Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare

Project Acronym: **ASCLEPIOS**
Project Contract Number: **826093**

Programme**: Health, demographic change and wellbeing**
Call: **Trusted digital solutions and Cybersecurity in Health and Care to protect privacy/data/infrastructures**
Call Identifier: **H2020-SC1-FA-DTS-2018-2020**

Focus Area: **Boosting the effectiveness of the Security Union**
Topic**: Toolkit for assessing and reducing cyber risks in hospitals and care centres**
Topic Identifier: **H2020-SC1-U-TDS-02-2018**

Funding Scheme: **Research and Innovation Action**

Start date of project: 01/12/2018 Duration: 36 months

Deliverable:

D2.2 Attribute-Based Encryption, Dynamic Credentials and Ciphertext Delegation and Integration in Medical Devices

Due date of deliverable: 28/02/2020 Actual submission date: 27/02/2020

WPL: Antonis Michalas

Dissemination Level: Public

Version: final

1 Table of Contents

2 List of Figures and Tables

Figures

Tables

3 Status, Change History and Glossary

Status:	Name:	Date:	Signature:
Draft:	**Ruben Groot Roessink, Arash Vahidi Alexandros Bakas, Alexandr Zalitko, Antonis Michalas**	07/02/2020	
Reviewed:	**Nicolae Paladi**	15/02/2020	
Approved:	**Tamas Kiss**	27/02/2020	

Table 1: Status Change History

Version	Date	Pages	Author	Modification
V0.1	1.6.2019	7	Ruben Groot Roessink	Started working on Attribute-Based Encryption
V0.2	15.6.2019	10	Alexandros Bakas	Bilinear-pairing and Access Policies
V0.3	24.6.2019	14	Ruben Groot Roessink	System Model
V0.4	28.6.2019	17	Arash Vahidi	CP-ABE requirements
V0.5	2.07.2019	17	Antonis Michalas	Reviewed and formatted text
V0.6	15.7.2019	21	Ruben Groot Roessink	Started working on FAME
V0.7	22.7.2019	24	Arash Vahidi	Advantages and Disadvantages of Fame
V0.8	12.8.2019	32	Alexandros Bakas	Started working on DAC-MACS
V0.9	20.8.2019	36	Antonis Michalas	Advantages and Disadvantages of DAC-MACS
V1.0	14.9.2019	42	Ruben Groot Roessink	Efficiency Analysis of RD-ABE
V1.1	29.9.2019	45	Ruben Groot Roessink	Advantages and Disadvantages of RD-ABE
V1.2	3.10.2020	46	Arash Vahidi	Started working on the comparison between ABE schemes
V1.2	14.10.2020	46	Antonis Michalas	Reviewed and formatted text

Version	Date	Pages	Author	Modification
V1.2	26.10.2019	49	Alexandr Zalitko	Requirement analysis
V1.3	8.11.2019	53	Alexandros Bakas	Ciphertext Delegation
V1.4	20.11.2019	54	Arash Vahidi	Architecture for combining SSE with ABE
V1.5	3.12.2019	57	Alexandr Zalitko	Designed a protocol for combining SSE with ABE
V1.6	18.12.2019	59	Antonis Michalas	Protocol Security
V1.7	29.12.2019	61	Alexandros Bakas	Simulation-Based Security
V1.8	8.1.2020	63	Alexandr Zalitko	Detailed experiments for measuring master public/secret key pairs
V1.9	16.1.2020	63	Arash Vahidi	Detailed experiments for measuring users' keys generation time
V2.0	25.1.2020	63	Alexandr Zalitko	Detailed experiments for measuring encryption and decryption times
V2.1	4.2.1010	65	Antonis Michalas	Reviewed and formatted text

Table 2: Deliverable Change History

Glossary

ct	A ciphertext on message m
m	A plaintext message
$\mathcal{P}$	Access policy
$(A, \rho(x))$	(LSSS) Access matrix A and function $\rho(x)$ describing an access policy $\mathcal{P}$
A	l (rows) × n (columns) matrix
$\rho(x)$	Function $\rho(x)$ maps a row x in A to its corresponding attribute α
v	Vector $v = (s, v_2, \dots, v_n)$ is used in LSSS schemes to mask a secret s
λ_x	$A_x * v$, part of a linear secret sharing scheme
ω_x	Weight used in LSSS scheme $\sum_{x \in A} \omega_x * A_x = (1,0, \dots, 0)$
x	A row in an access matrix A, corresponding with an attribute α
α	An attribute
$\mathbb{G}_p$	Algebraic group of order p
$\in_G \mathbb{G}$	Generator of group $\mathbb{G}$
$\in_R \mathbb{G}$	Chosen uniformly at random from group $\mathbb{G}$
CA	Central Authority
AA_k	The kth Attribute Authority (k in total, $k \leq 1$)
SI	System Initializer
CSP	Cloud Service Provider
u_i	Data owner
u_j	A user which is allowed to decrypt a file
u_μ	A revoked user
S_α	Global attribute universe
S_{α_k}	The attribute set managed by Attribute Authority AA_k
$S_{\alpha_k, j}$	The attribute set managed by Attribute Authority AA_k, which user Attribute Authority u_j is entitled to
S_U	The set of all users in the system
S_A	The set of all Attribute Authorities in the system
E_{key}	An encryption, using key key. Usually denotes asymmetric encryption in this report
$\mathcal{H}$	Hash function
uid_j	A global user identity of user u_j
aid_k	A global Attribute Authority identity of AA_k
AES	Advanced Encryption Standard
ABE	Attribute-Based Encryption
KP-ABE	Key-Policy Attribute-Based Encryption
CP-ABE	Ciphertext-Policy Attribute-Based Encryption

IBE	Identity-Based Encryption
TCP/IP	Internet protocol suite
SSE	Symmetric Searchable Encryption
LSSS	Linear Secret Sharing Schemes
FE	Functional Encryption
SSE	Symmetric Searchable Encryption
IND-CPA	Indistinguishable Against Chosen Plaintext Attacks
IND-CCA	Indistinguishable Against Chosen Ciphertext Attacks
RD-ABE	Revocable and Decentralized Attribute-Based Encryption

Table 3: Glossary

1 Introduction

The main goal of ASCLEPIOS is to design and develop an e-health framework that will allow patients to store and share their medical records in a secure and privacy-preserving way. The ASCLEPIOS project is meant to "*boost the effectiveness of the Security Union*", meaning that in the end the ASCLEPIOS framework should be deployable throughout the European Union.

An ever-changing population within the European Union introduces difficulties in sharing (encrypted) medical records. When using more traditional encryption schemes, such as AES, either the same key would have to be given to every user which should be able to decrypt (a set of) medical records or a unique ciphertext would have to be created for every (authorized) user. This either means that a lot of medical practitioners have the same key to decrypt someone's medical records, introducing the security risk of a key being lost or given to someone who should not be able to decrypt those medical records, or the sharing domain would become much less efficient as creating a unique ciphertext for every (authorized) unique user of the system per data record means a large additional amount of costly computations, communications and storage. Also, as new medical practitioners are added to the EU work force every day this means that the framework should either send the key to all new users in the system or a new ciphertext and key have to be created for each encrypted document for every new authorized user. Both situations are not ideal.

A promising new technique to encrypt data without having to know the users beforehand is called ABE (ABE, Section 4). ABE encrypts a file using a so-called access policy which specifies the attributes a user should be entitled to before being able to decrypt a file. This also has the advantage that encryption can be done by the data owner who decides for itself which attributes together should allow for decryption of its data. A more detailed description of ABE schemes is given in Section 4.

ABE was first introduced in 2004 by Sahai and Waters[1], but more and more ABE schemes have been proposed ever since. This report will research the state-of-the-art by looking at (relatively) new ABE schemes, their efficiency (the order of the amount of computations and communications) and advantages and disadvantages of three schemes. In Sections 5, 6 and 7 we compare different ABE scheme in order to decide which one is best suited for the needs of ASCLEPIOS.

In Section 8, we present a brief presentation of ciphertext delegation while in Section 9, we design a protocol based on ABE and Symmetric Searchable Encryption (SSE) according to the reference architecture that was proposed in D1.2 and the SSE scheme from D2.1. The experimental evaluation of the proposed protocol is presented in Section 10.

In Section 11, we describe how some functionalities of our protocol could be integrated in constrained devices, followed by some empirical results in Section 12. Finally, Section 13 concludes the document.

2 Attribute-Based Encryption

2.1 Introduction

ABE is a concept introduced in 2004 by Sahai and Waters[1]. Based on IBE, it allows a user to encrypt data so that it can only be decrypted by users with certain **attributes**.

Definition 1 (Attribute) *Characteristic of an object or entity. In Attribute-Based Encryption it is a characteristic that can be used to define who should be able to decrypt a ciphertext and who should not.*

For example, assume someone wants to encrypt a file so that it can only be read by someone else who is a manager or is an employee in the administrative department with access to the financials of the specific company. This is useful as this means that there is no need to generate public/private key pair for every user the file needs to be send to and someone completely removes the need for a key distribution between users. Next to that ABE also has the advantage that the data owner does not necessarily need to know all users which should be able to decrypt a ciphertext at the time of encryption, instead, if a user obtains the right attributes later on in the system's lifetime it is also able to decrypt the ciphertext.

It works as follows: A user u_i wants to encrypt a message m so that it can only be read by managers or read by employees in the administrative department that have access to the company's finances. The so-called **policy** $\mathcal{P}$ that will be used to encrypt the file is therefore: $isManager()$ OR ($inAdministrativeDepartment()$ AND $hasFinancialAccess()$).

Definition 2 (Policy) *Union or intersection ('OR'/ 'AND') of different attributes indicating which attributes together should allow for decryption. Policies can be expressed as a boolean formula, as seen in the example, but can also just as easily be expressed in a (binary) tree, where (internal) nodes define either a union or an intersection between its children.*

ABE allows a data owner u_i to encrypt a message m into encrypted message ct. User u_i should now be able to share ct with anyone, as ABE only allows authorized users to recover the original message m by decrypting ct. Therefore, if encrypted using ABE, ct can be sent to the users who should receive it, but it can just as easily be hosted somewhere online using a CSP.

In general, in ABE schemes there is no need for u_i to stay online during the decryption phase, which can be quite useful in certain use cases. Such a use case might, for example, be that a medical practitioner in another EU country, can still retrieve medical records if someone is unconscious after an accident in that country. This is permitted by the General Data Protection Regulation, namely in Art. 6 as it states that processing of "*personal data*" is allowed if it is necessary for the vital interests of the data subject or any other natural person[1]. Someone who is entitled these two attributes ($isDoctor()$, $isLifeThreatening()$) is able to retrieve the secret key which it can use to decrypt ct to obtain message m. ABE does not specify, when encrypting a message, which users should be able to decrypt the ciphertext and instead specifies which attributes a user should have before being able to decrypt a ciphertext. This means that data

[1] Art. 6(1)(d) GDPR

can also be decrypted by users that will become entitled to those attributes after the original message was encrypted.

ABE was mentioned for the first time in [1], although more formal definitions were given by Goyal et al.[2] KP-ABE and Bethencourt et al.[3] CP-ABE. This report believes CP-ABE is best suited for use in the ASCLEPIOS project and thus will focus on CP-ABE in describing the sub algorithms of a standard ABE scheme. The reason why CP-ABE is believed to be better equipped to be used in the ASCLEPIOS project is given in Section 5.2.1.1. Basically, every CP-ABE scheme at least exists of four algorithms (which are similar to the algorithms used in Fuzzy Identity-Based Encryption[1]), namely .**Setup()**, .**Encryption()**, .**Key Generation()** and .**Decryption()**. The following descriptions of the different sub algorithms were inspired by the descriptions in the paper by Bethencourt et al.[3] and some notations are changed for the sake of consistency throughout this report:

.Setup() - This algorithm takes no input other than the implicit security parameter. It outputs the public parameters PP and a master key MK.

.Encryption() - This algorithm takes as input the public parameters PP, a message m and an access policy $\mathcal{P}$ over the universe of attributes. A ciphertext ct is generated from m and the access policy $\mathcal{P}$ is embedded into ct.

.Key Generation() - This algorithm takes as input the master key MK and a set of attributes S_α. It outputs a private key SK.

.Decrypt() - This algorithm takes as input PP, ct and SK. If S_α satisfies access policy $\mathcal{P}$ the algorithm is able to decrypt ct and returns message m.

This document will analyse multiple ABE schemes to make a comparison between them in terms of efficiency and advantages & disadvantages (including security) on the methods used in [4] and [5]. This means that the complexity of these schemes will be split out in terms of computational complexity and communication complexity (in the so-called big O notation). This report will not create a new scheme, it will only analyse schemes based on the characteristics of that scheme as mentioned in the paper proposing that scheme or subsequent schemes that try to improve said scheme.

It is important to note that while this report tries to standardize the description of the different schemes and if possible describe them according to the four algorithms described above, this is not always possible as different authors use different structures, but more importantly use different notations within their schemes. Notations might differ among schemes. This report tries to explain the schemes as uniformly as possible but does not deviate too much from the notations used by the authors of all the specific schemes as that would remove any logical connection between the description in this report and the original paper.

2.2 Efficiency Analysis

The efficiency of different schemes is denoted as the computational complexity and communication complexity of the specific protocol steps of different schemes. The complexities are expressed per entity in the scheme, in order to be able to make a comparison between the complexity per entity per protocol step and not only between the complexities (computational and communication) per protocol step.

By computational complexity we mean the cost of the (mathematical) operations that are calculated in each steps of the scheme(s). Only the most costly computation of a protocol step is regarded, meaning that the computational complexity of each protocol step is expressed in the number of bilinear pairings (Section 4.4.1) or exponentiations. As bilinear pairings are

much more expensive to compute than exponentiations the bilinear pairings will determine the computational complexity of a protocol step if present, otherwise the computational complexity of a protocol is expressed in exponentiations. The complexity is expressed in growth rates, so for example, if a protocol step needs two bilinear pairings, independent of the amount of users, attributes and attribute authorities, etc., the complexity is constant and therefore expressed as $O(1)$. If the computational complexity grows linearly with the amount of users it would be $O(u)$ (where u denotes the amount of users).

The communication complexity focuses on the number of messages that need to be exchanged between different entities within the scheme per protocol step. This step will be measured in m to indicate the number of messages each entity sends. The analysis will not consider splitting messages into several messages due to underlying network protocols used, such as TCP/IP.

2.3 Advantages & Disadvantages

The comparison section of a scheme lists the advantages and the disadvantages of the different schemes. The decision was made to include security into Section Advantages & Disadvantages as in general security and usability are intertwined. A more secure system usually has limitations regarding its usability as more expensive operations are used to protect against more capable adversaries or because the system design heavily depends on the security level the system tries to achieve.

This report takes the security notions described in the papers for granted, unless stated otherwise, and therefore will not further describe the security games in the different schemes. This decision was made as it would not be constructive to just cite the security game as the scheme in question and end up with the same conclusion the authors of the paper already came to.

Advantages & disadvantages of are based on the system model described in Section 5. This section on a high-level scheme describes the intended hybrid combination between an ABE and a SSE schemes. Next to that Section 5. also incorporates a subsection regarding requirements that were set out based on the system model. This subsection will also be used while describing advantages and the disadvantages of the different schemes.

2.4 Important notions

This section describes a few (mathematical) components/notations used in more recent ABE schemes. These notations/components are explained in a more general and high-level approach to allow the concept to be grasped by the reader.

2.4.1 Bilinear pairings

A map or pairing e from two source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to a target group $\mathbb{G}_T$, all three of them multiplicative and of size λ, is called bi-linear if for all $a, b \in \mathbb{Z}_N$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. An efficient algorithm to calculate a specific kind of bilinear pairings (Weil Pairings) is described in [6] and [7].

The bilinear pairings used in protocols described in this report is of one of two forms. DAC-MACS (Section 7) makes use of Type-I pairings, meaning that there is only a single source group ($\mathbb{G}$). The bilinear pairings e is therefore of the form $e(g, g)$, where $g \in \mathbb{G}$ is a generator of group $\mathbb{G}$, meaning that any number in $\mathbb{G}$ can be expressed as g^a. For any $a, b \in \mathbb{Z}_N$, $e(g^a, g^b) = e(g, g)^{ab}$ as explained above. The mapping is explained as $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. So-called Type-II pairings and Type-III pairings both have two different source groups, usually denoted as $\mathbb{G}$ and $\mathbb{H}$ and therefore the mapping is expressed as $e: \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$ (Equation 1). The difference between Type-II and Type-III pairings is that in the first case there exists an efficiently computable homomorphism between the two source groups and in the latter case there does not exists such an homomorphism which is efficiently computable. Agrawal and Chase[1] use Type-III pairings in their FAME scheme, as explained in Section 6.

RD-ABE (Section 8) makes use of a so-called composite order group $\mathbb{G}$, where $\mathbb{G}$ is of order $N = p_1 * p_2 * p_3$. p_1, p_2 and p_3 are distinct (large) primes and so subgroups $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ of $\mathbb{G}$ of order p_1, p_2 and p_3 respectively can be found. $\mathbb{G}_{p_1}$ functions as the source group of bilinear pairings in RD-ABE as generator g_1 of group $\mathbb{G}_{p_1}$ is used in this scheme.

2.4.2 Access policies

Access policies in ABE are usually expressed as Boolean formulas, for example $\mathcal{P} = (\alpha_1$ OR α_2 AND $\alpha_3)$, where α_i denotes a specific attribute. The example formula describes an access policy ($\mathcal{P}$) where a user should be able to decrypt data, encrypted with policy $\mathcal{P}$, if it is entitled to α_1 or if it is entitled to α_2 and α_3. Any Boolean formula can be transformed into a binary Boolean formula in an easy manner. For example, $\mathcal{P}$ can be transformed into $\mathcal{P} = (\alpha_1$ OR (α_2 AND α_3)) (if binary operators are assumed right associative). $\mathcal{P}$ can now be expressed as a graph (shown in Figure 1). A more tangible example might be where α_1 denotes someone being in the management of a company ($isManager()$) and α_2 and α_3 denote someone being in the administrative department ($inAdministrativeDepartment()$) and being responsible for the financial state of the company ($hasFinancialAccess()$). Invoices to the company can now be encrypted using these attributes and only managers or employees responsible for the finances of the company can decrypt them. If the attributes of a user or a subset of them together adhere to the policy, meaning that the Boolean formula evaluates to true, the combination of those attributes is called accepting'.

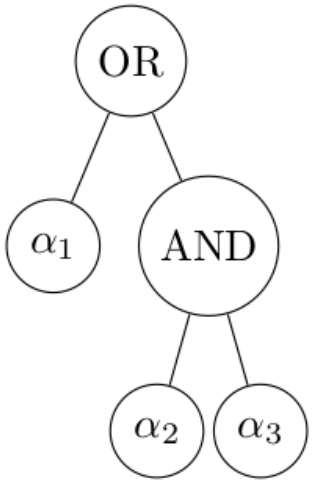


Figure 1: Boolean representation of an example access policy

One important notion **monotonicity** is assumed when talking about access policies. The notion is explained below.

Definition 3 (Monotonicity) *If a (non-empty) subset of any set of attributes is 'accepting', then the larger is set is "accepting" is as well. Assume a collection $\mathbb{A}$ containing all authorized ("accepting") sets. $\mathbb{A}$ is monotone if $\forall \mathbb{B}, \mathbb{C}$: if $\mathbb{B} \in \mathbb{A}$ and $\mathbb{B} \in \mathbb{C}$, then $\mathbb{C} \in \mathbb{A}$. This corresponds with the logical definition that if a user has the right attributes to decrypt a ciphertext and obtains more attributes then it still can decrypt data [2].*

2.4.3 Linear Secret Sharing Scheme

Current ABE schemes make use of a LSSS to reconstruct a secret based on a certain amount of shares a user holds. In general such a scheme is used to allow users to combine their shares to retrieve a secret. In recently introduced ABE schemes however, the shares correspond to attributes and therefore allow a user to combine them to reconstruct the secret. First a definition of LSSS is given, cited from [9]:

Definition 4 (LSSS) *A secret sharing scheme over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if*

- *The shares for each party form a vector over $\mathbb{Z}_p$*
- *There exists a matrix A called the share-generating matrix. The matrix has l rows and n columns. The xth row of A is labeled by a party $\rho(x)$, where ρ is a function from the rows in A to $\mathcal{P}$. If there exists column vector $v = (s, v_2, \dots, v_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $v_2, \dots, v_n \in \mathbb{Z}_p$ are chosen, then $A * v$ is the vector λ of shares of the secret s. The share $\lambda_x = A_x * v$ belongs to party $\rho(x)$.*

Definition 5 (Linear reconstruction property) *Assume an LSSS defining access policy $\mathcal{P}$ described in access matrix A. The vector is $(1,0,\dots,0)$ the span of rows of A and there exist constants $\{\omega_x \in \mathbb{Z}_p\}_{x\in A}$, such that, for any valid shares $\{\lambda_x\}_{x\in A}$ of a secret s, then:* $\sum_{x\in A} \omega_x \lambda_x = s$.

An example of this would be the following (in Section 4.4.4) it is explained how to generate an LSSS matrix from a (binary) access structure/policy): $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}$, corresponding to access policy $\mathcal{P} = (\alpha_1 \text{ AND } (\alpha_4 \text{ OR } (\alpha_2 \text{ AND } \alpha_3)))$

The rows correspond to attributes α_1, α_2, α_3, α_4 respectively. Attributes sets $\{\alpha_1, \alpha_2, \alpha_3\}$ and $\{\alpha_1, \alpha_4\}$ are "*accepting*", as the span of α_1, α_2 and α_3 and the span of α_1 and α_4 are both $(1,0,0)$.

