M.Tech 4th Semester Thesis Project Report

Efficient and Secure Access Control for Sensitive Healthcare Data

Under the Supervision of:

Prof. Dr. Ir. Bart Preneel
Electrical Engineering Department
Katholieke Universiteit Leuven, Belgium

Prof. Bimal Kumar Roy
Applied Statistics Unit
Indian Statistical Institute, Kolkata

Submitted by:

Asmita Samanta
Roll no.- CrS1902
M.Tech CrS 4th Semester Student
Indian Statistical Institute

July 8, 2021

Declaration

We do hereby declare that **Miss Asmita Samanta** has done her Master Thesis under our guidance and this project report entitled **"Efficient and Secure Access Control for Sensitive Healthcare Data"** has been submitted for the partial fulfilment of the Internship in M.Tech 4th Semester for the year 2021 at Indian Statistical Institute, Kolkata.

Prof. Dr. Ir. Bart Preneel
Electrical Engineering Department
Katholieke Universiteit Leuven, Belgium

Prof. Bimal Kumar Roy
Applied Statistics Unit
Indian Statistical Institute, Kolkata

Acknowledgement

I take immense pleasure in thanking my primary supervisor **Prof. Bart Preneel**, Electrical Engineering Department, Katholieke Universiteit Leuven, Belgium, and my secondary supervisor **Prof. Bimal Kumar Roy**, Applied Statistics Unit, Indian Statistical Institute, Kolkata, for giving me a wonderful opportunity to carry out this project work.

Special thanks to my daily supervisors **Seyed Farhad Aghili** and **Dave Singelée** for their advice, guidence and useful suggestions, which helped me a lot.

Finally, yet importantly, I would like to express my thanks from the core of my heart to my beloved family for their blessings and to my friends for their help and wishes.

Asmita Samanta
Student
M.Tech in Cryptology and Security
Indian Statistical Institute
Kolkata

Abstract

Healthcare services produce and use a great deal of sensitive personal data. But the fact is that this healthcare data has very high black market value. Now to easily access the healthcare data we can think about an access control server. So if we want to make an accesss control server for healthcare data then it has to be very secure. On the other hand, this data also needs to be easily accessible by the patient itself and authorized care givers.

In this thesis we have studied an existing token-based access control solution which is being applied to protect medical data in a hospital and observed its security limitations. After that we modify that model using Multi-Authority CP-ABE, as a building block, to overcome the security limitations. We have proposed two modified models in our paper.

Our first model relies on centralized MA-CP-ABE, which is based on composite order bilinear group. Since it is a centralized model, there is an central authority. In my case External IAM plays the role of Central Authority. I have used External IAM and Policy Decision Point as my two attribute authorities. This MA-CP-ABE is computed on a composite order bilinear group. According to the security analysis, my first model is adaptively secure. We have done this security analysis in standard model.
Our second model relies on decentralized MA-CP-ABE, which is based on prime order bilinear group. Since it is an decentralized scheme so there is no central authority. Here also I have used External IAM and Policy Decision Point as my two attribute authorities. This MA-CP-ABE is computed on a prime order bilinear group. According to the security analysis, my second model is CPA secure. We have done this security analysis in random oracle model.
Our second model is more efficient according to the computation cost than the first model whereas our first model is more efficient according to the communication cost than the second model.

We have implemented the decentralized Multi-Authority CP-ABE scheme, which is the building block of our second model, to use in modified Access Control Model. We have implemented the code in Python and used Charm-crypto framework for the implementation. Because of using decryption out-sourcing our final decryption time has become constant, it does not depend on the size of the data consumer's attribute set or on the number of attributes in access policy. Also we have implemented a modified LSSS in our thesis which is more efficient than Charm's LSSS.
We have also introduced revocation property in the scheme and provided insights on how to implement the whole access control model in this thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Full Form
DO	Data Owner
DU	Data User
DC	Data Consumer
RD	Raw Data
RK	Random Key
EK	Encrypted Key
ED	Encrypted Data
WK	Wrapped Key
ABE	Attribute Based Encryption
CP-ABE	Ciphertext Policy Attribute Based Encryption
MA-CP-ABE	Multi-Authority Ciphertext Policy Attribute Based Encryption
KMS	Key Management Server
KWS	Key Wrapping Server
External IAM	External Identity and Access Management (IAM) server
LSSS	Linear Secret Sharing Scheme
CA	Central Authority
GID / gid	Global Identifier
AA	Attribute Authority
GS	Global Setup
AS	Authority Setup
KG	Key Generation for a user
EC	Encrypt
DE	Decrypt
ACS	Access Control System
U	User
ACL	Access Control List
RBAC	Roll-based Access Control

Chapter 1

Introduction

First we are going to discuss what is the motivation of this thesis, which problem we are going to address and then what is our solution idea.

1.1 Motivation

Healthcare services produce and use a great deal of sensitive personal data. This data has a high black market value and therefore is a lucrative target for data theft and ransomware attacks. Indeed, reports have shown that a healthcare record may be valued at up to $250 per record on the black market [23], which is significantly more than for example stolen financial records. It is obvious that this healthcare data needs to be strongly protected. However, on the other hand, this data also needs to be easily accessible by the patient itself and authorized care givers.

In this thesis I have studied a specific Token-Based access control system (recent access control model / basic model) of a hospital toolkit, its components and their functionalities. After that I identified the security limitations (threat) of this basic model and what we can do to avoid the threats of sensitive healthcare data leaking. Then I have studied about Attribute Based Encryption (ABE) schemes and various Access Structures. After getting these basic concepts, I have modified the basic model using multi authority ciphertext policy attribute based encryption (MA CP-ABE) and I have used linear secret sharing scheme (LSSS) as my access structure.

1.2 Recent Access Control Model and Security Limitations

At first I am going to discuss about basic idea of some simple access control models Access Control List (ACL), Roll-based Access Control (RBAC) etc [20] and why we can not use them for sensitive healthcare data. Then I shall explain the idea of the recent Access Control model for Health Care data, i.e. functions, components, responsiblities, and the possible security limitations with our solution idea.

1.2.1 Basic Idea of Some Simple Access Control Model

In the most simple access control model we can have three components, namely Data Owner (DO), Data User (DU) and one Cloud Storage associated with an access handling server.
The DO stores its data to the cloud and in most cases the admin of the system defines a list mentioning who can access this data. The access handling server associated with cloud storage remembers who can access this data.
Now whenever a DU wants to access some data, the access handling server associated with cloud storage checks whether he is authorized to access that data or not and then depending on that

the access handling server sends the data or error message $\perp$ to DU.

There were many access control model before token based access control model, like ACL, RBAC etc.

Access Control List (ACL) : In the early days (1970), to give access to a computing resource, the list of legitimate users was appended to the resource itself. This was then called the Access Control List (ACL) [22]. But the main problems of this model are :
(i) each time a resource was added, the administrator had to list all legitimated users again;
(ii) each time a new user is added, the administrators have to add him/her to the access list of each resource he/she may need.

Role-Based Access Control (RBAC) : Role-Based Access Control (RBAC) [21] has been introduced to grant access based on the roles that users own in their organization. If a user has a certain role in the organization, he/she must be granted to access a definite, but variable, list of resources. The roles are often associated to the group of users. Users are assigned to groups, and then groups are associated to roles and those roles are associated to resources. RBAC was a vast improvement compared to the management of simple lists of users for every resource. It also offeres an improved security.

But in those models all the raw data are stored in the cloud server. So there was a confidentiality issue. Now I am going to describe a token-based access control model where this confidentiality issue has been taken care partially (but not fully, actually we are going to modify this model and solve the confidentiality issue fully in this thesis). At first I am going to discuss about all the components and their responsibilities of our recent token-based access control model.

1.2.2 Components and their responsibilities

The following figure gives an idea about the components of recent token-based access control model. Here the both sided arrows indicate that there is a communication channel in between the components.

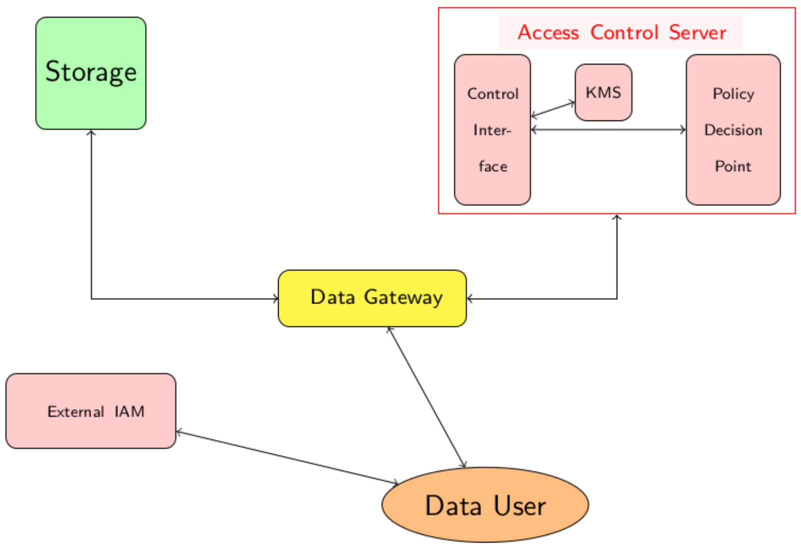


Figure 1.1: Components of Recent Access Control Model

At first I want to say what is Cloud Server, because many of our components are cloud components. The “Cloud" refers to the servers which can be accessed over the Internet. The softwares or databases which actually run on those cloud servers are cloud components in my case. These cloud servers are located in data centers all over the world. By using cloud computing, users and organisations don’t have to manage physical servers by themselves or run any software applications, which are cloud components, on their own machines. Now I am going to explain about the components.

1. **External IAM (External Identity and Access Management (IAM) server)** : It is a semi-honest (It only does its assigned jobs nothing else but it is curious. I have discussed the definition in the Section 2.6) server which verifies the identity of all users and generates token corresponding to there identity attributes. User uses this token as a proof of his/her identity to the other components.

2. **Data Gateway** : It is a semi-honest server and it is in cloud, i.e. it ia a cloud component. Data Owner and Data User communicate with Data Gateway to store the data or to access the data.
   At the time of data storing, Data Owner sends its raw data (RD) and its token (token_o) to the data gateway. After that Data Gateway generates a random key RK to encrypt the raw data using this key RK and get the encrypted data (ED). Then it also encrypts the encryption key (RK) and get encrypted encryption key (EK). After that it sends EK along with data owner’s token (token_o) to the Access Control Server to wrap EK.
   At the data consuming, Data Consumer sends its token (token_c) to Data Gateway. After that Data Gateway collects (ED, WK) from storage and sends the wrapped key along with data consumer’s token (token_c) to Access Control server and gets the unwrapped key EK or “Access Denied". If it gets EK, then it decrypts EK and get RK and then using RK it decrypts ED to get RD. It sends RD or “Access Denied" to Data Consumer.

3. **Access Control Server** : It is a semi-honest server, it is in cloud and it has three different components in it :

   (a) **Control Interface** : At the time of data storing Data Gateway sends encrypted encryption key EK and data owner’s token (token_o) to Control Interface and Control Interface sends EK to KMS and data owner’s token to Policy Decision Point. After the completion of key wrapping process Control Interface sends the wrapped key (WK) to the data gateway.
   At the time of data consuming, Data Gateway sends WK and data consumer’s token (token_c) to the Control Interface and Control Interface sends WK to KMS and consumer’s token (token_c) to Policy Decision Point. Control Interface gets unwrapped key EK and data owner’s parameter from KMS and it sends data owner’s parameters to Policy Decision Point. Control Interface finally gets either “Access granted" or “Access Denied" (I have explained the process at the time of explanation about Policy Decision Point) and send either EK or “Access Denied" to Data Gateway. It is basically a communication component of Access Control Server, i.e. if any other component like Data Gateway wants to communicate with KMS or Policy Decision Point of Access Control Server, then it has to continue the communication through Control Interface.

   (b) **KMS (Key Management Server)** : KMS basically wraps EK with the data owner’s parameters and unwrap WK.
   At the time of data storing, it gets EK from Control Interface and data owner’s parameters from Policy Decision Point and wrap EK with those parameters to produce

WK. KMS stores EK and sends WK to Control Interface.
At the time of Data consuming KMS gets WK from Control Interface and unwrap WK (using its key storage) to get EK and data owner's parameters. KMS sends those to control interface.

(c) **Policy Decision Point** : It actually handles all the access policies. Policy Decision Point decides that which data will be consumed by whom.
At the time of data storing Policy Decision Point gets data owner's token (token_o) from Control Interface and recovers data owner's parameters and sends these to KMS.
At the time of data consuming Policy Decision Point gets data consumer's token (token_c) and data owner's parameters from Control Interface. Policy Decision Point first recovers data consumer's parameters from its token (token_c). After that depending on data owner's parameter and data consumer's parameter, Policy Decision Point decides whether the consumer should get the access of the data or not.

4. **Storage** : This is the component in cloud in which data gateway stores the (ED,WK) pair for future access. Storage is not trusted at all and anyone can access the stored data.

The other two components in this access model is **Data Owner** and **Data Consumer** but they don't take part in message encryption or decryption.

This access control model uses the following algorithms :

1. **DataEncryption (RD) $\rightarrow$ (ED)** : Data Gateway runs this algorithm with raw data RD as input. Then it chooses a random encryption key RK to encrypt RD and to produce an encrypted data ED.

2. **KeyEncryption (RK) $\rightarrow$ (EK)** : Data Gateway runs this algorithm with encryption key RK as input to produce encrypted form of the key EK.

3. **KeyWrapping (EK, $token_o$) $\rightarrow$ (WK)** : Access Control server runs this algorithm with encrypted key EK and data owner's token $token_o$ as input. It takes help of KMS and Policy Decision Point to wrap EK (use the parameters of $token_o$) and produced wrapped key WK. It sends WK to the DataGateway.

4. **KeyUnWrapping (WK, $token_c$) $\rightarrow$ (EK)** : Access Control server runs this algorithm with wrapped key WK and data consumer's token $token_c$ as input. It takes help of KMS and Policy Decision Point to unwrap WK if and only if $token_c$ satisfies the data accessing criterias (set by Policy Decision Point) and produced either unwrapped key EK or $\perp$. It sends EK or $\perp$ to the DataGateway.

5. **KeyDecryption (EK) $\rightarrow$ (RK)** : Data Gateway runs this algorithm with unwrapped encrypted key EK as input to produce decrypted form of the key RK.

6. **DataDecryption (ED, RK) $\rightarrow$ (RD)** : Data Gateway runs this algorithm with encrypted data ED and encryption key RK as input to produce the decrypted data RD.

1.3 Security Limitation in Recent Model

This recent access control model seems very secure at a glance but if we observe it properly then we can see that here KMS has stored all EK's in its memory and anyone can access the stored pair (ED, WK) in Storage. Now if any attacker gets access of KMS somehow and can successfuly attack the Data Gateway, then using those access it actually get all the raw data from stored pairs (ED, WK) which is not expected at all.

Now since KMS and Data Gateway are in cloud, we can't gurantee the full security of those two components. A powerful Attacker may try to attack them any time. Here the actual problem is the Data owner sends its raw data to store in cloud and all the encryptions are computed in cloud and also all the decryption keys can be recovered by attacking the cloud components only.

1.4 Our Contribution (Solution Idea)

- We want to avoid the thing that the Data Gateway and KMS together can decrypt all the data stored in cloud storage. So, the data owner should encrypt the data before sending it to the Data Gateway.
- This creates a new problem. The problem is how can the data consumer decrypt this data. Using of simple private or public key encryption does not work here. Because if we use private key encryption scheme then Data Owner has to send the secret decryption key to all the Data Consumers by a secret channel, or if we use public key encryption scheme then data consumers have to send public keys corresponding to their secret decryption key to data owner and data owner has to encrypt a single data with many more keys, which are not good ideas. Moreover, the data owner does not necessarily know the data consumer in advance.
- As a solution to the above problem is to use Attribute Based Encryption (ABE), where the data owner can specify which users could decrypt the data, based on specific access control policies and any communication in between data owner and data consumer is not needed at all.

Now in ABE, we have two categories. The first is ciphertext-policy ABE (CP-ABE), and the second category is key-policy ABE (KP-ABE).
However, CP-ABE is much more appropriate than KP-ABE in our case because the access policy determination in CP-ABE is put on the data owner's hand which we actually want.
In CP-ABE, we also have two type of models : i) Single Authority CP-ABE, ii) Multi-Authority CP-ABE. In Single authority model we have to have a fully trusted authority. But in the basic model we don't have any fully trusted component and also we don't want to impose such strong condition. So, we discard single authority model and choose the Multi-Authority CP-ABE model.In multi-authority model we also have two categories : i) Centralized Multi-authority, ii) Decentralized Multi-Authority.
In Centralized models, there are many schemes which have to have one fully trusted central authority but I notice that the scheme in the paper [11], don't need a fully trusted central authority. It is sufficient to have a semi-honest central authority. So, I choose this scheme for my first modified model.
But the thing is this scheme uses composite order bilinear group operations which takes much more time than prime order bilinear group operations. So I have taken the idea of the papers [10, 12], and modify the access control model second time with a Decentralized multi-authority scheme.
To decrease the computation cost of data consumer, we choose decryption outsourcing property in our scheme. In decryption outsourcing, the data consumer does not do all the computations for message decryption. The message is partially decrypted in some cloud component and sent to the data consumer. The data consumer does very little amount of computation to get the final decrypted message.
Again to penalize malicious users, we have used revocation property. Actually whenever we

detect that some user of the system is doing some malicious thing we revoke the necessary attributes from that user to prevent the malicious work.

1.5 Research Methodology

Literature Review : At first I have studied the recent access cotrol model and tried to find out its security limitations. After that I have gone through many existing literatures. First I have gone through the paper [1] to get an overview of the topic "Attribute Based Encryption" (ABE) and its variations. After that I have gone through the paper [2] from where I can understand the motivation behind ABE and how we can use the idea of Identity Based Encryption IBE to construct ABE. But there the access policy was attached with the secret key which is basically the idea of KP-ABE. Here they also have used tree structure for access policy. Then I have gone through the paper [3] where I get the clear idea about Ciphertext Policy ABE (CP-ABE) and here I have also introduced with LSSS access structure. Then I have gone through some more CP-ABE schemes from paper [7, 8]. After that I have gone through the papers [4, 5, 6] to fully understand the LSSS access structure. From paper [5] I also got very clear idea about the transformation of other access structures into LSSS. They actually presents a more efficient version of LSSS matrix generation. After that I have gone through the paper [9], where I get the motivation of Multi-Authority CP-ABE but they have used tree structure for their access policy. Actually among all the structures, LSSS is more efficient for access policy because it is easy to implement and all the other access structures can be converted into LSSS easily. After that I have gone through papers [10, 11, 12, 13]. All these papers are on Multi-Authority CP-ABE (MA-CP-ABE). But the schemes of papers [10, 12] are decentralized MA-CP-ABE whereas the schemes of the paper [11] is centralized MA-CP-ABE. The scheme of paper [13] does not have any central authority but here the attribute authorities communicates in between them. Though the schemes of the paper [11] is centralized MA-CP-ABE but we can make that central authority a semi-honest component and also can merge it with one of the attribute authorities. From here actually I get the idea of my first scheme. I get the idea of my second scheme from the papers [10, 12].

Requirement & Design : Then I have figured out how to use this MA-CP-ABE scheme to modify recent access control model and design two different schemes to modify the recent access control model. I have also modified those schemes to achive revocation property also. I have compared the schemes with many other existing schemes.

Implementation : I have implemented one of the schemes and have compared with some other schemes. After that I have implemented the LSSS matrix generation using paper [5]. Then I used it to modify my implementations.

1.6 Thesis Organization

In the next chapter, I shall discuss about ABE architecture and some preliminaries (chapter 2). After that I shall introduce my centralized model (chapter 3) and decentralized model (chapter 4) and also analyse their security. Then I shall compare the models to see the efficiency (chapter 5). After that I shall discuss about the implementation of the scheme (chapter 6). Next I shall further modify both the model to introduce revocation property (chapter 7). Then I shall give my idea about implementation of the full access control server (chapter 8).

Chapter 2

Background

2.1 Attribute Based Access Control

At first we have to know what is Logical Access control. The main motive of Logical Access control is to protect object (which is data, some executable applications, network devices, services owned by some individual or some organizations) from unauthorized operations like reading or creating or editing or deleting objects by unauthorized persons. Owners of the objects actually have the right to define policies to describe who can operate which operation on which object.If the subject satisfies the access control policy defined by the object owner, then the subject is authorized to perform the desired operation on that object. The Access Control Mechanism (ACM) is the logical component which receives the access request from the subject, then decides and enforces the access decision. Attribute Based Access Control (ABAC) is one of the Access Control Mechanisms. Based on NIST standard ABAC model [15], we get the formal definition of ABAC as follows :

Attribute Based Access Control (ABAC) : A logical access control methodology where authorization to perform a set of operations is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes.

Here, Attributes are characteristics that define specific aspects of the subject, object, environment conditions, and/or requested actions that are predefined and preassigned by an authority.

2.2 Attribute Based Encryption

2.2.1 Idea of ABE

When we store some data in cloud storage then it becomes accessable by anyone. So, it is very necessary to ensure that only authorized users can access the data. To ensure this we can store encrypted data in cloud storage, instead of the raw data. For the encryption we can use either the symmetric encryption technology or the traditional public-key encryption technology. In a symmetric encryption-based access control system, when a new data user (DU) enters into the system, the data owner (DO) has to share the secret key that acts as a shared key with the new DU also such that new DU can access the data. Similarly, in the traditional public-key encryption-based access control, the DO is required to encrypt his data again via the new DU's public key, so the DO has to encrypt a single data multiple time with different public keys of the DU's. And also in both these cases the DO has to have a detail knowledge about the DU's in advance.

So obviously these two access control mechanisms lack flexibility and scalability (flexibility and scalability refer to the expressiveness of access control policies and the impact of newly joined data users on the access control system, respectively). But luckily the Attribute-Based Encryption (ABE) technology plays a key role in realizing access control systems with fine granularity and scalability.

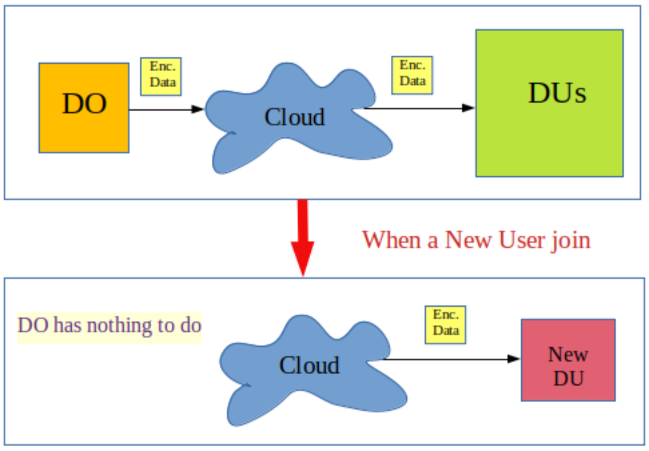


Figure 2.1: ABE based Access Control System

As shown in the above figure of access control system (based on ABE), the flexible attributes are embedded into the ciphertext and the DO does not need to know the identities of specific DUs before encryption. When a new DU joins the system, DOs have to do nothing. Therefore, both flexibility and scalability are enabled in the ABE-enabled access control system.

Sahai and Waters introduced the ABE notion for the first time and it is a promising cryptographic primitive and has successfully attracted considerable research efforts [1]. ABE has two categories, ciphertext-policy ABE (CP-ABE), and key-policy ABE (KP-ABE).

In **CP-ABE**, a user's attribute secret key is associated with an attribute list, and a ciphertext specifies an access policy that is defined over an attribute universe of the system. A ciphertext can be decrypted by a user if and only if the user's attribute list matches the ciphertext's access policy.
In **KP-ABE**, an access policy, which is defined over the system's attribute universe, is encoded into a user's attribute secret key and a ciphertext is created with respect to an attribute list. A ciphertext can be decrypted by a user if and only if the corresponding attribute list matches the access policy associated with the user's attribute secret key.

However, we have already mentioned that CP-ABE is much more appropiate than KP-ABE in our case because the access policy determination in CP-ABE is put on the data owner's hand.

