att1, gid)@k) & (Ex gid #l.
        CreateKey(u, att2, gid)@l))"
```

Code A.20: only_decrypt_with_right_attributes lemma ODABE

secret_user_key This lemma is exactly the same as in the ABE TAMARIN model, lemma A.4.

not_two_AA_same_attribute This lemma is exactly the same as in the DABE TAMARIN model, lemma A.12.

secret_AA_key This lemma is exactly the same as in the ABE TAMARIN model, lemma A.5.

encrypt_cpnode This lemma checks that the user that sends the values to the computational node is also the one who finishes the encryption. It states that for a user *u* that performs encryption, there also exists a trace where the user started the encryption by preparing and sending values to the computational node *cp*, who performs pre-encryption.

```
lemma encrypt_cpnode:
    "All u #i. EncryptUser(u)@i ==> (Ex id #j #k. StartEncrypt(id,
        u)@j & CPPreEncrypt(id, u)@k)"
```

Code A.21: encrypt_cpnode lemma ODABE

decrypt_cpnode This lemma checks that the user that sends the values to the computational node is also the one who finishes the decryption. It states that for a user *u* that performs decryption, there also exists a trace where the user started the decryption by preparing and sending values to the computational node *cp*, who performs pre-decryption.

```
lemma decrypt_cpnode:
    "All u #i. DecryptUser(u)@i ==> (Ex id #j #k. StartDecrypt(id,
        u)@j & CPPreDecrypt(id, u)@k)"
```

Code A.22: decrypt_cpnode lemma ODABE

sameCT This lemma is exactly the same as in the DABE TAMARIN model, lemma A.13.

secret_variable This lemma proves the confidentiality of the variables that are created. It states that for all variables k, the adversary cannot learn variable k.