A numerical example: Secret s is chosen as 5 and v_2 and v_3 are chosen uniformly at random from $\mathbb{Z}_8$ as 2 and 3 respectively, therefore vector $v = (5,2,3)$. The resulting vector is:

$$A * v = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} * \begin{pmatrix} 5 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1*5+1*2+0*3 \ mod\ 8 \\ 0*5-1*2+1*3 \ mod\ 8 \\ 0*5+0*2-1*3 \ mod\ 8 \\ 0*5-1*2+0*3 \ mod\ 8 \end{pmatrix} = \begin{pmatrix} 7 \ mod\ 8 \\ 1 \ mod\ 8 \\ -3 \ mod\ 8 \\ -2 \ mod\ 8 \end{pmatrix} = \begin{pmatrix} 7 \\ 1 \\ 5 \\ 6 \end{pmatrix}$$

The vector $\lambda = (7,1,5,6)$ corresponds with the shares that can be used by parties that hold them to reconstruct the secret. Only 'accepting' sets of attributes, meaning that the span of the vectors in A that correspond to these attributes is of the form $(1,0,\dots,0)$, are able to reconstruct the secret. The dot product of the span $(1,0,0)$ and the vector $v = (5,2,3)$ (containing the secret $s = 5$) is: $1*5+0*2+0*3 = 5$ and thus any 'accepting' set of attributes is able to reconstruct the secret s.

To reconstruct the secret based on the shares the shares the system calculates constants (weights) $\{\omega_x\}_{x\in A}$. These weights can be found in polynomial time with respect to the size of the share-generating matrix A[9]. s can be calculated using the following equation: $s = \sum_{x\in A} \omega_x \lambda_x$.

It should be noted that schemes using an LSSS scheme to define attribute shares might be vulnerable to collusions, meaning if a user has the attribute share for $inAdministrativeDepartment()$ and another user has the share for $hasFinancialAccess()$, while both do not have the other shares, it is possible that the users could just combine their shares to get unauthorized access to a document. ABE schemes should have a mechanism to prevent against such collusions.

2.4.4 Converting Boolean formulas to LSSS

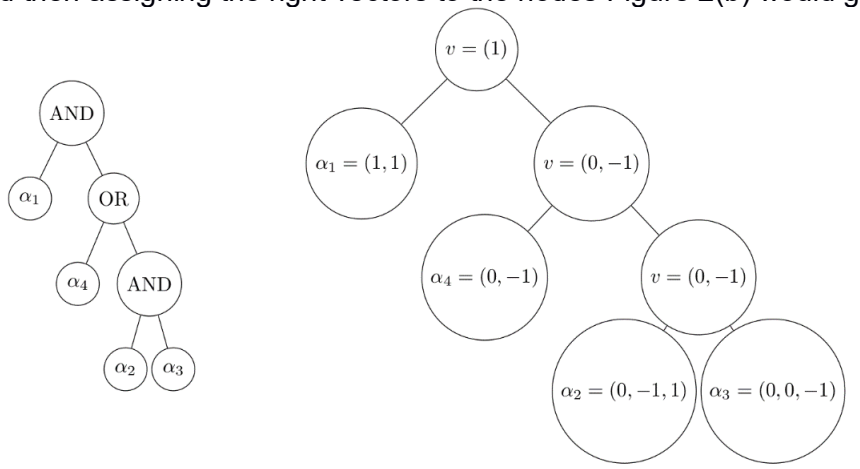
The following is cited from [9]:

This section describes a general algorithm for converting a boolean formula into an equivalent LSSS matrix. The Boolean formula is considered as an access tree, where interior nodes are AND and OR operations and the leaf nodes correspond to attributes and vector $(1,0,\dots,0)$ is used as the sharing vector for the LSSS matrix. First, the root node of the tree is labeled as a vector (1) (a vector of length 1). Then the algorithm goes down through the tree and labels

each node with a vector determined by the vector assigned to its parent node. A global counter variable c is maintained, initialized at 1.

If the parent node is an OR operation labeled by the vector v, then its children are also labeled by v (and the value of c stays the same). If the parent node is an AND operation labeled by the vector v, 0's are padded at the end of v, if necessary, to make it of length c. Then one of its children is labeled with the vector $v|1$ (where | denotes concatenation) and the other one with the vector $(0,\ldots,0)|-1$, where $(0,\ldots,0)$ denotes the zero vector of length c. Note that these two vectors sum to $v|0$. is then incremented by 1. Once the entire tree is labeled, the vectors labeling the leaf nodes form the rows of the LSSS matrix. If these vectors have different lengths, the shorter ones are padded with 0's at the end to arrive at vectors of the same length.

Transforming example α_1 **AND (α_4 OR (α_2 AND α_3))** into the Boolean formula Figure 2(a) and then assigning the right vectors to the nodes Figure 2(b) would give:



(a) Tranformation into the Boolean formula

(b) Assigning the right vectors to the nodes

Figure 2: LSSS matrix example

$$\text{Access structure} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

The generated access matrix is shown above. The combinations that evaluate to true in the graph/formula are $\{\alpha_1, \alpha_2, \alpha_3\}$ and $\{\alpha_1, \alpha_4\}$. This can be seen in the matrix as α_1, α_2 and α_3 together add up to $(1,0,0)$, which is the wanted sharing vector. The same holds true for attributes α_1 and α_4 when summated. Non-accepting attribute set $\{\alpha_1, \alpha_2\}$ gives the vector $(1,0,1)$ and therefore does not allow for decryption.

3 System model

3.1 High level overview

As the ASCLEPIOS project is still in a relatively early phase, meaning that system requirements/a system model have not been finalized this reports briefly defines a model of (part of) the system to ensure certain assumptions could be made while writing this report. Keep in mind while reading this chapter that this chapter will in no way defines the final system model/the requirements of the ASCLEPIOS project.

The final deliverable of ASCLEPIOS will include three Proof of Concepts (POCs) which all simulate a system that (in part) combines both SSE and ABE. The POCs are meant to show that it is possible to securely and efficiently share medical data, while also allowing the user to search among encrypted data, using SSE, so that only data that adheres to the search query has to be decrypted. Next to that, using ABE would allow the users to define a "*profile*" of the users that should be able to decrypt data, while not having the need to specify that specific users are able to decrypt medical data. This would be useful, for example, in a situation where someone gets hurt and is unconscious during a holiday in another EU country. It is assumed no one wants, for example, a medical practitioner in another country to be able to decrypt their medical data at any moment in time, but in life-threatening situations it would be quite useful to have a system in place that allows the medical practitioner to see which medication someone use on a daily basis as to be able to give that someone the best of care.

So, the ASCLEPIOS POCs in their core functionality (use case) should allow a user to upload data, encrypt it using a hybrid ABE/SSE scheme, and a medical practitioner should be able to decrypt the data, if it is entitled to the right attributes. Next to that, the ASCLEPIOS project, as one of its requirements, should be deployable on devices with low(er) computational capabilities so that medical data can be uploaded continuously and securely to the cloud. An example for such a device with a low computational capability would be a pacemaker that encrypts and uploads a data entry containing the state of its user's heart every amount of time, or when something out of the ordinary is taking place. Authorized practitioners should be able to decrypt this data when someone is rushed to the hospital with symptoms of a stroke.

This means, that the pacemaker in the example should be able to encrypt and upload a data entry every once in a while. As most of the data will be encrypted using the same policy, so that the same users are able to decrypt them, it seems logical to use much faster symmetric encryption. In general ABE schemes all use asymmetric encryption and are therefore far less efficient, but some have chosen to use a symmetric key for decrypting data items and while using an ABE scheme to decrypt the symmetric encryption key, thereby making the overall system much more efficient. Using a hybrid symmetric/asymmetric encryption scheme is not only much more efficient, but also allows its users to include other functionalities. One such example would be to include a Searchable Encryption (SE) scheme within the symmetric encryption as most SE use symmetric encryption to allow for search.

This means, that this report assumes a symmetric key is put on a device with low computational capabilities at the initialization of the system. This key is used by the device to encrypt new data entries symmetrically and upload them to a cloud service. The symmetric key is, after it has been added to the device, also uploaded using an ABE scheme, so that only authorized users are able to decrypt the symmetric key (content key) and are able to decrypt all data items encrypted using the content key. The decryption process is allowed to be more computationally heavy as the decryption process can be executed by devices with much more computational capabilities, although only having to decrypt the content key is not an expensive operation.

In summary the system model is as follows: a device with low computational power, in general attached to its physical user, periodically measures a medical state, encrypts it using a symmetric key and uploads it to a cloud server. Next to that the symmetric key is encrypted using an ABE scheme, where the ciphertext of the encrypted key in some way incorporates a policy who should be able to do that. Another user then, if authorized, should be able to retrieve the symmetric key using the ABE scheme and, after downloading all the periodic medical states, should be able to decrypt them and use them for whatever purpose. This system model does not make assumptions about the entities necessary for the ABE scheme.

It should be noted that the described system model does not include a part about FE which allows for statistical analysis of encrypted data. This report does not include that, as the early versions of ASCLEPIOS only include the FE part after data decryption by authorized users. Early versions describe a work flow, where authorized users decrypt the data and encrypt it again using FE, so that (medical) researchers might execute statistical queries on the (newly) encrypted data.

3.2 Requirements

This chapter describes the different requirements that the ASCLEPIOS should adhere to, at least according to the high-level overview of the system model in the previous section. Once again it should be noted that the high-level overview and these requirements are simply assumptions that had to be made to be able to make a good comparison between the different schemes, as while they all fall into the category "*ABE*"they all have different characteristics and intended goals.

3.2.1 Strong requirements

The following requirements were drawn up and are suggested to be "*Strong*" requirements, meaning that any ABE scheme used in a similar setting should at least implement these requirements. Most of them were drawn up from the requirements listed in [4].

3.2.1.1 CP-ABE

According to Bethencourt et al.[3] CP-ABE allows the encryptor to determine which users are able to decrypt certain ciphertexts by setting the access policy when generating said ciphertexts. This is opposed to KP-ABE where the key issuer determines which policy is used to generate the key and so there is an additional need to trust the key issuer. The use of CP-ABE is a strong requirement as this allows the user, instead of the key issuer, to determine which attributes together should allow the decryption of which ciphertexts, by stating the access policy based on which attributes a user needs to have to be able to decrypt the ciphertext and therefore there is less need to trust the key issuer.

3.2.1.2 Collusion resistance

Another important requirement is that any scheme should be collusion resistant. In ABE schemes collusion resistance means that users are not able to decrypt data encrypted with an

access policy which does not evaluate to True for any of the users. So for example, if one user has a Dutch nationality and is 15 years old, while another user has a Polish nationality and is 37 years old, they together should not be able to decrypt a ciphertext that was encrypted using a policy $\mathcal{P} = hasDutchNationality()$ AND $isAboveTwenty()$ as none of the users actually is in possession of all the attributes to decrypt the ciphertext.

3.2.1.3 Access revocation

For the system to be real-world applicable the system needs to have some sort of revocation process. Schroer mentions the notions of "*indirect*" and "*direct revocation*" in [4], meaning that the revocation takes effect after a certain amount of time or is enforced directly, respectively. The former has the negative property that revocations are not enforced directly, meaning that there is a window of time where a revoked user still has access to the data in the system, but has the positive property that re-encryption or other costly methods do not have to be used, whereas these properties are reversed in direct encryption.

A form of a revocation method might adapt policies by adding/removing policies form the ciphertext. While this is straightforward in KP-ABE, this is not in CP-ABE. In CP-ABE schemes adaptable policies almost always need some form of re-encryption which, in general, is usually quite expensive. Adaptable policies might be necessary in a real-life scenario, as for example, when a new hospital is added to the system it will most likely be added as a new attribute authority. The hospital is a trusted party that would be able to hand out attributes for its employees depending on their designated job. These new attributes might need to be added to existing ciphertexts to ensure the employees of the newly added hospital are able to decrypt the ciphertext when necessary.

3.2.1.4 Scalability

One of the most important requirements is that the system should be scalable. Scalability in terms of computations/communications denotes the possibility to, for example, add new hardware to the system so that the system in its entirety is able to perform more computations in the same amount of time. This ensures the system is not limited to a maximum amount of computations/communications per unit of time. Scalability in the example case denotes scaling an existing and running system.

Another way the system should be scalable regards "*adding users*" and to a lesser degree also "*adding authorities*". As the ASCLEPIOS project aims to design POCs for a medical data sharing system that can be used EU wide, it should be a strong requirement that new entities, necessary for the system to function properly, can be added after initialization as not all participants might be known beforehand. This includes for example the addition of new users and new Attribute Authorities during runtime.

3.2.2 Weak Requirements

This section describes recommended properties of the system which are not necessarily required, but might make the system more secure or more usable in a real-life scenario. Most of these are based on the weak requirements mentioned by Schroer in [4].

3.2.2.1 Multiple Authorities

For an application that is will be used EU wide it might be important to have the possibility to have more than one entity that serves as Attribute Authority AA_k, where $k \in S_A$. S_A denotes the set of Attribute Authorities. This would solve the so-called key escrow problem, meaning that there is no single Attribute Authority holding the decryption keys for all attributes and so the attributes can be chosen in such a way that a user needs to decrypt a message using the attributes of at least two AAs. This potentially increases the security of the system as k Attribute Authorities would have to be compromised in order to illegitimately decrypt a ciphertext which is encrypted using attribute keys of k Attribute Authorities, assuming the scheme is collusion resistant (Section 3.2.1.2). This would also be very useful in a real-world scenario as, for example, real-life authorities within the health-care domain, such as the Dutch BIG registration (registration service for the licences of medical practitioners and so on), hospitals and first aid carers can take on the task of being an Attribute Authority in the ASCLEPIOS POCs. In other (digital) medical data sharing systems these entities are already used as key issuers/authorities to prove that one is allowed to have access to certain data, so using these in the system proposed by ASCLEPIOS as AAs seems to be a logical choice.

3.2.2.2 Large Attribute Universe

For a medical data sharing mechanism to be usable in a real-life scenario, for example that the system is deployed with in the EU and crossing multiple borders, it is assumed to be necessary to have the possibility to have a large attribute universe. The system would not be very usable if the attribute universe is limited in size as that would mean there is a lot less control on who has access to what.

3.2.2.3 Regranting Access

It is necessary to be able to revoke access to certain users if, for example, the time expires in which someone needs access to specific ciphertexts and be able to decrypt them. The possibility to re-grant access is therefore seen as a weak requirement as that might also be necessary in a real-life setting. Access management is quite important for any deployed application to be useful and so re-granting access should be a possibility.

3.2.2.4 Multiple Access Controls

The difference between read and write access should be addressed as any database-managed application that is used in real-life differs in read and write access for specific users. In the case of ASCLEPIOS some medical practitioners should only be able to read data entries, but others might need to be able to also write data entries, such as comments to other data entries.

4 FAME: Fast Attribute-based Message Encryption

4.1 Introduction

FAME[8] is an ABE scheme developed by Agrawal and Chase. This scheme is the first scheme that allows using arbitrary string as attributes as they are mapped to group element in a group $\mathbb{H}$. The scheme is also the first scheme that includes elements from both groups $\mathbb{G}$ and $\mathbb{H}$ in every part of the ciphertext and the key. The authors state that they use Type-III pairings (Section 4.4.1) as they "*are generally faster*" and "*are the recommended choice by experts*". The fact that the decryption process is constant (in amount of needed computations) makes it usable for a large universe of users/attributes. The relatively fast encryption process, as no bilinear pairings but only multiplication and encryption are used, makes FAME usable on embedded devices with low computational power.

FAME is a CP-ABE scheme, meaning that the ciphertext is associated with a policy $\mathcal{P}$. The advantage of using CP-ABE instead of KP-ABE (key associated with $\mathcal{P}$) is that the data owner can specify the attributes a user should be entitled to before being able to decrypt a ciphertext, whereas in KP-ABE the CA is the one that makes the "*decision*" who is able to decrypt or not, by embedding the access policy $\mathcal{P}$ into the keys issued to users.

FAME itself does not allow for revoking access to a ciphertext. The policy ($\mathcal{P}_{old}$) is embedded in the ciphertexts and can only be changed by generating a new ciphertext using another policy $\mathcal{P}_{new}$.

FAME has the possibility that arbitrary strings can be used to denote attributes, which makes it in theory possible to encrypt ciphertexts for attributes of different instances. For example, the attribute $isDoctor()$ might be issued by a government-regulated authority that holds the records of all medical practitioners within that country, while the attribute $isNeurosurgeon()$ might be issued by the hospital the practioner works at. However, as there still is a central authority necessary to generate all decryption keys within FAME this means that this CA needs to be fully trusted as it can generate every key imaginable (within the key space).

As FAME does not allow for revoking access, it also does not allow for regranting access. As there is also no access control to distinguish between read and write access rights or ownership proof within the system it might not be usable in a real-life scenario, although it points out some interesting features which might be usable in a real-life setting.

The following (impactful) substitutions were made while describing the FAME ABE scheme. Keep in mind that this table only includes substitutions which were specific for this scheme:

Notation in this report	Notation in the original paper
v	l (denoting an arbitrary number used in hash function $\mathcal{H}$)

Table 4: Notation substitutions specific to FAME

4.2 Efficiency

4.2.1 .Setup()

The .Setup() algorithm of FAME is used to set up the system parameters. FAME does not make use of multiple authorities and thus the scheme is between users and a single CA. $GroupGen(1^{\lambda})$ is an external algorithm that finds a Type-III bilinear pairing: $e(g^a, h^b) = e(g,h)^{ab}$ or $\mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$, where $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ are three groups of prime order p.

c_1: CA runs $GroupGen(1^{\lambda})$ and obtains public parameters $PP = (p, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e(g,h), g \in_G \mathbb{G}, h \in_G \mathbb{H})$ (cost: **1 bilinear pairing**).

c_2: CA picks $a_1, a_2 \in_R \mathbb{Z}_p^*$ and $d_1, d_2, d_3 \in_R \mathbb{Z}_p$(cost: **negligible**).
c_3: CA computes $H_1 = h^{a_1}, H_2 = h^{a_2}$ (cost: **2 exponentiations**).
c_4: CA computes $T_1 = e(g,h)^{d_1 a_1 + d_3}, T_2 = e(g,h)^{d_2 a_2 + d_3}$(cost: **2 exponentiations**).
c_5: CA picks $b_1, b_2 \in_R \mathbb{Z}_{p^*}$ (cost: **negligible**).
c_6: CA computes $g^{d_1}, g^{d_2}, g^{d_3}$(cost: **3 exponentiations**).
c_7: CA sets its public key as $PK_{CA} = (h, H_1, H_2, T_1, T_2)$(cost: **negligible**).
c_8: CA sets its master secret key as $MSK_{CA} = (g, h, a_1, a_2, b_1, b_2, g^{d_1}, g^{d_2}, g^{d_3})$ (cost: **negligible**).
m_1: CA publishes its PK_{CA} and PP (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
CA	$O(1)$	$O(1)$

Table 5: Complexities .Setup() FAME

Both complexities are $O(1)$ as they are not linearly dependent on the amount of users, attributes and ciphertexts as the CA is only set up once.

4.3 .KeyGen()

The .KeyGen() algorithm uses hash functions to express an attribute as a string. The hash function is given below:

Definition 6 (Hash function $\mathcal{H}$) *FAME uses a hash function to get a deterministic value in group $\mathbb{G}$. Agrawal and Chase (FAME[8]) use the hash function to get from any arbitrary string, describing an attribute, to a value in a pre-determined group I, which can be used further on in the scheme.*
Hash function $\mathcal{H}$ has two types of input which will be explained in this section.

- $\mathcal{H}(\alpha, y, t)$, *where α denotes an arbitrary string (attribute), $y \in \{1,2,3\}$ and $t \in \{1,2\}$. A string 'Doctor' therefore could be used as an attribute by calculating the hash function as $\mathcal{H}('Doctor', y, t)$. The string 'Doctor' is of course first changed into a numeric value and y and t are concatenated to that value. y and t are used to define multiple different values that are all used to generate the ciphertext/decryption key for a single attribute.*
- $\mathcal{H}(j, y, t)$, *where j is a positive integer, corresponding with a column number in the access matrix A as explained before. y and t are the same as explained above. From now on this input type will be denoted as $\mathcal{H}(0, j, y, t)$ or $\mathcal{H}(0jyt)$, to distinct from the first input type.*

The .KeyGen() algorithm is used to generate a secret key based on a set of attributes ($S_{\alpha,j}$) user u_j is entitled to. It takes a list of attributes and the MSK as inputs.

m_2: $u_j \Rightarrow CA$, request for a secret key by sending (CA) (cost: **1 message**).

c_9: CA chooses $r_1, r_2 \in_R \mathbb{Z}_p$(cost: **negligible**).

c_{10}: CA computes $sk_0 = \left(h^{b_1 r_1}, h^{b_2 r_2}, h^{r_1+r_2}\right)$ (cost: **3 exponentiations**).

c_{11}: CA chooses $\{\sigma_\alpha \in Z_p\}_{\alpha \in S_{\alpha,j}}$(cost: **negligible**).

c_{12}: CA computes $\{sk_{\alpha,t} = \mathcal{H}(\alpha 1t)^{\frac{b_1 r_1}{a_t}} * \mathcal{H}(\alpha 2t)^{\frac{b_2 r_2}{a_t}} * \mathcal{H}(\alpha 3t)^{\frac{r_1+r_2}{a_t}} * g^{\frac{\sigma_\alpha}{a_t}}\}_{\alpha \in S_{\alpha,j}, t \in \{1,2\}}$

(cost: **7 exponentiations** per attribute).

c_{13}: CA computes $\{sk_\alpha = \left(sk_{\alpha,1}, sk_{\alpha,2}, g^{-\sigma_\alpha}\right)\}_{\alpha \in S_{\alpha,j}}$ (cost: **1 exponentiation**).

c_{14}: CA picks $\sigma' \in_R \mathbb{Z}_p$(cost: **negligible**).

c_{15}: CA computes$\{sk'_t = g^{d_t} * \mathcal{H}(011t)^{\frac{b_1 r_1}{a_t}} * \mathcal{H}(012t)^{\frac{b_2 r_2}{a_t}} * \mathcal{H}(013t)^{\frac{r_1+r_2}{a_t}} * g^{\frac{\sigma'}{a_t}}\}_{t \in \{1,2\}}$ (cost: **10 exponentiations**).

c_{16}: CA sets $sk' = \left(sk'_1, sk'_2, g^{d_3} * g^{-\sigma'}\right)$(cost: **1 exponentiation**).

m_2: $CA \Rightarrow u_j$ sends (sk_0, $\{sk_\alpha\}_{\alpha \in S_{\alpha,j}}$, sk') as its key(cost: **1 message** per attribute).