The basic CP-ABE has four components, namely the cloud service provider (CSP), the attribute authority (AA), the DO, and the DU [1]. The main four algorithms which are used in basic CP-ABE are as follows :

1. **SetUp** $(\lambda) \rightarrow$ **(PK, MK)** : This algorithm is known as the system setup algorithm. The Attribute authority (AA) runs this algorithm at the begining. It takes security parameter λ as an input, and produce system public key PK and master key MK.

2. **KeyGeneration (PK, MK, L) → (SK$_L$)** : This algorithm is known as the attribute key generation algorithm, which is also performed by the AA. The AA takes system public key PK , master key MK, and an attribute list L as inputs and generates SK$_L$ as the attribute secret key corresponding to L. Here the attribute list is basically the set of attributes of an user and SK$_L$ is called the attribute secret key of that user.

3. **Encryption (PK, M, $\mathbb{A}$) → (CT$_\mathbb{A}$)** : This algorithm is performed by the data owner (DO). The DO first chooses an access policy $\mathbb{A}$ for the desired message M, and then takes PK, M, and $\mathbb{A}$ as inputs and produces a ciphertext CT$_\mathbb{A}$ of message M associated with the access policy $\mathbb{A}$. This ciphertext is stored on the cloud service provider CSP.

4. **Decryption (PK, CT$_\mathbb{A}$, SK$_L$) → (M or $\perp$)** : This algorithm is performed by the data user DU (DU is basically the data consumer). It takes system public key PK, a ciphertext CT$_\mathbb{A}$ of M associated with $\mathbb{A}$, and an attribute secret key SK$_L$ corresponding to DU's attribute list L, and returns M if L satisfies the access policy $\mathbb{A}$. Otherwise outputs the error symbol $\perp$ as an indication of failure of decryption.

2.2.2 Variations of CP-ABE

From paper [1] we can observe that according to the requirements of access control in different application scenarios, CP-ABE schemes are further divided into many categories. Each category has some specific property. In my model I will use CP-ABE with some properties. My CP-ABE will be multi-authority, revocable, hierarchial with decryption outsourcing. For future work, I also want to introduce accountability and policy updating property in it (I do not do this in this paper but left it as future work). Now I am going to describe these properties in brief.

1. **Revocable CP-ABE** : The functionality of revocation is realized in revocable CP-ABE. According to the graininess, revocation mechanisms fall into user revocation and attribute revocation. Actually when we can detect that any user is doing some malicious things, we just revoke some attributes of the user or all the attributes of the user to prevent that malicious work. However, according to the effect to non-revoked users, revocation mechanisms are divided into indirect revocation and direct revocation. In my case I have used direct revocation, i.e. whenever a revocation list is published, the authorities directly send the required update informations to the cloud.

2. **Accountable CP-ABE** : The functionality of accountability is realized in accountable CP-ABE. Both the user traceability and the attribute authority accountability are involved in accountable CP-ABE. Mainly by this property we can detect a malicious user. My model does not have this property but it will be very interesting future work to add this property also.

3. **CP-ABE with Policy Updating** : In basic CP-ABE, it is impossible to change a ciphertext's access policy. Considering access control in emergencies, CP-ABE with policy updating can be adopted to update the access policy in an involved ciphertext. Suppose the authorized person is not present to recive the message and he wants to transfer the authority of message reciving to someone he trusts. Using this property we basically can handle this senario. But this property is also left for future work.

4. **Multi-authority CP-ABE** : With this type of CP-ABE construction, distributed access privilege can be realized. According to whether a central authority exists or not, multi-authority CP-ABE schemes are divided into centralized multi-authority CP-ABE constructions and decentralized multi-authority CP-ABE constructions. In my case I have made two models, one is centralized and another is decentralized.

5. **Hierarchical CP-ABE** : As for hierarchical CP-ABE constructions, the delegation of access privilege is organized in a hierarchical manner. Suppose A is authorized to access some data and the attribute set of B is basically a super set of the attribute set of A, then B can also access that data. My model has this property as I have used monotone access structure.

6. **Outsourced CP-ABE** : To support data users (respectively, data owers and the authority) with constrained computation resources, outsourced CP-ABE is proposed to outsource laborious computation in decryption (respectively, encryption and key generation) to third-party servers. My model actually outsources the decryption.

2.3 Access Structure

In CP-ABE we encrypts the data under some Access Policy. This Access policy is actually an access structure and any consumer can access the data iff it satisfies the access structure. We mainly prefer monotone Access Structure for our model. Now I will give the formal definition.

Access Structure : We denote $P = \{P_1, P_2, ..., P_T\}$ as a set of parties. A collection $A \subseteq 2^{\{P_1, P_2, ..., P_T\}}$ is monotonic if $\forall A_1, A_2$: if $A_1 \in A$ and $A_1 \subseteq A_2$, then $A_2 \in A$. An (monotone) access structure is a (monotone) collection A of non-empty subsets of $P = \{P_1, P_2, ..., P_T\}$. That is, $A \subseteq 2^{\{P_1, P_2, ..., P_T\}} - \{0\}$. So, an access structure A is basically a non-empty set of subsets of P. If a subset B of P is in A, then we said that that subset B is an authorized set. But if B is not in A, then we said that subset B is unauthorized.

Here in our case these parties are actually the attributes of an user.
There are various types of Access Structure like tree, threshold, AND-OR, LSSS etc. But I have decided to use LSSS as my Access Structure because it is easy to compute with LSSS access structure.

2.3.1 Linear Secret Sharing Scheme (LSSS)

Linear Secret Sharing Scheme (LSSS) : Let P denote a set of parties, $s \in \mathbb{Z}_p$ be the shared secret. A secret sharing scheme Π over P is linear (over $\mathbb{Z}_p$) if it has the following properties :

1. The shares of s for each party form a vector over $\mathbb{Z}_p$.

2. There is a matrix $W \in \mathbb{Z}_p^{l \times n}$ which is called the share-generating matrix for Π. For all $i = 1, ..., l$, a function $\rho \in \mathcal{F}([l] \mapsto P)$ associates the row W_i with a party. To generate the shares, we choose a column vector $\vec{v} = (s, r_2, ..., r_n)^T$, where $r_2, ..., r_n$ are randomly picked from $\mathbb{Z}_p$, then $W \cdot \vec{v}$ is the vector of l shares of s according to Π. The share $(W \cdot \vec{v})_i$ belongs to the party $\rho(i)$.

Every linear secret sharing scheme mentioned before has the following linear reconstruction property : assume that $\mathbb{A}$ is an access structure. Π is an LSSS for $\mathbb{A}$. So, $\mathbb{A} = (W, \rho)$. Now assume S denotes an authorized set, i.e. S satisfies the access structure $\mathbb{A}$. Then let $I = \{i : \rho(i) \in S\}$ be the index set of rows whose labels are in S. There exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ such that: if the shares $\{\lambda_i = (W \cdot \vec{v})_i\}$ are valid, then we have $\Sigma_{i \in I} w_i \lambda_i = s$. But for unauthorized sets, no such constants exist.

This property is very important to decrypt the message.

Also we can transform any accees structure into threshold access structure and any threshold access structure can be converted into LSSS. Actually using a simple algorithm from paper [5, 6] we can do it efficiently.

Example : Here we will use Shamir's secret sharing scheme and the construction of the following theorem from paper [5] :

Theorem : Let A_1 and A_2 be monotone access structures defined on participant sets $\mathcal{P}_1$ and $\mathcal{P}_2$, realized by LSSS $(M^{(1)}, \rho^{(1)})$ of size m_1 and $(M^{(2)}, \rho^{(2)})$ of size m_2, respectively. Let $P_z \in \mathcal{P}_1$. There exists an LSSS (M, ρ) of size $m_1 + (m_2 - 1) \cdot q$ realizing the access structure $A_1(P_z \to A_2)$, where q is the number of rows labeled by P_z in $(M^{(1)}, \rho^{(1)})$.

Suppose the access policy is $((A \wedge B) \vee (c \wedge D)) \wedge E$. Let M is the access matrix and L is the vector whose co-ordinates are attributes. Initially we let $M = (1)$ and $L = (((A, B, 2), (C, D, 2), 1), E, 2)$.

Step - 1 : $M = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$ and $L = \begin{pmatrix} ((A, B, 2), (C, D, 2), 1) \\ E \end{pmatrix}$

Step - 2 : $M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$ and $L = \begin{pmatrix} (A, B, 2) \\ (C, D, 2) \\ E \end{pmatrix}$

Step - 3 : $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 2 & 0 \end{pmatrix}$ and $L = \begin{pmatrix} A \\ B \\ (C, D, 2) \\ E \end{pmatrix}$

Step - 4 : $M = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 2 \\ 1 & 2 & 0 & 0 \end{pmatrix}$ and $L = \begin{pmatrix} A \\ B \\ C \\ D \\ E \end{pmatrix}$

Here M is the generating matrix of LSSS correcponding to the given access policy and ρ maps i-th row of M to i-th co-ordinate of L.

Now let our secret is s and we take a vector $\vec{v} = (s, r1, r2, r3)$.

Now we just compute all the secret shares for the attributes A, B, C, D and E one by one.
Secret share for A (λ_A) is $(s + r1 + r2)$
Secret share for B (λ_B) is $(s + r1 + 2 \cdot r2)$
Secret share for C (λ_C) is $(s + r1 + r3)$
Secret share for D (λ_D) is $(s + r1 + 2 \cdot r3)$
Secret share for E (λ_E) is $(s + 2 \cdot r1)$

Now just consider a set of attributes $S = \{A, B, E\}$

At the time of decrtption we first compute the matrix just following the algorithm of the paper [5].

Step - 1 : $M^{'} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$ and $L^{'} = \begin{pmatrix} ((A,B,2),(C,D,2),1) \\ E \end{pmatrix}$

Step - 2 : $M^{'} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$ and $L^{'} = \begin{pmatrix} (A,B,2) \\ (C,D,2) \\ E \end{pmatrix}$

Step - 3 : $M^{'} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 2 & 0 \end{pmatrix}$ and $L^{'} = \begin{pmatrix} A \\ B \\ (C,D,2) \\ E \end{pmatrix}$

Step - 4 : $M^{'} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 0 \end{pmatrix}$ and $L^{'} = \begin{pmatrix} A \\ B \\ E \end{pmatrix}$

Now we will compute a vector $\vec{c} = (c1, c2, c3)$ such that $\vec{c} \cdot M^{'} = (1, 0, 0)$.
It can be easily computed that $\vec{c} = (4, -2, -1)$ serves our purpose. It also implies that S is an authorized set of attributes.

To get the secret just compute $(c1 \cdot \lambda_A + c2 \cdot \lambda_B + c3 \cdot \lambda_E)$.

$c1 \cdot \lambda_A + c2 \cdot \lambda_B + c3 \cdot \lambda_E$
$= 4 \cdot (s + r1 + r2) + (-2) \cdot (s + r1 + 2 \cdot r2) + (-1) \cdot (s + 2 \cdot r1)$
$= s \cdot (4 - 2 - 1) + r1 \cdot (4 - 2 - 2) + r2 \cdot (4 - 4)$
$= s$

So it is very easy to compute the secret using the linear reconstruction property of LSSS.

2.4 Bilinear Mapping

We will discuss bilinear mapping for both prime order bilinear group and composite order bilinear group here. We have used both of them in different models.

2.4.1 Bilinear Mapping in Prime order bilinear group

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of the same prime order p. Then a bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ such that $\forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2, a, b \in \mathbb{Z}_p$, $e(P^a, Q^b) = e(P, Q)^{ab}$.
The map is called a bilinear mapping if
i) $e(g_1, g_2)$ generates the group $\mathbb{G}_T$, where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively,
ii) e is efficiently computable.
The map we consider here is symmetric, with $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$, where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of the same prime order p.

2.4.2 Bilinear Mapping in Composite order bilinear group

Here the group generator $\mathcal{G}$ takes a security parameter λ as an input and produce the terms $(N = p_1p_2p_3, \mathbb{G}, \mathbb{G}_T, e)$, where p_1, p_2 and p_3 are different primes, $N = p_1 \cdot p_2 \cdot p_3$ is the order of cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$, and $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$ is a map with such properties :

i) $\forall p, q \in \mathbb{G}, a, b \in \mathbb{Z}_N$, $e(p^a, q^b) = e(p, q)^{ab}$.
ii) $\exists g \in \mathbb{G}$ such that $e(g, g)$ generates the group $\mathbb{G}_T$
iii) e is efficiently computable.

$\mathbb{G}_{p_i}$ denotes the subgroup of order p_i in $\mathbb{G}$. The subgroups $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ have the orthogonality property. That is, for $h_i \in \mathbb{G}_{p_i}$ and $h_j \in \mathbb{G}_{p_j}$, if $i \neq j$, we have $e(h_i, h_j) = 1$.

2.5 System Model

We will use the same components of recent access model except the KMS. But their functionalities will be different for two models. So, I will discuss about them in detail when I will explain the models (in chapter 3 and chapter 4).

2.6 Trust Model

First I am going to describe about the Honest component and the Semi-Honest component.

Honest Component : This type of components basically do only the works which are assigned to it. It computes nothing more than that and also not curious about any more informations.

Semi-Honest Component : This type of components basically do all the works which are assigned to it. It computes nothing more than that but it is curious about to know more. It may gather all the extra informations and save them.

In my case trust model is basically same with the recent access model with some extra restrictions. I will discuss about it also in detail when I will explain the models (in chapter 3 and chapter 4).

2.7 Security Requirements

Mainly we need collusion resistance property in our model but we also need some security assumptions and some more properties like :
i) All the communication channels are very safe and secure.
ii) Each user has a very secure memory to store their keys.
iii) External IAM and Policy Decision Point can not collude in between them.
iv) All the access policies must containts components of both the Authorities.

I will discuss more about it at the time of model description.

Chapter 3

Centralized Multi-Authority Model (First Model)

I have tried to fit my model with the recent structure's components but add some more responsibilities & functionalities to them. I am going to explain about it.

3.1 Modified Components

<u>**External IAM**</u> : In my model External IAM plays a big role. It serves as Central Authority (CA) and also one of the two Multi-Authorities besides its basic works (which it does in recent basic model).
In my model this component has to be **Semi-Honest** and it **can not collude with Policy Decision Point** (which is another Attribute Authority in my model).
It basically verify the attributes based on the global id of the users and gives a **token** corresponding to their id (basic model functionality) and gives **Central Authority Public & Secret Key** and one **Decryption Secret Key** with respect to the global id (work as CA) and also one **Authority Attribute Secret Key** based on the user's attributes which are handled by it (work as AA). Here it is very important to mention that **It does not store any information about Decryption Secret Key in its memory or Central Authority Public & Secret Key**, after giving the keys to the user, it removes those information from its memory.

<u>**Policy Decision Point**</u> : In my model Policy Decision Point serves also as another Attribute Authority besides its basic work.
This compoent must be **Semi-Honest** and have to handle some attributes of the Access Policies (since for Multi-Authority CPABE encryption we have only two authorities and if one of them only containts all attributes of the Access Policy then it will be no more secure).
This Policy Decision Point mainly set the Access Policy of Key Wrapping and supply one of the attribute secret key to user (as it is an AA).

<u>**Data Owner**</u> : He/she first encrypts the raw data using AES and then encrypts the AES encryption key with the two Attribute Public Key under his/her Access Policy $\mathbb{A}_1$.

<u>**Data Consumer**</u> : Data Consumer use his/her own **Decryption Secret Key** to Decrypt the Pre-Decryted data. After that he actually get the AES encryption key and the raw data encrypted with that key. He/she easily decryts the encrypted message and get the raw data.

<u>**Data Gateway**</u> : It does all its basic works on two-time encrypted data in stead of

the raw data. But besides these it also does pre-decryption of the user's message. As Basic Model, it is also **Semi-Honest** in my model.

Key Wrapping Server (KWS) : It wraps the encrypted key and also unwraps the wrapped key using MA-CP-ABE and it is also **Semi-Honest**. It is basically a replacement of KMS of recent system.

3.2 System Definition

Our Multi-Authority Access Control System consists of a Multi-Authority CP-ABE scheme with decryption outsourcing property which is taken from the paper [11]. This Multi-Authority CP-ABE scheme is a collection of the following 8 algorithms:

1. **GlobalSetup** $(\lambda) \rightarrow$ **(GPK)** : This algorithm takes the security parameter λ as an input and produces the global parameters GPK for the system.
2. **CASetup (GPK) $\rightarrow$ (CPK , CMK)** : The CA runs this algorithm with GPK as input to produce its public parameter CPK and the corresponding master secret key CMK. CPK will be used by AAs only and CMK will be used to generate user's secret key by CA.
3. **AASetup (GPK , f ,** $U_f) \rightarrow (APK_f, AMK_f)$: Each AA_f runs this algorithm with GPK and its attribute domain U_f as input to produce the public parameter APK_f and the corresponding master secret key AMK_f. For $i \neq j$, $U_i \cap U_j = \phi$.
4. **Encrypt (M , $\mathbb{A}$, GPK ,** $\cup APK_f) \rightarrow$ **(CT)** : This algorithm takes in GPK, a message M, an access structure $\mathbb{A}(W, \rho)$ and the set of public parameters of relevant AAs. It produces a ciphertext CT. We assume the access structure $\mathbb{A}$ is implicitly included in CT.
5. **CAKeyGen (GPK , gid)** $\rightarrow (DSK_{gid}, CASK_{gid}, CAPK_{gid})$: This algorithm takes in GPK and the user's gid. It then outputs a decryption key DSK_{gid} , a gid-related private key $CASK_gid$ and a gid-related public key $CAPK_{gid}$, where DSK_{gid} will be used by the user, $CASK_{gid}$ will be used in pre-decrypting the ciphertext and $CAPK_{gid}$ will be used to generate the attribute-related secret keys by the AAs.
6. **AAKeyGen** $(S_{gid,f}, GPK, CPK, CAPK_{gid}, AMK_f)$ß$(ASK_{S,gid,f})$: When a user submits a set of attributes $S_{gid,f}$ belongs to AA_f to request the attribute-related key $ASK_{gid,f}$, AA_f runs this algorithm with $S_{gid,f}$, GPK, CPK, $CAPK_{gid}$ and AMK_f as input. If $CAPK_{gid}$ is invalid, it outputs $\perp$. Otherwise, it outputs $ASK_{S,gid,f} = \{ASK_{i,gid} | i \in S_{gid,f}\}$. We let $ASK_{S,gid} = \cup ASK_{S,gid,f}$ denote the attribute-related key of S_{gid} , where $S_{gid} = \cup S_{gid,f}$. We assume the set S_{gid} is implicitly included in $ASK_{S,gid}$.
7. **Pre-Decrypt (CT , GPK ,** $CASK_{gid}, ASKS, gid) \rightarrow$ **(PDKEY)** : This algorithm takes in CT, GPK, $CASK_{gid}$ and $ASK_{S,gid}$. It outputs the pre-decryption key PDKEY of CT if and only if S_{gid} satisfies A .
8. **Decrypt (CT , PDKEY ,** $DSK_{gid}) \rightarrow$ **(M)** : This algorithm takes in CT, PDKEY and DSK_{gid} . It outputs the plaintext message M.

We use this Multi-Authority CP-ABE scheme as a building block in our new modified model. Table-3.1 lists the notations that are used in the system definition.

Table : Notations	
Notations	Descriptions
M, RD	M is the Plaintext (actually the raw data of Data Owner). RD is the Data send to the Data Gateway by Data Owner. In recent basic model RD = M
RK	Randomly choosen encrytion key by Data Gateway
ED	Encrypted data of Data Gateway
EK	Encrypted encryption key
WK	Wrapped key EK
LSSS	Linear Secret Sharing Scheme
CA	Central Authority
gid	Global identifier
AA_f	Attribute authority with index f
GPK	System global public parameters
CPK	The CA's public parameters
CMK	The CA's master secret key
U_f	The attribute universe governed by AA_f
APK_f	Public parameters of AA_f
AMK_f	Master secret key of AA_f
CT	Ciphertext (the encrypted data which the data owner send to data gateway in modified model. it is basically the pair (encrypted data private key encryption, encrypted private key))
$\mathbb{A}(W, \rho)$	Access structure (policy) expressed by LSSS matrix W and map function ρ
DSK_{gid}	User decryption key for gid
$CAPK_{gid}$	gid-related public key
$CASK_{gid}$	gid-related private key
$S_{gid,f}$	The set of attributes of user (gid) governed by AA_f
$ASK_{S,gid,f}$	The private key for $S_{gid,f}$
PDKEY	Pre-decryption key
λ	The security parameter of the system
N	Product of 3 different primes p_1, p_2, p_3
$\mathbb{Z}_N$	The ring of integers modulo N (without 0)

Table 3.1: List of Notations used in System Definition

3.3 Fuctionalities used in the System

Global Setup : Let $\mathbb{G}$ and $\mathbb{G}_1$ denote two bilinear groups of order $N = p_1p_2p_3$ (a product of 3 different primes). Let $\mathbb{G}_{p_i}$ be the subgroup of order p_i in $\mathbb{G}$. Here the subgroups $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ have to have the orthogonality property. That is, for $h_i \in \mathbb{G}_{p_i} and h_j \in \mathbb{G}_{p_j}$, if $i \neq j$, we have $e(h_i, h_j) = 1$. Let $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ denote a bilinear map. g is a random chosen element from $\mathbb{G}_{p_1}$. Additionally, choose an UF-CMA (unforgeable under adaptive chosen message attacks) secure signature system $\Sigma_{sign} = (KeyGen, Sign, Verify)$. The GPK is broadcasted as GPK = (N, e, g, Σ_{sign}).

Central Authority Setup : The CA runs the $KenGen$ algorithm of Σ_{sign} . It sets the sign-key as CMK (Central Master Key) and verify-key as CPK (Central Public Key). The

CPK will be used by the AAs only. The CMK is used by only CA to generate Keys (only for one time) for Users.

Attribute Authorities Setup : Each AA_f governs its attribute universe U_f. For each $i \in U_f$, it chooses a random exponent $t_{f,i} \in \mathbb{Z}_N$ and computes $T_{f,i} = g^{t_{f,i}}$. It also chooses two random exponents $\alpha_f, a_f \in \mathbb{Z}_N$. Finally, the public parameter of AA_f is broadcasted as : $APK_f = (g^{a_f}, e(g,g)^{\alpha_f}, T_{f,i}\ \forall i)$. The master secret key of AA_f is $AMK_f = (\alpha_f, a_f, t_{f,i}\ \forall i)$.

Encrypt : Here the access policy is defined by $\mathbb{A} = (W, \rho)$, where W is a LSSS matrix with l rows and n columns, and ρ associates each row W_x to attribute $\rho(x)$.
Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Data Owner's encryptor uses those public keys and encrypts M under $\mathbb{A}$ as follows :

1. Chooses a random vector $\vec{v} = (s, v_2, ..., v_n) \in \mathbb{Z}_N^n$, where s is the secret value.
2. For each $x \in [l]$, it selects a random exponent $r_x \in \mathbb{Z}_N$.
3. Computes $C_1 = M \cdot (\Pi_{f=1}^{2} e(g,g)^{\alpha_f})^s$, $C_{01} = g^s$. For each $x \in [l]$ it also computes $C_{x1} = g^{(\Sigma_{f=1}^{2} a_f)\cdot W_x \cdot \vec{v}} \cdot T_{\rho(x)}^{-r_x}$, $D_{x1} = g^{r_x}$.
4. Set $CT = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]})$.

When a new user joins in the system, he/she has to register himself/herself and will obtain a unique gid. By running the CAKeyGen algorithm, the CA provides the gid-related keys to the users. Then, each AA runs the AAKeyGen algorithm and gives the attribute-related keys to the users.

CAKeyGen : For each user, the CA first chooses two random exponents $b_{gid}, c_{gid} \in \mathbb{Z}_N$, two random elements $R_{gid}, R_{gid,0} \in \mathbb{G}_{p_3}$ and computes $CASK_{gid} = L_{gid} = g^{b_{gid}/c_{gid}} \cdot R_{gid}$, $L_{gid,0} = g^{1/c_{gid}} \cdot R_{gid,0}$. After that, it uses CMK to sign on the string $(CMK, gid||CASK_{gid}||L_{gid,0})$ and gets a signature σ_{gid}. Let $CAPK_{gid} = (gid, CASK_{gid}, L_{gid,0}, \sigma_{gid})$. Finally, it sends the $DSK_{gid} = c_{gid}, CASK_{gid}$ and $CAPK_{gid}$ to the user. After sending the keys to user CA delete $b_{gid}, c_{gid}, R_{gid}R_{gid,0}$ from its memory.