```
lemma secret_variable:
    "All k #i. SecretVar(k)@i
    ==> ( not (Ex #k. K(k)@k))"
```

Code A.23: secret_variable lemma ODABE

gid_hiding This lemma is exactly the same as in the DABE TAMARIN model, lemma A.14.

check_correct_AAs_encrypt This lemma is exactly the same as in the DABE TAMARIN model, lemma A.15.

Appendix B

Labels TAMARIN models

This appendix contains the facts, action facts and trace restrictions of the TAMARIN models. They all have a small description and between the brackets after the description you can see in which model (ABE, DABE, ODABE, or ODABE-Dec) this (action) fact / trace restriction is used.

B.1 Facts

- `AA_SecretKey`($\$AA, sk$): stores the entity AA and the secret key sk of this entity. (ABE, DABE, ODABE, ODABE-Dec)
- `AA_publicKey`($\$AA, pubk$): stores the entity AA and the public key $pubk$ of this entity. (ABE)
- `UserID`(GID, $\$A$): stores the GID of user A. (ABE, DABE, ODABE, ODABE-Dec)
- `UserKeyCombi`(GID, $\$A$, keyGID, att): stores the secret key, keyGID, of user A with GID for attribute att. (ABE)
- `AttributeAuthority`($att, \$AA$): couples the attribute $attr$ to attribute authority AA. (ABE)
- `APState`($att1, att2$, accessPolicy): stores the access policy consisting of attributes $att1$ and $att2$. (ABE, DABE, ODABE, ODABE-Dec)
- `AA_pubkey`($\$AA, pubk$): stores the entity AA and the public key $pubk$ of this entity. (DABE, ODABE, ODABE-Dec)
- `AA_keys`($\$AA, pubk, sk$): stores the entity AA, the public key $pubk$ of this entity, and the secret key sk of this entity. (DABE, ODABE, ODABE-Dec)

- `UserKeyCombi`(GID, keyGID, *att* $\$AA, \$A$): stores the secret key, keyGID, of user A with GID for attribute *att* belonging to attribute authority AA. (DABE, ODABE, ODABE-Dec)
- `AttributeAuthority`($att, \$AA, pk$): couples the attribute *attr* to attribute authority AA with public key *pk*. (DABE, ODABE, ODABE-Dec)
- `Vars`(x, y, z, GID): stores the variables x, y, z and the GID of the user. (ODABE, ODABE-Dec)
- `UserValues`(*varGen*, GID, AA, pk): stores the variables, *varGen*, that attribute authority AA with public key *pk* setup for user with GID. (ODABE, ODABE-Dec)
- `CPNode`($nodeID$): stores the $nodeID$ of the computational node that is created. (ODABE, ODABE-Dec)
- `CP_Encrypt`($x, y, z, nodeID$, GID, $AA1, AA2$): stores the computational node's $nodeID$ that performs the pre-encryption with x, y, z for user with GID where the attributes of attribute authorities $AA1$ and $AA2$ are used in the access policy. (ODABE, ODABE-Dec)
- `PreEncrypt`($preComp, nodeID$, GID, $AA1, AA2$): stores the result of pre-encryption, $preComp$, performed by computational node with $nodeID$ for user with GID and involved attribute authorities $AA1$ and $AA2$. (ODABE, ODABE-Dec)
- `CP_Decrypt`($ct, nodeID$, GID, keyGID1p, keyGID2p, hGIDp): stores the computational node's $nodeID$ that performs the pre-decryption with keyGID1p, keyGID2p, hGIDp, and *ct* for user with GID. (ODABE-Dec)
- `DecryptorPrep`($ct, att1, att2, U$, GID, accessPolicy, $AA1, AA2$): stores a state for the decryptor U with GID with the ciphertext *ct*, attributes $att1$ and $att2$ together with the accessPolicy and the attribute authorities $AA1$ and $AA2$ associated with the attributes. (ODABE-Dec)
- `PreDecrypt`($preCompDec, nodeID$, GID): stores the result of pre-decryption, $preCompDec$, performed by computational node with $nodeID$ for user with GID. (ODABE-Dec)

B.2 Action facts

- `AA_Setup`($\$AA$): entity AA ran the attribute authority setup algorithm. (ABE, DABE, ODABE, ODABE-Dec)

- `SecretKey_AA`($\$AA, sk$): couples the entity AA to secret key sk. (ABE)
- `RevealAA`($\$AA$): the keys of entity AA are revealed. (ABE, DABE, ODABE, ODABE-Dec)
- `CreateKey`($\$A, \$AA$, GID, att): a secret key is created for user A with GID for attribute att that belongs to attribute authority AA. (ABE)
- `SecretKey`(keyGID): keyGID is kept secret for an adversary. (ABE, DABE, ODABE, ODABE-Dec)
- `Create_AP`(): an access policy is created. (ABE)
- `Encrypt`($'AA', m$): encrypt message m using the public key of attribute authority AA. (ABE)
- `Secret_m`(m): the message is kept secret for an adversary. (ABE, DABE, ODABE, ODABE-Dec)
- `Decrypt`(message): message is the result of the decrypt algorithm. (ABE)
- `SecretKey_AA_key`($\$AA, sk$): couples the entity AA to secret key sk. (DABE, ODABE, ODABE-Dec)
- `Public_key_AA`($\$AA, pubk$): couples the entity AA to public key $pubk$. (DABE, ODABE, ODABE-Dec)
- `UserGID`(GID): user with GID is created. (DABE, ODABE, ODABE-Dec)
- `CreateKey`($\$A, att$): a secret key is created for user A for attribute att. (DABE, ODABE)
- `AttrAA`($att, \$AA$): couples the attribute authority AA with attribute att that belongs to this AA. (DABE, ODABE, ODABE-Dec)
- `Create_AP`($att1, att2$): an access policy with attributes $att1$ and $att2$ is created. (DABE, ODABE, ODABE-Dec)
- `Encrypt`($AA1, AA2$): the encryption algorithm used attributes from attribute authorities $AA1$ and $AA2$. (DABE, ODABE, ODABE-Dec)
- `EncryptAtt`($att1, att2$): the encryption algorithm uses attributes $att1$ and $att2$ to encrypt. (DABE, ODABE, ODABE-Dec)
- `OutCT`(ct): the ciphertext ct is the output of the encryption algorithm. (DABE, ODABE, ODABE-Dec)

- `EncryptCheck`($AA1, AA2, att1, att2, pk1, pk2$): the encryption algorithm used attributes $att1$ and $att2$ from respectively attributes authorities $AA1$ and $AA2$ with respectively public keys $pk1$ and $pk2$ to encrypt the message. (DABE, ODABE, ODABE-Dec)

- `Decrypt`($AA1, AA2$): the attribute authorities $AA1$ and $AA2$ that contain the attributes needed for decryption. (DABE, ODABE, ODABE-Dec)

- `DecryptAttUser`($att1, att2, U$): user U decrypts the message using attributes $att1$ and $att2$. (DABE, ODABE, ODABE-Dec)

- `InCT`(ct): the ciphertext ct is the input of the decryption algorithm. (DABE, ODABE, ODABE-Dec)

- `SecretVar`($x/y/z/varGen/preComp/p/preCompDec/preCompEnc$): value $x/y/z/varGen/preComp/p/preCompDec/preCompEnc$ is kept secret for an adversary. (ODABE, ODABE-Dec)

- `AA_Setup_var`(AA): attribute authority AA setup the variables. (ODABE, ODABE-Dec)

- `StartEncrypt`($nodeID$, GID): the encryption prepare is performed by user with GID and will be send to the computational node with $nodeID$. (ODABE, ODABE-Dec)

- `CPPreEncrypt`($nodeID$, GID): the pre-encryption is performed by computational node with $nodeID$ for user with GID. (ODABE, ODABE-Dec)

- `EncryptUser`(GID): the GID of the user that encrypted the message. (ODABE, ODABE-Dec)

- `CreateKey`($\$A, att$, GID): a secret key is created for user A with GID for attribute att. (ODABE-Dec)

- `StartDecrypt`($nodeID$, GID): the decryption prepare is performed by user with GID and will be send to the computational node with $nodeID$. (ODABE-Dec)

- `CPPreDecrypt`($nodeID$, GID): the pre-decryption is performed by computational node with $nodeID$ for user with GID. (ODABE-Dec)

- `DecryptUser`(GID): user with GID decrypts the message. (ODABE-Dec)

B.3 Trace restrictions

- `Equality`(x, y): checks that values x and y are the same. (ABE, DABE, ODABE, ODABE-Dec)