Entity	Computational Complexity	Communication Complexity
CA	$O(a)$	$O(a)$
u_j	-	$O(1)$

Table 6: Complexities .KeyGen() FAME

The CA has to compute eight exponentiations for every attribute it received. This means that the computational complexity for the CA grows linearly with the amount of attributes and therefore it can be best expressed as $O(a)$. The communication complexity for the user is $O(1)$ as the user needs to send a single message to the CA to request its secret key(s) and the CA needs to send back a secret key which linearly grows with the amount of attributes embedded in said key. Keep in mind that this algorithm has to be run for every user per file he/she wants to decrypt (except if multiple files are encrypted using the same policy).

4.3.1 .Encrypt()

The .Encrypt() function is used by a data owner, user u_i, to encrypt a single message m into a ciphertext ct, embedded with an access policy $\mathcal{P}$ so that only a user which is entitled to the right attribute is able to decrypt the ciphertext. It should be noted that it is also possible to use FAME to share a symmetric encryption key which was used to encrypt a number of files, making the system much more efficient. The .Encrypt() algorithm takes public key PK of the CA, access matrix $(A, \rho(x))$ and a message m as input. Access matrix A (l rows $\times$ n columns) is part of a linear secret sharing scheme as explained in Section 4.4.3. $\rho(x)$ maps

a row A_x in the matrix A to its corresponding attribute α.
c_{17}: u_i (data owner) chooses $s_1, s_2 \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{18}: u_i computes $ct_0 = \left(H_1^{s_1}, H_2^{s_2}, h^{s_1+s_2}\right)$ (cost: **3 exponentiatons**).

c_{19}: u_i computes $\{ct_{\alpha,y} = \mathcal{H}(\rho(x)y1)^{s_1} * \mathcal{H}(\rho(x)y2)^{s_2} * \prod_{j=1}^{n}[\mathcal{H}(0jy1)^{s_1} * \mathcal{H}(0jy2)^{s_2}]^{A_{xj}}\}_{x\in A, y\in\{0,1,2\}}$ (cost: **2*n exponentiations** per attribute).

c_{20}: u_i sets $ct_\alpha = \left(ct_{\alpha,1}, ct_{\alpha,2}, ct_{\alpha,3}\right)$ as the ciphertext of a single attribute (cost: **negligible**).

c_{21}: u_i computes $ct' = T_1^{s_1} * T_2^{s_2} * m$ (cost: **2 exponentiatons**).
m_4: $u_i \Rightarrow CSP$ stores ciphertext $ct = (ct_0, \{ct_\alpha\}_{\alpha\in\mathcal{P}}, ct')$ (cost: **1 message** per attribute).

Entity	Computational Complexity	Communication Complexity
u_i	$O(a^2)$	$O(a)$

Table 7: Complexities .Encrypt() FAME

The complexity grows quadratically with the amount of attributes, as matrix A is generated based only on the attributes. Both the amount of rows and the amount of columns in matrix A are dependent on the amount of attributes and thus the computational complexity is $O(a^2)$. It needs to be noted however that this step has to be repeated for every encryption of a new message (as this might otherwise leak data about s_1 and s_2). The amount of messages (to store a ciphertext) is linearly dependent on the amount of attributes.

4.4 .Decrypt()

The .Decrypt() algorithm is used by a user u_i to decrypt a message that was encrypted using access policy $\mathcal{P}$, by using an access matrix A describing said policy. In order for this to work u_j computes the following parts of the decryption. Note that the decryptor has to be in the possession of the secret key in order to be able to decrypt any message. $sk_{0,i}$ denotes the ith part of sk_0 and $ct_{0,i}$ denotes the ith part of ct_0. If the attributes in attribute set $S_{\alpha,j}$ satisfy access policy $\mathcal{P}$, then there exist weights ω_x, such that $\sum_{x\in A}\omega_x * A_x = (1,0,\dots,0)$.
m_5: $u_j \Rightarrow CSP$, message requesting a ciphertext (cost: **1 message**).

m_6: $CSP \Rightarrow u_j$, requested ciphertext (cost: **1 message** per attribute).

c_{22}: u_j computes $num_1 = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,1}^{\omega_x}, sk_{0,1}\right) = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,1}^{\omega_x}, h^{b_1 r_1}\right)$ (cost: **1 bilinear pairing**).

c_{23}: u_j computes $num_2 = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,2}^{\omega_x}, sk_{0,2}\right) = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,2}^{\omega_x}, h^{b_2 r_2}\right)$ (cost: **1 bilinear pairing**).

c_{24}: u_j computes $num_3 = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,3}^{\omega_x}, sk_{0,3}\right) = e\left(\prod_{\alpha\in\mathcal{P}} ct_{\alpha,3}^{\omega_x}, h^{r_1+r_2}\right)$ (cost: **1 bilinear pairing**).

c_{25}: u_j computes $den_1 = e\left(sk_1' * \prod_{\alpha\in\mathcal{P}} sk_\alpha^{\omega_x}, ct_{0,1}\right) = e\left(sk_1' * \prod_{\alpha\in\mathcal{P}} sk_\alpha^{\omega_x}, H_1^{s_1}\right)$ (cost: **1 bilinear pairing**).

c_{26}: u_j computes $den_2 = e\left(sk_2' * \prod_{\alpha\in\mathcal{P}} sk_\alpha^{\omega_x}, ct_{0,2}\right) = e\left(sk_2' * \prod_{\alpha\in\mathcal{P}} sk_\alpha^{\omega_x}, H_2^{s_2}\right)$ (cost: **1 bilinear pairing**).

c_{27}: u_j computes $den_3 = e\left(sk'_3 * \prod_{\alpha \in \mathcal{P}} sk_\alpha^{\omega_x}, ct_{0,1}\right) = e\left(g^{d_3 - \sigma'} * \prod_{\alpha \in \mathcal{P}} sk_\alpha^{\omega_x}, h^{s_1+s_2}\right)$ (cost: **1 bilinear pairing**).

c_{28}: u_j obtains $m = \frac{ct' * num_1 * num_2 * num_3}{den_1 * den_2 * den_3}$ (cost: **negligible**).

Entity	Computational Complexity	Communication Complexity
u_j	$O(1)$	$O(1)$
CSP	-	$O(a)$

Table 8: Complexities .Decrypt() FAME

While the decryption process might take a bit longer if the amount of attributes increases, this only influences the amount of exponentiations and multiplications, which are far less expensive in terms of computations as bilinear pairings are. In the end any decryptor only needs six bilinear pairings to decrypt a message and therefore the computational complexity of decrypting one ciphertext is constant and thus is $O(1)$. The CSP sends a ciphertext, upon request from user u_j, to u_j. As the size of the ciphertext linearly increases with the amount of attributes used in its encryption this means that the communication complexity of the CSP is $O(a)$ per ciphertext.

4.5 Advantages & Disadvantages

First the requirements will be mentioned and whether FAME meets them in Section 6.5.1. After that, based on the requirements analysis, a list of advantages and a list of disadvantages will be mentioned in Sections 6.5.2 and 6.5.3.

4.5.1 Requirement analysis

This section will briefly mention the requirements as were listed in Section 5.2. Each paragraph will mention a requirement, whether it is strong or weak and whether FAME8 meets that requirement.

CP-ABE (Strong) - CP-ABE is a form of ABE where the access policy $\mathcal{P}$ is embedded into the ciphertext encryption so that only users entitled to the right attributes are able to decrypt the encrypted file. FAME is a CP-ABE scheme where the encryption of data items includes the use of LSSS to embed an access policy into the ciphertext.

Collusion resistance (Strong) - Collusion resistance ensures that different users are not able to combine their attributes/keys to decrypt a ciphertext if none of the users is able to decrypt the file on its own. In FAME every ABE key, which in fact are six general keys and a key for every attribute necessary for the decryption of a file, contains at least some of the values, which are random per user, r_1, r_2, σ_α (for every attribute α) and σ'. If these ABE keys are used together in the decryption of a ciphertext these keys combined nullify the random values, ensuring the original secret can be retrieved. Inserting a secret key of another user does not have this property and so FAME is collusion resistant.

Access revocation (Strong) - An access revocation mechanisms allows for revocation of an attribute, the revocation of an attribute of a user and/or a method to update the access policy in the encryption of a ciphertext. This ensures certain users or users with certain attributes are

no longer able to decrypt a ciphertext. This can be done either mathematically or system wise. Unfortunately FAME does not have a method to revoke user access.

Scalability (Strong) - Scalability ensures the system will not run out of computational resources as new computational resources can be added to the system during run time. This ensures the system is not limited in the amount of attributes, users or attribute authorities it can handle. ASCLEPIOS aims to design a system that makes it possible for on-person medical devices to continuously upload (encrypted) data to a CSP, meaning that the encryption phase is the most important operation to look at to be able to say anything about its scalability as these on-person devices, such as pacemakers, are usually very limited in the amount of computations they can handle. The complexity of the encryption step grows linearly with the amount of attributes used to encrypt a message and the most significant computation it has to perform is an exponentiation. If used in a hybrid system where a content key κ is used to (symmetrically) encrypt data items and the content key is shared using FAME it depends on the amount of attributes used to encrypt κ if applicable. FAME allows for the addition of users and attributes to the system after its original initialization. Existing ciphertexts are not updated when a new attribute is added.

Multiple Authorities (Weak) - Having multiple Attribute Authorities (AAs) removes the need for trust in a single CA as each of the AAs individually decides whether a user is entitled to a specific attribute α or not. If a ciphertext is encrypted using the attribute keys of multiple AAs it would need all of the authorities to collude together for them to be able to decrypt the ciphertext. This potentially increases security. FAME, however, does not describe a method to decentralize attribute management and has a single CA that is needed for the generations of decryption key.

Regranting Access (Weak) - Regranting access is only applicable if the system incorporates a revocation method. Regranting access ensures attribute/user revocation does not have to be permanent. The simplest way to implement such a functionality is by giving revoked users new attributes that allow for decryption of a ciphertext once again. FAME does not mention a revocation mechanism and thus regranting access is not mentioned as well.

Multiple Access Controls (Weak) - Multiple Access Controls is not trivial in Attribute-Based Encryption schemes, but might just as well be necessary for real-life usage of any ABE scheme. Usually ABE schemes only deal with read access, meaning that once a user has decrypted a file it is only able to view the contents of said file. When dealing with medical files it might, for example, be necessary for a medical practitioner to be able to add a comment to a data entry, meaning that he should be able to change the ciphertext, i.e. he should have write access to the file. FAME does not include the possibility to make use of different access controls.

4.5.2 Advantages

User Addition - FAME does not need to define all users during the initialization process which should be a requirement for any such system to be deployable in a real-life scenario as not all user of a system can be known beforehand.

4.5.3 Disadvantages

Full trust in CA - In FAME, as there are no Attribute Authorities, the CA functions as the entity that sets up the system and is therefore an " all powerful " entity. It has access to all keys and attributes and therefore should be fully trusted.

Revocation not mentioned - In the paper regarding FAME, no revocation method is mentioned. While a revocation method should not necessarily depend on mathematical structures, but might also be implemented system wise, the FAME paper does not mention revocation at all.

5 DAC-MACS: Data Access Control for Multi-Authority Storage Systems

5.1 Introduction

DAC-MACS[10] was first described by Yang et al. in 'Effective Data Access Control for Multi-Authority Cloud Storage Systems'. The paper was published at INFOCOM 2013 and was written by Yang et al. The authors propose a Multi-Authority CP-ABE scheme that achieves both backward and forward security and is efficient as it outsources part of the decryption process to the CSP without allowing the CSP to learn anything about the actual message that was decrypted. It is noted that this chapter describes the 2013 paper with the title DAC-MACS, published at INFOCOM 2013.

The system has a CA that is in charge of distributing global identifiers uid and aid, to registered users and attribute authorities, respectively. Next to that the CA also generates a global secret/public key for new users. The CA " *is not involved in any attribute management* " [10].

Attribute Authorities (AAs) are " independent attribute authorities responsible for issuing, revoking and updating user's attributes. " *Each* AA " can manage an arbitrary number of attributes' and 'has full control over the structure and semantics of its attributes. " *Each* AA_k *generates its own public attribute keys and secret keys for each user 'reflecting their attributes* " [10].

The CSP stores (encrypted) data of certain users and allows other users to request specific data. Part of the decryption of a stored ciphertext is done by the CSP to also allow less powerful devices to retrieve data from the server, without compromising the data security/privacy, meaning that the CSP does not learn anything about the underlying plaintexts.

Data owners encrypt specific data items using symmetric encryption with a so-called content key. This content key is then encrypted using an access policy $\mathcal{P}$, thereby indicating which attributes together should be able to obtain the content key for decryption of the original data items. This design significantly increases decryption time as symmetric encryption/decryption is much faster than asymmetric encryption/decryption. The encrypted data and the encrypted content key are sent to the CSP which allows every legal user of the system to retrieve them as users which do not have the right attributes are not able to decrypt the encrypted content key and then decrypt the data.

The DAC-MACS protocol uses bilinear pairings for encryption and decryption of certain ciphertexts. First of all every protocol step will be analysed in Section 7.2 based on the communication cost and the computation cost of the protocol steps for each entity in the system.

The following (large) substitutions were made while describing the DAC-MACS ABE scheme. Keep in mind that this table only includes substitutions which were specific for this scheme:

Notation in this report	Notation in the original paper
σ_{CA}	sk_{CA} (The signing/secret key of the Central Authority)
v_{CA}	vk_{CA} (The verificative key of the Central Authority)

ϵ_k	α_k (Part of secret key of AA_k)
$t_{k,j}$	$t_{j,k}$ (Changed to better reflect it being generated by AA_k, used in key generation)

Table 9: Notation substitutions specific to DAC-MACS

5.2 Efficiency

5.2.1 .Setup()

The .Setup() algorithm exists of two parts, namely .CASetup(), which sets up a CA and uses sub algorithms .UserRegistration() to register a user and .AARegistration() to set up a new Attribute Authority. Both users and AAs are set up during the system initialization and so no new users or AAs can be added later on in the system's lifetime. The .AASetup() algorithm is used to setup a new Attribute Authority at the system initialization. Both algorithms need groups $\mathbb{G}$ (with generator g) and $\mathbb{G}_T$, both of prime order p. A Type-I bilinear pairings $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is used in some of the calculations. The system also defines a hash function $\mathcal{H}$, which maps random bit strings to element in $\mathbb{G}$ ($\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{G}$).

5.2.1.1 .CASetup()

The .CASetup() algorithm is used to set up the (centrally trusted) CA. The CA takes security parameter λ as input and outputs master key MSK, system parameter SP and a singing and verificative key pair (σ_{CA}, v_{CA}). For each user uid_j it generates a global public/secret key pair (GPK_{uid_j}, GSK_{uid_j}) and a certificate $Cert(uid_j)$.

c_1: CA chooses large prime p (and thus chooses $\mathbb{Z}_p$) (cost: **negligible**).

c_2: CA chooses generator $g \in_G \mathbb{G}$, where $\mathbb{G}$ is an algebraic group op order p (cost: **negligible**).

c_3: CA computes bilinear pairing $e(g,g)$ (cost: **1 bilinear pairing**).

c_4: CA chooses hash function $\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{G}$ (cost: **negligible**).

c_5: CA chooses value $a \in_R \mathbb{Z}_p$ uniformly at random as its Master Secret Key (MSK) (cost: **negligible**).

c_6: CA generates System Parameter (SP) as g^a (cost: **1 exponentiation**).

c_7: CA chooses its signing key and computes its verificative key (σ_{CA}, v_{CA}) (cost: **1 exponentiation**).

m_1: CA publishes p, g, $\mathbb{G}$, $\mathcal{H}$, $e(g,g)$ $SP = g^a$ (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
CA	$O(1)$	$O(1)$

Table 10: Complexities .CASetup() DAC-MACS

Both complexities are $O(1)$ as they do not grow with the amount of users, attributes and ciphertexts as the CA is only set up once.

5.2.1.2 .UserRegistration() (sub-algorithm of .CASetup())

In this step the user registers itself to the CA during the system initialization. User u_j obtains its global (user) identifier uid_j, global public key GPK_{uid_j} and global secret key GSK_{uid_j}.
m_2: User $u_j \Rightarrow CA$, request user identifier uid_j and its user keys (cost: **1 message**).

c_8: CA assigns global unique user identity (uid_j) to u_j (cost: **negligible**).

c_9: CA chooses $u_{uid_j} \in_R \mathbb{Z}_p$ and computes $GPK_{uid_j} = g^{u_{uid_j}}$ (cost: **negligible**).

c_{10}: CA chooses $z_{uid_j} \in_R \mathbb{Z}_p$ and computes $GSK_{uid_j} = z_{uid_j}$ (cost: **1 exponentiation**).

c_{11}: CA generates certificate $Cert(uid_j) = E_{\sigma_{CA}}(uid_j, u_{uid_j}, g^{\frac{1}{z_{uid_j}}})$, by using its signing key (sk_{CA}) (cost: **1 exponentiaton**).
m_3: $CA \Rightarrow u_j$, a message containing: GPK_{uid_j}, GSK_{uid_j} and $Cert(uid_j)$ (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
CA	$O(1)$	$O(1)$
u_j	-	$O(1)$

Table 11: Complexities .UserRegistration() DAC-MACS

Every user has to request an uid_j and their global key pair by sending a single message per user to the CA, therefore the complexity of every user in this protocol step is $O(1)$ as it does not grow with to the amount of attributes, users or ciphertexts. The CA has to compute several things, but everytime this algorithm is executed the amount of computations and communications is constant for the CA. It should be noted that this algorithm is executed for every user uid_j and thus both the computational and communication complexity of the CA become $O(u)$ as they grow linearly with the amount of users.

5.2.1.3 .AARegistration() (subalgorithm of .CASetup())

In this step an Attribute Authority (AA_k) registers itself to the CA during system initialization.

m_4: $AA_k \Rightarrow CA$, AA_k requests its global identifier aid_k (cost: **1 message**).

c_{12}: CA assigns global authority identity aid_k to AA_k (cost: **negligible**).

m_5: $CA \Rightarrow AA_k$, aid_k (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity

CA	-	$O(1)$
AA_k	-	$O(1)$

Table 12: Complexities .AARegistration() DAC-MACS

Each AA_k has a communication complexity of $O(1)$ as each one needs to send a single message to CA requesting to be assigned its own aid_k. The CA has a communication complexity of $O(1)$ as the amount of messages send by the CA to AA_k is constant. It should be noted that the communication complexity for the CA linearly grows with the amount of attribute authorities in the system, and so in the overall system increases to $O(k)$.

5.2.1.4 .AASetup()

This step sets up an Attribute Authority. Each Attribute Authority AA_k runs algorithm .AARegistration() to be assigned a global aid. S_{α_k} denotes set of all attributes managed by authority AA_k.

c_{13}: AA_k chooses $\epsilon_k, \beta_k, \gamma_k \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{14}: AA_k sets authority secret key $SK_k = (\epsilon_k, \beta_k, \gamma_k)$ (cost: **negligible**).

c_{15}: AA_k chooses attribute version key $\{VK_\alpha = v_\alpha \in_R \mathbb{Z}_p\}_{\alpha \in S_{\alpha_k}}$ (cost: **negligible**).

c_{16}: AA_k generates public attribute key $\{PK_\alpha = (g^{v_\alpha}\mathcal{H}(\alpha))^{\gamma_k}\}_{\alpha \in S_{\alpha_k}}$ (cost: **2 exponentiations** per attribute).

c_{17}: AA_k generates authority public key as $PK_k = (e(g,g)^{\epsilon_k}, g^{\frac{1}{\beta_k}}, g^{\frac{\gamma_k}{\beta_k}})$ (cost: **3 exponentiations**).

m_6: AA_k publishes PK_k and $\{PK_\alpha\}_{\alpha \in S_{\alpha_k}}$ (cost: **1 message** per attribute).

Entity	Computational Complexity	Communication Complexity
AA_k	$O(a)$	$O(a)$

Table 13: Complexities .AASetup() DAC-MACS

Each AA_k has a computational complexity of $O(a)$ as it grows linearly with the amount of attributes of the AA_k. It has the communication complexity of $O(a)$ as it needs to publish a public key for every attribute it is manages. This means that both complexities grow linearly with the amount of attribute an authority AA_k manages. It should be noted that every AA_k should run this algorithm at system initialization, and thus the total complexity is $O(a * k)$.

5.2.2 .SKeyGen()

In this step an AA_k generates a secret key for every attribute requesting user u_j is entitled to. $S_{\alpha_k,j}$ denotes the set of all attributes managed by AA_k user u_j is entitled to (according to AA_k).

m_7: User $u_j \Rightarrow AA_k$, message containing $Cert(uid_j)$ (cost: **1 message**).

c_{18}: AA_k verifies $E_{\sigma_{CA}}(uid_j, u_{uid_j}, g^{\frac{1}{z_{uid_j}}})$ in $Cert(uid_j)$ using its verification key v_{CA} (cost: **1 exponentiation**).

c_{19}: AA_k authenticates user u_j and otherwise aborts (cost: **negligible**).

c_{20}: AA_k assigns a set of attributes $S_{\alpha_k,j}$ to u_j based on its identity (cost: **negligible**).

c_{21}: AA_k chooses $t_{k,j} \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{22}: AA_k runs secret key generation algorithm .SKeyGen() and obtains user's secret key $SK_{k,j} = (K_{k,j}, L_{k,j}, R_{k,j} = g^{a*t_{k,j}}, \{K_{\alpha,j}\}_{\alpha \in S_{\alpha_k,j}})$, where:

$K_{k,j} = g^{\frac{\epsilon_k}{z_{uid_j}}} * g^{a*u_{uid_j}} * g^{\frac{a}{\beta_k} t_{k,j}}$ (cost: **3 exponentiations**).