AAKeyGen : After receiving the submitted key $CAPK_{gid}$, the AA_f first uses the CPK to verify whether the $CAPK_{gid}$ is valid. If not, it aborts. Otherwise, it issues the user a set of attributes $S_{gid,f}$. It randomly selects $R_{gid,f,0} \in \mathbb{G}_{p_3}$ and computes $K_{gid,f} = L_{gid,0}^{\alpha_f} \cdot L_{gid}^{a_f} \cdot R_{gid,f,0} = g^{\alpha_f/c_{gid}} \cdot g^{a_f \cdot b_{gid}/c_{gid}} \cdot R_{gid,f}$, where $R_{gid,f} = R_{gid,f,0} \cdot R_{gid,0}^{\alpha_f} \cdot R_{gid}^{a_f}$. For each attribute $i \in S_{gid,f}$, it randomly picks $R'_{gid,f,i} \in \mathbb{G}_{p_3}$ and computes $K_{gid,f,i} = L_{gid}^{t_{f,i}} \cdot R'_{gid,f,i} = T_{f,i}^{b_{gid}/c_{gid}} \cdot R_{gid,f,i}$, where $R_{gid,f,i} = R_{gid}^{t_{f,i}} \cdot R'_{gid,f,i}$. It finally sends $ASK_{S,gid,f} = (K_{gid,f}, \{K_{gid,f,i}\}_{i\in S_{gid,f}})$.

PreDecrypt : The Data consumer sends $ASK_{S,gid,f} = (K_{gid,f}, \{K_{gid,f,i}\}_{i\in S_{gid,f}})$ and $CASK_{gid} = L_{gid}$ to the cloud server and asks it to pre-decrypt the CT. Policy Decision Point computes $K2 = \Pi_{f\in 1,2} K_{gid,f}$ and constants $y_x \in \mathbb{Z}_N$, such that $\Sigma_{\rho(x)\in S_{gid}}(y_x \cdot W_x) = (1, 0, ..., 0)$. Then computes

PDKEY $= \frac{e(K2, C_{01})}{\Pi_{\rho(x)\in S_{gid}}(e(C_{x1}, L_{gid})\cdot e(D_{x1}, K_{\rho(x)}))^{y_x}} = e(g,g)^{\Sigma_{f=1}^{2}\alpha_f \cdot s/c_{gid}}$.

Cloud server sends PDKEY to Data consumer.

Decrypt : Data Consumer's decryptor computes $M = \frac{C_1}{(PDKEY)^{c_{gid}}}$.

We now want to change Key wrapping and unwrapping procedure of the recent model, we will use MA-CPABE here also. In this case we replace the component KMS (Key Management Server) of basic model by KWS (Key Wrapping Server).

3.4 Data Storing

In this section I am presenting data storing Architecture of my model with step by step explanation.

3.4.1 Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point

1. Data Owner sends his/her Global ID (gid) to the External IAM.
2. External IAM sends a token to Data Owner after verifying all of its attributes.
3. Data Owner choose the message M and then using AES encryption it encrypts M and get $enc(M)$. The AES encryption key (AK) and the choosen access policy $\mathbb{A} = (W, \rho)$, where W is a LSSS matrix with l rows and n columns, and ρ associates each row W_x to attribute $\rho(x)$, to the encryptor. In my model I forcefully assume that **Data Owner's Access Policy must include attributes of both Attribute Authorities**.
4. Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Encryptor uses those public keys and encrypts AK under $\mathbb{A}_1$ using **Encrypt** algorithm and produce CT_0.
5. Encryptor Sends CT_0 to the Data Owner.
6. Data Owner sends $CT = (enc(M), CT_0) = (enc(M), \mathbb{A}_1, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]})$ to the Data Gateway along with own token.
7. Data Gateway take $RD = CT$. Now it does following things (as basic model) on RD :
   (a) Chooses a random key RK and encrypts RD with RK and gets ED.
   (b) Then also encrypts RK with its some special secret key and get EK.
8. Data Gateway sends EK with Data Owners token to the Control Interface.
9. Control Interface sends EK to the KWS and token to the Policy Decision Point.
10. Policy Decision Point first retrieve data owner's identity from the token and decides an access policy $\mathbb{A}_2 = (A', \rho')$ (where A' has l' many rows). It sends $\mathbb{A}_2 = (A', \rho')$ to the KWS.
11. KWS runs the **Encrypt** algorithm on EK under the access policy $\mathbb{A}_2 = (A', \rho')$ and get the wrapped key (doubly encrypted encryption key) $WK = (\mathbb{A}', C_2, C_{02}, \{C_{x2}, D_{x2}\}_{x\in[l']})$.

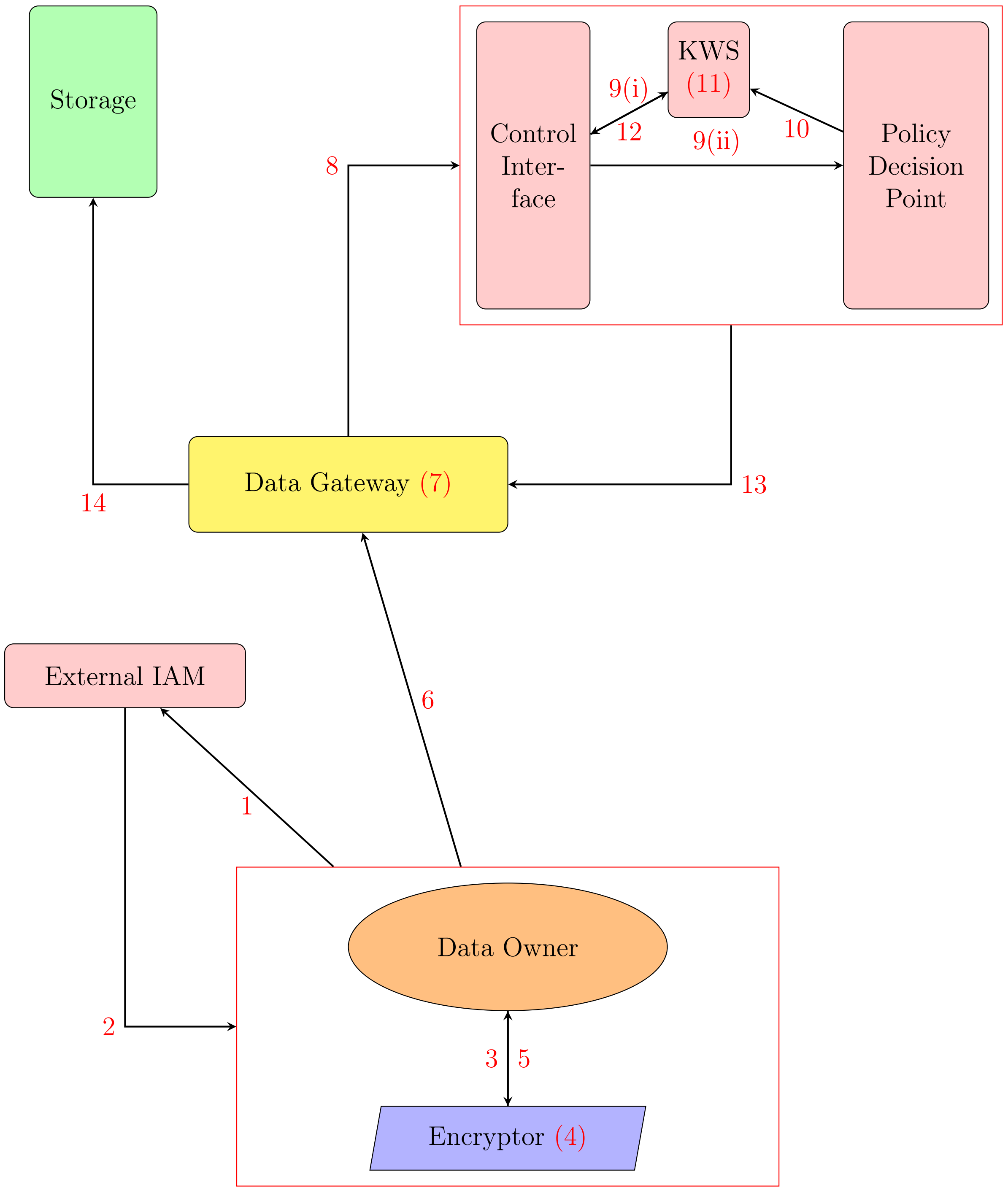


Figure 3.1: Rough Architecture of Data storing in first (Centralized) model

12. KWS sends WK to Control Interface.
13. Control Interface sends WK to the Data Gateway.
14. Data Gateway sends ED and WK to the Storage.

3.5 Data Consuming

In this section I am presenting data consuming Architecture of my model with step by step explanation.

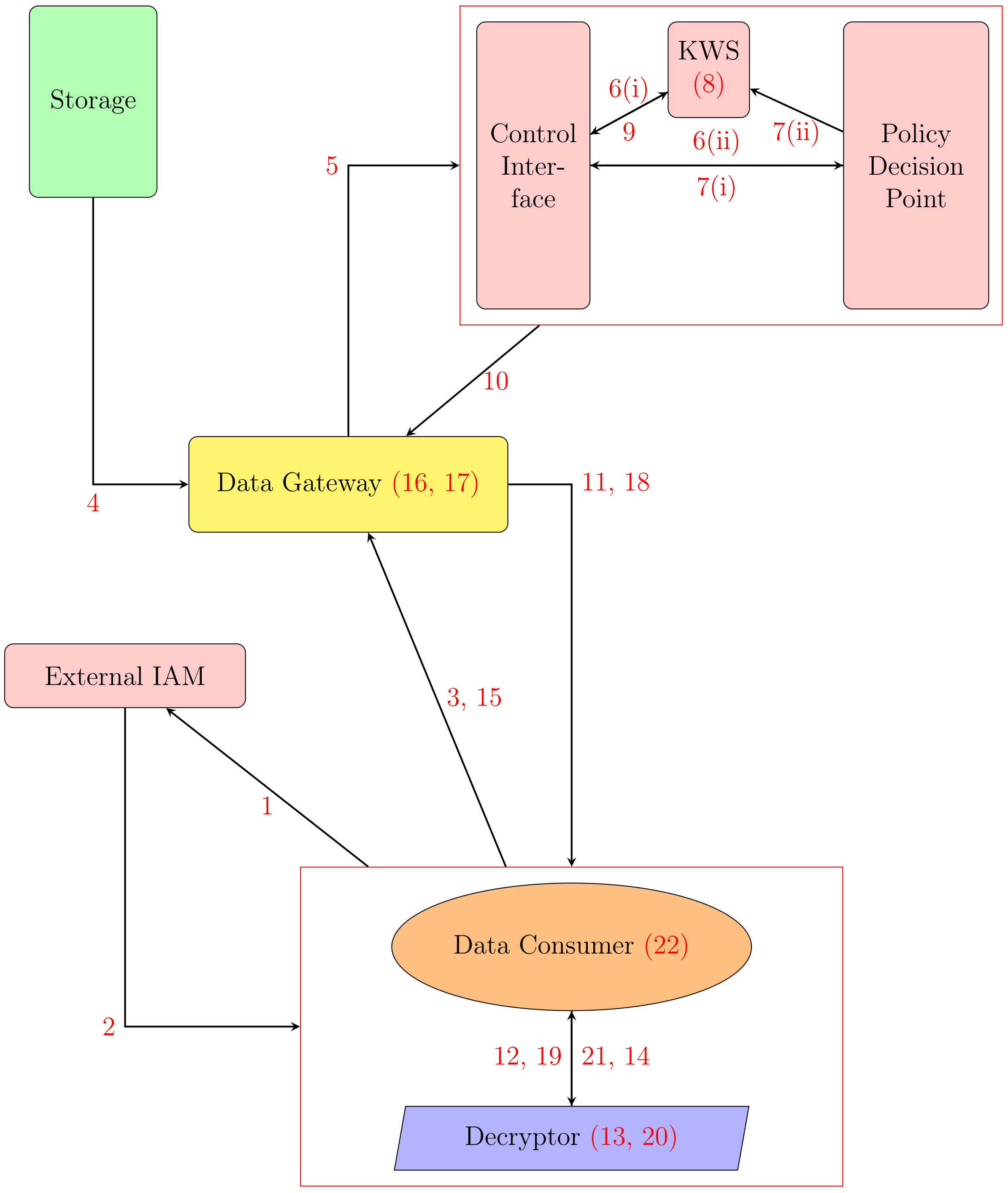


Figure 3.2: Rough Architecture of Data consuming in first (Centralized) model

3.5.1 Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point.

1. Data Consumer sends Global ID (gid) to the External IAM.
2. External IAM sends token, $CAPK_{gid}, CASK_{gid}, DSK_{gid}, ASK_{S,gid,1} = (K_{gid,1}, \{K_{gid,1,i}\}_{i \in S_{gid,1}})$ to the Data Consumer.
3. Data Consumer sends $ASK_{S,gid,1} = (K_{gid,1}, \{K_{gid,1,i}\}_{i \in S_{gid,1}})$, token, $CASK_{gid}$ and $CAPK_{gid}$ to the Data Gateway.

4. Data Gateway collects ED and WK from the Storage, where $WK = (\mathbb{A}^{'}, C_2, C_{02}, \{C_{x2}, D_{x2}\}_{x\in[l']})$.

5. Data Gateway sends WK, $CAPK_{gid}, CASK_{gid}, ASK_{S,gid,1}$ and consumer's token to Control Interface of Access Control Server.

6. Control Interface sends WK, $ASK_{S,gid,1}$ to KWS and sends $CAPK_{gid}$, consumer's token to Policy Decision Point.

7 Policy Decision Point runs **AAKeyGen** and sends $ASK_{S,gid,2}$ to control interface and to KWS both.

8 KWS runs **PreDecrypt** on WK with $ASK_{S,gid,1}, ASK_{S,gid,2}, CASK_{gid}$ and computes $EK^{'}$ = PreDecryption key (PDKEY) for decrypting WK.

9 KWS sends $EK^{'}$ to Control Interface.

10 Control Interface sends $EK^{'}, ASK_{S,gid,2}$ to the Data Gateway.

11 Data Gateway sends $EK^{'}, C_2$ to the Data Consumer.

12 Data Consumer sends $EK^{'}, C_2, DSK_{gid}$ to its Decryptor.

13 Decrytor runs **Decrypt** on $EK^{'}, C_2, DSK_{gid}$ (with PDKEY $= EK^{'}$) and get EK.

14 Decryptor sends EK to Data Consumer.

15 Data Consumer sends EK to Data Gateway.

16 Data Gateway decrypts EK and gets RK, and after that using that RK decrypts ED and gets $RD = CT = (enc(M), \mathbb{A}_1, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]}) = (enc(M), CT_0)$.

17 Data Gateway runs **PreDecrypt** with CT_0, $ASK_{S,gid,1}$, $ASK_{S,gid,2}$ and $CASK_{gid}$ as input and produce $CT^{'}$ = PreDecryption key (PDKEY) for decrypting CT_0.

18 Data Gateway sends $enc(M), CT^{'}$ and C_1 to Data Consumer.

19 Data consumer sends $CT^{'}$, C_1 and $DSK_{gid} = c_{gid}$ to its Decryptor.

20 Decryptor runs **Decrypt** algorithm on $CT^{'}$, C_1 and $DSK_{gid} = c_{gid}$ (with PDKEY $= CT^{'}$) as input and retrieve the AES encryption key AK.

21 Decryptor sends AK to Data Consumer.

22 Data Consumer decrypts $enc(M)$ using AK and gets M.

3.6 Security Analysis

The security proof which is used in the paper [11] implies that our model is also adaptively secure. I have given the proof in Appendix-A.

Our Scheme is also collusion resistant because $L_{gid}, L_{gid,0}$ is different for each users, so they can't combine their Authority Attribute Secret keys to access an unauthorized data.

Suppose Alice or Bob does not have an authorized set of attributes but union of their attribute sets is an authorized set and they are planning to collude.

Let us consider all the keys of both Alice and Bob.

Alice's Key :

$CASK_{gid} = L_{gid} = g^{b_1/c_1} \cdot R_1$
$L_{gid,0} = g^{1/c_1} \cdot R_{1,0}$
$DSK_{gid} = c_1$
$K_{gid,f} = g^{\alpha_f/c_1} \cdot g^{a_f \cdot b_1/c_1} \cdot R_{1,f,0} \cdot R_{1,0}^{\alpha_f} \cdot R_1^{a_f}.$
$K_{gid,f,i} = T_{f,i}^{b_1/c_1} \cdot R_1^{t_{f,i}} \cdot R'_{1,f,i}, \forall i \in U_f$

Bob's Key :

$CASK_{gid} = L_{gid} = g^{b_2/c_2} \cdot R_2$
$L_{gid,0} = g^{1/c_2} \cdot R_{2,0}$
$DSK_{gid} = c_2$
$K_{gid,f} = g^{\alpha_f/c_2} \cdot g^{a_f \cdot b_2/c_2} \cdot R_{2,f,0} \cdot R_{2,0}^{\alpha_f} \cdot R_2^{a_f}.$
$K_{gid,f,i} = T_{f,i}^{b_2/c_2} \cdot R_2^{t_{f,i}} \cdot R'_{2,f,i}, \forall i \in U_f$

Suppose after mixing up their keys there is two attributes $i, j \in U_f$ where i is only Alice's attribute and j is only Bob's attribute. In that case they will produce $CASK_{gid}, L_{gid,0}, K_{gid,f}$ of only one of them but produce $K_{gid,f,i}$'s from both of them.

Let Alice gives all its keys with $K_{gid,f,j}$ of Bob and when we add attribute j to Alice's attribute set it becomes an authorized set.

But then if we calculate the pre-decryption key PDKEY, it will be PDKEY $= (e(g,g)^{\Sigma_{f=1}^{2} \alpha_f \cdot s/c_1}) / (e(g,g)^{r_j t_{f,j}(\frac{b_2}{c_2} - \frac{b_1}{c_1})})$ instead of PDKEY $= e(g,g)^{\Sigma_{f=1}^{2} \alpha_f \cdot s/c_1}$.
Then Alice will decrypt the message using its secret key c_1.
So, at the end of decryption we will get $M \cdot e(g,g)^{r_j t_{f,j}(\frac{b_2 c_1}{c_2} - b_1)}$ instead of the message M.

Similarly for the other case also we will get a wrong result. Therefore two unauthorized user can't get any facility after a successful collusion. Hence Our scheme is collusion resistant.

Chapter 4

Decentralized Multi-Authority Model (Second Model)

Firstly the MA-CP-ABE, which we have used in our first model is actually based on the composite order bilinear groups. Now if we can change it with a MA-CP-ABE based on prime order bilinear group, then we can achive more efficiency [14]. For this I take the idea of the paper [10, 12], and construct another modified model (using decentralized MA-CP-ABE).
As my first model here we also use KWS instead of KMS. For key-wrapping and unwrapping in KWS we use another CP-ABE without decryption outsourcing. Actually here the full computation for key-unwrapping will be in cloud (KWS does the computations and it is a cloud component). So we do not need a CP-ABE with decryption outsourcing for key-wrapping and unwrapping.

Here I will omit the concept of Central Authority.

4.1 Modified Components and Their Uses

In modified version I have changed the responses of some components. So I am explaining all the components functionalities once again.

External IAM : In this model External IAM serves as one of the two Multi-Authorities besides its basic works.
This component has to be **Semi-Honest** and it **can not collude with Policy Decision Point** (which is another Attribute Authority in my model).
It basically verify the attributes based on the global id of the users and gives a **token** corresponding to their id (basic model functionality) and one **Authority Attribute Secret Key** based on the user's attributes which are handled by it (work as AA).

Policy Decision Point : In this model Policy Decision Point serves also as another Attribute Authority besides its basic work.
This compoent must be **Semi-Honest** and have to handle some attributes of its Access Policy (since for Multi-Authority CP-ABE encryption we have only two authorities and if one of them only containts all attributes of the Access Policy then it will be no more secure).
It basically produce one another access policy $\mathbb{A}_2$ for key wrapping and computes the authority attribute secret key for Data Consumer and all the **Pre-Decryption** operations in cloud.

Data Owner : He/she first encrypts the raw data using AES and then encrypts the AES encryption key with the two Attribute Public Key under his/her Access Policy $\mathbb{A}_1$.

Data Consumer : Data Consumer use his/her own **Decryption Secret Key** to Decrypt the Pre-Decryted data. After that he actually get the AES encryption key and the raw data encrypted with that key. He/she easily decryts the encrypted message and get the raw data.

Data Gateway : It does all its basic works on encrypted data in stead of the raw data and it unwraps the wrapped key. But besides these it also does pre-decryption of the user's message. As Basic Model, it is also **Semi-Honest** in my model.

Key Wrapping Server (KWS) : It wraps the encrypted key and also unwraps the wrapped key using MA-CP-ABE and it is also **Semi-Honest**.

4.2 Full System Definition

Following functions are basically combination of two different MA-CP-ABE (one with decryption outsourcing and another is without decryption outsourcing).

Global Setup $(\lambda) \rightarrow$ **GP** : This algorithm takes the security parameter λ as an input and outputs the global parameters GP for the system.

Authority Setup (f, GP)$\rightarrow$ $(PK_f, PK_f^{'}, MSK_f, MSK_f^{'})$: Authority AA_f (which is indexed by f) takes GP as input and runs the algorithm to generate public parameters $(PK_f, PK_f^{'})$ and secret parameters $(MSK_f, MSK_f^{'})$.
In my case I have two authorities, so $f \in \{1, 2\}$. AA_1 = External IAM and AA_2 = Policy Decision Point.

Encrypt (M, $\mathbb{A}_1$, GP, $\{PK_1, PK_2\}$) $\rightarrow$ CT : Data owner's Encryptor takes message M, data owner's access policy $\mathbb{A}_1$, GP, $\{PK_1, PK_2\}$ as input and compute the encrypted data CT.

ReEncrypt (CT) $\rightarrow$ (ED) : Data Gateway runs this algorithm with encrypted data CT as input. Then it chooses a random encryption key RK to encrypt CT and to produce an encrypted data ED.

KeyEncryption (RK) $\rightarrow$ (EK) : Data Gateway runs this algorithm with encryption key RK as input to produce encrypted form of the key EK.

Wrap (EK, $\mathbb{A}_2$, GP, $\{PK_1^{'}, PK_2^{'}\}$) $\rightarrow$ WK : KWS takes encrypted encryption key EK, policy decision point's access policy $\mathbb{A}_2$, GP, $\{PK_1^{'}, PK_2^{'}\}$ as input and compute the wrapped key WK.

AAKeyGen1 (GP, GID, f, $S_{GID,f}, MSK_f$) $\rightarrow$ $K_{f,GID}$: Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter MSK_f as input and computes secret attribute key $K_{f,GID}$ which is used for message decryption.

AAKeyGen2 (GP, GID, f, $S_{GID,f}, MSK_f^{'}$) $\rightarrow$ $K_{f,GID}^{'}$: Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter $MSK_f^{'}$ as input and computes secret attribute key $K_{f,GID}^{'}$ which is used for key unwrapping by Data Gateway.

KeyTransform (GID, $K_{f,GID}$) $\rightarrow$ ($T_{f,GID}$, DSK) : Data consumer runs this algorithm to transfer attribute secret keys $K_{f,GID}$ to $T_{f,GID}$ using its own decryption secret key DSK.

UnWrap (GP, WK, $\{K^{'}_{1,GID}, K^{'}_{2,GID}\}$) $\rightarrow$ EK : This Algorithm takes global parameter GP, wrapped key WK, secret attribute keys $\{K^{'}_{1,GID}, K^{'}_{2,GID}\}$ as input and compute unwrapped key (actual encrypted encryption key) EK.

KeyDecryption (EK) $\rightarrow$ (RK) : Data Gateway runs this algorithm with unwrapped encrypted key EK as input to produce decrypted form of the key RK.