```
restriction Equality:
  "All x y #i. Eq(x,y) @i ==> x = y"
```

Code B.1: Equality trace restriction

- `Once`(x): checks that the trace is only created once, so if there are two occassions of `Once`(x), they happen at the same time. (ABE, DABE, ODABE, ODABE-Dec)

```
restriction Once:
  "All X #i #j. Once(X)@i & Once(X)@j ==> #i = #j"
```

Code B.2: Once trace restriction

- `Inequality`(x, y): checks that values x and y are not the same. (DABE, ODABE, ODABE-Dec)

```
restriction Inequality:
  "All x #i. Neq(x,x) @ #i ==> F"
```

Code B.3: Inequality trace restriction

- `Attribute`(x, y, AP): checks that the access policy AP equals $< x, y >$ or $< y, x >$, so it ensures the attributes x and y are part of the access policy AP. (DABE, ODABE, ODABE-Dec)

```
restriction Attribute:
  "All x y ap #i. Attribute(x, y, ap) @i ==> (<x, y> = ap | <y,
      x> = ap)"
```

Code B.4: Attribute trace restriction

List of Figures

2.1 Example of Key-Policy Attribute-based Encryption (KP-ABE) and Ciphertext-Policy Attribute-based Encryption (CP-ABE) 7

3.1 Conversion from access tree to Linear Secret Sharing Scheme (LSSS) matrix for access policy 'A AND B AND (C OR D)' 16
3.2 DABE overview 20
3.3 ODABE Encryption overview 22

4.1 ODABE Decryption overview 26
4.2 ODABE Decryption setup overview 30
4.3 DABE and ODABE decryptor execution time 31

List of Tables

2.1 Supported properties ABE schemes 11

4.1 Number of heavy computations in decryption schemes, $i = |\Lambda|$ and $x = |\mathcal{T}|$ 29

4.2 Decryptor execution time of decryption schemes 30

List of Algorithms and Protocols

1 LSSS Conversion algorithm [42] 16
2 ODABE Secret decryption parameter generation 25
3 ODABE Decryption 25

List of Codes

A.1 executable_without_decrypt lemma ABE 52
A.2 executable lemma ABE 52
A.3 secret_message lemma ABE 53
A.4 secret_user_key lemma ABE 53
A.5 secret_AA_key lemma ABE 53
A.6 executable_setup_encrypt lemma DABE 53
A.7 executable_encrypt_decrypt lemma DABE 54
A.8 executable lemma DABE 54
A.9 secret_message lemma DABE 54
A.10 collusion_resistant lemma DABE 55
A.11 only_decrypt_with_right_attributes lemma DABE 55
A.12 not_two_AA_same_attribute lemma DABE 56
A.13 sameCT lemma DABE 56
A.14 gid_hiding lemma DABE 56
A.15 check_correct_AAs_encrypt lemma DABE 56
A.16 executable_setup_encrypt lemma ODABE 57
A.17 executable lemma ODABE 57
A.18 authority_setup_correct lemma ODABE 58
A.19 collusion_resistant lemma ODABE 58
A.20 only_decrypt_with_right_attributes lemma ODABE 59
A.21 encrypt_cpnode lemma ODABE 59
A.22 decrypt_cpnode lemma ODABE 59
A.23 secret_variable lemma ODABE 60
B.1 Equality trace restriction 65
B.2 Once trace restriction 65
B.3 Inequality trace restriction 65
B.4 Attribute trace restriction 65