$L_{k,j} = g^{\frac{\beta_k}{z_{uid_j}} t_{k,j}}$ (cost: **2 exponentiations**).

$R_{k,j} = g^{a*t_{k,j}}$ (cost: **1 exponentiations** per attribute).

$\{K_{\alpha,j} = g^{\frac{\beta_k * \gamma_k}{z_{uid_j}} t_{k,j}} * (g^{v_\alpha} * \mathcal{H}(\alpha))^{\gamma_k * \beta_k * u_{uid_j}})\}_{\alpha \in S_{\alpha_k,j}}$ (cost: **2 exponentiations** per attribute).

m_8: $AA_k \Rightarrow u_j$, messages containing secret attribute keys (cost: **1 message** per attribute).

Entity	Computational Complexity	Communication Complexity
AA_k	$O(a)$	$O(a)$
u_j	-	$O(1)$

Table 14: Complexities .SKeyGen() DAC-MACS

Each time this protocol is called user u_j has a complexity of sending and receiving 1 message, step, meaning that its computational complexity is $O(1)$. This, however, is only if this protocol is executed with a single user and Attribute Authority, otherwise the communication of the user u_j is linearly dependent with the amount of Attribute Authorities ($O(k)$). The Attribute Authority AA_k then computes a secret key for every attribute the user u_j is entitled to, meaning that its complexity is $O(a)$, both computationally and communication wise. However, the .SKeyGen() algorithm is called by every user u_j for each AA_k, meaning that each AA_k has a computation and communication complexity of $O(a * u)$ and each user u_j has a communication complexity of $O(k)$. Both the overall computation and communication complexity grow with the amount of users, the amount of attribute authorities and the amount of attributes each authority manages, denoted as of $O(k * a * u)$.

5.2.3 .Encrypt()

This .Encrypt() algorithm is used to encrypt a set of files owned by user u_i (data owner) using content key κ. The .Encrypt() function will also encrypt the content key. An access policy $\mathcal{P}$ is converted to an access matrix $(A, \rho(x))$, where $\rho(x)$ maps a row x in A to its corresponding attribute α. S_A denotes the set of all Attribute Authorities involved in the encryption of content key κ.

c_{23}: u_i generates content key κ (cost: **negligible**).

c_{24}: u_i generates access matrix (A, ρ) (size $l \times n$), according to access policy $\mathcal{P}$ (cost: **negligible**).

c_{25}: u_i symmetrically encrypts files $\{f_1, \ldots, f_z\}$ and obtains $\{ct_1, \ldots, ct_z\}$ using key κ (cost: **z symmetric encryptions**).

c_{26}: u_i chooses secret $s \in \mathbb{Z}_p$ (cost: **negligible**).

c_{27}: u_i chooses $\{v_2, \ldots, v_n\} \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{28}: u_i sets $v = (s, v_2, \ldots, v_n)$ (cost: **negligible**).

c_{29}: u_i computes $\lambda_x = v * A_x$ for all rows in A (cost: **negligible**).

c_{30}: u_i chooses $\{r_\alpha\}_{\alpha \in \mathcal{P}} \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{31}: u_i computes ciphertext $ct = (C, C', C'', \{C_\alpha\}_{\alpha \in \mathcal{P}}, \{D_{1,\alpha}\}_{\alpha \in \mathcal{P}}, \{D_{2,\alpha}\}_{\alpha \in \mathcal{P}})$, where:

$C = \kappa * (\prod_{k \in S_A} e(g, g)^{\epsilon_k})^s$ (cost: **k multiplications**).

$C' = g^s$ (cost: **1 exponentiation**).

$C'' = g^{\frac{s}{\beta_k}}$ (cost: **1 exponentiation**).

$\{C_\alpha = g^{\alpha * \lambda_x} * ((g^{v_\alpha} * H(\alpha))^{\gamma_k})^{-r_\alpha}\}_{\alpha \in \mathcal{P}}$ (cost: **2 exponentiations** per attribute).

$\{D_{1,\alpha} = g^{\frac{r_\alpha}{\beta_k}}\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

$\{D_{2,\alpha} = g^{-\frac{\gamma_k}{\beta_k} r_\alpha}\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

m_9: $u_i \Rightarrow CSP$, all the ciphertexts that were encrypted using content key κ (cost: **z messages**).

m_{10}: $u_i \Rightarrow CSP$, ct (cost: **l messages**).

Entity	Computational Complexity	Communication Complexity
u_i	$O(a)$	$O(a)$

Table 15: Complexities .Encrypt() DAC-MACS

User u_i encrypts all the files, using content key κ, but as this is symmetric encryption it is relatively fast and does not contribute too much to the overall complexity. The exponentiations in this step however are non-negligible operations and thus attribute to the computational complexity, which is $O(a)$, meaning that the amount of computations linearly increases with the amount of attributes (in policy $\mathcal{P}$). The same holds for the communication complexity of u_i. It should be noted that this step has to be repeated for every unique access policy $\mathcal{P}$ a data owner uses to encrypt a content key κ.

5.2.4 .Decrypt()

Algorithm .Decrypt() exists of two sub algorithms. First a user asks the CSP (with more computational capabilities) to generate a token using the secret attribute keys of the user in sub algorithm .TKGen() which is then sent to the u_j. The user then uses its secret value z_{uid_j} to retrieve content key κ.

5.2.4.1 .TKGen()

User u_j sends its secret keys $\{SK_{k,j}\}_{k\in S_A}$ to the server which generates a token thereby outsourcing part of the decryption of ciphertext CT. S_A denotes all involved Attribute Authorities, whereas S_{A_k} denotes the set of attributes of AA_k. N_A denotes the number of involved AAs.

m_{11}: $u_j \Rightarrow CSP$, $\{SK_{k,j}\}_{k\in S_A}$ (cost: **1 message** per attribute).

c_{32}: CSP calculates $TK = \prod_{k\in S_A} \frac{e(C\prime,K_{k,j})*e(R_{k,j},C'')^{-1}}{\prod_{\alpha\in S_{A_k}}(e\left(C_\alpha,GPK_{uid_j}\right)*e\left(D_{1,\alpha},K_{\alpha,j}\right)*e\left(D_{2,\alpha},L_{k,j}\right))^{\omega x*N_A}}$

$= \frac{e(g,g)^{a*u_{uid_j}*s*N_A}*\prod_{k\in S_A} e(g,g)^{\frac{\epsilon_k}{z_{uid_j}}s}}{e(g,g)^{u_{uid_j}*a*N_A*\sum_{x\in A}\lambda_x\omega_x}} = \prod_{k\in S_A} e\,(g,g)^{\frac{\epsilon_k}{z_{uid_j}}s}$ (cost: **2 bilinear pairings** per attribute and **3 bilinear pairings** per attribute belonging to Attribute Authority AA_k).

m_{12}: $CSP \Rightarrow u_j$, TK (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
u_i	-	$O(a)$
CSP	$O(a)$	$O(1)$

Table 16: Complexities .TKGen() DAC-MACS

First of all user u_j sends all its keys to the CSP at a communication complexity of $O(a)$. The user sends a key per Attribute Authority, but as these keys exist of a sub key per attribute per AA_k, this means that the amount of messages linearly increases with the amount of attributes in the system. The CSP then generates a decryption token TK. The CSP can optimize its calculation of 2 bilinear pairings per attribute and 3 bilinear pairings per attribute belonging to Attribute Authority AA_k into 5 bilinear pairings per attribute and so the computational complexity linearly increases with the amount of attributes used for encrypting content key κ. The CSP in the end sends TK to user u_j at a constant communication complexity ($O(1)$). Keep in mind that the .TKGen() algorithm is run every time a user want to decrypt a content key.

5.2.4.2 .UserDecrypt()

User u_j obtains content key κ by decrypting ciphertext CT using TK and its global secret key GSK_{uid_j}.

c_{33}: $\kappa = CT/TK^{z_{uid_j}}$ ($z_{uid_j} = GSK_{uid_j}$) (cost: **1 exponentiation**).

Entity	Computational Complexity	Communication Complexity
u_i	$O(1)$	-

Table 17: Complexities .UserDecrypt() DAC-MACS

u_j has a computation complexity of $O(1)$ as the complexity does not grow with any of the parameters of the system, as the computation is only a single (modular) division.

5.2.5 .Update()

An attribute α user u_μ is entitled to is revoked by its authority AA_k .Update(), meaning that both the secret key and the ciphertext need to be updated. This step guarantees backwards as well as forward security. The .Update() algorithms is called by an AA_k if it decides a user no longer should be entitled to a specific attribute α.

5.2.5.1 .UKeyGen()

.UKeyGen() is run by AA_k to generate a user's key update key $KUK_{\alpha,j}$ for every non-revoked user en the ciphertext update key CUK_α.

c_{34}: AA_k generates a new attribute version key $VK'_\alpha = v'_\alpha \in_R \mathbb{Z}_p$ (cost: **negligible**).

c_{35}: AA_k computes Attribute Update Key $AUK_\alpha = \gamma_k(v'_\alpha - v_\alpha)$ (cost: **negligible**).

c_{36}: AA_k computes user's Key Update Key $\{KUK_{\alpha,j} = g^{u_{uid_j}\beta_k AUK_\alpha}\}_{u_j \in S_U, u_j \neq u_\mu}$ (cost: **3 exponentiations** per user).

c_{37}: AA_k computes Ciphertext Update Key $CUK_\alpha = \frac{\beta_k}{\gamma_k} * AUK_\alpha$ (cost: **negligible**).

c_{38}: AA_k updates public attribute key $PK'_\alpha = PK_\alpha * g^{AUK_\alpha}$ (cost: **1 exponentiation**).

m_{13}: AA_k publishes PK'_α as the replacement of PK_α (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
AA_k	$O(u)$	$O(1)$

Table 18: Complexities .UKeyGen() DAC-MACS

The Attribute Authority computes Key Update Keys $KUK_{\alpha,j}$ for each non-revoked user u_j, meaning that the complexity is linearly dependent on the amount of users and thus is denoted as $O(u)$. It then broadcasts that one of its attributes has been changed/revoked at cost $O(1)$. It should be noted that this algorithm only revokes an attribute of one user, but could be optimized to allow for revocation of one single attribute for multiple users at the same cost.

5.2.5.2 .SKUpdate()

For every user $u_j \neq u_\mu$ that has not been revoked AA_k has generated Key Update Key ($KUK_{\alpha,j}$) and sends it to them. Each user $u_j \neq u_\mu$ updates it secret key. S_U denotes the set of all users.

m_{14}: $AA_k \Rightarrow u_j$ $(u_j \in S_U, u_j \neq u_\mu)$, message containing $KUK_{\alpha,j}$ associated with uid_j (cost: **u messages**).

c_{39}: Each $u_j \neq u_\mu$ changes K_{α_j} in ct to $K'_{\alpha,j} * KUK_{\alpha,j}$ for changed attribute α (cost: **negligible** per user).

Entity	Computational Complexity	Communication Complexity
AA_k	-	$O(u)$
u_j	$O(1)$	-

Table 19: Complexities .SKUpdate() DAC-MACS

AA_k sends a single message to every user containing their specific Key Update Key (KUK) at communication cost $O(1)$. User u_j now computes a single multiplication to obtain its updated Secret Key for (revoked) attribute x. The computation cost per user is constant and thus $O(1)$. Every time an attribute gets revoked every user updates its key corresponding to said attribute.

5.2.5.3 .CTUpdate()

Algorithm .CTUpdate() is run by the CSP to update every ciphertext associated with (revoked) attribute α.

m_{15}: $AA_k \Rightarrow CSP$, CUK_α (cost: **1 message**).

c_{40}: CSP changes C_α in ct to $C_\alpha{}' = C_\alpha * D_{2,\alpha}^{CUK_\alpha}$ (cost: **1 exponentiation**).

Entity	Computational Complexity	Communication Complexity
AA_k	-	$O(1)$
CSP	$O(ct)$	-

Table 20: Complexities .CTUpdate() DAC-MACS

AA_k sends a single message to the CSP containing a ciphertext update key with constant complexity $O(1)$. The CSP has to update every ciphertext (containing symmetric encryption key κ). The amount of ciphertexts that need to be updated are dependent on the amount of users that use that attribute for encryptions and the amount of different policies used per user. Therefore, the computational complexity of the CSP has been set at $O(ct)$, where ct is dependent on the amount of users that use that specific attribute and the amount different policies the users each use.

5.3 Advantages & Disadvantages

First the requirements will be mentioned and whether DAC-MACS[10] meets them in Section7.3.1. After that, based on the requirements analysis, a list of advantages and a list of disadvantages will be mentioned in Sections 7.3.2. Requirement analysis

This section will briefly mention the requirements as were listed in Section 5.2. Each paragraph will mention a requirement, whether it is strong or weak and whether DAC-MACS meets that requirement.

CP-ABE (Strong) - CP-ABE is a form of ABE where the access policy $\mathcal{P}$ is embedded into the ciphertext encryption so that only users entitled to the right attributes are able to decrypt the encrypted file. DAC-MACS is a CP-ABE scheme where the encryption of data items includes the use of LSSS (Section 4.4.3) to embed an access policy into the ciphertext.

Collusion resistance (Strong) - Collusion resistance ensures that different users are not able to combine their attributes/keys to decrypt a ciphertext if none of the users is able to decrypt the file on its own. In DAC-MACS a secret key for an attribute is made up of three general keys that all contain a random value, while one of them also includes some identifier for a user. The secret key for an attribute also contains the random value and the user identifier. The users therefore cannot collude by combining the keys for different attributes.

Access revocation (Strong) - An access revocation mechanisms allows for revocation of an attribute, the revocation of an attribute of a user and/or a method to update the access policy in the encryption of a ciphertext. This ensures certain users or users with certain attributes are no longer able to decrypt a ciphertext. This can be done either mathematically or system wise. DAC-MACS incorporates an interesting revocation method. If an attribute should be revoked for a user the corresponding Attribute Authority first updates the attribute, by choosing a new version key, which in its turn is used to update the Attribute Update Key (AUK). The AUK is used to compute a Key Update Key (KUK) for every non-revoked user and a Ciphertext Update Key (CUK). The AA also updates the public attribute key using AUK. The KUKs are sent to the users who each use their KUK to update the secret attribute key using a single multiplication. The CUK is used by the CSP to update all of the ciphertexts that were encrypt using the attribute. The revoked user does not receive an update key and is not able to update its attribute key(s) accordingly.

Scalability (Strong) - Scalability ensure the system will not run out of computational resources as new computational resources can be added to the system during runtime. This ensures the system is not limited in the amount of attributes, users or attribute authorities it can handle. ASCLEPIOS aims to design a system that makes it possible for on-person medical devices to continuously upload (encrypted) data to the CSP and thus the encryption phase is the most important operation to look at to be able to say anything about its scalability as these on-person devices, such as pacemakers, are usually very limited in the amount of computations they can handle. The complexity of the encryption step grows linearly with the amount of attributes in DAC-MACS, but does require bilinear pairings and four exponentiations per encryption, which might be too expensive for devices with low computational power. Benchmarking might explain how much attributes would be feasible for the system. DAC-MACS does describe some sort of hybrid system as a symmetric content key κ is used for fast encryption/decryption of data items while the heavier ABE scheme is used to encrypt and share the content key. It should also be noted that DAC-MACS assumes all users and attribute authorities are known when initializing the system, although a few alterations to the scheme might allow for the addition of users and AAs after system initialization.

Multiple Authorities (Weak) - Having multiple Attribute Authorities (AAs) removes the need for trust in a Central Authority as each of the AAs individually decides whether a user is entitled to a specific attribute α or not. If a ciphertext is encrypted using the attribute keys of multiple AAs it would need all of the authorities to collude together for them to be able to decrypt the

ciphertext. This potentially increases security and might be more useful in a real-life setting. DAC-MACS describes a method to include multiple AAs that assign users different attributes.

Regranting Access (Weak) - Regranting access is only applicable if the system incorporates a revocation method. Regranting access ensures attribute/user revocation does not have to be permanent. The simplest way to implement such a functionality is by giving revoked users new attributes that allow for decryption of a ciphertext once again. Regranting Access has not been described in DAC-MACS, but a Key Update Key for an attribute of a revoked user can just as easily be calculated when necessary, thereby allowing a revoked user to once again decrypt ciphertexts encrypted using that attribute (assuming it is entitled to all other attributes necessary for decryption as well).

Multiple Access Controls (Weak) - Multiple Access Controls is not trivial in Attribute-Based Encryption schemes, but might just as well be necessary for real-life usage of any ABE scheme. Usually ABE schemes only deal with read access, meaning that once a user has decrypted a file it is only able to view the contents of said file. When dealing with medical files it might, for example, be necessary for a medical practitioner to be able to add a comment to a data entry, meaning that he should be able to change the ciphertext, i.e. he should have write access to the file. DAC-MACS does not include the possibility to make use of different access controls.

5.3.1 Advantages

Multiple Attribute Authorities - One of the advantages of the DAC-MACS scheme is that the authors describe a system that allows for multiple AAs. Using multiple AAs is seen as an advantage as no single authority is able to decrypt a ciphertext if it was encrypted using the attributes of multiple authorities. Furthermore, in a (European) Union wide setting, creating a CA that issues attributes for all different hospitals, governmental regulation agencies in the healthcare domain and other medical instances is a difficult problem on its own.

Revocation Method & Regranting Access - DAC-MACS describes an interesting method for the revocation of the attribute of a single user. The revocation process does not require all users to be online during the process, only the CSP and the revoking AA need to be online for this process. Other users can just request their own update key when needing it for decrypting an item. The way DAC-MACS implements revocations also allows for regranting access to an 'attribute' later on.

Outsourced Decryption - Decryption in normal use cases should not be outsourced as it usually allows the decryptor to learn information regarding the contents of the ciphertexts. However, the authors of DAC-MACS found a clever way in which part of the decryption is executed by the CSP, but a secret element belonging to the user wanting to decrypt the file is still necessary to obtain the original message sent using the DAC-MACS scheme. As this means that the CSP cannot learn anything about the contents of the ciphertext, this is seen as an advantage of the DAC-MACS scheme.

5.3.2 Disadvantages

No addition of authorities/users - DAC-MACS does not support the addition of AAs or users after the system has been initialized. As the total user population of a system cannot be known beforehand it is seen as a disadvantage that the authors of DAC-MACS mention 'that all users/AAs should be set beforehand'. It is noted however that, while the authors state that all users and AAs are known at the set up phase, it is believed that the scheme can also be used in a setting where users and Attribute Authorities are added later in the life-time of the system.

Not Backwards Secure - The revocation process of DAC-MACS only influences the decryption of the content key κ. After revoking an attribute for a specific user this user is no longer capable of obtaining κ. But, if κ has already been decrypted by that user then the user is still able to decrypt all data items that were symmetrically encrypted using κ. The system is therefore not backwards secure. It should also be noted that κ should be changed after revoking a user as, if that user already obtained κ, it is also able to decrypt newly added data items, uploaded after being revoked.

Update collusion attacks - DAC-MACS is vulnerable to collusion attacks between users, but only in the revocation process. Consider user u_j receives User Update Key $UUK_{\alpha,j} = g^{u_{uid_j} * AUK_\alpha * \beta_k}$ for attribute α which was updated and user u_j. In this formula u_{uid_j} denotes the identity of the user, AUK_α denotes the Attribute Update Key for attribute α. β_k denotes one of the secret values of Attribute Authority AA_k. Every User Update Key can therefore be expressed as $UUK_{\alpha,b} = g^{(AUK_\alpha * \beta_k) * b}$ for every user u_b, which for simplicity will be expressed as g^{ab} from now on. As b is known to all users this means that it is possible to obtain g^a by finding the modular inverse of b (b^{-1}) and calculating $(g^{ab})^{b^{-1}} = g^a$. g^a in this case denotes $g^{AUK_\alpha \beta_k}$. So, every non-revoked user can find g^a and share it with users which where revoked for this attribute, meaning such a revoked user u_i can simply calculate a valid User Update Key $UUK_{\alpha,i} = (g^a)^{u_{uid_i}}$ ($u_i \neq u_j$). This means that any u_i can collude with any u_j for a specific attribute (where u_i is revoked for and u_j is not) to obtain a valid secret key for said attribute and is able to decrypt ciphertexts if it colludes with u_j. This only works if u_i previously was entitled to attribute α and thus at one point in time has received a first secret key for that attribute from AA_k.

Another collusion attack is possible between a revoked user u_i and the CSPThis attack uses the CUK_α obtained from the honest-but-curious $CSP$and allows u_i to get the old version of a ciphertext, so that it can be decrypted by u_i using its non-updated secret key(s). The authors of this follow-up paper therefore conclude that DAC-MACS does not meet the backwards security requirement the authors state in [11].