PartialDecryption (ED, RK) $\rightarrow$ (CT) : Data Gateway runs this algorithm with encrypted data ED and encryption key RK as input to produce the partially decrypted data CT.

PreDecrypt (GP, CT, $\{T_{1,GID}, T_{2,GID}\}$) $\rightarrow CT^{'}$: This algorithm takes GP, CT, transformed keys $\{T_{1,GID}, T_{2,GID}\}$ as input and compute pre-decrypted ciphertext $CT^{'}$.

Decrypt (DSK, $CT^{'}$) $\rightarrow$ M : Data Consumer takes its own decryption secret key DSK, pre-decrypted cipher text $CT^{'}$ as input and runs this algorithm to compute actual message M.

4.3 Fuctionalities used in the System

Global Setup : Let $\mathbb{G}$ and $\mathbb{G}_1$ denote two bilinear groups of prime order p. Let $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ denote a bilinear map. Choose a generator g of the group $\mathbb{G}$. Also computes $e(g, g)$. After that choose a collision resistant hash function $H : \{GID\} \mapsto \mathbb{G}$. Set global parameter GP $= \{g, p, e, H, e(g, g)\}$.

Authority Setup : AA_f chooses $\alpha_i, y_i, \beta_i, z_i \in \mathbb{Z}_p \ \forall i \in U_f$ (U_f is the set of attributes handled by AA_f). Then it computes $e(g, g)^{\alpha_i}, g^{y_i}, e(g, g)^{\beta_i}, g^{z_i} \ \forall i \in U_f$. It sets $PK_f = \{(e(g, g)^{\alpha_i}, g^{y_i})\}_{i \in U_f}$, $PK^{'}_f = \{e(g, g)^{\beta_i}, g^{z_i}\}_{i \in U_f}$ as public keys and sets $MSK_f = \{\alpha_i, y_i\}_{i \in U_f}$, $MSK^{'}_f = \{\beta_i, z_i\}_{i \in U_f}$ as secret keys.

Encrypt : Data Owner sends its data m and its own policy $\mathbb{A}_1 = (A, \rho)$, where A is a LSSS matrix with l rows and n columns, and ρ associates each row A_x to attribute $\rho(x)$, as input to its encryptor. After that it chooses some secret $s \in \mathbb{Z}_p$. Select two random vectors $\vec{v}, \vec{w} \in \mathbb{Z}_p^n$ such that 1st co-ordinate of $\vec{v}$ is s and 1st co-ordinate of $\vec{w}$ is 0. Now computes $\lambda_j = A_j \cdot \vec{v}$ and $w_j = A_j \cdot \vec{w}$ for all rows $j \in [l]$. Now it chooses $r_j \in \mathbb{Z}_p \ \forall j \in [l]$. Then it computes $C_0 = m \cdot e(g, g)^s$ and $\forall j \in [l]$ also computes $C_{1j} = e(g, g)^{\lambda_j} \cdot e(g, g)^{\alpha_{\rho(j)} r_j}$, $C_{2j} = g^{r_j}$, $C_{3j} = g^{y_j r_j} \cdot g^{w_j}$. Finally sets $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$.

ReEncrypt : It is same as basic model.

KeyEncryption : It is same as basic model.

Wrap : KWS takes encrypted encryption key EK from Data Gateway and Policy Decision Point's access policy $\mathbb{A}_2 = (A^{'}, \rho^{'})$, where $A^{'}$ is a LSSS matrix with $l^{'}$ rows and $n^{'}$ columns, and $\rho^{'}$ associates each row $A^{'}_x$ to attribute $\rho^{'}(x)$, as input. After that it chooses some secret $s^{'} \in \mathbb{Z}_p$. Select two random vectors $\vec{v^{'}}, \vec{w^{'}} \in \mathbb{Z}_p^{n^{'}}$ such that 1st co-ordinate of $\vec{v^{'}}$

is $s^{'}$ and 1st co-ordinate of $\vec{w'}$ is 0. Now computes $\lambda_j^{'} = A_j^{'} \cdot \vec{v'}$ and $w_j^{'} = A_j^{'} \cdot \vec{w'}$ for all rows $j \in [l^{'}]$. Now it chooses $r_j^{'} \in \mathbb{Z}_p \; \forall j \in [l^{'}]$. Then it computes $D_0 = EK \cdot e(g,g)^{s^{'}}$ and $\forall j \in [l^{'}]$ also computes $D_{1j} = e(g,g)^{\lambda_j^{'}} \cdot e(g,g)^{\beta^{'}_{\rho^{'}(j)} r_j^{'}}$, $D_{2j} = g^{r_j^{'}}$, $D_{3j} = g^{z_j r_j^{'}} \cdot g^{w_j^{'}}$. Finally sets $WK = (D_0, \{D_{1j}, D_{2j}, D_{3j}\}_{j \in [l^{'}]})$.

It is basically **the Encrypt Algorithm** with input EK and $\mathbb{A}_2$ with public parameters $PK_1^{'}, PK_2^{'}$. (Since it has used different public parameters, we just separate them by different names)

AAKeyGen1 : Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter MSK_f as input. Then it computes $K_{f,GID} = \{K_{i,GID}\}_{i \in S_{GID,f}} = \{g^{\alpha_i} \cdot H(GID)^{y_i}\}_{i \in S_{GID,f}}$. This the secret attribute key, which is used for Message decryption

AAKeyGen2 : Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter $MSK_f^{'}$ as input. Then it computes $K_{f,GID}^{'} = \{K_{i,GID}^{'}\}_{i \in S_{GID,f}} = \{g^{\beta_i} \cdot H(GID)^{z_i}\}_{i \in S_{GID,f}}$. This secret attribute key is used for Key Unwrapping

KeyTransform : Data Consumer takes the secret attribute keys $\{K_{i,GID}\}_{i \in S_{GID}}$ as input. After that it chooses $d \in \mathbb{Z}_p$, and set its own secret decryption key $DSK = d$. After that it trsforms the attribute keys as $\{T_{i,GID}\}_{i \in S_{GID}} = \{(K_{i,GID}^{\frac{1}{d}}, H(GID)^{\frac{1}{d}}\}_{i \in S_{GID}}$.

UnWrap : KWS takes wrapped key $WK = (D_0, \{D_{1j}, D_{2j}, D_{3j}\}_{j \in [l^{'}]})$ and attribute secret keys $K_{1,GID}^{'}, K_{2,GID}^{'}$ as input. Then computes constants $c_x^{'} \in \mathbb{Z}_p$, such that $\Sigma_{\rho^{'}(x) \in S_{gid}}(c_x^{'} \cdot A^{'}{}_x) = (1,0,...,0)$. Now it computes $C^{'} = \Pi_{j=1}^{l^{'}}((e(H(GID), D_{3j}) \cdot D_{1j})/(e(K^{'}_{\rho^{'}(j),GID}, D_{2j})))^{c_j^{'}}$. Finally computes $EK = \frac{D_0}{C^{'}}$. Then sends EK to Data Gateway.

KeyDecryption : It is same as basic model.

PartialDecryption : It is same as basic model.

PreDecrypt : Data Gateway takes cipher text $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$ and transformed secret keys $T_{1,GID}, T_{2,GID}$ as input. Then computes constants $c_x \in \mathbb{Z}_p$, such that $\Sigma_{\rho(x) \in S_{gid}}(c_x \cdot A_x) = (1,0,...,0)$. Here $T_{i,GID} = (K_{i,GID}^{\frac{1}{d}}, H(GID)^{\frac{1}{d}})$, $\forall i \in S_{GID}$. If 2nd component of each $T_{i,GID}$ is not same then it will return $\perp$.
Otherwise it computes $C_1 = \Pi_{j=1}^{l}((e(H(GID)^{\frac{1}{d}}, D_{3j}))/(e(T_{\rho(j),GID}, D_{2j})))^{c_j}$ and $C_2 = \Pi_{j=1}^{l}(C_{1j})^{c_j}$. Finally it sets $CT^{'} = (C_1, C_2)$ and sends $C_0, CT^{'}$ to Data Consumer.

Decrypt : Data consumer sends $C_0, CT^{'} = (C_1, C_2)$ and own decryption secret key $DSK = d$ to its Decryptor as input. Then it computes $C^{''} = ((C_2)^{\frac{1}{d}} \cdot C_1)^d$. Finally generates $m = C_0/C^{''}$.

Table : Notations	
Notations	Descriptions
M, RD	M is the Plaintext (actually the raw data of Data Owner). RD is the Data send to the Data Gateway by Data Owner. In recent basic model RD = M
RK	Randomly choosen encrytion key by Data Gateway
ED	Encrypted data of Data Gateway
EK	Encrypted encryption key
WK	Wrapped key EK
LSSS	Linear Secret Sharing Scheme
GID	Global identifier
AA_f	Attribute authority with index f
GP	System global public parameters
U_f	The attribute universe governed by AA_f
PK_f, $PK_f^{'}$	Public parameters of AA_f
MSK_f, $MSK_f^{'}$	Master secret key of AA_f
CT	Ciphertext (the encrypted data which the data owner send to data gateway in modified model. it is basically the pair (encrypted data private key encryption, encrypted private key))
$\mathbb{A}_1(A, \rho)$	Access structure (policy) expressed by LSSS matrix A and map function ρ and it is set decided by the data owner
$\mathbb{A}_2(A^{'}, \rho^{'})$	Access structure (policy) expressed by LSSS matrix $A^{'}$ and map function $\rho^{'}$ and it is set decided by the policy decision point
DSK	Data consumer's secret decryption key
$S_{gid,f}$	The set of attributes of user (gid) governed by AA_f
$K_{i,gid}$, $K_{i,gid}^{'}$	The attribute secret keys for the attribute $i \in S_{gid,f}$
$K_{f,gid}$, $K_{f,gid}^{'}$	The private keys for $S_{gid,f}$
$T_{i,gid}$	The transformed attribute secret key for the attribute $i \in S_{gid,f}$
$T_{f,gid}$	The transformed private key for $S_{gid,f}$
λ	The security parameter of the system
$\mathbb{Z}_p$	The ring of integers modulo p (without 0)

Table 4.1: List of Notations used in the System

4.4 Data Storing

In this section I am presenting data storing Architecture of my model with step by step explanation.

4.4.1 Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point

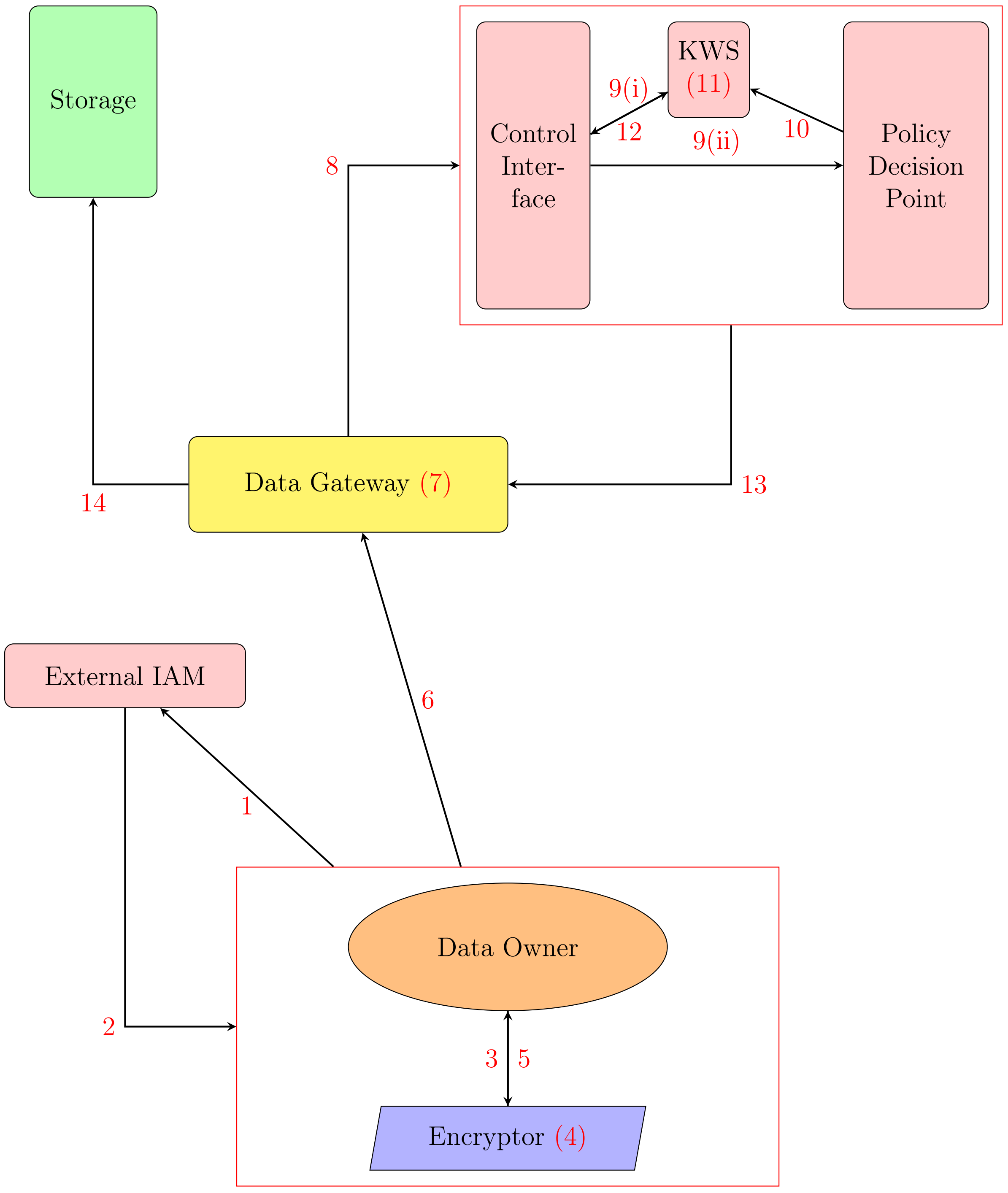


Figure 4.1: Rough Architecture of Data storing in Second (Decentralized) model

1. Data Owner sends his/her Global ID (gid) to the External IAM.
2. External IAM sends a token to Data Owner after verifying all of its attributes.
3. Data Owner choose the message M and then using AES encryption it encrypts M and get $enc(M)$. The AES encryption key (AK) and the choosen access policy $\mathbb{A}_1 = (A, \rho)$ to its encrypter. In my model I forcefully assume that **Data Owner's Access Policy must include attributes of both Attribute Authorities**.
4. Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Encryptor uses those public keys and encrypts AK under $\mathbb{A}_1$ using **Encrypt** algorithm and produce CT_0.

5. Encryptor Sends CT_0 to the Data Owner.

6. Data Owner sends $CT = (enc(M), CT_0) = (enc(M), \mathbb{A}_1, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x \in [l]})$ to the Data Gateway along with own token.

7. Data Gateway take $RD = CT$. Now it does following things (as basic model) on RD :

   (a) Chooses a random key RK and encrypts RD with RK and gets ED.

   (b) Then also encrypts RK with its some special secret key and get EK.

8. Data Gateway sends EK with Data Owner's token to the Control Interface.

9. Control Interface sends EK to the KWS and token to the Policy Decision Point.

10. Policy Decision Point sends another access policy $\mathbb{A}_2 = (A', \rho')$ to the KWS based on data owner's token.

11. KWS runs the **Wrap** (which is basically as same as **Encrypt** algorithm just with different public parameters) algorithm on EK and get the wrapped key (doubly encrypted encryption key) WK.

12. KWS sends WK to Control Interface.

13. Control Interface sends WK to the Data Gateway.

14. Data Gateway sends ED and WK to the Storage.

4.5 Data Consuming

In this section I am presenting data consuming Architecture of my model with step by step explanation.

4.5.1 Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point.

1. Data Consumer sends Global ID (gid) to the External IAM.

2. External IAM runs **AAKeyGen1, AAKeyGen2** and sends token, $K_{1,GID}, K'_{1,GID}$ to Data Consumer.

3. Data Consumer sends token, $K'_{1,GID}$ to the Data Gateway.

4. Data Gateway collects ED and WK from the Storage.

5. Data Gateway sends WK, $K'_{1,GID}$ and consumer's token to Control Interface of Access Control Server.

6. Control Interface sends WK, $K'_{1,GID}$ to KWS and sends consumer's token to Policy Decision Point.

7. Policy Decision Point runs **AAKeyGen1, AAKeyGen2** and sends $K_{2,GID}$ to control interface and $K'_{2,GID}$ to KWS.

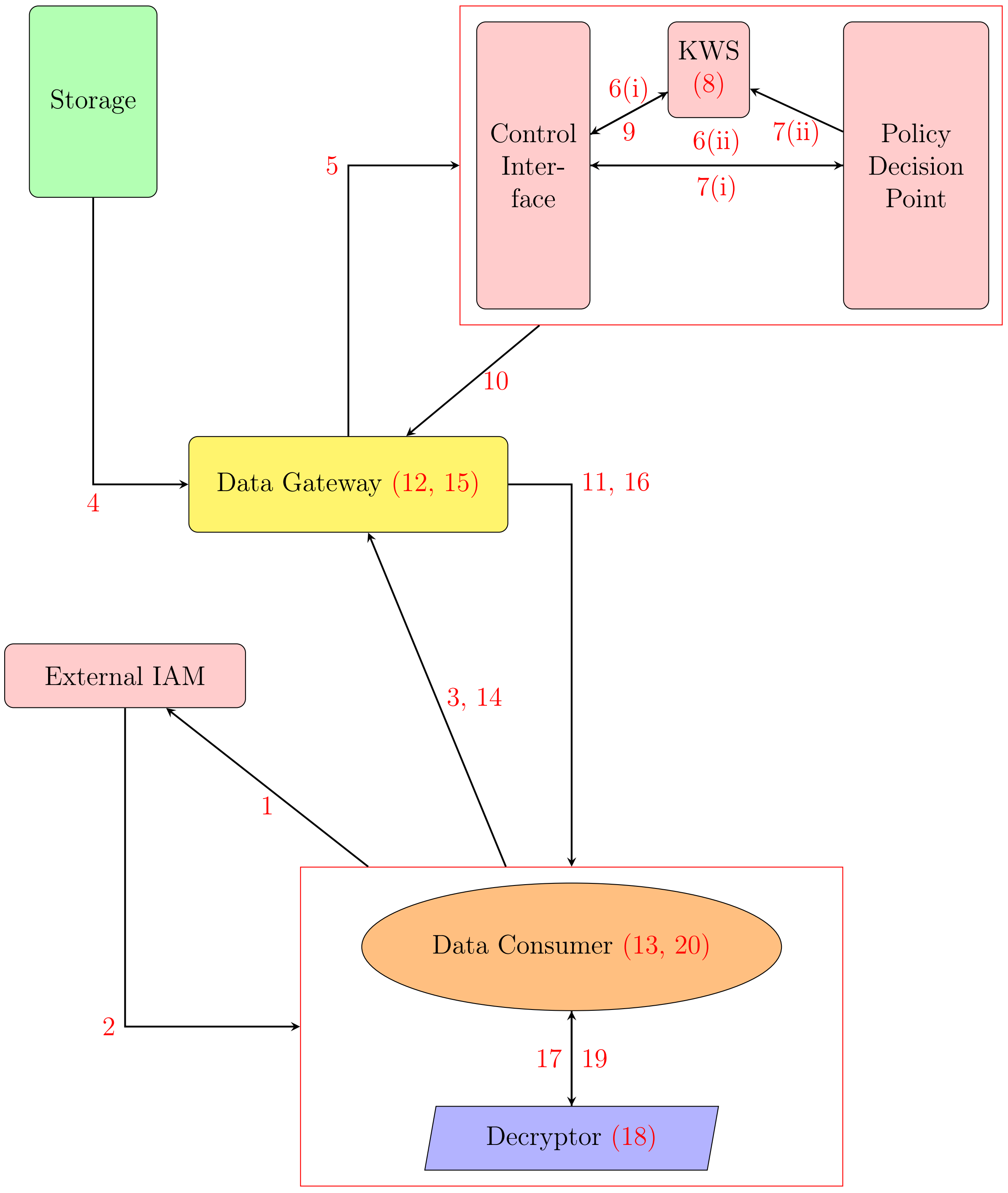


Figure 4.2: Rough Architecture of Data consuming in Second (Decentralized) model

8. KWS runs **UnWrap** and computes unwrapped key EK.
9. KWS sends EK to Control Interface.
10. Control Interface sends $K_{2,GID}$ and EK to the Data Gateway.
11. Data Gateway sends $K_{2,GID}$ to Data Consumer.
12. Data Gateway decrypts EK and gets RK, and after that using that RK decrypts ED and gets $RD = CT = (enc(M), \mathbb{A}_1, C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]}) = (enc(M), CT_0)$.
13. Data Consumer runs **KeyTransform** on $\{K_{i,GID}\}_{i \in S_{GID}} = \{K_{1,GID}, K_{2,GID}\}$ and computes $\{T_{i,GID}\}_{i \in S_{GID}}$ and DSK.

14. Data Consumer sends $\{T_{i,GID}\}_{i \in S_{GID}}$ to Data Gateway.

15. Data Gateway runs **PreDecrypt** on CT_0 and computes $CT^{'}$.

16. Data Gateway sends $enc(M), C_0, CT^{'}$ to Data Consumer.

17. Data consumer sends $CT^{'}, C_0$ and DSK to its Decryptor.

18. Decryptor runs **Decrypt** algorithm with $CT^{'}, C_0$ and DSK as input and retrieve the AES encryption key AK.

19. Decryptor sends AK to Data Consumer.

20. Data Consumer decrypts $enc(M)$ using AK and gets M.

4.6 Security Analysis

Here I will prove the security of the main MA-CP-ABE scheme (base of the construction) which I have used as building block in this second model. First I will describe the base MA-CP-ABE scheme.

Base MA-CP-ABE

This consists of the following algorithms :

Global Setup : Let $\mathbb{G}$ and $\mathbb{G}_1$ denote two bilinear groups of prime order p. Let $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ denote a bilinear map. Choose a generator g of the group $\mathbb{G}$. Also computes $e(g, g)$. After that choose a collision resistant hash function $H : \{GID\} \mapsto \mathbb{G}$. Set global parameter GP $= \{g, p, e, H, e(g, g)\}$.

Authority Setup : AA_f chooses $\alpha_i, y_i \in \mathbb{Z}_p \ \forall i \in U_f$ (U_f is the set of attributes handled by AA_f). Then it computes $e(g, g)^{\alpha_i}, g^{y_i} \ \forall i \in U_f$. It sets $PK_f = \{(e(g, g)^{\alpha_i}, g^{y_i})\}_{i \in U_f}$ as public key and sets $MSK_f = \{\alpha_i, y_i\}_{i \in U_f}$ as secret key.

Encrypt : Data Owner sends its data m and its own policy $\mathbb{A}_1 = (A, \rho)$, where A is a LSSS matrix with l rows and n columns, and ρ associates each row A_x to attribute $\rho(x)$, as input to its encryptor. After that it chooses some secret $s \in \mathbb{Z}_p$. Select two random vectors $\vec{v}, \vec{w} \in \mathbb{Z}_p^n$ such that 1st co-ordinate of $\vec{v}$ is s and 1st co-ordinate of $\vec{w}$ is 0. Now computes $\lambda_j = A_j \cdot \vec{v}$ and $w_j = A_j \cdot \vec{w}$ for all rows $j \in [l]$. Now it chooses $r_j \in \mathbb{Z}_p \ \forall j \in [l]$. Then it computes $C_0 = m \cdot e(g, g)^s$ and $\forall j \in [l]$ also computes $C_{1j} = e(g, g)^{\lambda_j} \cdot e(g, g)^{\alpha_{\rho(j)} r_j}$, $C_{2j} = g^{r_j}$, $C_{3j} = g^{y_j r_j} \cdot g^{w_j}$. Finally sets $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$.

AAKeyGen : Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter MSK_f as input. Then it computes $K_{f,GID} = \{K_{i,GID}\}_{i \in S_{GID,f}} = \{g^{\alpha_i} \cdot H(GID)^{y_i}\}_{i \in S_{GID,f}}$.

KeyTransform : Data Consumer takes the secret attribute keys $\{K_{i,GID}\}_{i \in S_{GID}}$ as input. After that it chooses $d \in \mathbb{Z}_p$, and set its own secret decryption key $DSK = d$. After that it trsforms the attribute keys as $\{T_{i,GID}\}_{i \in S_{GID}} = \{(K_{i,GID}^{\frac{1}{d}}, H(GID)^{\frac{1}{d}}\}_{i \in S_{GID}}$.

PreDecrypt : Data Gateway takes cipher text $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$ and transformed secret keys $T_{1,GID}, T_{2,GID}$ as input. Then computes constants $c_x \in \mathbb{Z}_p$, such

that $\Sigma_{\rho(x)\in S_{gid}}(c_x \cdot A_x) = (1,0,...,0)$. Here $T_{i,GID} = (K_{i,GID}^{\frac{1}{d}}, H(GID)^{\frac{1}{d}})$, $\forall i \in S_{GID}$. If 2nd component of each $T_{i,GID}$ is not same then it will return $\perp$.
Otherwise it computes $C_1 = \Pi_{j=1}^{l}((e(H(GID)^{\frac{1}{d}}, D_{3j}))/(e(T_{\rho(j),GID}, D_{2j})))^{c_j}$ and $C_2 = \Pi_{j=1}^{l}(C_{1j})^{c_j}$. Finally it sets $CT^{'} = (C_1, C_2)$ and sends $C_0, CT^{'}$ to Data Consumer.

Decrypt : Data consumer sends $C_0, CT^{'} = (C_1, C_2)$ and own decryption secret key $DSK = d$ to its Decryptor as input. Then it computes $C^{''} = ((C_2)^{\frac{1}{d}} \cdot C_1)^d$. Finally generates $m = C_0/C^{''}$.

To prove the security of this scheme I will prove that if any attacker can successfully attack this scheme then we can generate one attacker who can successfully attack the OO-MA-DO-CPABE scheme of the paper [12]. Here I will prove that our model is CPA secure.

4.6.1 Security Game Description

A CP-ABE scheme is said to be secure against *chosen plaintext attacks* (CPA) if no probabilistic polynomial-time adversaries have non-negligible advantage in this following game (DC-MA-CP-ABE_$Game_{\mathcal{A},\pi}(S,\lambda)$).

- **Init** : The adversary chooses the challenge access structure W and gives it to the challenger.
- **SetUp** : The challenger runs the Setup algorithm and gives the public parameters PK to the adversary.
- **Phase-1** : The adversary submits Attribute set S for a KeyGen query. Provided $S \nvDash W$, the challenger answers with a secret key SK for S. This can be repeated adaptively.
- **Challenge** : The adversary submits two messages M_0 and M_1 of equal length. The challenger chooses $\mu \in \{0,1\}$ at random and encrypts M_μ to W . The resulting ciphertext CT is given to the adversary.
- **Phase-2** : Same as Phase-1.
- **Guess** : The adversary outputs a guess μ_0 of μ.

Definition : Security of scheme is CPA-secure for attribute universe S if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that :

$$\Pr[\text{ DC-MA-CP-ABE\_}Game_{\mathcal{A},\pi}(S,\lambda) = 1] \leqslant 1/2 + negl(\lambda).$$

4.6.2 Security Game

Theorem : Suppose the construction of OO-MA-DO-CPABE scheme is CPA-secure. Then our proposed scheme is also CPA-secure with due to the Definition.

Proof : If possible let $\mathcal{A}$ be the adversary who can successfully attack our scheme with probability ϵ. Now I will construct an adversary $\mathcal{B}$ who will attack [SS] scheme successfully with probability ϵ. For adversary $\mathcal{A}$, adversary $\mathcal{B}$ plays the role of challenger.

- **Init** : $\mathcal{A}$ chooses some access structure $W = (\mathbb{A}, \rho)$, and send this to $\mathcal{B}$.
  $\mathcal{B}$ sends this $W = (\mathbb{A}, \rho)$ to the challenger.

- **SetUp** : Challenger runs Global Setup and Authority Setup Algorithm and sends all the public keys GP, PK_1 and PK_2 to $\mathcal{B}$.
  $\mathcal{B}$ sends public keys GP, PK_1 and PK_2 to $\mathcal{A}$.
- **Phase-1** : Now $\mathcal{A}$ queries to $\mathcal{B}$ for secret keys for the set $S \nvDash W$ and $\mathcal{B}$ passes those same queries to challenger.
  Challenger sends the query responses to $\mathcal{B}$ and $\mathcal{B}$ sends those responses directly to $\mathcal{A}$.
- **Challenge** : Now $\mathcal{A}$ chooses two messages m_0, m_1 of same length and send those to $\mathcal{B}$.
  $\mathcal{B}$ sends m_0, m_1 to the challenger.
  Challenger chooses random $b \in \{0,1\}$ and encrypts m_b and gets
  $CT' = ((\mathbb{A},\rho), C_0, \{C'_{1j}, C'_{2j}, C'_{3j}, C'_{4j}, C'_{5j}, CT_{1j}, CT_{2j}\}_{j\in[1,p]})$. Challenger sends CT' to $\mathcal{B}$.
  $\mathcal{B}$ computes $C_{1j} = C'_{1j} \cdot CT_{1j} \cdot e(g,g)^{C'_{4j}}$, $C_{2j} = C'_{2j}$ and $C_{3j} = C'_{3j} \cdot CT_{2j} \cdot g^{C'_{5j}}$ and sends $CT = ((\mathbb{A},\rho), C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j\in[1,p]})$ to $\mathcal{A}$.
- **Phase-2** : Again $\mathcal{A}$ queries to $\mathcal{B}$ for secret keys for the set $S \nvDash W$ and $\mathcal{B}$ passes those same queries to challenger.
  Challenger sends the query responses to $\mathcal{B}$ and $\mathcal{B}$ sends those responses directly to $\mathcal{A}$.
- **Guess** : $\mathcal{A}$ guess $b' \in \{0,1\}$ and sends b' to $\mathcal{B}$.
  $\mathcal{B}$ sends b' to Challenger.

Since $C_0 = m_b \cdot e(g,g)^s$ is same in both CT and CT', we can say that win probability of $\mathcal{B}$ is same as $\mathcal{A}$. This completes my proof.

$\blacksquare$

This Scheme is also collusion resistant because $H(GID)$ is different for each users with very high probability (since H is collision resistant), so they can't combine their Authority Attribute Secret keys to access an unauthorized data.

Suppose Alice or Bob does not have an authorized set of attributes but union of their attribute sets is an authorized set and they are planning to collude.

Let us consider all the keys of both Alice and Bob.

Alice's Key : $\{K_{i,GID_A}\}_{i\in S_{GID_A},f} \;= \{g^{\alpha_i} \cdot H(GID_A)^{y_i}\}_{i\in S_{GID_A},f}$

Bob's Key : $\{K_{i,GID_B}\}_{i\in S_{GID_B},f} \;= \{g^{\alpha_i} \cdot H(GID_B)^{y_i}\}_{i\in S_{GID_B},f}$

Suppose after mixing up their keys there is two attributes $i, j \in U_f$ where i is only Alice's attribute and j is only Bob's attribute. In that case they will use $H(GID)$ of only one of them but use $K_{i,GID}$'s from both of them to make transform keys. They can not use $H(GID)$ of both of them to generate transform keys because the last component of all transformed keys have to be same, otherwise they will be caught.

Let Alice gives all its keys with K_{j,GID_B} of Bob and when we add attribute j to Alice's attribute set it becomes an authorized set.

But then if we calculate $C_1 = \Pi_{j=1}^{l}((e(H(GID)^{\frac{1}{d}}, D_{3j}))/(e(T_{\rho(j),GID}, D_{2j})))^{c_j}$, we will get $C_1 = (1/e(g,g)^{\frac{\alpha_j r_j c_j}{d}}) \cdot (\frac{e(H(GID_A),g)}{e(H(GID_B),g)})^{\frac{y_j r_j c_j}{d}}$ instead of getting $C_1 = (1/e(g,g)^{\frac{\alpha_j r_j c_j}{d}})$. And So when we will calculate $C'' = ((C_2)^{\frac{1}{d}} \cdot C_1)^d$ we will get $C'' = e(g,g)^s \cdot (\frac{e(H(GID_A),g)}{e(H(GID_B),g)})^{y_j r_j c_j}$ instead of getting $C'' = e(g,g)^s$. And finally after decryption we will get $M \cdot (\frac{e(H(GID_B),g)}{e(H(GID_A),g)})^{y_j r_j c_j}$ instead of getting the actual message M.

Now $M \cdot (\frac{e(H(GID_B),g)}{e(H(GID_A),g)})^{y_j r_j c_j} = M$ will happen iff $H(GID_A) = H(GID_B)$ (as alice and bob can be choosen arbitrarily). But $H(GID_A) = H(GID_B)$ where $GID_A \neq GID_B$ implies that we have found a collision for the hash function H, which is not possible as H is collision resistant. So, Alice do not get the actual message M.

Similarly for the other case also we will get a wrong result. Therefore two unauthorized user can not get any facility after a successful collusion. Hence Our scheme is collusion resistant.

Chapter 5

Comparison of Efficiency

I am going to compare my two models, Centralized Multi-Authority Model (First Model) and Decentralized Multi-Authority Model (Second Model), and the models with the schemes (if we use those schemes in our model) from the papers “Decentralizing Attribute-Based Encryption" by Allison Lewko and Brent Waters [LW][10] (scheme with prime order bilinear group [LWP] and scheme with composite order bilinear group [LWC]), “Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption" by Yannis Rouselakis and Brent Waters [RW][17] and “Efficient Decentralized Attribute Based Access Control for Mobile Clouds" by Sourya Joyee De and Sushmita Ruj [SS][12].

For simplicity of expression, we assume that

u = number of user's attribute
i = number of user's attribute handled by External IAM
p = number of user's attribute handled by Policy Decision Point
(So, clearly $i + p = u$.)
l = number of rows of the Data Owner's Access Policy matrix
r = number of rows of the Data Owner's Access Policy matrix which actually indicates user's attribute
s = number of rows of the Policy Decision Point's Access Policy matrix
v = number of rows of the Policy Decision Point's Access Policy matrix which actually indicates user's attribute
exp = exponentiation

We ignore the costs of Basic Model, i.e. we will ignore the cost of those cmputations which are in the Recent Basic Model.

5.1 Comparison of Computation Cost

We have divided all the communication costs in 3 parts, Key Generation, Data Storing and Data Consuming (we deon't consider revocation here as all the schemes of those papers are not revocable and if we include our revocation process in those schemes then the computation cost for revocation will be same in each schemes).

5.1.1 Key Generation

Table : Key Generation Computation Cost		
Schemes	External IAM	Policy Decision Point
[LWP] [10]	4i exp	4p exp
[LWC] [10]	4i exp	4p exp
[RW] [17]	4i exp	4p exp
[SS] [12]	4i exp	4p exp
First	$(i+4)$ exp	$(p+2)$ exp
Second	4i exp	4p exp

Table 5.1: Computation costs for Key Generation

5.1.2 Data Storing

Table : Data Storing Computation Cost		
Schemes	Data Owner's Encryptor	KWS
[LWP] [10]	$(5l + 1)$ exp + 1 pairing	$(5s + 1)$ exp + 1 pairing
[LWC] [10]	$(5l + 1)$ exp + 1 pairing	$(5s + 1)$ exp + 1 pairing
[RW] [17]	$(6l + 1)$ exp + 1 pairing	$(6s + 1)$ exp + 1 pairing
[SS] [12]	$(9l + 1)$ exp + 1 pairing	$(9s + 1)$ exp + 1 pairing
First	$(3l + 2)$ exp + 1 pairing	$(3s + 2)$ exp + 1 pairing
Second	$(5l+1)$ exp	$(5s+1)$ exp

Table 5.2: Computation costs for Data Storing

5.1.3 Data Consuming

Table : Data Consuming Computation Cost				
Schemes	Data Consumer	Data Consumer's Decryptor	Data Gateway	KWS
[LWP] [10]	–	2r pairing + r exp	–	2v pairing + v exp
[LWC] [10]	–	2r pairing + r exp	–	2v pairing + v exp
[RW] [17]	–	3r pairing + r exp	–	3v pairing + v exp
[SS] [12]	4u exp	4 exp	2r pairing + 2r exp	2v pairing + 2v exp
First	–	2 exp	$(2r + 1)$ pairing + r exp	$(2v + 1)$ pairing + v exp
Second	2u exp	2 exp	2r pairing + 2r exp	2v pairing + v exp

Table 5.3: Computation costs for Data Consuming

From the above tables it seems that my first model is more efficient than others but that is not true. We know that cost of pairing and exponentiation in composite order bilinear group is much more than in prime order bilinear group. From the paper [17] we can get the idea. I have also explained about it in Appendix-C.

So from that fact and my comparison tables I can say that my second model i.e. the decentralized model is more efficient than others acccording to computation costs.

5.2 Comparison of Communication Cost

Here we will consider only those communications which involves more than basic model. For simplicity of expression, we assume that

$|\mathbb{G}|$ = The bit length of the element in $\mathbb{G}$
$|\mathbb{G}_1|$ = The bit length of the element in $\mathbb{G}_1$
$|N|$ = The bit length of the element in $\mathbb{Z}_N$
$|EK|$ = The bit length of encrypted encryption key which is choosed by Data Gateway
Here we have assume that the pairing map is $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$.

5.2.1 Data Storing

Table : Data Storing Communication Cost			
Schemes	Data Owner to Data Gateway	Control Interface to Data Gateway	Data Gateway to Storage
[LW] [10]	$2l\|\mathbb{G}\| + (l+1)\|\mathbb{G}_1\|$	$2s\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$2(l+s)\|\mathbb{G}\|+(l+s+2)\|\mathbb{G}_1\|$
[RW] [17]	$3l\|\mathbb{G}\| + (l+1)\|\mathbb{G}_1\|$	$3s\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$3(l+s)\|\mathbb{G}\|+(l+s+2)\|\mathbb{G}_1\|$
[SS] [12]	$3l\|\mathbb{G}\|+(2l+1)\|\mathbb{G}_1\|$	$3s\|\mathbb{G}\|+(2s+1)\|\mathbb{G}_1\|$	$3(l+s)\|\mathbb{G}\| + 2(l+s+1)\|\mathbb{G}_1\|$
First	$(2l+1)\|\mathbb{G}\| + \|\mathbb{G}_1\|$	$(2s+1)\|\mathbb{G}\| + \|\mathbb{G}_1\|$	$2(l+s+1)\|\mathbb{G}\| + 2\|\mathbb{G}_1\|$
Second	$2l\|\mathbb{G}\| + (l+1)\|\mathbb{G}_1\|$	$2s\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$2(l+s)\|\mathbb{G}\|+(l+s+2)\|\mathbb{G}_1\|$

Table 5.4: Communication costs for Data Storing

5.2.2 Key Accessing

Table : Key Accessing Communication Cost			
Schemes	External IAM to Data Consumer	Control Interface to Data Gateway	Data Gateway to Data Consumer
[LW] [10]	$2i\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$
[RW] [17]	$4i\|\mathbb{G}\|$	$4p\|\mathbb{G}\|$	$4p\|\mathbb{G}\|$
[SS] [12]	$2i\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$
First	$\|N\| + (i+4)\|\mathbb{G}\|$	$(p+1)\|\mathbb{G}\|$	$(p+1)\|\mathbb{G}\|$
Second	$2i\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$	$2p\|\mathbb{G}\|$

Table 5.5: Communication costs for Key Accessing

5.2.3 Data Consuming

Table : Data Consuming Communication Cost					
Schemes	Storage to Data Gateway	Data Consumer to Data Gateway	Data Gateway to Control Interface	Control Interface to Data Gateway	Data Gateway to Data Consumer
[LW] [10]	$2(l+s)\|\mathbb{G}\| + (l+s+2)\|\mathbb{G}_1\|$	$u\|\mathbb{G}\|$	$(2s+u)\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$\|EK\|$	$2l\|\mathbb{G}\| + (l+1)\|\mathbb{G}_1\|$
[RW] [17]	$3(l+s)\|\mathbb{G}\| + (l+s+2)\|\mathbb{G}_1\|$	$2u\|\mathbb{G}\|$	$(3s+2u)\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$\|EK\|$	$3l\|\mathbb{G}\| + (l+1)\|\mathbb{G}_1\|$
[SS] [12]	$3(l+s)\|\mathbb{G}\| + 2(l+s+1)\|\mathbb{G}_1\|$	$2u\|\mathbb{G}\| + \|EK\|$	$(3s+u)\|\mathbb{G}\| + (2s+1)\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$	$4\|\mathbb{G}_1\|$
First	$2(l+s+1)\|\mathbb{G}\| + 2\|\mathbb{G}_1\|$	$(u+2)\|\mathbb{G}\| + \|EK\|$	$(2s+u+3)\|\mathbb{G}\| + \|\mathbb{G}_1\|$	$\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$
Second	$2(l+s)\|\mathbb{G}\| + (l+s+2)\|\mathbb{G}_1\|$	$2u\|\mathbb{G}\|$	$(2s+u)\|\mathbb{G}\| + (s+1)\|\mathbb{G}_1\|$	$\|EK\|$	$2\|\mathbb{G}_1\|$

Table 5.6: Communication costs for Data Consuming

From the above comparison tables it is very clear that my first model i.e. the centralized model is more efficient than others acccording to communication costs.

Chapter 6

Implementation

In this thesis one of the main goal is to implement the Multi-Authority CP-ABE scheme. I have implemented my Second Multi-Authority CP-ABE scheme i.e. Decentralized Multi-Authority CP-ABE scheme. To implement this I have choosen python language and Charm-crypto framework.

I have not implemented my first Multi-Authority CP-ABE scheme i.e. Centralized Multi-Authority CP-ABE scheme. Actually that scheme is based on composite order bilinear group which are several orders of magnitude slower than the prime order groups that provide the same security level. And another important reason of not implementing my first scheme is Charm does not support composite order groups. Although my first scheme takes more computation time than my second scheme, my first scheme is more secure than my second scheme. It is adaptively secure in standard model. Also my first scheme needs less communication cost. Because of these strong positive things we don't discard this scheme from our thesis. Actually if there is a scenario where to get adaptive security in standard model is much more important than computation cost, then our first scheme is a very good fit for that.

6.1 Framework

We implemented our scheme in Charm-crypto [18], a framework developed for convenience rapid prototyping of cryptographic schemes and protocols. It is based on the Python language which allows the programmer to write code similar to the theoretical implementations. However, the routines that implement the dominant group operations use the PBC library [16] (written natively in C) and the time overhead imposed by the use of Python is usually less than 1%.

We tested several ABE constructions on all elliptic curve bilinear groups provided by Charm-crypto, i.e. two super-singular “SS" symmetric EC groups and three “MNT" asymmetric EC groups. In Appendix-B, we have presented the approximate security level each group provides with respect to the discrete log problem. All our benchmarks were executed on a Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz with 4GB RAM running Ubuntu 20.04 and Python3.8.5.

6.2 Implementation Details and Comparisons

All Charm-crypto routines use formally asymmetric groups (although the underlining groups might be symmetric) and therefore we translated our schemes to the asymmetric setting. Namely, we have three groups G1, G2 and GT and the pairing e is a function from $G1 \times G2$ to GT.

To get the bilinear group and the pairing, I have imported all functions like PairingGroup, pair, G1,G2, ZR from “charm.toolbox.pairinggroup". Here initially I have used “SS512" as group object but it can be done with some other group objects also. I have implemented that CP-ABE with and without decryption outsourcing. Here I am attaching my programs :

- Decentralized Multi-Authority CP-ABE without Decryption Outsourcing
- Decentralized Multi-Authority CP-ABE with Decryption Outsourcing

I have compared my program with some other decentralized multi-authority CP-ABE programs which are based on the papers [10, 17]. I have denoted paper [10] (the prime version) by [LWP10] and paper [17] by [RW17]. I have compared their Setup time, Key Generation time, cipher text size, Encryption time and Decryption time. I get some diagrams which I am attaching here and with each diagram I will describe the parameters.

Setup Time : Here we observe that how the setup time differs when we increase the total number of attributes of the system. So it is basically a graph for Setup time Vs Total Number of Attributes.

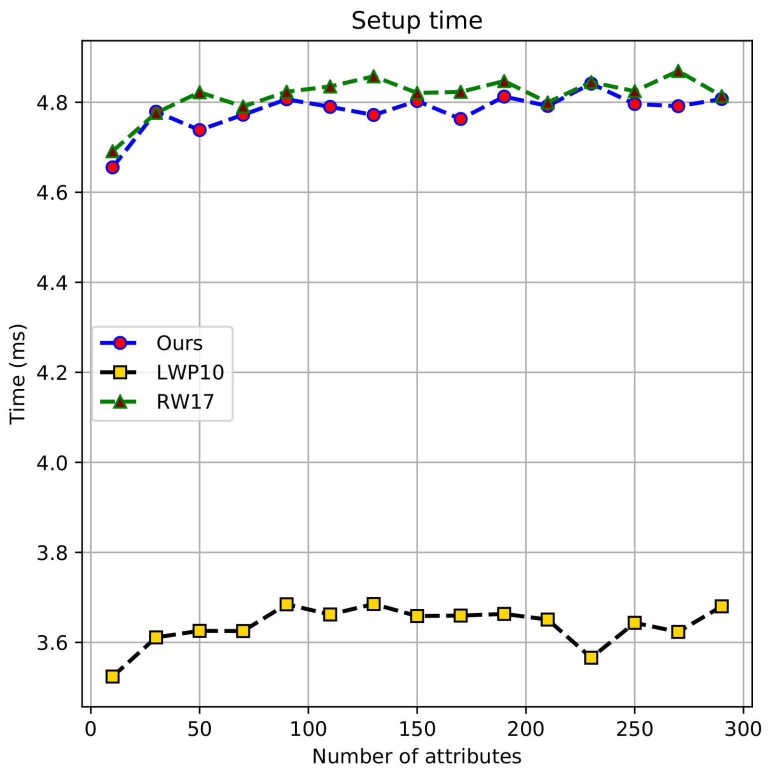


Figure 6.1: Comparision of Setup Time

Here we can observe that our setup time is more than [LWP10] but we need this Setup Algorithm only once at the time of initializing the model. So, there is nothing to worry about it.

Authority Setup Time : Here we observe that how the authority setup time differs when we increase the total number of attributes of the system. So it is basically a graph for Authority Setup time Vs Total Number of Attributes.

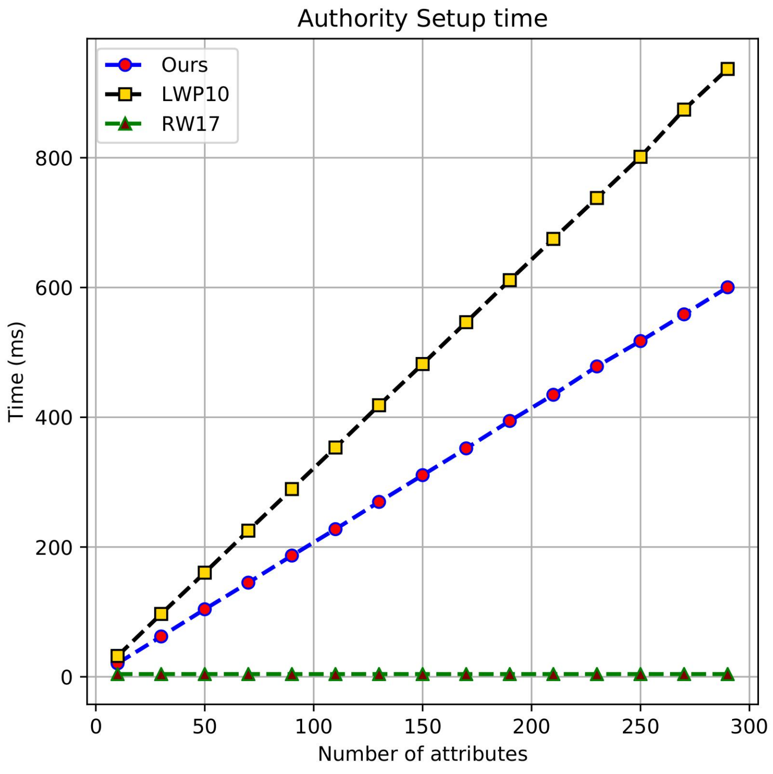


Figure 6.2: Comparision of Authority Setup Time

Here we can observe that our authority set up time is more than [RW17] but we need this Authority Setup Algorithm only once at the time of initializing the Attribute Authorities in the model. And also in our application we are not dealing with very large number of attributes, so here also we do not need to be worried.