6 RD-ABE: Revocable and Decentralized Attribute-Based Encryption

6.1 Introduction

RD-ABE[12] is a scheme which was published in The Computer Journal (2016). It was written by Cui and Deng at the Singapore Management University in Singapore. The authors propose a revocable and decentralized ABE scheme, meaning that there is no CA. The system makes use of Attribute Authorities (AAs), which do not have any communication with each other, except for a few global public parameters set at the initialization of the system, a global time element and a list of global identifiers of users. Any party can become an AA by generating its own authority public/secret key pair. Cui and Deng[12] do not mention a server CSP where ciphertexts can be stored for anyone to access, but for the sake of consistency we included it into this report as an actor in the system as other schemes do include this actor as well and ABE usually are used in Cloud settings. The authors describe a scheme which uses indirect revocation to revoke a user from decrypting a ciphertext. The scheme embeds a time element in certain elements within the scheme to ensure keys have to be updated to new time periods every period of time, otherwise encryption will fail.

Each AA_k has a set of attributes as its own attribute universe (S_{α_k}). For each of these attributes AA_k generates a public/secret key pair and publishes the public key. Next to that, the AAs generate a secret attribute key per attribute per authorized user which includes a time period and they send it to the corresponding user over a secure channel. At the start of a new time period the AAs generate new secret attribute keys for every attribute/user pair. Because the hash of a global identifier (uid_j) of a corresponding user u_j is embedded into these secret attribute keys, keys for different attributes of different users cannot be combined to decrypt a ciphertext, thereby preventing collusion attacks. This ensures that only users that adhere to an access policy $\mathcal{P}$ are able to decrypt a ciphertext.

A data owner u_i encrypts a ciphertext using the public keys of the attributes according to an access policy $\mathcal{P}$. When another user u_i wants to decrypt a ciphertext it uses the keys it received from the AAs for attributes which are included in the access policy $\mathcal{P}$ and obtains message m. It should be noted that a data owner u_i has to update the ciphertext as well, as a time element is embedded into the ciphertext. This also means that a previous ciphertext needs to be deleted by the CSP.

The RD-ABE scheme uses bilinear pairings in the encryption and decryption of ciphertexts and uses LSSS (Section 4.4.3) to embed access policies. The efficiency of the protocol will be explained in Section 8.2 based on the communication cost and the cost of the computations of different protocol steps. In Section 8.3 the advantages and disadvantages of RD-ABE will be elaborated upon.

The following (large) substitutions were made while describing the RD-ABE scheme. Keep in mind that this table only includes substitutions which were specific for this scheme:

Notation in this report	Notation in the original paper
ϵ_α	α_i (Denoting a secret value for an attribute α)

6.2 Efficiency

6.2.1 .GSetup()

The .GSetup() algorithm is used by the RD-ABE scheme to set the global system parameters (global public parameter GP as mentioned in the paper). First the system initializer (SI) chooses a composite order bilinear group $\mathbb{G}$ of order $N = p_1 p_2 p_3$, where p_1, p_2 and p_3 are distinct big primes. $g_1 \in_G \mathbb{G}_{p_1}$ which is a subgroup of $\mathbb{G}$ of order p_1. The initializer also chooses hash functions $\mathcal{H}_0: \{0,1\}^* \rightarrow \mathbb{Z}_N$ and $\mathcal{H}_1: \{0,1\}^* \rightarrow \mathbb{G}$.

c_1: SI chooses (distinct) big primes p_1, p_2 and p_3 (cost: **negligible**).

c_2: SI chooses group $\mathbb{G}$ of order $N = p_1 p_2 p_3$ (cost: **negligible**).

c_3: SI chooses subgroup $\mathbb{G}_{p_1}$ of $\mathbb{G}$ of order p_1 (cost: **negligible**).

c_4: SI chooses generator $g_1 \in_G \mathbb{G}_{p_1}$ (cost: **negligible**).

c_5: SI chooses hash function $\mathcal{H}_0: \{0,1\}^* \rightarrow \mathbb{Z}_N$ (cost: **negligible**).

c_6: SI chooses hash function $\mathcal{H}_1: \{0,1\}^* \rightarrow \mathbb{G}$ (cost: **negligible**).

c_7: SI computes bilinear pairing $e(g_1, g_1)$ (cost: **1 bilinear pairing**).

m_1: SI publishes global public parameter $GP = (N, g_1, e(g_1, g_1), \mathcal{H}_0, \mathcal{H}_1)$ (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
SI	$O(1)$	$O(1)$

Table 21: Complexities .GSetup() RD-ABE

During the setup phase the System Initializer (SI) publishes global public parameters GP. The generation of the GP is a constant cost and therefore the computational complexity of this protocol step is $O(1)$. Next to that SI publishes GP at a constant cost and thus the communication complexity is denoted as $O(1)$.

6.2.2 .ASetup()

The .ASetup() algorithm is used to set up a new Attribute Authority (AA_k) using global public parameters GP generated in the previous algorithm. AA_k executes this step for each attribute $\alpha \in S_{\alpha_k}$ it manages, where S_{α_k} denotes the attribute universe managed by AA_k.

c_8: AA_k chooses $\{\epsilon_\alpha, \gamma_\alpha \in_R \mathbb{Z}_N\}$ (cost: **negligible**).

c_9: AA_k computes $g_1^{\gamma_\alpha}$ (cost: **1 exponentiation** per attribute).

c_{10}: AA_k computes $e(g_1, g_1)^{\epsilon_\alpha}$ (cost: **1 exponentiation** per attribute).

c_{11}: AA_k sets the private key of attribute α as $ask_\alpha = (\epsilon_\alpha, \gamma_\alpha)$ (cost: **negligible**).

m_2: AA_k publishes the public key of attribute α ($apk_\alpha = (e(g_1, g_1)^{\epsilon_\alpha}, g_1^{\gamma_\alpha})$) (cost: **1 message** per attribute).

Entity	Computational Complexity	Communication Complexity
AA_k	$O(a)$	$O(a)$

Table 22: Complexities .ASetup() RD-ABE

For each of its attributes AA_k generates a secret key and a private key. The cost to generate these keys is a single exponentiation and a single bilinear pairing per attribute. Therefore the computational complexity of an Attribute Authority in this step is linearly dependent on the amount of attributes in its own attribute universe S_{α_k}, namely $O(a)$. For each of its attributes AA_k publishes the public key apk_α at a cost of a single message per attribute, meaning that the total communication complexity for an Attribute Authority in this protocol step is $O(a)$ (linearly dependent with the amount of attributes the AA_k manages). The overall system complexity of this protocol step is dependent on the amount of attributes in the system, setting the computational and communication complexity at $O(a * k)$.

6.2.3 .KeyGen()/.KeyUpd()

The .KeyGen() algorithm is used to generate an attribute-based private key for an attribute α for a user (u_j) with global identifier uid_j, who is in possession of said attribute (α). The .KeyUpd() algorithm is the same, except for that it takes a revocation list rl_α for that attribute as input, ensuring that revoked users will not get an updated key for attribute α. A time element $Time$ is embedded and thus this key is valid for a certain period of time. After that period has expired the .KeyUpd() algorithm is used to generate a new secret key corresponding to user u_j with global identifier uid_j, time period $Time_{new}$ and attribute α.

c_{12}: AA_k computes $\mathcal{H}_1(uid_j||Time)$ (cost: **negligible**).

c_{13}: AA_k computes $\mathcal{H}_1(uid_j||Time)^{\gamma_\alpha}$ (cost: **1 exponentiation**).

c_{14}: AA_k computes $\mathcal{H}_0(Time)$ (cost: **negligible**).

c_{15}: AA_k computes $g_1^{\epsilon_\alpha * \mathcal{H}_0(Time)}$ (cost: **1 exponentiation**).

c_{16}: AA_k computes secret key $sk_{\alpha,uid_j}^{Time} = g_1^{\epsilon_\alpha * \mathcal{H}_0(Time)} \mathcal{H}_1(uid_j||Time)^{\gamma_\alpha}$ (cost: **negligible**).

m_3: AA_k sends sk_{α,uid_j}^{Time} to user u_i (cost: **1 message**).

Entity	Computational Complexity	Communication Complexity
AA_k	$O(a * u * t)$	$O(a * u * t)$

Table 23: Complexities .KeyGen() RD-ABE

For each attribute $\alpha \in S_{\alpha_k}$, AA_k generates a key for each user entitled to said attribute α. The complexity for a single key generation is constant at the cost of 2 exponentiations per key, but as this protocol step is executed every time period, for every attribute and for every user entitled to that attribute the computational complexity of this protocol step is $O(a * u * t)$. AA_k also has to sent the secret key to its corresponding user, meaning that the communication complexity is also $O(a * u * t)$ as a single message per attribute, per user has to be sent every time period.

6.2.4 .Encrypt()

Algorithm .Encrypt() is used by a data owner (u_i) to encrypt message m (which might be a symmetric key, used on its turn to encrypt large amounts of data). The algorithm takes access matrix $(A, \rho(x))$ with l rows and n columns as input, which corresponds with access policy $\mathcal{P}$. A_x denotes a row of the matrix and function $\rho(x)$ maps row A_x of A to attribute α, where $\rho(x)$ is injective, meaning that each attribute occurs as a row A only once. As a time element is included in the ciphertext(s) this means that the user has to encrypt message m every period in time using the new timestamp and ensure the old encrypted version of m is deleted.

c_{17}: u_i chooses secret value $s \in \mathbb{Z}_N$ (cost: **negligible**).

c_{18}: u_i chooses $\{v_2, \ldots, v_n\} \in_R \mathbb{Z}_N$ (cost: **negligible**).

c_{19}: u_i sets $v = (s, v_2, \ldots, v_n)$ (cost: **negligible**).

c_{20}: u_i chooses $\{w_2, \ldots, w_n\} \in_R \mathbb{Z}_N$ (cost: **negligible**).

c_{21}: u_i sets $w = (0, w_2, \ldots, w_n)$ (cost: **negligible**).

c_{22}: u_i chooses $r_x \in_R \mathbb{Z}_N$ for all rows in A (cost: **negligible**).

c_{23}: u_i computes $C_0 = m * e(g_1, g_1)^s$ (cost: **1 exponentiation**).

c_{24}: u_i computes $\mathcal{H}_0(Time)$ (cost: **negligible**).

c_{25}: u_i computes $e(g_1, g_1)^{\mathcal{H}_0(Time)}$ (cost: **1 exponentiation**).

c_{26}: u_i computes $\{(e(g_1, g_1)^{\mathcal{H}_0(Time)})^{\epsilon_\alpha r_x} = e(g_1, g_1)^{\epsilon_\alpha \mathcal{H}_0(Time) r_x}\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

c_{27}: u_i computes $\{e(g_1, g_1)^{\lambda_x}$ where $\lambda_x = A_x * v\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

c_{28}: u_i computes $\{C_{1,\alpha} = e(g_1, g_1)^{v_x} e(g_1, g_1)^{\epsilon_\alpha \mathcal{H}_0(Time) r_x}$ (cost: **negligible**).

c_{29}: u_i computes $\{C_{2,\alpha} = g_1^{r_x}\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

c_{30}: u_i computes $\{C_{3,\alpha} = g_1^{\gamma_\alpha r_x} g_1^{w_x}\}_{\alpha \in \mathcal{P}}$ where $w_x = A_x * w$ (cost: **1 exponentiation** per attribute).

m_4: u_i publishes ciphertext $ct = (C_0, \{C_{1,\alpha}\}_{\alpha \in \mathcal{P}}, \{C_{2,x}\}_{\alpha \in \mathcal{P}}, \{C_{3,x}\}_{\alpha \in \mathcal{P}})$ (cost: **3 messages** per attribute).

Entity	Computational Complexity	Communication Complexity
u_i	$O(a)$	$O(a)$

Table 24: Complexities .Encrypt() RD-ABE

User u_i encrypts message m using access policy $\mathcal{P}$. The computational complexity for the user is linearly dependent on the amount of attributes used in the access policy $\mathcal{P}$ and thus is $O(a)$. The amount of messages u_i needs to send is also linearly dependent on the amount of attributes as the ciphertext ct contains an element per row x in $(A, \rho(x))$. The communication complexity therefore is also $O(a)$.

6.2.5 .Decrypt()

The .Decrypt() algorithm is used to decrypt the ciphertext ct containing message m, which was encrypted at time $Time$. If user u_j (uid_j) is in possession of a set of attributes with adheres to the access policy of the same time $Time$ the user is able to decrypt the ciphertext.

m_5: $u_j \Rightarrow CSP$, request ciphertext ct from the CSP (cost: **1 message**).

m_6: $CSP \Rightarrow u_j$, reply with ciphertext ct (cost: **3 messages** per attribute).

c_{31}: u_j computes $\mathcal{H}_1(uid_j||Time)$ (cost: **negligible**).

c_{32}: u_j computes $\{C_{1,\alpha} e(\mathcal{H}_1(uid_j||Time), C_{3,\alpha})\}_{\alpha \in \mathcal{P}}$ (cost: **1 bilinear pairing** per attribute).

c_{33}: u_j computes $\{e(sk^{Time}_{\alpha,uid_j}, C_{2,\alpha})\}_{\alpha \in \mathcal{P}}$ (cost: **1 bilinear pairing** per attribute).

c_{34}: u_j computes $\{\frac{C_{1,\alpha} e(\mathcal{H}_1(uid_j||Time), C_{3,\alpha})}{e(sk^{Time}_{\alpha,uid_j}, C_{2,\alpha})} = e(g_1, g_1)^{v_x} e(\mathcal{H}_1(uid_j||Time), g_1)^{w_x}\}_{\alpha \in \mathcal{P}}$ (cost: **negligible**).

c_{35}: u_j chooses $\{c_x \in \mathbb{Z}_N\}_{x \in A}$ such that $\sum_x c_x A_x = (1,0,\ldots,0)$ (cost: **negligible**).

c_{36}: u_j computes $\{(e(g_1, g_1)^{v_x} e(\mathcal{H}_1(uid_j||Time), g_1)^{w_x})^{c_x} = e(g_1, g_1)^{v_x c_x} e(\mathcal{H}_1(uid_j||Time), g_1)^{w_x c_x}\}_{\alpha \in \mathcal{P}}$ (cost: **1 exponentiation** per attribute).

c_{37}: u_j computes $\prod_{x \in A}(e(g_1, g_1)^{v_x c_x} e(\mathcal{H}_1(uid_j||Time), g_1)^{w_x c_x}) = e(g_1, g_1)^{(\sum_x v_x c_x)} e(\mathcal{H}_1(uid_j||Time), g_1)^{(\sum_x w_x c_x)} = e(g_1, g_1)^{s} e(\mathcal{H}_1(uid_j||Time), g_1)^{0} = e(g_1, g_1)^{s}$ (cost: **negligible**).

c_{38}: u_j computes $\frac{C_0}{e(g_1, g_1)^s} = m$ (cost: **negligible**).

Entity	Computational Complexity	Communication Complexity
u_j	$O(a)$	$O(1)$
CSP	-	$O(a)$

Table 25: Complexities .Decrypt() RD-ABE

The user requests ciphertext C by sending a single message to the CSPat the communication complexity of $O(1)$. The CSP replies with the ciphertext. As the amount of components of the ciphertext is linearly dependent on the amount of attributes used in the access policy $\mathcal{P}$, the amount of messages the CSP sends is as well, setting the communication complexity of the CSP at $O(a)$. User u_j then decrypts the ciphertext and obtains message m at a constant cost of 2 bilinear pairings per attribute in access policy $\mathcal{P}$. The computational complexity therefore is $O(a)$.

6.3 Advantages & Disadvantages

First the requirements will be mentioned and whether RD-ABE[12] meets them in Section 8.3.1. After that, based on the requirements analysis, a list of advantages and a list of disadvantages will be mentioned in Sections 8.3.2 Requirement analysis

This section will briefly mention the requirements as were listed in Section 5.2. Each paragraph will mention a requirement, whether it is strong or weak and whether RD-ABE meets that requirement.

CP-ABE (Strong) - CP-ABE is a form of ABE where the access policy $\mathcal{P}$ is embedded into the ciphertext so that only users entitled to the right attributes are able to decrypt the encrypted file. RD-ABE is a CP-ABE scheme where the encryption of data items includes the use of LSSS to embed an access policy into the ciphertext.

Collusion resistance (Strong) - Collusion resistance ensures that different users are not able to combine their attributes/keys to decrypt a ciphertext if none of the users is able to decrypt the file on its own. In RD-ABE every ABE key includes the secret key of an attribute, a time period and the (global) identity of the user. Because the user identity is embedded in the ciphertext different users cannot combine their attribute keys to collude to decrypt a ciphertext. RD-ABE is therefore collusion resistant.

Access revocation (Strong) - An access revocation mechanism allows for revocation of an attribute, the revocation of an attribute of a user and/or a method to update the access policy in the encryption of a ciphertext. This ensures certain users or users with certain attributes are no longer able to decrypt a ciphertext. This can be done either mathematically or system wise. RD-ABE achieves revocation by updating the secret attribute keys of users using the .KeyUpd() algorithm every period of time and re-encrypting messages using the new time period as the time period is embedded in the ciphertext as well. It should be noted that re-encryption of messages can only be done by the data owner. Revocation in RD-ABE does not prevent a user from decrypting a ciphertext which was decrypted at a moment in time where the user was able to decrypt the ciphertext, even if the time period has been long passed. In order to prevent a user decrypting an old ciphertext the ciphertext should either be deleted from the CSP or the CSP should not allow revoked user to access the old ciphertext. This however, does not prevent a (revoked) user from decrypting an old ciphertext if it obtained the ciphertext from a different source than the CSP.

Scalability (Strong) - Scalability ensure the system will not run out of computational resources as new computational resources can be added to the system during run time. This ensures the system is not limited in the amount of attributes, users or attribute authorities it can handle. As ASCLEPIOS aims to design a system that makes it possible for on-person medical devices to continuously upload (encrypted) data to a CSPmeaning that the encryption phase is the most important operation to look at to be able to say anything about its scalability as these on-person devices, such as pacemakers, are usually very limited in the amount of computations they can handle. The complexity of the encryption step grows linearly with the amount of attributes used to encrypt a message and the most significant computation it has to perform is an exponentiation (assuming the bilinear pairings $e(g_1, g_1)$ is calculated as a public parameter). If used in a hybrid system where a content key κ is used to (symmetrically) encrypt data items and the content key is shared using RD-ABE it really depends on the amount of attributes used to encrypt κ if applicable. RD-ABE allows for the addition of users and attributes to the system after its original initialization. Existing ciphertexts are not updated when a new attribute is added, but have to be updated every new period of time.

Multiple Authorities (Weak) - Having multiple Attribute Authorities (AAs) removes the need for trust in a central authority as each of the Authorities on itself decides whether a user is entitled to a specific attribute or not. If a ciphertext is encrypted using the attribute keys of multiple AAs it would need all of the authorities to collude together for them to be able to

decrypt the ciphertext, meaning that if one AA was not compromised the system would still function correctly. This potentially increases security. RD-ABE describes a method to decentralize attribute management.

Regranting Access (Weak) - Regranting access is only applicable if the system incorporates a revocation method. Regranting access ensures attribute/user revocation does not have to be permanent. RD-ABE has a possibility to regrant access by generating a new update key for a specific attribute using the global identity of the user and the current time period.

Multiple Access Controls (Weak) - Multiple Access Controls is not trivial in ABE schemes, but might just as well be necessary for real-life usage of any ABE scheme. Usually, ABE schemes only deal with read access, meaning that once a user has decrypted a file it is only able to view the contents of said file. When dealing with medical files it might, for example, be necessary that a medical practitioner is able to add a comment to a data entry, meaning that he should be able to change the ciphertext, i.e. he should have write access to the file. RD-ABE does not include the possibility to make use of different access controls.

6.3.1 Advantages

Exponentiations in Encryption - In the Encryption phase of RD-ABE the most significant operation is the exponentiation (assuming the bilinear pairing is calculated as a global parameter). While relatively expensive in computation time it is far less expensive than finding a bilinear pairing, such as the DAC-MACS scheme uses.

User Addition - RD-ABE does not need to define all users during the initialization process which should be a requirement for any such system to be deployable in a real-life scenario. As not all schemes allow for such additions this is seen as an advantage of using RD-ABE.

Attribute Authority Addition - RD-ABE allows any entity at any moment during the lifetime of an RD-ABE-based program to function as an Attribute Authority by simply generating its own public/secret key pair. As not all schemes allow an Attribute Authority to be added during run time this is seen as one of the advantages of RD-ABE. It is up to a data owner to choose which attributes to use encrypt a ciphertext and thus the addition of new Attribute Authority does not influence the security of the scheme.

Revocation Method - One of the advantages of using RD-ABE is that the scheme includes a revocation method. Revocation is achieved by including a time element in ciphertexts and secret attribute keys of users in the system. By simply not updating a key and re-encrypting (part of) the original message a revoked user is not able to decrypt the re-encrypted ciphertext using its new attribute keys. It should be noted that the user is still able to decrypt the previous ciphertext if it is in possession of all the previous attribute keys. It might therefore be better for the scheme to only generate a secret attribute key and send it to the corresponding user on request. This would also greatly influence the efficiency of the scheme.

6.3.2 Disadvantages

Not backwards secure - The RD-ABE scheme is not backwards secure, meaning that once a user was able to decrypt a specific version (corresponding to a certain time period) of a ciphertext (i.e. had the right secret attribute keys) it will be able to do so as long as it is in possession of the ciphertext and the keys. It would therefore be better to only issue keys to users once requested as this would mean that a revoked user who did not receive a secret attribute key for a specific time period is not able to decrypt a ciphertext encrypted using a

policy containing said attribute (unless another combination of attributes without said attribute is "*accepting*" as well).

Re-encryption of a ciphertext - As the time period is also embedded in (part of) the ciphertext this means that, for a user to be able to decrypt a ciphertext and obtain the original message, using its updated secret attribute keys, the ciphertext needs to be updated, while embedding the new time period. This means that (part of) the ciphertext needs to be re-calculated and this amounts to a single exponentiation per attribute included in the access policy $\mathcal{P}$. As this is quite an expensive operation this is seen as a disadvantage of the RD-ABE scheme.