Key Generation Time : Here we observe that how the Key Generation Time increases when we increase the number of user's attributes. So it is basically a graph for Key Generation Time Vs Number of User's Attributes.

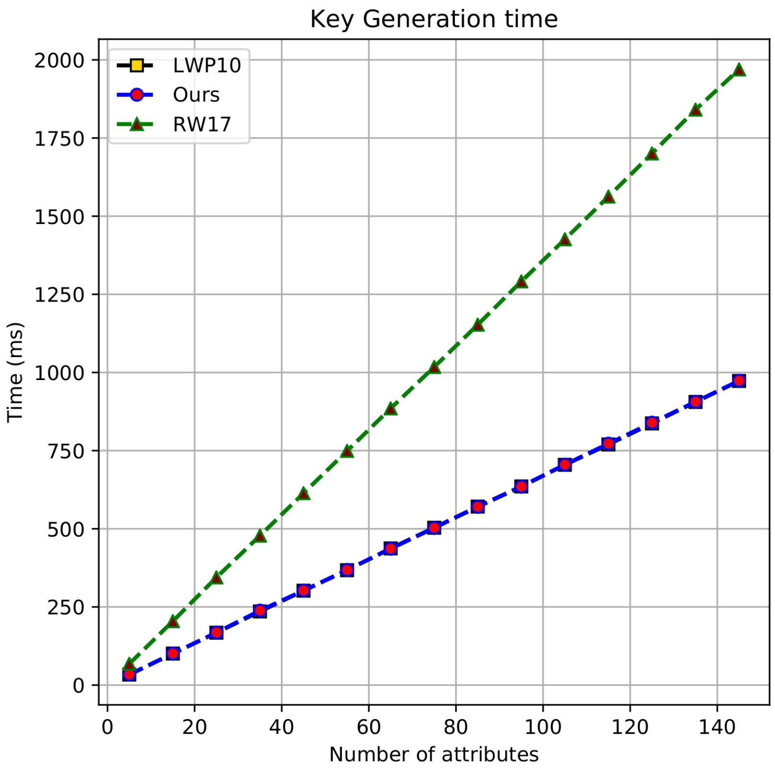


Figure 6.3: Comparision of Key Generation Time

We can observe that key generation algorithm of our model is taking the least time among all of them. So it is a very positive thing for our model.

Encryption Time : Here we observe that how the Encryption Time differs when we increase the number of attributes in the access policy. So it is basically a graph for Encryption Time Vs Number of Attributes in Access Policy.

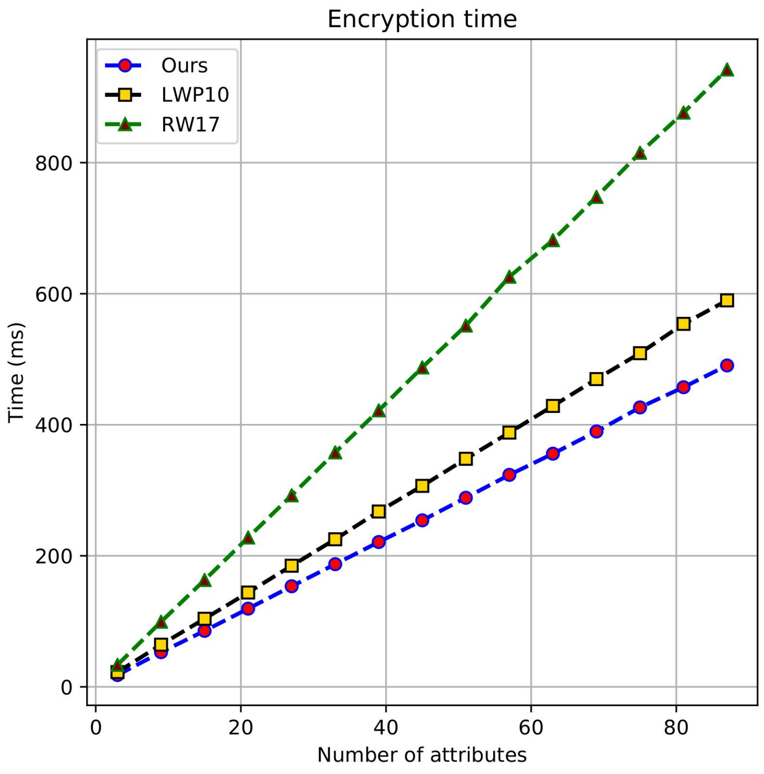


Figure 6.4: Comparision of Encryption Time

We can observe that encryption algorithm of our model is taking the least time among all of them. So it is also a positive thing for our model.

Cipher-text Size : Here we observe that how the Cipher-text Size differs when we increase the number of attributes in the access policy. So it is basically a graph for Cipher-text Size Vs Number of Attributes in Access Policy.

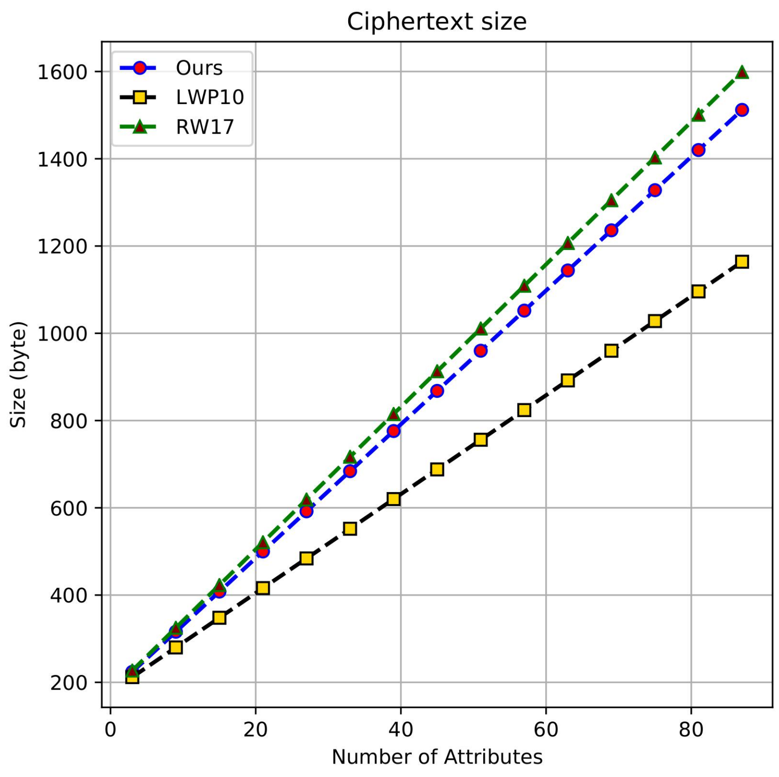


Figure 6.5: Comparision of Cipher-text Size

Here we can see that the ciphertext size of our model is more than [LWP10], but for our application we do not need large number of attributes in the access policy. So it will not be that much problematic. And scince the encryption time is less we can say that our model is a good fit for our application scenario.

Decryption Time : Here we observe that how the Decryption Time differs when we increase the number of user's attributes. So it is basically a graph for Decryption Time Vs Number of User's Attributes.

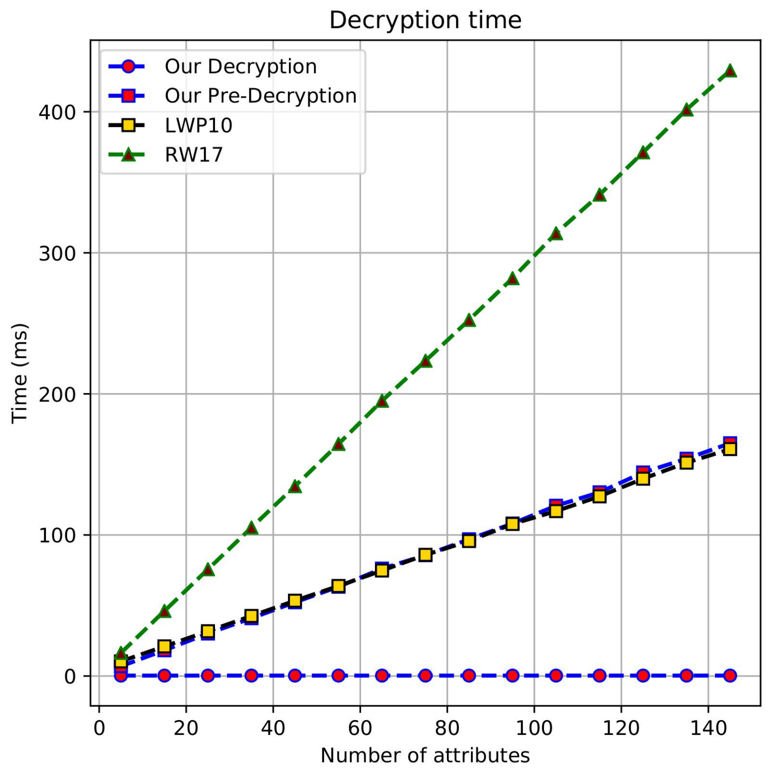


Figure 6.6: Comparision of Decryption Time

Here we have shown our pre-decryption time and final decryption time in this graph. Our pre-decryption process is computed in the cloud component Data Gateway and our final decryption process is computed on the data consumer's personal device. In the graph, our decryption time is constant because we have used the scheme with decryption outsourcing. So for our final decryption we just have to do only two exponentiations, one multiplication and one devision irrespective of number of user's attribute (see Section 4.3). And also here we can observe that the pre-decryption time is also good.

After that I have modified my programs to achieve more efficiency. For that at first I have implemented the paper [5]. I have implemented a program which can transfer any threshold access structure into LSSS. After that I have further modify the program for generating LSSS matrix at the time of decryption. I am attaching my code links here :

- Program to generate LSSS
- Modified Program to generate LSSS for decryption

I have also given the algorithm and python code to implement the LSSS of the paper [5] in Appendix-D.

Then using this two programs I have modified my code. Now I am attaching my modified programs here :

- Modified Program without decryption outsourcing
- Modified Program with decryption outsourcing

After using this modified LSSS the implementation of the CP-ABE scheme becomes more efficient. One can easily observe that from the following table :

Charm's LSSS Vs. Modified LSSS				
Number of Attributes in Access Policy	Encryption Time with Charm's LSSS (ms)	Pre-Decryption Time with Charm's LSSS (ms)	Encryption Time with Modified LSSS (ms)	Pre-Decryption Time with Modified LSSS (ms)
8	48.56	18.21	45.58	13.07
12	70.96	26.41	67.70	21.07
16	94.37	35.53	89.64	29.25
20	119.56	43.99	112.06	34.72
24	139.58	52.17	133.70	38.98
28	162.99	58.97	158.76	55.42

Table 6.1: Charm's LSSS Vs. Modified LSSS : Comparison of Efficiency

Now I am going to give a comparison table for our modified CP-ABE code and the other two CP-ABE codes for better understanding. In the following table, average running times are in milliseconds(ms). The algorithms are denoted as GS: Global setup, AS: Authority setup, KG: Key generation for a user, EC: Encrypt, DE: Decrypt. The numbers in parentheses refer to the number of attributes in key generation, the number of rows of the policy in encryption, and the number of rows utilized during decryption.

Our Decentralized scheme with decryption outsourcing					
Curves	GS	AS(19)	KG(6)	EC(8)	DC(6)
SS512	4.66	41.60	42.32	46.45	0.32
SS1024	33.65	477.42	272.17	579.50	5.13
MNT159	9.49	120.51	7.32	138.46	2.24
MNT201	12.23	152.30	10.57	176.17	2.86
MNT224	15.11	189.16	13.95	220.27	3.51
[LWP10][10]					
Curves	GS	AS(19)	KG(6)	EC(8)	DC(6)
SS512	3.48	63.42	42.06	58.88	21.22
SS1024	2.44	1068.31	272.45	856.04	384.61
MNT159	5.60	189.86	7.17	174.64	53.32
MNT201	7.30	245.67	10.48	223.14	68.10
MNT224	9.23	304.86	13.41	279.95	84.11
[RW17][17]					
Curves	GS	AS(19)	KG(6)	EC(8)	DC(6)
SS512	4.70	4.21	84.27	89.87	45.91
SS1024	32.95	51.96	545.85	762.12	575.17
MNT159	9.47	4.11	240.67	194.88	160.79
MNT201	12.18	5.35	363.22	265.61	221.77
MNT224	15.28	6.18	371.98	300.69	249.80

Table 6.2: Comparison Table of Running Times of Schemes

The modified CP-ABE code with 'Modified LSSS' uses 'numpy' to get the solution of a system of linear equations (for linear reconstruction property). Actually in our application senario we do not need large number of attributes in access policy, so this modified CP-ABE code with 'Modified LSSS' works very good for our application.

But if anyone want to apply our modified CP-ABE code with 'Modified LSSS' in an application where they have to work with large number of attributes in access policy, then they have to think about an efficient way to solve a system of linear equations with large number of equations because 'numpy' does not give accurate results for large number of equations. If one can find a efficient way to solve a system of linear equations with large number of equations then they can apply our modified CP-ABE code with 'Modified LSSS' for their application also.

Chapter 7

Further Modification

In reality there may be some malicious user in access control model. In that case it is very much necessary to revoke them from the system or revoke some attributes of them.
But in previous models we have no scope for revocation, so now we want to introduce revocable access model just modyfing some components and functionalities. We will introduce this property in both of my models. However, the revocation phase is not efficient because the proxy server (which I have introduced in the next section) needs to re-encrypt the data.

7.1 First Model with Revocation

7.1.1 Components and Their Uses

I add one extra component and add some more responsibilities & functionalities to the components. The other components are same as First Model (Centralized Multi-Authority Model).

External IAM : In my model External IAM plays a big role. It serves as Central Authority (CA) and also one of the two Multi-Authorities besides its basic works (which it does in recent basic model).
In my model this component has to be **Semi-Honest** and it **can not collude with Policy Decision Point** (which is another Attribute Authority in my model).
It basically verify the attributes based on the global id of the users and gives a **token** corresponding to their id (basic model functionality) and gives **Central Authority Public & Secret Key** and one **Decryption Secret Key** with respect to the global id (work as CA) and also one **Authority Attribute Secret Key** based on the user's attributes which are handled by it (work as AA).

Policy Decision Point : In my model Policy Decision Point serves also as another Attribute Authority besides its basic work.
This compoent must be **Semi-Honest** and have to handle some attributes of its Access Policy (since for Multi-Authority CPABE encryption we have only two authorities and if one of them only containts all attributes of the Access Policy then it will be no more secure).
It basically handle all policies, produce one another access policy $\mathbb{A}_2$ for key wrapping, publish revocation list and computes the authority attribute secret key for Data Consumer.

Data Owner : He/she first encrypts the raw data with the two Attribute Public Key under his/her Access Policy $\mathbb{A}$.

Data Consumer : Data Consumer use his/her own **Decryption Secret Key** (which he/she gets from External IAM) to Decrypt the Pre-Decryted data.

Data Gateway : It does all its basic works on two-time encrypted data in stead of the raw data. But besides these it also does pre-decryption of the user's message. As Basic Model, it is also **Semi-Honest** in my model.

KWS : It wraps the encrypted key and also unwraps the wrapped key using MA-CPABE and it is also **Semi-Honest**.

Proxy Server : We introduce a Proxy server which is basically in cloud and it stores all necessary keys related to revocation and also update cipher texts properly. It is also **Semi-Honest and can't colude with any Revoked User ever**.

7.1.2 Fuctionalities used in the System

Global Setup : Let $\mathbb{G}$ and $\mathbb{G}_1$ denote two bilinear groups of order $N = p_1p_2p_3$ (a product of 3 different primes). Let $\mathbb{G}_{p_i}$ be the subgroup of order p_i in $\mathbb{G}$. Moreover, the subgroups $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ have the orthogonality property. That is, for $h_i \in \mathbb{G}_{p_i} and h_j \in \mathbb{G}_{p_j}$, if $i \neq j$, we have $e(h_i, h_j) = 1$. Let $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ denote a bilinear map. g is a random chosen element from $\mathbb{G}_{p_1}$. Additionally, choose an UF-CMA (unforgeable under adaptive chosen message attacks) secure signature system $\Sigma_{sign} = (KeyGen, Sign, Verify)$. The GPK is broadcasted as GPK = (N, e, g, Σ_{sign}).

Central Authority Setup : The CA runs the $KenGen$ algorithm of Σ_{sign} . It sets the sign-key as CMK (Central Master Key) and verify-key as CPK (Central Public Key). The CPK will be used by the AAs only. The CMK is used by only CA to generate Keys (only for one time) for Users.

Attribute Authorities Setup : Each AA_f governs its attribute universe U_f. For each $i \in U_f$, it chooses a random exponent $t_{f,i} \in \mathbb{Z}_N$ and computes $T_{f,i} = g^{t_{f,i}}$. It also chooses two random exponents $\alpha_f, a_f \in \mathbb{Z}_N$. Finally, the public parameter of AA_f is broadcasted as : $APK_f = (g^{a_f}, e(g,g)^{\alpha_f}, T_{f,i}\ \forall i)$. The master secret key of AA_f is $AMK_f = (\alpha_f, a_f, t_{f,i}\ \forall i)$.
Additionally, each AA_f also maintains a binary tree $TREE_f$. In $TREE_f$, each node j is associated with a different encryption key $KEK_{f,j}$ and each leaf node is labeled by a user (gid). Such a tree with height h can accommodate at most 2^h users. Moreover, there is a path P_j from j to the root node. When a new user comes to AA_f for requesting the attribute-related keys, AA_f not only generates the requested keys, but also adds the user to the leftmost leaf node j and gives him/her the path keys in the path P_j. For each attribute $i \in U_f$, AA_f establishes an attribute-user group $G_{f,i}$, which is a set of the users who own this attribute i. We let $Gn_{f,i}$ denote the minimum set of nodes whose descendant nodes cover all the users in $G_{f,i}$. At the beginning of system initialization, each AA_f shares a unique attribute group key $AK_{f,i} \in \mathbb{Z}_N$ with the proxy server (in cloud) for each $i \in U_f$. Whenever a new user registered in the system or a revocation list is published it updates all $AK_{f,i}$.

Encrypt : Here the access policy is defined by $\mathbb{A} = (W, \rho)$, where W is a LSSS matrix with l rows and n columns, and ρ associates each row W_x to attribute $\rho(x)$.
Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Data Owner's encryptor uses those public keys and encrypts M under $\mathbb{A}$ as follows :

1. Chooses a random vector $\vec{v} = (s, v_2, ..., v_{n_1}) \in \mathbb{Z}_N^n$, where s is the secret value.
2. For each $x \in [l]$, it selects a random exponent $r_x \in \mathbb{Z}_N$.
3. Computes $C_1 = M \cdot (\Pi_{f=1}^2 e(g,g)^{\alpha_f})^s$, $C_{01} = g^s$. For each $x \in [l]$ it also computes $C_{x1} = g^{(\Sigma_{f=1}^2 a_f)\cdot W_x \cdot \vec{v}} \cdot T_{\rho(x)}^{-r_x}$, $D_{x1} = g^{r_x}$.
4. Set $CT = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x \in [l]})$.

After encryption, at the time of data storing first $CT = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]})$ is sent to proxy server and proxy server computes $D'_{x1} = D_{x1}^{AK_{\rho(x)}}$ and sets $CT' = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D'_{x1}\}_{x\in[l]})$ and sends CT' for storing.

When a new user joins in the system, he/she has to register himself/herself and will obtain a unique gid. By running the CAKeyGen algorithm, the CA issues the gid-related keys to the users. Then, each AA runs the AAKeyGen algorithm and gives the attribute-related keys to the users.

CAKeyGen : For each user, the CA first chooses two random exponents $b_{gid}, c_{gid} \in \mathbb{Z}_N$, two random elements $R_{gid}, R_{gid,0} \in \mathbb{G}_{p_3}$ and computes $CASK_{gid} = L_{gid} = g^{b_{gid}/c_{gid}} \cdot R_{gid}$, $L_{gid,0} = g^{1/c_{gid}} \cdot R_{gid,0}$. After that, it uses CMK to sign on the string $(CMK, gid||CASK_{gid}||L_{gid,0})$ and gets a signature σ_{gid}. Let $CAPK_{gid} = (gid, CASK_{gid}, L_{gid,0}, \sigma_{gid})$. Finally, it sends the $DSK_{gid} = c_{gid}, CASK_{gid}$ and $CAPK_{gid}$ to the user. After sending the keys to user CA delete $b_{gid}, c_{gid}, R_{gid} R_{gid,0}$ from its memory.

AAKeyGen : After receiving the submitted key $CAPK_{gid}$, the AA_f first uses the CPK to verify whether the $CAPK_{gid}$ is valid. If not, it aborts. Otherwise, it issues the user a set of attributes $S_{gid,f}$. It randomly selects $R_{gid,f,0} \in \mathbb{G}_{p_3}$ and computes $K_{gid,f} = L_{gid,0}^{\alpha_f} \cdot L_{gid}^{a_f} \cdot R_{gid,f,0} = g^{\alpha_f/c_{gid}} \cdot g^{a_f \cdot b_{gid}/c_{gid}} \cdot R_{gid,f}$, where $R_{gid,f} = R_{gid,f,0} \cdot R_{gid,0}^{\alpha_f} \cdot R_{gid}^{a_f}$. For each attribute $i \in S_{gid,f}$, it randomly picks $R'_{gid,f,i} \in \mathbb{G}_{p_3}$ and computes $K_{gid,f,i} = L_{gid}^{t_{f,i}} \cdot R'_{gid,f,i} = T_{f,i}^{b_{gid}/c_{gid}} \cdot R_{gid,f,i}$, where $R_{gid,f,i} = R_{gid}^{t_{f,i}} \cdot R'_{gid,f,i}$.
In addition, for each unrevocked attribute $i \in S_{gid,f}$, the AA_f sets the set $G_{f,i}$ and $Gn_{f,i}$. It then encrypts $AK_{f,i}$ by $KEK_{f,j}$ if $j \in Gn_{f,i}$ is the ancestor node of the leaf node in which the user is assigned. It finally sends $ASK_{S,gid,f} = (K_{gid,f}, \{K_{gid,f,i}\}_{i \in S_{gid,f}})$ and the encrypted $\{AK_{f,i}| \, unrevocked \; i \in S_{gid,f}\}$ to the user.

TransformKey : The Data consumer first use path decryption keys (path keys of AA_f) to get $\{AK_{f,i}|i \in S_{gid,f}\}$ and computes $K'_{gid,f,i} = K_{gid,f,i}^{1/AK_{f,i}}$, $\forall i \in S_{gid,f}$. Then it sets $ASK'_{S,gid,f} = (K_{gid,f}, \{K'_{gid,f,i}\}_{i \in S_{gid,f}})$.

PreDecrypt : The Data consumer sends $ASK'_{S,gid,f} = (K_{gid,f}, \{K'_{gid,f,i}\}_{i\in S_{gid,f}})$ and $CASK_{gid} = L_{gid}$ to the cloud server and asks it to pre-decrypt the CT'. Policy Decision Point computes $K2 = \Pi_{f \in 1,2} K_{gid,f}$ and constants $y_x \in \mathbb{Z}_N$, such that $\Sigma_{\rho(x)\in S_{gid}}(y_x \cdot W_x) = (1, 0, ..., 0)$. Then computes

$$\text{PDKEY} = \frac{e(K2, C_{01})}{\Pi_{\rho(x)\in S_{gid}}(e(C_{x1}, L_{gid}) \cdot e(D'_{x1}, K'_{\rho(x)}))^{y_x}} = \frac{e(K2, C_{01})}{\Pi_{\rho(x)\in S_{gid}}(e(C_{x1}, L_{gid}) \cdot e(D_{x1}, K_{\rho(x)}))^{y_x}}$$
$$= e(g,g)^{\Sigma_{f=1}^2 \alpha_f \cdot s/c_{gid}}.$$

Cloud server sends PDKEY to Data consumer.