7 Comparison

7.1 Introduction

This chapter describes the comparison of the three schemes described in this report (FAME[8], DAC-MACS[10] and RD-ABE[12]). First a comparison of each of the traditional protocol steps of an ABE scheme (.Setup(), .Encrypt(), .KeyGen() and .Decrypt()) is made by using the efficiency analysis which was described in their respective chapters in Section 9.2. Then, Non-traditional protocol steps of the specific schemes are compared in. After that a comparison between the schemes is made based on how/whether each scheme adheres to the requirements set in Section 5.2. In the end a conclusion is drawn in Section 9.4.

7.2 Efficiency

First the computational and communication complexity and the most significant operation (only applies to the computational complexity), per scheme, per entity are listed. Sub algorithms are aggregated. After that a brief description is given of the efficiencies of the different schemes and the comparison between the scheme is given as well. This in the end is used to draw a conclusion in Section 9.4.

7.2.1 .Setup()

Scheme	Entity	Comp. complex.	Most significant	Comm. complex.
FAME	CA	$O(1)$	Bilinear pairing	$O(1)$
DAC-MACS	CA CA u_j	$O(1)$ $O(u)$ -	Bilinear pairing Exponentiation -	- $O(u+k)$ $O(1)$
RD-ABE	SI AA_k	$O(1)$ $O(a)$	Bilinear pairing Exponentiation	$O(1)$ $O(a)$

Table 26: Complexities Comparison .Setup()

Table 26 shows the computational and communication efficiency of the .Setup() algorithm, per scheme, per entity. It should be noted that the CA is mentioned twice at the DAC-MACS scheme, but this was added as a single bilinear pairing is computed in all schemes and thus the complexity would not be complete if that pairing was not added to the scheme. It is noted that FAME is far more efficient than DAC-MACS and RD-ABE which are dependent on the amount of users/attribute authorities and the amount of Attribute Authorities (and they in their turn on the amount of attributes they manage), respectively. It is however assumed that a system in real-life deployment has far more users than Attribute Authorities/attributes and thus DAC-MACS has (in practice) the least efficient .Setup() algorithm.

7.2.2 .KeyGen()

Scheme	Entity	Comp. complex.	Most significant	Comm. complex.

FAME	CA u_j	$O(a*u)$ -	Exponentiation -	$O(a*u)$ $O(1)$
DAC-MACS	AA_k u_j	$O(a*u)$ -	Exponentiation -	$O(a*u)$ $O(k)$
RD-ABE	AA_k	$O(a*u*t)$	Exponentiation	$O(a*u*t)$

Table 27: Complexities Comparison .KeyGen()

Table 27 shows the computational and communication complexity of the .KeyGen() algorithm per scheme. It can be seen that the FAME ABE scheme is again relatively efficient, as u_j only needs to send a single message to retrieve its key, whereas in DAC-MACS, with the same computational complexities, a user has to sent a message to each involved Attribute Authority (AA_k). It should however be noted that if the global amount of attributes in the system is the same in both schemes DAC-MACS allows for the distribution of computations among multiple Attribute Authorities (and so in total less computations per entity), whereas in FAME the CA performs every computation. DAC-MACS has the same complexities except for the fact that a user has to sent multiple messages to each involved AA_k as mentioned before. The RD-ABE crypto scheme is the least efficient on of the three schemes as a new key has to be generated for every attribute and every user each new period of time.

7.2.3 .Encrypt()

Scheme	Entity	Comp. complex.	Most significant	Comm. complex.
FAME	u_i	$O(a^2)$	Exponentiation	$O(a)$
DAC-MACS	u_i	$O(a)$	Exponentiation	$O(a)$
RD-ABE	u_i	$O(a)$	Exponentiation	$O(a)$

Table 28: Complexities Comparison .Encrypt()

Table 28 shows the computational and communication complexity of the .Encrypt() algorithm being the same for the DAC-MACS and RD-ABE schemes as the method in both schemes is linearly dependent on the amount of attributes. FAME however is much less efficient as the complexity grows quadratically with the amount of attributes. Especially the .Encrypt() step in ASCLEPIOS should be efficient as this protocol step is likely executed on a device with low computational capabilities, such as a pace maker, making FAME possibly a less useful candidate depending on the actual amount of attributes featured in the system.

7.2.4 .Decrypt()

Scheme	Entity	Comp. complex.	Most significant	Comm. complex.
FAME	u_j CSP	$O(1)$ -	Bilinear pairing -	$O(1)$ $O(a*u)$
DAC-MACS	u_j CSP	$O(1)$ $O(a*u)$	Exponentiation Bilinear pairing	$O(a)$ $O(a)$

RD-ABE	u_j CSP	$O(a)$ -	Bilinear pairings -	$O(1)$ $O(a*u)$

Table 29: Complexities Comparison .Decrypt()

Table 29 shows the computational and communication complexity of the .Decrypt() algorithm per scheme. It shows that there are enormous differences between the complexities of the different schemes. FAME is the most efficient scheme as the communication complexity of the CSP is dependent on the amount of attributes (influences the size of the ciphertext) and users when talking about the communication complexity. Each user however only sends a single message requesting the ciphertext and computes a constant amount (6) bilinear pairings per decryption. The second most efficient scheme, for the users, is the DAC-MACS crypto scheme, which only requires the user to send a key for each of the attributes involved to the CSP, which the CSP uses to generate a token TK at the cost of 5 bilinear pairings per attribute (per user requesting such a token). A user only needs to compute a single encryption to decrypt the ciphertext. In RD-ABE on its turn, the CSP has no mathematical operations to compute, but each user computes 2 bilinear pairings per attribute involved. This makes RD-ABE globally more efficient, but a CSP usually has more computational power. It is left up to the reader to decide what he prefers.

7.2.5 Other

7.2.5.1 DAC-MACS - Update/Revocation method

DAC-MACS – Update/Revocation method

Scheme	Entity	Comp. complex.	Most significant	Comm. complex.
DAC-MACS	u_j AA_k CSP	$O(1)$ $O(u)$ $O(ct)$	Exponentiation Exponentiation Exponentiation	- $O(u)$ -

Table 30: Complexities Comparison .Update()

Table 30 shows the computational cost of the .Update() algorithm in DAC-MACS. No clear comparison can be drawn as the other two schemes do not incorporate such a protocol step. It should however be taken into mind that this revocation method requires the CSP, the involved AA_k and each involved, non-revoked user to perform exponentiations every time an attribute gets revoked for a user.

7.3 Requirement analysis

Requirement	FAME	DAC-MACS	RD-ABE
CP-ABE	Yes	Yes	Yes
Collusion resistance	Yes	Partly	Yes
Access revocation	No	Direct	Indirect

Scalability	Partly	Partly	Partly

Table 31: Adherence to strong requirements

Table 31 shows the strong requirements which were set out in Section 5.2.1 and the adherence of the different schemes to them. All three schemes described are CP-ABE schemes (and not KP-ABE schemes) and thus the decision which attributes together should allow for decryption lies with the user and not the key issuer. Both FAME and RD-ABE are collusion resistant, although one should keep in mind that in FAME there is a Central Authority, which is able to generate any key it wants and thus has to be fully trusted. DAC-MACS has a revocation method, but it is possible to, as a user revoked for a specific attribute, to obtain the update key for said attribute by colluding with a non-revoked user (for that attribute) or the CSP. This means however, that the revocation method has a flaw and not the overall crypto scheme.

FAME does not allow for the revocation of an attribute, whereas DAC-MACS and RD-ABE respectively allow for direct and indirect revocation of an attribute for a specific user. Both direct and indirect revocation have their advantages/disadvantages. It is left up to the reader to decide which he prefers. The last strong requirement is that the schemes should be scalable which is only entirely the case for the .Setup() of all three schemes. The .Encrypt() algorithm is relatively efficient in the DAC-MACS and RD-ABE scheme, but in FAME the computational complexity quadratically increases with the amount of attributes and is thus less useful when deployed on devices with low computational capabilities, such as pace makers. The difference between the schemes becomes clear in the comparison of the .KeyGen() algorithm and the .Decrypt() algorithm as the RD-ABE scheme needs to resend a new attribute key every once in a while to every non-revoked user (whereas DAC-MACS only sends an update to all involved users once an attribute is revoked for a user). FAME and DAC-MACS have roughly the same computational complexity. The .KeyGen() algorithm however only deals with exponentiations and is therefore still quite scalable, whereas the .Decrypt() algorithm deals with Bilinear pairings. The complexity of FAME is constant and thus is by far the most efficient scheme when decrypting a ciphertext. Both RD-ABE and DAC-MACS have the same global computational complexity, but RD-ABE allows for the distribution of these computations among all involved users, whereas these computations are done by the CSP in DAC-MACS. Both situations have their advantages, but as these complexities grow more than linear the question remains whether DAC-MACS and RD-ABE are scalable or not.

Requirement	FAME	DAC-MACS	RD-ABE
Multiple Authorities	No	Yes	Yes
Regranting access	-	Yes	Yes
Multiple Access Controls	No	No	No

Table 32: Adherence to weak requirements

Table 32 shows the weak requirements which were set out in Section 5.2.2 and the adherence of the different schemes to them. First of all, none of the scheme allows for multiple access controls and so this requirements is not met. The FAME crypto scheme does not make use of multiple authorities and so the Central Authority always has to be trusted. As FAME does not

allow for revocation, the scheme does also not describe a method to regrant access. Both DAC-MACS and RD-ABE distribute the management of attributes among different Attribute Authorities and have a method which allows for regranting access for a specific attribute to a revoked user.

8 Ciphertext Delegation

In this section we briefly describe ciphertext delegation as it is defined in [13]. Ciphertext delegation is a process by which a ciphertext can be made harder to decrypt using only public operations in a more efficient way than decrypting and re-encrypting under a more restrictive policy.

In particular, we want a user that has access to only the ciphertext and public key to process this information into a completely new encryption under a more restrictive access policy. We say that a ciphertext with a given access policy can be *delegated* to a more restrictive policy if there is a procedure that given any valid encryption of a message under the first policy produces an *independent and uniformly chosen* encryption of the same message under the new access policy. It is important to stress out that delegation is required to produce a new encryption of the same message that is independent of the randomness and access policy of the original ciphertext being delegated from. This requirement is crucial in multiple delegations from the same base ciphertext are used in a scheme. Without this guarantee, multiple delegations may have correlated randomness and the security of the underlying scheme would not imply any security in these applications. Before we continue, we need to recall some notation defined previously.

8.1 Ciphertext Policy Delegation

The most important analysis for delegation comes when considering CP-ABE scheme as the ciphertexts may be associated with complex access policies. Some of the most prominent CP-ABE schemes (including the ones presented in this document) are built upon an underlying secret sharing scheme corresponding to their access policy. The first step in the encryption procedure in these schemes is to share a uniformly chosen secret according to the implied secret sharing scheme with the shares of the secret embedded into certain components of the ciphertext. The encryption scheme is said to be based on a given secret sharing scheme if it falls into the above paradigm for this secret sharing scheme.

Notation: A linear secret sharing scheme over a field $\mathbb{F}$ for a set of players $\mathbb{P}$ is defined through a pair (A, ρ) with A the share generating matrix of dimension $n \times \ell$ and ρ the assignment function from $[n] \to \mathbb{P}$. To evaluate the shares of a secret, the vector $u = (s, k_1, \dots, k_{\ell-1})$ is produced, where s is the secret to be shared and $k_1, \dots, k_{\ell-1}$ are chosen uniformly at random, The share vector is defined to be $\vec{v} = Au$ with party i receiving all $\vec{v}[j]$ such that $\rho(j) = i$.

We say that a secret sharing scheme S consists of two polynomial time algorithms. A sharing algorithm **Share** and a reconstruction algorithm **Rec**. The reconstruction algorithm is responsible for reconstructing the shares returned by the parties. However, the sharing algorithm, instead of returning one share to each party, it outputs $(\vec{v}, \alpha)$ where the components of v are elements on the share space and α is an assignment of indices of $\vec{v}$ to $[n]$, where n is the number of users that determines which components of $\vec{v}$ should be sent to which user.

8.1.1 Delegation Procedure

It is observed that most known CP-ABE schemes have the access structure embedded into the ciphertexts in the form of a secret sharing scheme. Informally, such schemes work as follows:

- The encryption algorithm generates a secret s and shares it according to the policy it is being encrypted under.
- The decryption algorithm uses all the shares that correspond to the possessed attributes to reconstruct s. Reconstructing s implies recovering the plaintext message.

With this in mind, we make use of a share extractor X that recovers the shares built-in the ciphertexts.

Informally, we say that a CP-ABE scheme is secret sharing based for a secret sharing scheme S with share space S is there is a share extractor X that on input a valid ciphertext c_p outputs $(s, (\vec{v}, \alpha))$ where $s \in \mathbb{S}$, $(\vec{v}, \alpha) \leftarrow S.Share(s, P)$. If $F(C) = (s, (\vec{v}, \alpha))$ with $(\vec{v}, \alpha) \leftarrow S.Share(s, P)$ then C is a valid ciphertext.

8.1.2 Elementary Delegation Properties

For a CP-ABE scheme to support ciphertext delegation, it needs to allow certain operations to be performed directly on the ciphertexts that manipulate the shares of the shared secret.

1. ***Property 1***. A well-formed ciphertext under a given access policy can be re-randomized to an independent encryption of the same message under the same policy.
2. ***Property 2***. There exists a probabilistic polynomial time algorithm **Combine** such that for any $i, j \leq len(\vec{v})$ with $(\alpha_i) = (\alpha_j)$ we have **Combine**$\big(C, i, j, a_i, b_j, d\big) = C'$ with $F(C') = \big(s, (\vec{v}', \alpha)\big)$ where:

$$\overrightarrow{v'[k]} = \begin{cases} \vec{v}[k], & \forall\ k \neq i \\ a_i \vec{v}[i] + b_j \vec{v}[j] + d, & if\ k = i \end{cases}$$

3. ***Property 3***. There exists a probabilistic polynomial time algorithm **Delete** such that for any $i \leq len(\overrightarrow{v})$ we have **Delete**$(C, i) = C'$ where $F(C') = \big(s, (\vec{v}', \alpha)\big)$, $len\big(\overrightarrow{v'}\big) = len(\vec{v})$ and:

$$(\overrightarrow{v'}[k], \alpha'(k)) = \begin{cases} (\vec{v}[k], \alpha(k)), & \forall\ k < i \\ (\vec{v}[k+1], \alpha(k+1)), & if\ k \geq i \end{cases}$$

4. ***Property 4.*** There exists a probabilistic polynomial time algorithm **Add** such that for any $i \in [n]$ we have **Add**$(C, i) = C'$ where $F(C') = \big(s, (\vec{v}', \alpha)\big)$, $len\big(\overrightarrow{v'}\big) = len(\vec{v}) + 1$ and:

$$(\overrightarrow{v'}[k], \alpha'(k)) = \begin{cases} (\vec{v}[k], \alpha(k)), & \forall\ i < len(\vec{v}) \\ (0, i), & if\ k = len(\vec{v}) + 1 \end{cases}$$

5. ***Property 5.*** There exists a probabilistic polynomial time algorithm **Swap** such that for any $i, j \leq len(\vec{v})$ we have **Swap**$(C, i, j) = C'$ where $F(C') = \big(s, (\vec{v}', \alpha)\big)$ and:

$$\big(\overrightarrow{v'}[k], \alpha'(k)\big) = \begin{cases} (\vec{v}[k], \alpha(k)), \forall k \notin \{i, j\} \\ \big(\vec{v}[k], \alpha(i)\big), if\ k = j \\ \big(\vec{v}[k], \alpha(j)\big), if\ \ k = i \end{cases}$$

These ciphertext manipulations are basic for ciphertext delegation. If a ciphertext allows delegation of any well-formed ciphertext under policy P to a well-formed ciphertext under policy P' using the elementary delegation operations, this implies that any ciphertext encrypted with

a policy P may be delegated to a uniformly random encryption of the same message under a new policy P' by the re-randomization guarantee.

We will not go into further details for ciphertext delegation as depending on the secret sharing scheme the CP-ABE scheme is based on, the elementary delegation operations may have different capabilities.

9 Combining Symmetric Searchable Encryption and Ciphertext-Policy Attribute-Based Encryption

In this section we design a protocol that combines SSE D2.1 with CP-ABE with respect to the reference architecture proposed in D1.2.

9.1 Architecture

Before we procced to a detailed description of the protocol, we briefly recall some core entities that participate in it, as they were defined in D1.2. and in [16] and [17]

Cloud Service Provider (CSP): One of the common models of a cloud computing platform is Infrastructure-as-a-Service (IaaS). In its simplest form, such a platform consists of cloud hosts which operate virtual machine guests and communicate through a network. Often a cloud middleware manages the cloud hosts, virtual machine guests, network communication, storage resources, a public key infrastructure and other resources. Cloud middleware creates the *cloud infrastructure* abstraction by weaving the available resources into a single platform. In our system model we consider a cloud computing environment based on a trusted IaaS provider. The IaaS platform consists of cloud hosts which operate virtual machine guests and communicate through a network. In addition to that, we assume a Platform-as-a-Service (PaaS) provider that is built on top of the IaaS platform and can host multiple outsourced databases. Furthermore, the cloud service provider is responsible for storing users' data. Finally, the CSP must be TEE enabled since core entities of the protocol will be running in a trusted execution environment offered by SGX.

Master Authority (MS): MS is responsible for setting up all the necessary public parameters that are needed for the proper run of the underlying protocols. Furthermore, MS is responsible for generating and distributing ABE keys to the registered users. Finally, MS is considered as a single trusted authority. Thus, we assume that MS is TEE-enabled and is running in an enclave called the Master Enclave.

Key Tray (KeyTray): KeyTray is a key storage that exists in the CSP and stores ciphertexts of all the symmetric keys that have been generated by various data owners and are needed in order to decrypt data. Every registered user can contact the KeyTray directly and request access to the stored ciphertexts. Furthermore, the symmetric keys are encrypted with a CP-ABE scheme. Thus, a single symmetric key is encrypted only once and users with certain access rights and different keys are able to access it (i.e. decrypt it). Moreover, similar to MS, KeyTray is also TEE-enabled and is running in an enclave called the KeyTray Enclave.

Revocation Authority (REV): REV is responsible for maintaining a revocation list (*rl*) with the unique identifier of the users that have been revoked. At this point it is worth mentioning that a single user might own more than one CP-ABE secret key. Therefore, *rl* maintains a mapping of users with the CP-ABE keys they own. Every time that a key of a user is revoked, REV needs to update *rl*. This, as we will see later, will prevent revoked users from accessing ciphertexts that are not authorized anymore. Similar to MS and KeyTray, REV is also TEE-enabled and is running in an enclave called the Revocation Enclave.

Registration Authority (RA): RA is responsible for the registration of users in the CSP. Additionally, RA has a public/private key pair denoted as pk_{RA}/sk_{RA}. RA can run as a separate third party but can be also implemented as part of the CSP. The registration process is out of

the scope of this paper. Thus, we will not describe how the registration of a new user takes place. Instead, we will assume that a user has been already registered and has access to the remote storage and the services offered by the CSP.

9.2 Protocol

We are now ready to proceed with a formal and detailed description of the core algorithms. The SSE scheme we are using is based on the presented in [18].

ASCLEPIOS.Setup : Each entity from the described system model obtains a public/private key pair (pk, sk) for a CCA2 secure public cryptosystem and publishes its public key while it keeps the private key secret. Apart from that, all three entities that are running in an enclave generate a signing and a verification key. Furthermore, MS runs CPABE.Setup and generates a master public and private key. Below we provide the list of the generated key pairs:

- (pk_{CSP}, sk_{CSP})- public/private key pair for the cloud service provider.
- $(pk_{MS}, sk_{MS}), (sig_{MS}, ver_{MS}), (MPK, MSK)$-public/private, verification/signing and master key pairs for the Master Authority.
- $(pk_{KT}, sk_{KT}), (sig_{KT}, ver_{KT})$- public/private and verification/signing key pairs for the KeyTray.
- $(pk_{REV}, sk_{REV}), (sig_{REV}, ver_{REV})$- public/private and verification/signing key pairs for the Revocation Authority.

ASCLEPIOS.ABEUserKey : This phase is taking place between a registered user u_i that wishes to obtain a CP-ABE key and MS who is responsible for generating such keys. This is a probabilistic key-generation algorithm that runs in the master enclave and takes as input MSK, the identity of the user that is requesting a key and a list of attributes *A* that is derived from user's registered information. More precisely, u_i contacts MS and proves that she is a registered user. Then, attests MS and requests a new CP-ABE key. MS then runs CPABE.Gen and generates sk_{A,u_i}. This is then sent back to the user over a secure channel.

ASCLEPIOS.Store : After a successful registration, we assume that u_i has received a valid credential ($cred_i$) that can be used to login to a cloud service offered by the CSP. Additionally, u_i is now able to store data to the cloud storage. During this phase the communication takes place between the user and the CSP. First, u_i contacts the CSP by sending the following: $m_1 =< r_1, E_{pk_{CSP}}(Auth), StoreReq, H_1) >$ where r_1 is a random number generated by u_i, $Auth$ is an authenticator that allows u_i to prove to the CSP that is a legitimate/registered user and H_1 is the following hash $H(r_1||Auth||StoreReq)$. Upon reception, CSP verifies the freshness of the message, the identity of the user and starts processing the store request. To do so, CSP creates the message $m_2 = (r_2, \sigma_{CSP}(H_2))$, where H_2 is the following hash $H(r_2||u_i)$ and σ_{CSP} is a signature of CSP on H_2. Then, m_2 is sent back to u_i. Upon reception, u_i verifies both the freshness as well as the integrity of the message. Now, u_i simply generates a symmetric key K_i by running SSE.Gen. This key will be used to protect the data that will be stored in the cloud. The final step of this phase is the storage of encrypted files by u_i to a storage resource offered by the CSP. User u_i runs StoreFile – a deterministic algorithm that takes as input the symmetric secret key K_i that generated earlier and a collection of files $\boldsymbol{f_i}$ and outputs a collection of ciphertexts $\boldsymbol{c_i}$ as well as an encrypted index γ_i. Both γ_i and $\boldsymbol{c_i}$ are then send to the CSP via a secure channel. More precisely, u_i sends the following message to the CSP: $m_3 =< (r_3, E_{pk_{CSP}}(\gamma_i), \boldsymbol{c_i}, H_3)\sigma_{u_i}(H_3) >$, where $H_3 = H(r_3||\gamma_i||\boldsymbol{c_i})$. Upon reception, CSP verifies both the integrity and the freshness of m_3 and stores $\boldsymbol{c_i}$ along with the encrypted index γ_i in a local database.