Decrypt : Data Consumer's decryptor computes $M = \frac{C_1}{(PDKEY)^{c_{gid}}}$.

Revocation of User

Suppose the attribute $i^{'} \in U_{f'}$ is revoked from some users, $AA_{f'}$ randomly choose a new $AK_{f',i} \in \mathbb{Z}_N$. Whenever a user is about to losing an attribute $i^{'} \in U_{f'}$, $AA_{f'}$ sends a new attribute group key $AK^{'}_{f',i}$ to the proxy server via a secure channel and proxy server reencrypts the encrypted data stored in the storage. Meanwhile, it also defines a new set $Gn_{f',i'}$, which denotes the minimum set of nodes whose descendant nodes cover all the unrevoked users. It then encrypts $AK^{'}_{f',i}$ by $KEK_{f',j}$ for each $j \in Gn_{f',i'}$. Finally, it sends the encrypted $\{AK^{'}_{f',i}\}_{KEK_{f',j}}$ to the unrevoked users who is labeled by the node that is the descendant of j. After receiving $\{AK^{'}_{f',i}\}_{KEK_{f',j}}$, the user recovers the new attribute group key $AK^{'}_{f',i}$ and use it for further processes.

7.1.3 Data Storing

In this section I am presenting data storing Architecture of my model with step by step explanation.

Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point

1. Data Owner sends his/her Global ID (gid) to the External IAM.
2. External IAM sends a token to Data Owner after verifying all of its attributes.
3. Data Owner sends the message M (here it is basically AES encryption key) and the access policy $\mathbb{A} = (W, \rho)$, where W is a LSSS matrix with l rows and n columns, and ρ associates each row W_x to attribute $\rho(x)$ to the encryptor. In my model I forcefully assume that **Data Owner's Access Policy must include attributes of both Attribute Authorities**.
4. Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Encryptor uses those public keys and encrypts M under $\mathbb{A}_1$ using **Encrypt** algorithm and produce CT.
5. Encryptor Sends $CT = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]})$ to the Data Owner.
6. Data Owner sends this $CT = (\mathbb{A}, C_1, C_{01}, \{C_{x1}, D_{x1}\}_{x\in[l]})$ to the Data Gateway along with its own token.
7. Data Gateway take $RD = CT$. Now it does following things (as basic model) on RD :
   (a) Chooses a random key RK and encrypts RD with RK and gets ED.
   (b) Then also encrypts RK with its some special secret key and get EK.
8. Data Gateway sends EK with Data Owners token to the Control Interface.
9. Control Interface sends EK to the KWS and token to the Policy Decision Point.

10. Policy Decision Point sends another access policy $\mathbb{A}_2 = (A^{'}, \rho^{'})$ (where A' has l' many rows) to the KWS based on data owner's token.
11. KWS runs the **Encrypt** algorithm on EK under the access policy $\mathbb{A}_2 = (A^{'}, \rho^{'})$ and get the wrapped key (doubly encrypted encryption key) $PWK = (\mathbb{A}^{'}, C_2, C_{02}, \{C_{x2}, D_{x2}\}_{x \in [l']})$.
12. KWS sends PWK to Control Interface.
13. Control Interface sends PWK to the Data Gateway.
14. Data Gateway sends PWK to Proxy Server.
15. Proxy Server computes $WK = (\mathbb{A}^{'}, C_2, C_{02}, \{C_{x2}, D^{'}_{x2}\}_{x \in [l']})$ and sends WK to Data Gateway.
16. Data Gateway sends ED and WK to the Storage.

7.1.4 Data Consuming

In this section I am presenting data consuming Architecture of my model with step by step explanation.

Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point.

1. Data Consumer sends Global ID (gid) to the External IAM.
2. External IAM sends token, $CAPK_{gid}, CASK_{gid}, DSK_{gid}, ASK_{S,gid,1} = (K_{gid,1}, \{K_{gid,1,i}\}_{i \in S_{gid,1}})$ to the Data Consumer.
3. Data Consumer transform its key and sends $ASK^{'}_{S,gid,1} = (K_{gid,1}, \{K^{'}_{gid,1,i}\}_{i \in S_{gid,1}})$, token, $CASK_{gid}$ and $CAPK_{gid}$ to the Data Gateway.
4. Data Gateway collects ED and WK from the Storage, where $WK = (\mathbb{A}^{'}, C_2, C_{02}, \{C_{x2}, D^{'}_{x2}\}_{x \in [l']})$.
5. Data Gateway sends WK, $CAPK_{gid}, CASK_{gid}, ASK^{'}_{S,gid,1}$ and consumer's token to Control Interface of Access Control Server.
6. Control Interface sends WK, $ASK^{'}_{S,gid,1}$ to KWS and sends $CAPK_{gid}$, consumer's token to Policy Decision Point.
7. Policy Decision Point runs **AAKeyGen** and sends $ASK_{S,gid,2}$ to control interface.
8. Control Interface sends $ASK_{S,gid,2}$ to the Data Gateway.
9. Data Gateway sends $ASK_{S,gid,2}$ to the Data Consumer.
10. Data Consumer transforms $ASK_{S,gid,2}$ into $ASK^{'}_{S,gid,2}$.
11. Data Consumer sends $ASK^{'}_{S,gid,2}$, $ASK_{S,gid,1}$ to Data Gateway.
12. Data Gateway sends $ASK^{'}_{S,gid,2}$ to Control Interface of Access Control Server.

13. Control Interface sends $ASK^{'}_{S,gid,2}$ to KWS.

14. KWS runs **PreDecrypt** on WK with $ASK^{'}_{S,gid,1}, ASK^{'}_{S,gid,2}, CASK_{gid}$ and computes $EK^{'}$ = PreDecryption key (PDKEY) for decrypting WK.

15. KWS sends $EK^{'}$ to Control Interface.

16. Control Interface sends $EK^{'}$ to the Data Gateway.

17. Data Gateway sends $EK^{'}, C_2$ to the Data Consumer.

18. Data Consumer sends $EK^{'}, C_2, DSK_{gid}$ to its Decryptor.

19. Decrytor runs **Decrypt** on $EK^{'}, C_2, DSK_{gid}$ (with PDKEY $= EK^{'}$) and get EK.

20. Decryptor sends EK to Data Consumer.

21. Data Consumer sends EK to Data Gateway.

22. Data Gateway decrypts EK and gets RK, and after that using that RK decrypts ED and gets $RD = CT^{'}$.

23. Data Gateway runs **PreDecrypt** with $CT^{'}$, $ASK_{S,gid,1}$, $ASK_{S,gid,2}$ and $CASK_{gid}$ as input and produce $CT^{''}$ = PreDecryption key (PDKEY) for decrypting CT..

24. Data Gateway sends $CT^{''}$ and C_1 to Data Consumer.

25. Data consumer sends $CT^{''}$, C_1 and $DSK_{gid} = c_{gid}$ to its Decryptor.

26. Decryptor runs **Decrypt** algorithm on $CT^{''}$, C_1 and $DSK_{gid} = c_{gid}$ (with PDKEY $= CT^{''}$) as input and produce M.

27. Decryptor sends M to Data Consumer.

7.1.5 Security Analysis

It is adaptively secure and collusion resistant (see Section 3.6). It also has forward security.

Forward secrecy : When an attribute is revoked from a user at some time instance, the corresponding $AK_{f,i}$ will be updated and transmitted to the unrevoked users. Meanwhile, the relevant ciphertext components are also re-encrypted under the new $AK_{f,i}$ by the cloud server. In this way, the ciphertext associated with new attribute group key cannot be decrypted by the private keys labeled by old $AK_{f,i}$. Thus, forward secrecy is guaranteed.

7.2 Second Model with Revocation

7.2.1 Modified Components and Their Uses

In this modified version (which I am going to describe) I add one extra component and add some more responsibilities & functionalities to the components. The other components are same as Second Model (Decentralized Multi-Authority Model).

External IAM : In this model External IAM serves as one of the two Multi-Authorities

besides its basic works.
This component has to be **Semi-Honest** and it **can not collude with Policy Decision Point** (which is another Attribute Authority in my model).
It basically verify the attributes based on the global id of the users and gives a **token** corresponding to their id (basic model functionality) and one **Authority Attribute Secret Key** based on the user's attributes which are handled by it (work as AA).

Policy Decision Point : In this model Policy Decision Point serves also as another Attribute Authority besides its basic work.
This compoent must be **Semi-Honest** and have to handle some attributes of its Access Policy (since for Multi-Authority CPABE encryption we have only two authorities and if one of them only containts all attributes of the Access Policy then it will be no more secure).
It basically publish the revocation list, produce one another access policy $\mathbb{A}_2$ for key wrapping and computes the authority attribute secret key for Data Consumer and all the **Pre-Decryption** operations in cloud.

Data Owner : He/she first encrypts the raw data with the two Attribute Public Key under his/her Access Policy $\mathbb{A}_1$.

Data Consumer : Data Consumer use his/her own **Decryption Secret Key** to Decrypt the Pre-Decryted data.

Data Gateway : It does all its basic works on encrypted data in stead of the raw data and it unwraps the wrapped key. But besides these it also does pre-decryption of the user's message. As Basic Model, it is also **Semi-Honest** in my model.

Key Wrapping Server (KWS) : It wraps the encrypted key and also unwraps the wrapped key using MA-CPABE and it is also **Semi-Honest**.

Proxy Server : We introduce a Proxy server which is basically in cloud and it stores all necessary keys related to revocation and also update cipher texts properly. It is also **Semi-Honest and can't colude with any Revoked User ever**.

7.2.2 Fuctionalities used in the System

Global Setup : Let $\mathbb{G}$ and $\mathbb{G}_1$ denote two bilinear groups of prime order p. Let $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ denote a bilinear map. Choose a generator g of the group $\mathbb{G}$. Also computes $e(g, g)$. After that choose a collision resistant hash function $H : \{GID\} \mapsto \mathbb{G}$. Set global parameter GP = $\{g, p, e, H, e(g, g)\}$.

Authority Setup : AA_f chooses $\alpha_i, y_i, \beta_i, z_i \in \mathbb{Z}_p\ \forall i \in U_f$ (U_f is the set of attributes handled by AA_f). Then it computes $e(g, g)^{\alpha_i}, g^{y_i}, e(g, g)^{\beta_i}, g^{z_i}\ \forall i \in U_f$. It sets $PK_f = \{(e(g, g)^{\alpha_i}, g^{y_i})\}_{i \in U_f}$, $PK_f^{'} = \{e(g, g)^{\beta_i}, g^{z_i}\}_{i \in U_f}$ as public keys and sets $MSK_f = \{\alpha_i, y_i\}_{i \in U_f}$, $MSK_f^{'} = \{\beta_i, z_i\}_{i \in U_f}$ as secret keys.
Additionally, each AA_f also maintains a binary tree $TREE_f$. In $TREE_f$, each node j is associated with a different encryption key $KEK_{f,j}$ and each leaf node is labeled by a user (gid). Such a tree with height h can accommodate at most 2^h users. Moreover, there is a path P_j from j to the root node. When a new user comes to AA_f for requesting the attribute-related keys, AA_f not only generates the requested keys, but also adds the user to the leftmost leaf node j and gives him/her the path keys in the path P_j. For each attribute $i \in U_f$, AA_f establishes an attribute-user group $G_{f,i}$, which is a set of the users who own this attribute i. We let $Gn_{f,i}$

denote the minimum set of nodes whose descendant nodes cover all the users in $G_{f,i}$. At the beginning of system initialization, each AA_f shares a unique attribute group key $AK_{f,i} \in \mathbb{Z}_N$ with the proxy server (in cloud) for each $i \in U_f$. Whenever a new user registered in the system or a revocation list is published it updates all $AK_{f,i}$.

Encrypt : Data Owner sends it raw data M and its own policy $\mathbb{A}_1 = (A, \rho)$, where A is a LSSS matrix with l rows and n columns, and ρ associates each row A_x to attribute $\rho(x)$, as input to its encryptor. After that it chooses some secret $s \in \mathbb{Z}_p$. Select two random vectors $\vec{v}, \vec{w} \in \mathbb{Z}_p^n$ such that 1st co-ordinate of $\vec{v}$ is s and 1st co-ordinate of $\vec{w}$ is 0. Now computes $\lambda_j = A_j \cdot \vec{v}$ and $w_j = A_j \cdot \vec{w}$ for all rows $j \in [l]$. Now it chooses $r_j \in \mathbb{Z}_p \; \forall j \in [l]$. Then it computes $C_0 = M \cdot e(g,g)^s$ and $\forall j \in [l]$ also computes $C_{1j} = e(g,g)^{\lambda_j} \cdot e(g,g)^{\alpha_{\rho(j)} r_j}$, $C_{2j} = g^{r_j}$, $C_{3j} = g^{y_j r_j} \cdot g^{w_j}$. Finally sets $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$.

ReEncrypt : It is same as basic model.

KeyEncryption : It is same as basic model.

Wrap : KWS takes encrypted encryption key EK from Data Gateway and Policy Decision Point's access policy $\mathbb{A}_2 = (A', \rho')$, where A' is a LSSS matrix with l' rows and n' columns, and ρ' associates each row A'_x to attribute $\rho'(x)$, as input. After that it chooses some secret $s' \in \mathbb{Z}_p$. Select two random vectors $\vec{v'}, \vec{w'} \in \mathbb{Z}_p^{n'}$ such that 1st co-ordinate of $\vec{v'}$ is s' and 1st co-ordinate of $\vec{w'}$ is 0. Now computes $\lambda'_j = A'_j \cdot \vec{v'}$ and $w'_j = A'_j \cdot \vec{w'}$ for all rows $j \in [l']$. Now it chooses $r'_j \in \mathbb{Z}_p \; \forall j \in [l']$. Then it computes $D_0 = EK \cdot e(g,g)^{s'}$ and $\forall j \in [l']$ also computes $D_{1j} = e(g,g)^{\lambda'_j} \cdot e(g,g)^{\beta'_{\rho'(j)} r'_j}$, $D_{2j} = g^{r'_j}$, $D_{3j} = g^{z_j r'_j} \cdot g^{w'_j}$. Finally sets $PWK = (D_0, \{D_{1j}, D_{2j}, D_{3j}\}_{j \in [l']})$.
After encryption, at the time of data storing $PWK = (D_0, \{D_{1j}, D_{2j}, D_{3j}\}_{j \in [l']})$ is sent to proxy server and proxy server computes $D'_{3j} = D_{3j}^{AK_{\rho(j)}}$ and sets $WK = (D_0, \{D_{1j}, D_{2j}, D'_{3j}\}_{j \in [l']})$ and sends WK for storing.

AAKeyGen1 : Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter MSK_f as input. Then it computes $K_{f,GID} = \{K_{i,GID}\}_{i \in S_{GID,f}} = \{g^{\alpha_i} \cdot H(GID)^{y_i}\}_{i \in S_{GID,f}}$. This the secret attribute key, which is used for Message decryption.
In addition, for each unrevocked attribute $i \in S_{gid,f}$, the AA_f sets the set $G_{f,i}$ and $Gn_{f,i}$. It then encrypts $AK_{f,i}$ by $KEK_{f,j}$ if $j \in Gn_{f,i}$ is the ancestor node of the leaf node which the user is associated with. It finally sends $K_{f,GID}$ and the encrypted $\{AK_{f,i} | \; unrevocked \; i \in S_{gid,f}\}$ to the user.

AAKeyGen2 : Authority AA_f (which is indexed by f) takes global parameter GP, unique global id GID of the user, set of attributes of the user $S_{GID,f}$ which are handled by AA_f, secret parameter MSK'_f as input. Then it computes $K'_{f,GID} = \{K'_{i,GID}\}_{i \in S_{GID,f}} = \{g^{\beta_i} \cdot H(GID)^{z_i}\}_{i \in S_{GID,f}}$. This secret attribute key is used for Key Unwrapping. It finally sends $K'_{f,GID}$ to the user.

KeyTransform : Data Consumer takes the secret attribute keys $\{K_{i,GID}\}_{i \in S_{GID}}$ as input. After that it chooses $d \in \mathbb{Z}_p$, and set its own secret decryption key $DSK = d$. After that it trsforms the attribute keys as $\{T_{i,GID}\}_{i \in S_{GID}} = \{(K_{i,GID}^{\frac{1}{d}}, H(GID)^{\frac{1}{d}}\}_{i \in S_{GID}}$ and

$\{T^{'}_{i,GID}\}_{i \in S_{GID}} = \{(K^{'}_{i,GID}, H(GID)^{\frac{1}{AK_i}}\}_{i \in S_{GID}}$.

UnWrap : KWS takes wrapped key $WK = (D_0, \{D_{1j}, D_{2j}, D^{'}_{3j}\}_{j \in [l']})$ and attribute secret keys $T^{'}_{1,GID}, T^{'}_{2,GID}$ as input. Here $T^{'}_{f,GID} = \{T^{'}_{i,GID}\}_{i \in S_{GID,f}}$ Then computes constants $c^{'}_x \in \mathbb{Z}_p$, such that $\Sigma_{\rho'(x) \in S_{gid}}(c^{'}_x \cdot A^{'}{}_x) = (1, 0, ..., 0)$. Now it computes $C^{'} = \Pi^{l^{'}}_{j=1}((e(H(GID)^{\frac{1}{AK_j}}, D^{'}_{3j}) \cdot D_{1j})/(e(K^{'}_{\rho'(j),GID}, D_{2j})))^{c^{'}_j}$. Finally computes $EK = \frac{D_0}{C'}$. Then sends EK to Data Gateway.

KeyDecryption : It is same as basic model.

PartialDecryption : It is same as basic model.

PreDecrypt : Data Gateway takes cipher text $CT^{'} = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{j \in [l]})$ and transformed secret keys $T_{1,GID}, T_{2,GID}$ as input. Then computes constants $c_x \in \mathbb{Z}_p$, such that $\Sigma_{\rho(x) \in S_{gid}}(c_x \cdot A_x) = (1, 0, ..., 0)$. Here $T_{i,GID} = (K^{\frac{1}{d}}_{i,GID}, H(GID)^{\frac{1}{d}})$, $\forall i \in S_{GID}$.
Now it computes $C_1 = \Pi^{l}_{j=1}((e(H(GID)^{\frac{1}{d}}, C_{3j}))/(e(K^{\frac{1}{d}}_{\rho(j),GID}, C_{2j})))^{c_j}$ and $C_2 = \Pi^{l}_{j=1}(C_{1j})^{c_j}$. Finally it sets $CT^{''} = (C_1, C_2)$ and sends C_0 and $CT^{''}$ to Data Consumer.

Decrypt : Data consumer sends $C_0, CT^{''} = (C_1, C_2)$ and own decryption secret key $DSK = d$ to its Decryptor as input. Then it computes $C^{''} = ((C_2)^{\frac{1}{d}} \cdot C_1)^d$. Finally generates $M = C_0/C^{''}$.

Revocation of User

Suppose the attribute $i^{'} \in U_{f'}$ is revoked from some users, $AA_{f'}$ randomly choose a new $AK_{f',i} \in \mathbb{Z}_N$. Whenever a user is about to losing an attribute $i^{'} \in U_{f'}$, $AA_{f'}$ sends a new attribute group key $AK^{'}_{f',i}$ to the proxy server via a secure channel and proxy server reencrypts the encrypted data stored in the storage. Meanwhile, it also defines a new set $Gn_{f',i'}$, which denotes the minimum set of nodes whose descendant nodes cover all the unrevoked users. It then encrypts $AK^{'}_{f',i}$ by $KEK_{f',j}$ for each $j \in Gn_{f',i'}$. Finally, it sends the encrypted $\{AK^{'}_{f',i}\}_{KEK_{f',j}}$ to the unrevoked users who is labeled by the node that is the descendant of j. After receiving $\{AK^{'}_{f',i}\}_{KEK_{f',j}}$, the user recovers the new attribute group key $AK^{'}_{f',i}$ and use it for further processes.

7.2.3 Data Storing

In this section I am presenting data storing Architecture of my model with step by step explanation.

Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point

1. Data Owner sends his/her Global ID (gid) to the External IAM.

2. External IAM sends a token to Data Owner after verifying all of its attributes.
3. Data Owner sends the message M (here it is basically AES encryption key) and the access policy $\mathbb{A}_1 = (A, \rho)$ to its encrypter. In my model I forcefully assume that **Data Owner's Access Policy must include attributes of both Attribute Authorities**.
4. Each Attribute Authority (External IAM, Policy Decision Point) has broadcasted their Public Keys previously. Encryptor uses those public keys and encrypts M under $\mathbb{A}_1$ using **Encrypt** algorithm and produce CT.
5. Encryptor Sends CT to the Data Owner.
6. Data Owner sends this CT to the Data Gateway along with its own token.
7. Data Gateway take $RD = CT$. Now it does following things (as basic model) on RD :
   (a) Chooses a random key RK and encrypts RD with RK and gets ED.
   (b) Then also encrypts RK with its some special secret key and get EK.
8. Data Gateway sends EK with Data Owner's token to the Control Interface.
9. Control Interface sends EK to the KWS and token to the Policy Decision Point.
10. Policy Decision Point sends another access policy $\mathbb{A}_2 = (A^{'}, \rho^{'})$ to the KWS based on data owner's token.
11. KWS runs the **Wrap** algorithm on EK and get the wrapped key (doubly encrypted encryption key) $PWK = (D_0, \{D_{1j}, D_{2j}, D_{3j}\}_{j\in[l']})$.
12. KWS sends PWK to Control Interface.
13. Control Interface sends PWK to the Data Gateway.
14. Data Gateway sends PWK to Proxy Server.
15. Proxy Server computes $WK = (D_0, \{D_{1j}, D_{2j}, D^{'}_{3j}\}_{j\in[l']})$ and sends WK to Data Gateway.
16. Data Gateway sends ED and WK to the Storage.

7.2.4 Data Consuming

In this section I am presenting data consuming Architecture of my model with step by step explanation.

Architecture Explained Step by Step

Here I assume AA_1 = External IAM, AA_2 = Policy Decision Point.

1. Data Consumer sends Global ID (gid) to the External IAM.
2. External IAM runs **AAKeyGen1, AAKeyGen2** and sends token, $K_{1,GID}, K^{'}_{1,GID}$ to Data Consumer.
3. Data Consumer sends token to the Data Gateway.

4. Data Gateway collects ED and WK from the Storage.
5. Data Gateway sends WK and consumer's token to Control Interface of Access Control Server.
6. Control Interface sends WK to KWS and sends consumer's token to Policy Decision Point.
7. Policy Decision Point runs **AAKeyGen1, AAKeyGen2** and sends $K_{2,GID}$ and $K'_{2,GID}$ to control interface.
8. Control Interface sends $K_{2,GID}$, $K'_{2,GID}$ to the Data Gateway.
9. Data Gateway sends $K_{2,GID}$, $K'_{2,GID}$ to Data Consumer.
10. Data Consumer runs **KeyTransform** on $\{K_{i,GID}\}_{i\in S_{GID}} = \{K_{1,GID}, K_{2,GID}\}$ and computes $\{T_{i,GID}\}_{i\in S_{GID}}$ and DSK. After that Data Consumer runs **KeyTransform** on $\{K'_{i,GID}\}_{i\in S_{GID}} = \{K'_{1,GID}, K'_{2,GID}\}$ and computes $\{T'_{i,GID}\}_{i\in S_{GID}}$
11. Data Consumer sends $\{T_{i,GID}\}_{i\in S_{GID}}$ and $\{T'_{i,GID}\}_{i\in S_{GID}}$ to Data Gateway.
12. Data Gateway sends $\{T'_{i,GID}\}_{i\in S_{GID}}$ to Control Interface.
13. Control Interface sends $\{T'_{i,GID}\}_{i\in S_{GID}}$ to KWS
14. KWS runs **UnWrap** and computes unwrapped key EK.
15. KWS sends EK to Control Interface.
16. Control Interface sends EK to Data Gateway.
17. Data Gateway decrypts EK and gets RK, and after that using that RK decrypts ED and gets $RD = CT'$.
18. Data Gateway runs **PreDecrypt** on CT' and computes CT''.
19. Data Gateway sends CT'', C_0 to Data Consumer.
20. Data consumer sends CT'', C_0 and DSK to its Decryptor.
21. Decryptor runs **Decrypt** algorithm with CT'', C_0 and DSK as input and produce M.
22. Decryptor sends M to Data Consumer.