ASCLEPIOS.KeyTrayStore : A key storage algorithm that allows an already logged-in user to safely store a symmetric secret key K_i, that generated earlier, in the Key- Tray. This is a probabilistic algorithm that takes as input a symmetric key K_i, MPK and a policy P and outputs an encrypted version of K_i which is associated with P. This is done by running $c_p^{K_i} \leftarrow CPABE.Enc(MPK, K_i, P)$. The generated ciphertext, is sent by u_i to the KeyTray who stores it locally. More precisely, u_i first attests the KeyTray and then sends the following message: $m_4 =< E_{pk_{KT}}(r_4), c_p^{K_i}, \sigma_i(H(r_4||c_p^{K_i}))>$. Additionally, the KeyTray generates a random number r_{K_i} encrypts it with pk_i and stores it next to $c_p^{K_i}$. As we will see later, this number will be used during the revocation phase to prove that u_i is the owner of K_i.

ASCLEPIOS.KeyShare : Now that u_i has stored an encrypted version of K_i to the KeyTray, other users should be able to access it. Hence, u_i must have a way to share the encrypted data $\boldsymbol{c_i}$ that stored earlier. Let's assume that there is another registered user u_j, $j \neq i$ that wishes to access $\boldsymbol{c_i}$. To do so, u_j needs to get access to K_i that is stored in the KeyTray. The important thing to notice here is that the data sharing will be done without the involvement of u_i. Therefore, after u_i stores $c_p^{K_i}$ to the KeyTray, she can be offline. In order for u_j to access K_i she first needs to get a special token from REV that will prove that u_j's access has not been revoked. To this end, u_j first attests REV and then sends the following message to obtain the token: $m_5 = (r_5, E_{pk_{REV}}(u_j), \sigma_j(H(r_5, u_j)))$. Upon reception, REV verifies the integrity and the freshness of the message and checks if $u_j \in rl$. In such case, REV drops the connection since u_j has been revoked. Otherwise, REV generates a token τ_{ks} and sends the following to u_j: $m_6 = (r_6, E_{pk_{KT}}(u_j, \tau_{ks}), \sigma_{REV}\left(H\left(r_6 \left|\right|u_j\right| \left|\tau_{ks}\right)\right))$. Upon reception, u_j forwards m_6 to the KeyTray who verifies the signature as well as the freshness and user's id and sends $c_p^{K_i}$ to u_j. At this point, u_j uses her private CP-ABE key to recover K_i.The decryption will only work if the attributes that are associated with u_j's key satisfy the policy that is associated with $c_p^{K_i}$. Apart from that, the KeyTray sends also the following to u_j: $m_7 = \left(E_{pk_{CSP}}(u_j, t), \sigma_{KT}\left(H(u_j \left|\right|t)\right)\right)$, where t is the time that u_j accessed $c_p^{K_i}$. As we will see in the next step, t plays a crucial role in the access control.

ASCLEPIOS.Search : Now that u_j has gained access to K_i, she can start searching directly over encrypted data. Let's assume that u_j wishes to search over the ciphertexts that have been encrypted with K_i, for a specific keyword w. To do so, she first forwards to the CSP m_7 that received in the previous step. Upon reception, CSP recovers u_j's identity and the timestamp t, verifies the signature and then checks if t is valid. We assume that there is a time interval since u_j got access to K_i, where she is eligible to access files that are stored in the CSP. After that time, u_j will have to run again the previous step in order to receive a fresh timestamp. This will guarantee that u_j has not been revoked since the last time that got access to K_i. Then, if all the verifications are successful, u_i runs SSE.SearchToken $(K_i, w) \rightarrow \tau_s(w)$ and obtains a search token $\tau_s(w)$. Then, she sends the generated token to the CSP who runs SSE.Search$(\gamma_i, \boldsymbol{c_i}, \tau_s(w)) \rightarrow I_w$ that outputs a sequence of file identifiers I_w, such that $I_w \subset \boldsymbol{c_i}$. In addition to that, all files in I_w contain the keyword w that u_j searched for. The resulted I_w is sent back to the user. Upon reception, u_j executes the SSE.Dec algorithm by giving as input K_i and the sequence of encrypted files that corresponds to the list of identifiers that received from the CSP. By doing this, u_j recovers the files that contain keyword w.

ASCLEPIOS.**Update** : Apart from storing data and searching over the encrypted data, users also need to be able to update stored data. Here, we consider the scenario where u_j wishes to add a new file f to the cloud storage. A naive approach that u_j could follow would be to run ACLEPIOS.Store again, generate the ciphertext of f and send it to the CSP. However, this would mean that u_j would also create a new encrypted index that would correspond to the encryption of file f. Such an approach is not efficient since the user would end-up with a long list of encrypted indexes that are not related to each other and every time that wishes to perform a search over her data would require from the CSP to search over all the encrypted indexes. To avoid this, u_j needs to store f but instead of creating a separate encrypted index she needs to update the current one in order to also include the newly added file. To achieve that, u_j first generates an add token by executing $(\tau_\alpha(f), c_f) \leftarrow$ AddToken(K_i, f) and sends it to the CSP. Upon reception, CSP executes SSE.Add (γ_i, $\boldsymbol{c_i}$, $\tau_\alpha(f)$, c_f) $\rightarrow$ (γ_i', $\boldsymbol{c_i}'$) and outputs an updated encrypted index γ_i' and an updated sequence of ciphertexts $\boldsymbol{c_i}'$ that corresponds to the data stored by u_j. Thus, by running SSE.Add, CSP stores the ciphertext of f and updates the existing encrypted index and ciphertext list of u_i.

ASCLEPIOS.**Delete** : Users must also be able to delete a file. Assume that u_j wishes to delete a file f . To do so, u_j runs SSE.DeleteToken which takes as input the symmetric key K_i and the file that needs to be deleted and outputs a delete token: $\tau_d(f) \leftarrow$DeleteToken (K_i, f) which is sent to the CSP. Upon reception, the CSP first checks that u_j is eligible to delete a file and she has not been revoked (this is done by opening m_7 and looking at the timestamp provided by the KeyTray). Then, the CSP runs SSE.Delete (γ_i, $\boldsymbol{c_i}$, $\tau_d(f)$) $\leftarrow$ (γ_i', $\boldsymbol{c_i}'$) which removes the requested file f and updates both the corresponding encrypted index and the sequence of ciphertexts.

ASCLEPIOS.**Revoke** : The last phase of our protocol allows a data owner to revoke access to a user. We assume that u_i wishes to revoke access to u_j. To do so, u_i contacts the revocation authority (REV) by sending $m_8 = \left(r_8, E_{pk_{REV}}(u_i, u_j, c_p^{K_i}), \sigma_i\left(H\left(r_8||u_i\left||u_j\right||c_P^{K_i}\right)\right)\right)$. Upon reception, REV checks the integrity and the freshness of the message and recovers the identity of data owner (u_i) as well as the user that needs to be revoked (u_j). Then, REV contacts the KeyTray by requesting the ciphertext of r_{K_i} that was stored next to $c_p^{K_i}$ during the run of ASCLEPIOS.KeyTrayStore. So, KeyTray sends the following message to REV: $m_9 = (r_9, E_{pk_{u_i}}(r_{K_i}), \sigma_{KT}\left(H\left(r_{K_i}\right||r_9)\right))$. Upon reception, REV forwards m_9 to u_i who recovers r_{K_i} and verifies that the message has been generated by the KeyTray (verifying the signature). Then, u_i signs r_{K_i} and sends it to the KeyTray through REV. KeyTray verifies the signature and is also convinced that u_i is the owner of K_i. Hence, KeyTray generates a fresh random number r_{K_i}' that replaces r_{K_i} and also sends an acknowledgement to REV that u_i has the right to revoke access to u_j for all files that are encrypted with K_i. Finally, REV adds the identity of u_j in rl. As a result, the next time that u_j will try to access any of the files that are encrypted with K_i access will be denied.

9.3 Protocol Security

We now analyze our protocol's behavior in the presence of a malicious adversary. We prove the security of the scheme by showing its resistance to a list of malicious behaviors. In this part the security analysis explicitly focuses on the described protocol and not on the underlying cryptographic schemes. This analysis is based the ones described in [19] and [20]

Realistic Assumption. We assume that all user ids have the same length (or at least they are not a prefix of each other). By ensuring this property is satisfied, we can avoid a prefix attack, such as the following:
Assume two users with ids u_0 = 001 and u_1 = 0011 respectively. Then if an adversary *ADV* gets a valid signature $\sigma_{KT}(H(u_1||t))$ (from a valid m_7) this will be the same as a valid signature on u_0 with a much larger time. However, by setting all users' ids to have the same length we avoid such an attack.

Proposition 1 (Compromise Revoked Users). Let $\mathcal{U}$ be the set of all users that have been given access to $\mathrm{K_i}$ and $\mathcal{R}$ the set of all users that their access to $\mathrm{K_i}$ has been revoked. Assume an adversary ADV corrupts n, $\mathrm{n} \leq |\mathcal{R}|$ users out of those in the set $\mathcal{R}$. Then ADV cannot infer any information about the files that have been encrypted with $\mathrm{K_i}$.

Proof. Here, we consider the case where ADV corrupts at least one user $\mathrm{u_c} \in \mathcal{R}$. In other words, ADV corrupts at least a user who in the past was eligible to use $\mathrm{K_i}$ and therefore she was able to decrypt all files from $\mathbf{c_i}$ that were encrypted with that key. ADV will try to use $\mathrm{u_c}$ in order to obtain the collection of ciphertexts $\mathbf{c_i}$ and access the contents of the files.
ADV trying to access the content of any file in $\mathbf{c_i}$ can succeed if all the following conditions hold:

1. Access the symmetric key K_i that used to encrypt the files.
2. Successfully bypass the authentication of CSP during the ASCLEPIOS.Search phase.
3. Access the latest ciphertexts list c_i^{fresh}

- Condition 1 is always true. We know that $uu_c \in \mathcal{R}$. Therefore, at some point in the past u_c was member of $\mathcal{U}$. Hence, we can safely assume that u_c was able to decrypt $c_p^{K_i}$ and recover K_i

- Condition *2* can only be true if the adversary convince the CSP that $u_c \notin \mathcal{R}_\mathrm{i}$. To do so, ADV needs to generate a valid m_8 message that will also contain a fresh timestamp t_c, that will proove that $u_c$received access to K_i recently and is still active. Generating a valid m_8 message can be done with the following two options:

    - *Replay Old Message:* First, we consider the case were ADV replays an older message m_7 in order to generate a valid m_8

that will allow her to bypass the checks of the CSP. To this end, ADV uses the following message that was received in the past: $m_7 = (E_{pk_{CSP}}(u_c, t), \sigma_{KT}(H(u_c, t)))$. This is a valid message that contains the identity of the corrupted user as well as a valid signature from the KeyTray. Then ADV generates a fresh random number r and creates a new m_8 that is sent to the CSP. Even though the structure of the generated message is correct, the CSP will drop the connection since it will identify it as an old message. This is due to the fact that the timestamp t contained in m_7 has expired. Therefore, the CSP cannot be sure if u_c's access right is still active. To bypass that, ADV will try to replace t with the current time t_c. To do so, the adversary will use pk_{CSP} to generate $E_{pk_{CSP}}(u_c, t_c)$ and replace the first part of m_7. However, the second part of the message has a signature from the KeyTray that contains the initial timestamp t. Replacing this with a valid signature on t_c fails due to the assumption of soundness of the signature scheme. Therefore, ADV will fail to bypass CSP's authentication.

- *Impersonate a Legitimate User* : The only remaining alternative for the adversary is to impersonate a legitimate user u_l from the set $\mathcal{U}$. To so, *ADV* overhears the communication between u_l and the CSP. By doing this, *ADV* intercepts the message m_8 that u_l sent to the CSP. This message is fresh and contains an acceptable timestamp t. However, it also contains (in m_7) the identity of u_l. This will be used at the end of the ASCLEPIOS.Search phase where the CSP will use pk_{u_l} to encrypt the data that will be sent to the user. Therefore, *ADV* will use pk_{CSP} and will replace $E_{pk_{CSP}}(u_l, t)$ with $E_{pk_{CSP}}(u_c, t)$ we denote the new message as m^c. In addition to that, she will calculate *a new signature that will be included in* m_8 *along with* m^c.Upon reception, CSP will verify the first signature but will fail to verify the one that is included inm^{cc}. This is due to the fact that *ADV* had to change the identity of the legitimate user to u_c but she could not generate a valid signature on the new message. Hence, the attack will fail.

- Condition 3 cannot be true. This is implied immediately from the exculpability of the previous attack. More precisely, in order fot ADV to access c_i^{fresh}, she needs to bypass the CSP's authentication. However, we showed that this is not possible.

Proposition 2 (Revoke Legitimate Users). Let $\mathrm{u_i}$ be the owner of data that has been encrypted with $\mathrm{K_i}$. Additionally, let $\mathcal{U}_{\mathrm{CSP}}$ the set of all users that have been given registered with the CSP and $\mathcal{U}$ be the set of all users that have been given access to $\mathrm{K_i}$. Assume an adversary ADV corrupts a user $\mathrm{u_c}$, $\mathrm{u_c} \in \mathcal{U}_{\mathrm{CSP}} \setminus \{\mathrm{u_l}\}$. Then ADV cannot successfully revoke access to any $\mathrm{u_l} \in \mathcal{U}$.

Proof. Here, we consider the case where *ADV* corrupts a user u_c such that $u_c \in \mathcal{U}\backslash\{u_l\}$. The attack will be successful if *ADV* manages to revoke access to data that has been encrypted with K_i for a legitimate user $u_l \in \mathcal{U}$. To do so, *ADV* needs to run ASCLEPIOS.Revoke and convince REV that she is the data owner. Hence, ADV generate the corresponding message and sends it to REV. Upon reception, REV checks the integrity and the freshness of the message and recovers the identity of u_c, who is pretending to act as data owner, as well as the id of user that needs to be revoked u_l. Then, REV contacts the KeyTray by requesting the ciphertext of r_{K_i} that was stored next to c_p^K. The ciphertext of the random number r_{K_i} is then forwarded to u_c who fails to recover it as it is encrypted with the public key of u_i. Therefore, the attack fails.

9.4 Simulation-based Security

In this section we capture the notion of security by using the standard real experiment versus ideal experiment formalization. In particular, in the real experiment the adversary observes the algorithms being executed honestly, while in the ideal experiment a simulator S simulates the functionalities of the protocol based on specified leakage from the SSE scheme.

Definition (Sim-Security). We consider the following experiments. In the real experiment, all algorithms run as defined in our construction, In the ideal experiment, a simulator $\mathcal{S}$ intecepts ADV's queries and answers with simulated responses.

Real Experiment:

1. $\boldsymbol{EXP^{real}(1^{\lambda})}$:
2. $\boldsymbol{(MPK, MSK) \leftarrow ASCLEPIOS.Setup(1^{\lambda})}$
3. $\boldsymbol{(\gamma, c) \leftarrow ADV^{SSE.Setup(K,f)}}$
4. $\boldsymbol{ASCLEPIOS.Search() \rightarrow I_w}$
5. $\boldsymbol{ASCLEPIOS.Update() \rightarrow (\gamma', c')}$
6. $\boldsymbol{ASCLEPIOS.Delete() \rightarrow (\gamma', c')}$
7. $\boldsymbol{Output\ b}$

Ideal Experiment:

1. $\boldsymbol{EXP^{ideal}(1^{\lambda})}$:
2. $\boldsymbol{(MPK) \leftarrow \mathcal{S}(1^{\lambda})}$
3. $\boldsymbol{(\gamma, c) \leftarrow ADV^{\mathcal{S}(\mathcal{L}_{in}(f))}}$
4. $\boldsymbol{\mathcal{S}(\mathcal{L}_{search}) \rightarrow I_w}$
5. $\boldsymbol{\mathcal{S}(\mathcal{L}_{add}) \rightarrow (\gamma', c')}$
6. $\boldsymbol{\mathcal{S}(\mathcal{L}_{del}) \rightarrow (\gamma', c')}$
7. $\boldsymbol{Output\ b}$

We say that the protocol is *Sim-Secure* if for all probabilistic polynomial time adversaries ADV:

$$|\Pr[(Real) = 1] - \Pr[(Ideal) = 1]| \leq negl(\lambda)$$

At a high-level, we will construct a simulator that will replace all of the ASCLEPIOS algorithms. $\mathcal{S}$ can simulate Key generation and encryption oracles. S is given the length of the challenge message as well as the leakage functions $\mathcal{L}$ as they were defined in D2.1. Moreover, since the data owner is the only one who can run ASCLEPIOS.Revoke we do not include that algorithms in the simulator. We start by defining the functionalities $\mathcal{S}$.

- $\mathcal{S}.Setup$: This algorithm will only generate MPK that will be given to ADV.
- $\mathcal{S}.Store$: $\mathcal{S}$ generates a dictionary that will enable it to consistently reply to search queries even after file additions and deletions.
- $\mathcal{S}.KeyShare$: $\mathcal{S}$ encrypts K_{adv} under MPK and sends it back to ADV. Moreover $\mathcal{S}$ simulates and sends to ADV the corresponding messages.
- $\mathcal{S}.Search/Update/Delete$: $\mathcal{S}$ gets as inputs the corresponding SSE leakage function and simulates the tokens.

Theorem: *Let $SKE = (Gen, Enc, Dec)$ be an IND-CPA secure symmetric key cryptosystem. Moreover, let Sign be an EUF-CMA secure signature scheme. Then, our construction is sim-secure.*

We will now use a hybrid argument to prove that ADV cannot distinguish between the real and ideal experiments.

Hybrid 0: Everything runs normally.

Hybrid 1: Like Hybrid 0 but $\mathcal{S}.Setup$ runs instead of $ASCLEPIOS.Setup$. These algorithms are identical from ADV's point of view since in both algorithms ADV is only given MPK and hence, they are indistinguishable.

Hybrid 2: Like Hybrid 1 but $\mathcal{S}.Store$ runs instead of $ASCLEPIOS.Store$. Nothing changes from ADV's perspective since the simulated index has exactly the same size and format like the real one. Moreover, the IND-CPA security of the symmetric encryption scheme ensures that ADV cannot distinguish between encryptions of files and encryptions of zeros.

Hybrid 3: Like Hybrid 2 but $\mathcal{S}.KeyShare$ runs instead of $ASCLEPIOS.KeyShare$.

Lemma*: Hybrid 3 is indistinguishable from hybrid 2.*

***Proof**:* By replacing the two algorithms, nothing changes from ADV's point of view. Moreover, if ADV can generate m_6 without having contacted REV before, then she can produce a valid REV's signature. However, given the security of the signature scheme, this can only happen with negligible probability. Hence, the hybrids are indistinguishable.

Hybrid 4: Like Hybrid 3 but $\mathcal{S}$ is now given as input the leakage functions from the SSE scheme and simulates search, add and delete tokens.

***Lemma**: Hybrid 4 is indistinguishable from hybrid 3.*

Proof: Proof is omitted as it is included in D2.1.

And this hybrid concludes our proof. We managed to construct a simulator $\mathcal{S}$ that can "fool" any PPT adversary ADV into thinking that she is running the real algorithms whereas she is only getting simulated responses.

10 Experiments

For the implementation of CP-ABE, we used Agrawal's scheme [8] shipped by Charm-Crypto Framework version 50.0 inside a Charm-Crypto Docker container. The Charm-Crypto Framework was developed in Python language. Therefore, these experiments were implemented in Python 3.6. All CP-ABE experiments were conducted on a desktop machine with Intel Core i7-8700 at 3.20GHz (6 cores), 32GB RAM.

10.1 Setup Phase

The first phase of the experiments, we devoted to measuring the execution time required to generate a pair of keys for a master entity. This is part of the setup phase for ASCLEPIOS, we consider the existence of at least one master entity responsible for the generation of CP-ABE keys. However, it can be argued that the overall security of the system cannot rely on one single master authority. Furthermore, in cases where it is required to achieve a high level of security, a master secret key pair might be generated for each data owner. Such an approach could also lead to a multi-authority ABE model, as described in [14].

In contrast, it has been observed that in multi-authority settings malicious adversaries may collude from the different authorities. Taking all cases into consideration, we ran a key generation algorithm increasing the number of generated master key pairs up to 200. The result of the experiment is illustrated in Figure 3(a). As can be seen in the graph, the time to generate 200 master key pairs took almost six seconds. Moreover, the time to generate one master key pair is less than a second, which is considered acceptable. Apart from that, in Figure 3(a), the time is growing linearly with the increasing number of key pairs that are being generated.

10.2 Users Key Generation

In the second phase of the experiments, we show the average time that it takes to generate the user's keys. It is essential to mention that we developed an algorithm which automatically generates a list of attributes with a different length since each ABE user’s key is associated with attributes. Namely, we measured execution time for several users’ keys, increasing the number of attributes associated with the key. The results showed that even for a very large organization and/or publicly available online services, the size of attributes bound to a key is considered acceptable. As can be seen in the Figure 3(b), the average time to generate a user key with 1,000 attributes took almost six-and-a-half seconds, and to generate a user key with 500 attributes took approximately three-and-a-half seconds. These results are suitable for covering even more complex cases where companies are required to generate large keys based on a wide variety of information. Thus, it can be stated that covering a long list of attributes is realistic and should not prevent an organization from adopting such an approach.

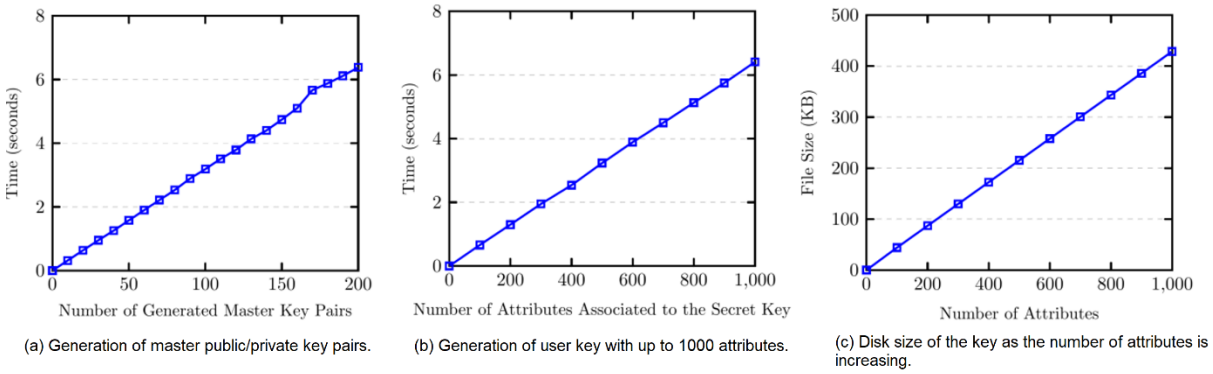
In addition to that, it is interesting to observe how the file size of the user’s key is changing with the increase in several attributes associated with the key. In Figure 3(c), we can see that the size of the generated user's key is increasing precisely in line with growth in the number of attributes. The results showed that the disk size of one key with 1,000 attributes is around 420KB, while a key with 500 attributes almost twice as less 210KB. Moreover, the key associated with 50 attributes has a size of 50KB on the disk.

10.3 Encryption/Decryption

In ASCLEPIOS, we use CP-ABE to encrypt a symmetric key and not large volumes of data. Hence, we measured the time needed to encrypt and decrypt a symmetric key under policies of different sizes. We used access policies of type *“1 and 2 and … and n”* similar to [15]. Such size is the most demanding as it requires a key to contain all attributes associated with the policy for the successful decryption. The experiment can be divided into two stages. In the first stage, we measured the encryption process. Namely, we ran an encryption algorithm on a message with different policies. In the second stage, we were decrypting the freshly generated ciphertexts with keys that are associated with a different number of attributes. In addition to that, we were adding access policies of a different structure in order to record the performance of the decryption not only when all conditions needed to be fulfilled (most demanding case), but also when a random number of attributes is needed to satisfy the underlying policy.

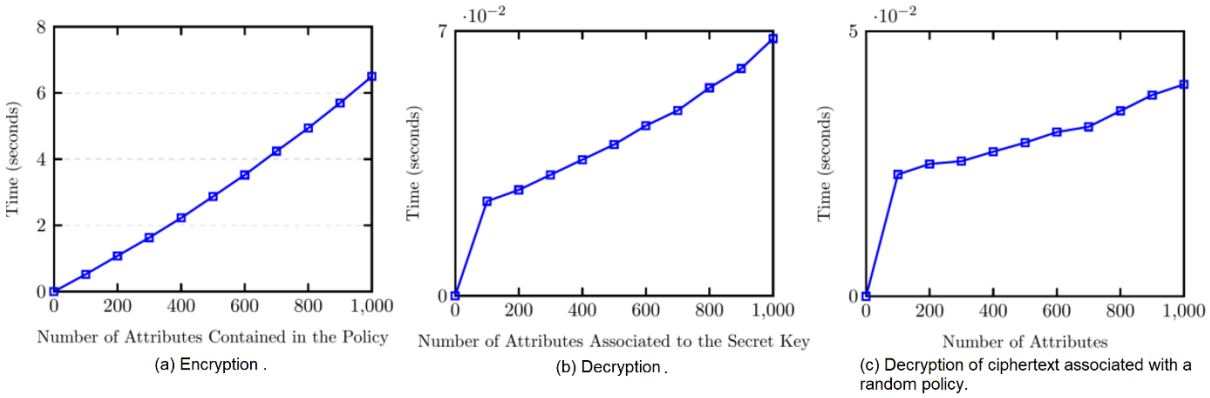
During the first stage of the experiment, we generated access policies of type *"1 and 2 and ... and n"* similar to the [15] and [8]. The policy is considered the most demanding case since all n attributes are required for successful decryption. Figure 4(a), demonstrates that the encryption time required to encrypt a message with a random policy of size up to 1,000. Figure 4(b) shows the time needed to decrypt the ciphertext of the message by using a key with up to 1,000 attributes where all were required to satisfy the policy. As it can be seen in the Figure 4(a, b), the time to encrypt and decrypt a message depends on the particular attributes available and the size of the policy. Namely, the encryption of the message with a policy size of 1,000 attributes took approximately 6.3 seconds where the decryption time took almost 0.070 seconds. However, for much more realistic scenarios where policies contain around 200 attributes, the encryption time took around a second and the decryption time was less than a half of the second. It is evident that the underlying CP-ABE scheme does not add any real computational burden to the overall performance of the protocol.

In the second stage of the experiment, we focused on analyzing the behavior of the underlying CP-ABE scheme. More precisely, we created an algorithm which randomly generates a policy that contains numerical attributes as well as conditions such as the following: *“(1 and 2) or (3 and 4)”*. This condition required that at least one of two parenthesis are satisfied by the attributes associated with the key that is trying to the decrypt a ciphertext. Figure 4(c) shows the time needed to decrypt a ciphertext previously bound with a policy size up to 1,000 but generated randomly. In the other words, unlike previous experiment, it does not require a key with all attributes. From the result shown in the graph, we can observe that the decryption time is linear regardless the randomness of the policy. This indicates that the decryption time is more less the same as long as the size of the policy is the same length.



(a) Generation of master public/private key pairs. (b) Generation of user key with up to 1000 attributes. (c) Disk size of the key as the number of attributes is increasing.

Figure 3: Processing time for the generation of user keys and measurement of the required disk space



(a) Encryption . (b) Decryption . (c) Decryption of ciphertext associated with a random policy.

Figure 4: CP-ABE encryption and decryption with a policy size of up to 1000

11 Integration in Constrained Devices

In this section, we are attempting to integrate parts of the proposed protocol to constrained devices. In particular, we are working with Zolertia devices that have very limited resources (512KB flash, 32MHz and 32KB of RAM). Hence, we can be sure that whatever runs on these devices will also run on medical devices. Apart from that, it should be mentioned that what we are presenting in this section is just a first step in integrating cryptographic software into medical devices and as such, at its current state may not be fully compatible with the needs of ASCLEPIOS.

11.1 Components

In this section, we present a detailed description of how the SSE parts of our protocol could be integrated in constrained devices. More specifically, we design a protocol consisting of two main algorithms; AddData and SearchData. The proposed algorithms are heavily influenced by the SSE scheme presented in D2.1. However, we need to make some modifications as in this case we are regard the Zolertia devices as the data owners. Before we proceed to the formal construction, we give a high level description of the different entities that participate in the construction, with respect to D1.2 and D2.1.

- Data owners: Let $\mathcal{D} = \{d_1, \dots d_n\}$ be the set of all sensor nodes in our environment deployed to register the occurrence of specific environmental events. The data owners in our system are able to add and update encrypted data using the proposed scheme. For the purposes of our implementation, we utilize the Zolertia Re-Mote board devices that are based on the Texas Instruments CC2538 ARM Cortex-M# system on chip (SoC). These boards feature a 2.4GHz IEEE 802.15.4 RF Interface, running up to 32MHz with 512KB of programmable flash and 32KB of RAM while possessing a built-in battery charger (500mA) with energy harvesting capabilities as well as a CC1200 868/915MHz RF transceiver which allows for dual band operation. The functions performed by a Zolertia device are:

    1. Register the occurrence of a sensed event (e.g. Temperature, Humidity, etc). Data about a sensed data is referred to as keyword throughout the rest of this section and is denoted by w_i.
    2. Generate a hash of data about every sensed event $h(w_i)$
    3. Generate a unique identifier id_j for each sensed event based on the sensing device's id, timestamp and the nature of the event (e.i. temperature, humidity, etc).
    4. Encrypt the unique identifier with a symmetric key K to generate a keyword value, $c_{id(w_i)}$, that corresponds directly to each keyword.

The sensor device then sends both $h(w_i)$ and $c_{id(w_i)}$ to a trusted authority TA.

- Trusted Authority (TA): TEE enabled storage that stores the following indexes:

    1. NoApp $[w_i]$, which contains a hash of the keyword along with the number of times it has received that keyword.
    2. NoSearch $[w_i]$, which contains a has of the keyword along with the number of times the keyword has been searched for

- Cloud Service Provider (CSP): TEE enabled cloud storage that contains a dictionary DICT. DICT contains a mapped between each keyword and a unique identifier of the sensor device that sent it.

11.2 Formal Construction

The AddData algorithm is undertaken by both a sensor device d and the TA. The sensor device is deployed to register the occurrence of an environment event event such as temperature, humidity, motion, etc. Data about a sensed event is referred to as a keyword w_i Once an event has been registered, w_i is hashed to produce $h(w_i)$. d then generates a unique identifier that will be used to identify the particular keyword. This unique identifier ID is made up on the sensor device's id, timestamp and the type of event being registered (i.e. temperature, humidity, etc). The ID is then encrypted with a secret shared key K to produce $c_{id(w_i)}$ (line 4 of algorithm 1). The sensor device sends $h(w_i)$ and $c_{id(w_i)}$ to the TA. Upon reception, TA retrieves the corresponding NoApp and NoSearch indexes from its local database. Based on these indexes TA can compute the DICT addresses that will be sent to the CSP.

Algorithm 1 Add Data

Sensor

1: Register data about a sensed event w_i
2: Compute hash of the data $h(w_i)$
3: Generate a unique identifier for the sensed data ID. (ID = SensorID||t||T), where SensorID is the unique id of the sensor, t is the timestamp, T is the type of the measurement.
4: Compute $c_{id(w_i)} = Enc(K, ID)$
5: Send $h(w_i), c_{id(w_i)}$ to the TA

TA

6: NoApp[$h(w_i)$] + +
7: K_{w_i} = $H(K_h, h(w_i)$||NoSearch[$h(w_i)$]) \\ Where K_h is the key for a keyed has function and K_w the keyword key as defined in D2.1
8: $addr_{w_i}$ = $h(K_{w_i}$, NoApp[$h(w_i$)]||0)
9: Map = Map ∪ $\{addr_{w_i}, c_{id(w_i)}\}$
10: Send [Map] to the CSP

CSP

11: Add Map into central Dict

The search algorithm enables users that posses the secret key (obtained as described in Section 9) to perform a search operation over the encrypted data. To do so, a user first hashes the keyword she wants to search for, and sends $h(w_i)$ to the TA. Upon reception, TA retrieves the NoApp and NoSearch indexes from its local database and computes the keyword key K_w as $K_w = h(K_h, h(w_i)||NoSearch[w_i])$ in order to calculate the addresses of all possible instances of $h(w_i)$ in DICT. More specifically, TA generates a list L_s containing all the addresses on DICT. As a next step, TA increases the NoSearch Times by one which is then used to compute a fresh K_w and the new addresses for DICT. The new addresses are stored in a list L'_s. Finally, the two lists are sent to the CSP along with the user's identity. Upon reception, the CSP uses L_s to find all the $c_{id(w_i)}$ and stores them in a list R. As a next step, it

removes the current addresses and inserts the new ones contained in L'_s. Finally, R is sent back to the user.

Algorithm 2 Search Data

User

1: Compute and send $h(w_i)$ to TA

TA

2: Retrieve the values NoApp[$h(w_i)$] and NoSearch[$h(w_i)$] from the local database
3: K_{w_i} = $H(K_h, h(w_i)||\text{NoSearch}[h(w_i)])$
4: NoSearch[w_i] + +
5: K'_{w_i} = $H(K_h, h(w_i)||\text{NoSearch}[h(w_i)])$
6: $L_s = \{\}$
7: for i = 1 to i = NoApp[$h(w_i)$] do
8: $addr_{w_i} = h(K_{w_i}, i||0)$
9: $L_s = L_s \cup \{addr_{w_i}\}$
10: $L'_s = \{\}$
11: for i = 1 to i = NoApp[$h(w_i)$] do
12: $addr'_{w_i} = h(K'_{w_i}, i||0)$
13: $L'_s = L'_s \cup \{addr'_{w_i}\}$
14: Send (L_s, L'_s) to the CSP

CSP

15: $R = \{\}$
16: for i = 1 to i = Sizeof(L_s) do
17: $c_{id(w_i)}$ = Dict[$L_s[i]$]
18: $R = R \cup \{c_{id(w_i)}\}$
19: Delete the row on Dict and update it according to the address in L'_s
20: Send R to the user

12 Experimental Evaluation in Constrained Devices

In this section, we present the results of experiments conducted to demonstrate the feasibility of our proposed work. Our experiments focused primarily on evaluating the performance of the algorithms de-scribed in section 5 on both the sensor device and the TA. For this work, we utilized a zolertia device with 512KB programmable flash and 32KB RAM as our sensor device while using an Intel i7 Ubuntu desk-top with 16GB RAM as the TA. To implement the necessary algorithms, we developed a Contiki-NG application on the sensor device written in C, using modified cryptographic functions from the Tinycrypt library [Wood, 2019]. On the TA, we developed a node js application to interact with a local database. With regards to this database, our dictionary is implemented as tables in a MySQL database hosted on the TA. Although existing works in the field of SSE rely on data structures such as arrays, maps, sets, lists, trees, etc, we opted for a relational database to represent a persistent storage. The experiments measure the performance of the core cryptographic components of our work on the re-source constrained sensor device, as well as the over-all performance of the add and search algorithms us-ing datasets of arbitrary sizes.

Datasets: To comprehensively measure the performance of both the search and add algorithms, it was important that we utilized datasets of different sizes. Due to the uniqueness of our work, the datasets had to be created as part of our experiments (i.e. using the add algorithm of the protocol). We left the sensor device to collect the temperature in a room every 5s for a varying number of hours and forwarded that in-formation to the TA. We did this for 1hr, 4hrs, 12hrs and 24hrs with a temperature range of 10-to-35 degrees (Figure 1).

Table 33: Dataset Size

	Duration (hrs)	Number of Entries
DS1	1	737
DS2	4	2,844
DS3	12	8,617
DS4	24	17,287

Add Algorithm: This part of our work consists of two phases and is performed on both the sensor device and the TA:

PH1: The sensor device collects data on a sensed event, generates a unique id, hashes the data about the sensed event, encrypts the unique id, and finally sends both the hashed data and the ciphertext to the TA;

PH2: The TA retrieves the NoApp and NoSearch from the database based on the hashed message received from the sensor device, builds the en-crypted index and generates the dictionary.

We measure the total performance of the add algorithm by evaluating the performance of the cryptographic components on the sensor device and the time taken by the TA to complete the algorithm.

Performance of Cryptographic Components on Sensor Device: As mentioned during the description of our dataset, the sensor device is left to collect measurements for a varying

period of time. From table 1, it is observed that, for a timespan of 24 hours, the sensor device and the TA node run through various portions of the add algorithm 17,287 times. The system time for the CC2538 platform for which the sensor device is based on is represented as CPU ticks. As a result of this limitation, the performance metrics on the sensor devices are recorded in ticks and externally converted to seconds. Specific figures are derived by dividing the number of ticks by 128 (CPU ticks per second [Kurniawan, 2018]). In 17,287 iterations of the first part of the add function, the sensor device takes an average of 4.5 ticks to generate the hash of the keyword (temperature) and the ciphertext of the unique filename. This corresponds to 0.035s

Execution Time on the TA: In this part of our experiments, we measured the time taken by the TA to build the index table and generate the encrypted dictionary. For 17,287 runs of our protocol, the TA takes an average of 14.516ms for each keyword hash received from the sensor device. This time includes the time taken to query and update the database. This is a very encouraging result as it illustrates that the TA will continue to be very efficient even if we increase the number of sensor devices that communicate with it. We acknowledge that the results for this section would better resemble real life scenarios if we utilized multiple sensor devices. Unfortunately, our current implementation supports just one sensor device per TA.

From the measurements described above, it can be seen that the add algorithm is quite efficient and fast. Hence, we can safely assume that there will be no backlog on both the sensor device and the TA even if the sensor device collects data every 1s and sends to the TA. The total execution time of the add algorithm is 0.0495s.

Search Algorithm: In this part of our experiments, we measured the total time taken to complete the search algorithm over the encrypted dictionary generated by the add algorithm. As described in Section 5, the search algorithm is performed on the TA in a local search and is performed on the CSP in a global search. For the purposes of our experiments, we assumed that the CSP has the same specifications as the TA. Hence, the performance of both the local and global search will only vary based on the size of the dataset. The search time is calculated by measuring the following:

1. Time taken by the TA to generate a search token from a hashed keyword sent by a user;
2. Time needed to find the respective matches in the database;
3. Generate a new keyword address to replace the address retrieved in the dictionary.

On average, the time taken to generate the search token is 0.066ms. The search algorithm involves gen-erating a new keyword address for every keyword value found. As such, the actual search time also in-cludes the time taken to generate new keyword ad-dresses. Searching for a keyword that appears 760 times in a database with 17,287 entries takes approx-imately 11.36s (i.e. time taken to find all the key-word values and generate new keyword addresses for all 760 instances of the keyword). The search al-gorithm for a keyword that appears 22 times in a database with 737 entries takes approximately 134ms. These two times represent the fastest and slowest search operation times for our experiments. The

search operation times correlate directly to the size of the dataset and the number of times the keyword appears in that dataset.

Table 34: Performance Summary

Function	Execution Time (ms)
Add (Sensor Side)	35
Add (TA Side)	14.516
Total Add Algorithm	49.51
Search Token Generation	0.066

13 Conclusion

Based on the efficiency and requirement analysis no clear 'winner' can be selected. The FAME crypto scheme is much more efficient, but has no revocation method and therefore does not meet the strong requirements listed in section 5.2.1. However, also DAC-MACS and RD-ABE do not meet all the strong requirements (or only partly) and those schemes are much less efficient than FAME. This means that none of the schemes, in their current form, are suited to be used in ASCLEPIOS, but they show interesting concepts which might be used, or at least taken into regard when designing a framework such as ASCLEPIOS.

Bibliography

[1] Amit Sahai, Brent Waters, Fuzzy Identity-Based Encryption, 2005
[2] Vipul Goyal, Omkant Pandey, Amit Sahai, Brent Waters, Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data, 2006
[3] John Bethencourt, Amit Sahai, Brent Waters, Ciphertext-Policy Attribute-Based Encryption, 2007
[4] Dennis Schroer, Feasability of End-to-end Encryption using Attribute Based Encryption in Health Care, 2016
[5] Christoph Bösch, Pieter Hartel, Willem Jonker, Andreas Peter, A survey of provably secure Searchable Encryption, 2015
[6] Victor Miller, The Weil pairing, and its efficient calculation, 2004
[7] Victor Miller and others, Short programs for functions on curves, 1986
[8] Shashank Agrawal and Melissa Chase, FAME: Fast Attribute-Based Message Encryption, 2017
[9] Allison Lewko and Brent Waters, Decentralizing Attribute-Based Encryption, 2011
[10] Kan Yang, Xiaohua Jia, Bo Zhang and Ruitao Xie, DAC-MACS: Effective data access control for multiauthority cloud storage systems, 2013
[11] Jianan Hong, Kaiping Xue and Wei Li, Comments on "DAC-MACS: effective data access control for multiauthority cloud storage systems"/security analysis of attribute revocation in multiauthority data access control for cloud storage systems, 2015
[12] Hui Cui and Robert Deng, Revocable and decentralized Attribute-Based Encryption, 2016
[13] Amit Sahai, Hakan Seyalioglu, Brent Waters, Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption. Annual cryptography conference, 2012
[14] Melissa Chase, Multi-authority Attribute Based Encryption, Proceedings of the 4th Conference on Theory of Cryptography, 2007
[15] Matthew Green, Susan Hohenberger and Bren Waters, Outsourcing the Decryption of ABE Ciphertexts, In Proceedings of the 20th USENIX Conference on Security, 2011
[16] Antonis Michalas, The Lord of the Shares: Combining Attribute-Based Encryption and Searchable Encryption for Flexible Data Sharing. 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19). Limassol, Cyprus 08 - 12 Apr 2019 ACM.
[17] Alexandros Bakas, Antonis Michalas, Modern Family: A Revocable Hybrid Encryption Scheme Based on Attribute-Based Encryption, Symmetric Searchable Encryption and SGX. 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019.
[18] Alexandros Bakas, Antonis Michalas, Multi-client symmetric searchable encryption with forward privacy, Cryptology ePrint Archive, Report 2019/813, https://eprint.iacr.org/2019/813(2019)
[19]Antonis Michalas, Alexandros Bakas, Hai-Van Dang, Alexandr Zalitko, MicroSCOPE: Enabling Access Control in Searchable Encryption with the use of Attribute-Based

Encryption and SGX. 24th Nordic Conference on Secure IT Systems, NordSec 2019, Aalborg, Denmark, November 18-20, 2019.
[20] Antonis Michalas, Alexandros Bakas, Hai-Van Dang, Alexandr Zalitko, Access Control in Searchable Encryption with the use of Attribute-Based Encryption and SGX, CCSW 19, Conference on Cloud Computing Security Workshop, November 2019.