7.2.5 Security Analysis

It is CPA secure and collusion resistant (see Section 4.6). It also has forward security (by the same argument as Subsection 7.1.5).

Chapter 8

Idea of Implementing whole Access Control Server

I was thinking that we can divide this full Access Control system in three phases : Initial Setup, Key Updation, Data Accessing.

1. **Initial Setup** : Whenever a new user (U) register into this system :
   (a) U gets a global unique ID and It will send this to External IAM.
   (b) After reciving the global unique ID of U, External IAM will send the token and the all keys corresponding to External IAM to U.
   (c) U will send that token and relevant keys (if it is needed) to Policy Decision Point through Data Gateway and Control Interface.
   (d) Policy Decision Point sends all the keys to U through Control Interface and Data Gateway.
2. **Key Updation** : Whenever a revocation list will be published :
   (a) Authorities will refresh their revocation keys.
   (b) Authorities will update those revocation keys in proxy server also.
   (c) U will ask for updated revocation keys corresponding to its unrevocked attributes.
   (d) U gets updated revocation keys corresponding to its unrevocked attributes (following the steps from 1.(b) to 1.(d)).
3. **Data Accessing** : At the time of data storing :
   (a) DO gets its token from External IAM and logs in to the system.
   (b) DO just encrypt its data and send it to Data Gateway. Rest of the part will be same (as I described in Subsection 7.2.3).

   At the time of Data Consuming :

   (a) DC gets its token from External IAM and logs in to the system.
   (b) DC sends transformed keys directly along with its other necessary keys to Data Gateway. Rest of the part will be same (as I described in Subsection 7.2.4, from step-12 to step-22).

Now I will describe my idea in details. Suppose we have an access control system (ACS).

Initial Setup : Now whenever a new user (U) will register in that system ACS, U produces all of its identity proof to the External IAM and gets a global unique ID (GID) and a token as a proof of its identity. This token will basicaaly have the information of all attributes of the user. This U may be DO or DC. External IAM also update its revocation tree (where actually it stores the keys related to the revocation process) and hands over all of his secret keys related with the authority External IAM. After that U sends his token, GID and all other neccessary informations to Policy Decision Point, through Data Gateway and Control Interface, to get all of his secret keys related with the authority Policy Decision Point and gets the required secret keys.

Key Updation : Whenever External IAM detects some malicious user, it revokes that user fully or revokes some of his attributes and publishes an revocation List. After publication of the revocation list the authorities updates all of their revocation related keys and also sends them to the proxy server (in our revocation model). Proxy server updates its storage where it stores all the keys related to revocation. After that Proxy server also access the cipher texts, strored in the cloud storage, which needs an update and updates those cipher-texts with new revocation keys. Here at the time of updation proxy server basically does re-encryption only on those ciphertexts.
All the users of the systems time to time check whether a revocation list has been published or not. If they notice that a revocation list has been published, they asks the authorities for new updated keys. They updates their storage with the updated keys.

Data Accessing : Since all the users have updated keys, at the time of data storing or consuming they do not ask the authorities for keys.
At the time of Data storing, the data owner (DO) just encrypts necessary data with the keys which he already has in his personal storage and sends the encrypted data to Data Gateway for further processing.
At the time of Data consuming, data consumer (DC) sends his secret transformed keys to Data Gateway which are neccessary to pre-decrypt the data. DC don't asks the authorities for secret keys as he already has all the keys in his personal storage. After the full cloud computation Data Gateway sends the partially decrypted data to DC and DC easily computes the raw data using his own decryption secret key.

If we divide the full access control structure in this way, then it will help to reduce the communication costs. Because here the DC is not asking for the keys to the attribute authorities at the time of each data consumption. Also if you see the Subsection 7.1.4 and Subsection 7.2.4, then you can observe that at first DC is asking for all keys, then after getting the keys DC transforms those keys and again sends those transformed keys to the Data Gateway. But here since the DC already has all the keys, he just sends the final transformed keys directly to the Data Gateway. So, the number of communications will be reduced.

Chapter 9

Conclusion

In this thesis I have studied the recent Token-Based access control model and found the security limitation in it. I have studied many constructions of ABE, CP-ABE, MA-CP-ABE, LSSS and then I have choosen suitable building blocks to design two access control model which can avoid those security limitations.

My first model is based on centralized MA-CP-ABE. In the first model I have used composite order bilinear group. Since it is a centralized model, there is an central authority. In my case External IAM plays the role of Central Authority. I have used External IAM and Policy Decision Point as my two attribute authorities. According to the security analysis, my first model is adaptively secure. We have done this security analysis in standard model.
My second model is based on decentralized MA-CP-ABE. For my second model I have used prime order bilinear group. Since it is an decentralized scheme so there is no central authority. Here also I have used External IAM and Policy Decision Point as my two attribute authorities. According to the security analysis, my second model is CPA secure. We have done this security analysis in random oracle model.
My second model is more efficient according to the computation cost than the first model whereas my first model is more efficient according to the communication cost than the second model. Also my first model has adaptive security in standard model whereas we have used random oracle model for the second model.

Then I have implemented the CP-ABE scheme of my Second model. I have also implemented a more efficient form of LSSS matrix generation and used that to make my scheme more efficient. I have only implemented the Second Multi-Authority CP-ABE scheme i.e. Deentralized Multi-Authority CP-ABE scheme because this is the efficient one (according to computation cost). Actually the first scheme i.e. the Centralized Multi-Authority CP-ABE scheme is based on composite order bilinear group which are several orders of magnitude slower than the prime order groups that provide the same security level. Another important reason of not implementing my first scheme is Charm does not support composite order groups.
Although my first scheme takes more computation time than my second scheme, my first scheme is more secure than my second scheme, it is adaptively secure in standard model. Also my first scheme needs less communication cost.

I have also introduced revocation property in the models to penalize the malicious users. But for this introduction of revocation property needs additional overhead. It also increases the computation and communication costs of the model. But to decrease the communication cost we have given an implementation idea for the whole access control model. For future work one can think about introducing accountability property or policy updating property in the models.

Bibliography

[1] “Attribute-based Encryption for Cloud Computing Access Control: A Survey" by YINGHUI ZHANG, ROBERT H. DENG, SHENGMIN XU, JIANFEI SUN, QI LI and DONG ZHENG.

[2] “Fuzzy Identity-Based Encryption" by Amit Sahai and Brent Waters.

[3] “Ciphertext-Policy Attribute-Based Encryption" by John Bethencourt, Amit Sahai and Brent Waters.

[4] “Secure Schemes for Secret Sharing and Key Distribution" by Amos Beimel.

[5] “Efficient Generation of Linear Secret Sharing Scheme Matrices from Threshold Access Trees" by Zhen Liu, Zhenfu Cao, and Duncan S. Wong.

[6] “New Monotone Span Programs from Old" by Ventzislav Nikov and Svetla Nikova.

[7] “Computationally Efficient Ciphertext-Policy Attribute-Based Encryption with Constant-Size Ciphertexts" by Yinghui Zhang, Dong Zheng, Xiaofeng Chen, Jin Li and Hui Li.

[8] “Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization" by Brent Waters.

[9] “Multi-authority Attribute Based Encryption" by Melissa Chase.

[10] “Decentralizing Attribute-Based Encryption" by Allison Lewko and Brent Waters.

[11] “Secure, efficient and revocable multi-authority access control system in cloud storage" by Qi Li, Jianfeng Ma, Rui Li, Ximeng Liu, Jinbo Xiong and Danwei Chen.

[12] “Efficient Decentralized Attribute Based Access Control for Mobile Clouds" by Sourya Joyee De and Sushmita Ruj.

[13] “Improving Privacy and Security in Multi-Authority Attribute-Based Encryption" by Melissa Chase and Sherman S.M. Chow.

[14] “Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves" by Aurore Guillevic.

[15] “Guide to Attribute Based Access Control (ABAC) Definition and Considerations" by Vincent C. Hu, David Ferraiolo, Rick Kuhn, Arthur R. Friedman, Alan J. Lang, Margaret M. Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone. NIST Special Publication 800-162.

[16] “The stanford pairing based crypto library" by Ben Lynn.

[17] “Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption" by Yannis Rouselakis and Brent Waters.

[18] “Charm: A framework for rapidly prototyping cryptosystems" by Joseph A. Akinyele, Matthew Green, and Avi Rubin. Cryptology ePrint Archive, Report 2011/617, 2011.

[19] Miracl crypto sdk. https://certivox.com/solutions/miracl-crypto-sdk/.

[20] “Token Based Authorization" by Giovanni Augusto Baruzzi.

[21] “Role-Based Access Control" by Ferraiolo D.F., Kuhn D.R.. 15th National Computer Security Conference: 554–563, 1992.

[22] “ACLA: A Framework for Access Control List (ACL) Analysis and Optimization" by Jiang Qian, Susan Hinrichs, and Klara Nahrstedt.

[23] https://www.securelink.com/blog/healthcare-data-new-prize-hackers/

Appendix A

Security Proof of Centralized Multi-authority CP-ABE (First) Model

Before giving out the proof, we have to introduce the definitions of two additional but necessary structures: semi-functional ciphertext (SF-CT) and semi-functional key (SF-Key), which will not be employed in the real constructions, but are necessary in the proof. For each attribute $i \in U_f$, we pick a random exponent $z_{f,i} \in \mathbb{Z}_N$.

Semi-functional ciphertext (SF-CT) : A SF-CT is formed in the following way. We let g_2 be a generator of G_{p_2}, c be a random chosen exponent from $\mathbb{Z}_N$. For each row $x \in [l]$, we randomly select $\gamma_x \in \mathbb{Z}_N$. In addition, we choose a random vector $\vec{y} \in \mathbb{Z}_N^n$. Then, we set $C_0 = g^s g_2^c$. For each $x \in [l]$:

$$C_x = g^{\Sigma_{f \in \{1,2\}} a_f \cdot W_x \cdot \vec{v}} \cdot T_{\rho(x)}^{-r_x} \cdot g_2^{W_x \cdot \vec{y} + \gamma_x z_{\rho(x)}}$$
$$D_x = g^{r_x} \cdot g_2^{-\gamma_x}$$

Semi-functional key (SF-Key) : For a gid, a SF-Key will be one of the two following forms:

- SF-Key of **type 1** : We pick random exponents $b, d_f \in \mathbb{Z}_N$ and set :

$$L_{gid} = g^{b_{gid}/c_{gid}} \cdot R_{gid} \cdot g_2^b$$
$$Lgid, 0 = g^{1/c_{gid}} \cdot R_{gid,0}$$
$$CAPK_{gid} = (gid, CASK_{gid}, L_{gid,0}, \sigma_{gid})$$
$$DSK_{gid} = c_{gid}$$
$$K_{gid,f} = g^{\alpha_f/c_{gid}} \cdot g^{a_f b_{gid}/c_{gid}} \cdot R_{gid,f} \cdot g_2^{d_f}$$
$$K_{gid,f,i} = T_{f,i}^{b_{gid}/c_{gid}} \cdot R_{gid,f,i} \cdot g_2^{bz_{f,i}}$$

- SF-Key of **type 2** : We pick random exponents $b, d_f \in \mathbb{Z}_N$ and set :

$$L_{gid} = g^{b_{gid}/c_{gid}} \cdot R_{gid}$$
$$Lgid, 0 = g^{1/c_{gid}} \cdot R_{gid,0}$$
$$CAPK_{gid} = (gid, CASK_{gid}, L_{gid,0}, \sigma_{gid})$$
$$DSK_{gid} = c_{gid}$$
$$K_{gid,f} = g^{\alpha_f/c_{gid}} \cdot g^{a_f b_{gid}/c_{gid}} \cdot R_{gid,f} \cdot g_2^{d_f}$$
$$K_{gid,f,i} = T_{f,i}^{b_{gid}/c_{gid}} \cdot R_{gid,f,i}$$

Observe that, if we use an SF-Key to decrypt a normal ciphertext or use a normal key to decrypt an SF-CT, $\Pi_{f\in\{1,2\}}e(g,g)^{\alpha_f s}$ can be correctly computed. However, if we attempt to employ an SF-Key to decrypt an SF-CT, it will come out an additional term : $e(g_2,g_2)^{c\Sigma_{f\in\{1,2\}}d_f - by_1}$, where y_1 is the first coordinate of $\vec{y}$. An SF-Key of type 1 is said to be nominal if $c\Sigma_{f\in\{1,2\}}d_f - by_1 = 0$. In this case, such an SF-key can decrypt the corresponding SF-CT.

Before going into the proof, we shall discuss about some security assumptions which will be used to proof that the first centralized MA-CP-ABE scheme is adaptively secure.

- **Assumption - 1** : Let $\mathcal{G}$ denote a group generator, which produces the group description $G = (\mathbb{G}, \mathbb{G}_1, N = p_1p_2p_3, e)$. Given the terms $D = (G, g, \tau_3)$, where $g \xleftarrow{R} G_{p_1}$ and $\tau_3 \xleftarrow{R} G_{p_3}$, the adversary has to distinguish the element $\phi_1 \xleftarrow{R} G_{p_1p_2}$ from $\phi_2 \xleftarrow{R} G_{p_1}$.
  The advantage with which an adversary $\mathcal{A}$ can break Assumption 1 is defined as :
  $Adv1_{\mathcal{G},\mathcal{A}} = |Pr[\mathcal{A}(D,\phi_1) = 1] - Pr[\mathcal{A}(D,\phi_2) = 1]|$
- **Definition - 1** : $\mathcal{G}$ satisfies Assumption 1 if the advantage $Adv1_{\mathcal{G},\mathcal{A}}$ is negligible for any PPT adversary $\mathcal{A}$.

- **Assumption - 2** : Let $\mathcal{G}$ denote a group generator, which produces the group description $G = (\mathbb{G}, \mathbb{G}_1, N = p_1p_2p_3, e)$. Given the terms $D = (G, g, \tau_1, \tau_2, \tau_3, \theta_2, \theta_3)$, where $g, \tau_1 \xleftarrow{R} G_{p_1}$, $\tau_2, \theta_2 \xleftarrow{R} G_{p_2}$ and $\tau_3, \theta_3 \xleftarrow{R} G_{p_3}$, the adversary has to distinguish the element $\phi_1 \xleftarrow{R} G$ from $\phi_2 \xleftarrow{R} G_{p_1p_3}$.
  The advantage with which an adversary $\mathcal{A}$ can break Assumption 2 is defined as :
  $Adv2_{\mathcal{G},\mathcal{A}} = |Pr[\mathcal{A}(D,\phi_1) = 1] - Pr[\mathcal{A}(D,\phi_2) = 1]|$
- **Definition - 2** : $\mathcal{G}$ satisfies Assumption 2 if the advantage $Adv2_{\mathcal{G},\mathcal{A}}$ is negligible for any PPT adversary $\mathcal{A}$.

- **Assumption - 3** : Let $\mathcal{G}$ denote a group generator, which produces the group description $G = (\mathbb{G}, \mathbb{G}_1, N = p_1p_2p_3, e)$. Given the terms $D = (G, g, g^\alpha\tau_2, \tau_3, g^s\theta_2, r_2)$, where $g \xleftarrow{R} G_{p_1}$, $\tau_2, \theta_2, r_2 \xleftarrow{R} G_{p_2}$ and $\tau_3 \xleftarrow{R} G_{p_3}$, the adversary has to distinguish the element $\phi_1 = e(g,g)^{\alpha s}$ from $\phi_2 \xleftarrow{R} G_1$.
  The advantage with which an adversary $\mathcal{A}$ can break Assumption 3 is defined as :
  $Adv3_{\mathcal{G},\mathcal{A}} = |Pr[\mathcal{A}(D,\phi_1) = 1] - Pr[\mathcal{A}(D,\phi_2) = 1]|$
- **Definition - 3** : $\mathcal{G}$ satisfies Assumption 3 if the advantage $Adv3_{\mathcal{G},\mathcal{A}}$ is negligible for any PPT adversary $\mathcal{A}$.

Now we are going to introduce the security game run between an adversary $\mathcal{A}$ and a simulator $\mathcal{B}$. If total number of AAs is F, then $\mathcal{A}$ is assumed to corrupt at most $F-1$ AAs (In my model $F = 2$). We let $\mathbb{F}_c$, $\mathbb{F}_{uc} = \mathbb{F} - \mathbb{F}_c$ denote the index set of corrupted and uncorrupted AAs, respectively.

Security Game :

- **Setup** : The simulator $\mathcal{B}$ runs the **GlobalSetup, CASetup** and **AASetup** algorithms. It then transmits the system public parameters GPK, CPK, and $\cup_{f=1}^{F} APK_f$ to the adversary $\mathcal{A}$. $\mathcal{A}$ appoints an index set of the AAs $\mathbb{F}_c$ which it wants to corrupt, where $\mathbb{F} - \mathbb{F}_c \neq \phi$. For $f \in \mathbb{F}_c$, $\mathcal{B}$ sends the master key $\{AMK_f | f \in \mathbb{F}_c\}$ to $\mathcal{A}$.

- **Phase-1** : The adversary can make adaptive secret key queries as follows:
  CAkey queries : To answer these queries, $\mathcal{B}$ responds by DSK_{gid}, $CASK_{gid}$ and $CAPK_{gid}$.
  AAkey queries : The adversary makes AAkey queries by submitting $\cup S_{gid,f}$ and $CAPK_{gid}$ to $\mathcal{B}$, where $f \in \mathbb{F}_{uc}$. $\mathcal{B}$ returns $\{ASK_{S_{gid,f}}\}_{f \in \mathbb{F}_{uc}}$.
- **Challenge** : The adversary declares two equal-length message M_0, M_1 and a challenge access structure $\mathbb{A}^*$. $\mathcal{B}$ first flips a random coin and choose $b \in \{0,1\}$. It then encrypts M_b under $\mathbb{A}^*$ and gets the challenge ciphertext CT^*. It gives CT^* to $\mathcal{A}$.
- **Phase-2** : The adversary $\mathcal{A}$ can make adaptive secret key queries as in Phase-1.
- **Guess** : $\mathcal{A}$ outputs its guess b' of b.

We note that the adversary $\mathcal{A}$ cannot make AAkey queries on the attribute set $S_{gid,f}$ such that $(\cup_{f \in \mathbb{F}_{uc}} S_{gid,f}) \cup (\cup_{f \in \mathbb{F}_c} U_f)$ can satisfy the challenge access structure $\mathbb{A}^*$. The advantage of $\mathcal{A}$ is defined as $(Pr[b' = b] - 1/2)$.

Now to prove the adaptive security of our centralized MA-CP-ABE scheme from Assumptions 1, 2, 3, a sequence of games are used. The detailed definitions are given in the following :

- **Game$_{Real}$** : The first game Game$_{Real}$ denotes the real security game. The challenge ciphertext and all users' keys are normal.
- **Game$_0$** : In this game, all users' keys are normal, but the challenge ciphertext is semi-functional.

We let q be the number of key queries that are requested by $\mathcal{A}$. For k from 1 to q, we consider :

- **Game$_{k,1}$** : In this game, the first $k-1$ keys are semifunctional form of type 2. The k-th key is semi-functional form of type 1. The remaining keys are normal.
- **Game$_{k,2}$** : In this game, the first k keys are semi-functional form of type 2. The other keys are normal form.
- **Game$_{Final}$** : In this game, all the keys are semi-functional form of type 2. Different from Game$_{k,2}$, the semi-functional challenge ciphertext is an encryption of a random (unknown) message, which is independent of M_0 and M_1.

To proof the that our centralized MA-CP-ABE is adaptively secure we need the following lemmas. These lemmas can be proved using the assumptions. You can see the paper [11] for the detail proof.

- **Lemma - 1** : Given a UF-CMA (unforgeable under adaptive chosen message attacks) secure signature scheme, suppose that there is a PPT adversary $\mathcal{A}$ with advantage : Game$_{Real}Adv_{\mathcal{A}}$ $-$ Game$_0Adv_{\mathcal{A}} = \epsilon$, we then can construct a PPT simulator $\mathcal{B}$ to break Assumption-1 with advantage ϵ.
- **Lemma - 2** : Given a UF-CMA (unforgeable under adaptive chosen message attacks) secure signature scheme, suppose that there is a PPT adversary $\mathcal{A}$ with advantage : Game$_{k-1,2}Adv_{\mathcal{A}}$ $-$ Game$_{k,1}Adv_{\mathcal{A}} = \epsilon$, we then can employ $\mathcal{A}$ to construct a PPT simulator $\mathcal{B}$ to break Assumption-2 with advantage negligibly approximate to ϵ.

- **Lemma - 3** : Given a UF-CMA (unforgeable under adaptive chosen message attacks) secure signature scheme, suppose that there is a PPT adversary $\mathcal{A}$ with advantage : $\text{Game}_{k,1}Adv_{\mathcal{A}} - \text{Game}_{k,2}Adv_{\mathcal{A}} = \epsilon$, we then can employ $\mathcal{A}$ to construct a PPT simulator $\mathcal{B}$ to break Assumption-2 with advantage negligibly approximate to ϵ.

- **Lemma - 4** : Given a UF-CMA (unforgeable under adaptive chosen message attacks) secure signature scheme, suppose that there is a PPT adversary $\mathcal{A}$ with advantage : $\text{Game}_{q,2}Adv_{\mathcal{A}} - \text{Game}_{Final}Adv_{\mathcal{A}} = \epsilon$, we then can employ $\mathcal{A}$ to construct a PPT simulator $\mathcal{B}$ to break Assumption-3 with advantage negligibly approximate to ϵ.

Now we shall proof the following theorem :

Theorem : Suppose that the signature system is existentially unforgeable against adaptive chosen message attack (UF-CMA) and Assumptions 1, 2, 3 hold. Then no polynomial-time adversary $\mathcal{A}$ can break our MA-CP-ABE scheme with a non-negligible advantage.

Proof : If Assumptions 1, 2 and 3 hold and the signature system Σ_{sign} is UF-CMA secure, using the mentioned 4 lemmas we get that Game_{Final} is indistinguishable from the real security game. In Game_{Final}, the value of b is information-theoretically hidden from $\mathcal{A}$. Thus, the adversary $\mathcal{A}$ cannot gain a non-negligible advantage in breaking our scheme.

Appendix B

Approximate Security Level of all Charm Elliptic Curves

In the following table we present the approximate security levels of all the elliptic curves supported by Charm. The results of the table provides an intuitive comparison between the security levels of the different instantiations [17].

Curve	Security Level (bits)
SS512	80
SS1024	112
MNT159	70
MNT201	90
MNT224	100

Table B.1: Approximate security levels of the utilized ECC groups [17]

“SS" are super singular curves (symmetric bilinear groups), while “MNT" are the Miyaji, Nakabayashi, Takano curves (asymmetric bilinear groups). The number after the type of the curve denotes the size of the base field in bits.

Appendix C

Prime vs Composite Order Group Operations

In the paper [17], in order to demonstrate the generic difference in the efficiency of prime order vs composite order implementations, they timed the group exponentiation (of a random group element with a random exponent) and pairing operations (on random group elements) in the MIRACL framework [19] for different security levels. The benchmarks were executed on a dual core Intel® Xeon® CPU W3503@2.40GHz with 2.0GB RAM running Ubuntu R10.04. The elliptic curve utilized for all benchmarks was the super-singular (symmetric) curve $y^2 = x^3 + 1$ mod p with embedding degree 2 for suitable primes p.

In the following table we can see the significant gap between the timings in prime and composite order groups for the same security levels. This is the main reason that we have implemented our second model which is based on prime order groups.

Group Exponentiation			
Security Level (bits)	Prime	Composite (2 primes)	Composite (3 primes)
80	3.5	66.9	201.6
112	14.8	448.1	1404.3
128	34.4	1402.5	4512.5
192	273.8	20097.0	66526.0
Group Pairing			
Security Level (bits)	Prime	Composite (2 primes)	Composite (3 primes)
80	13.9	245.3	762.3
112	65.7	1706.8	5485.2
128	176.6	5428.2	17494.4
192	1752.3	79046.8	263538.1

Table C.1: Average timing of group exponentiations and pairings in MIRACL [17]

Here timing results are in milliseconds over 100 repeats of group exponentiations and pairings in MIRACL.

Appendix D

Implementation of LSSS

D.1 Implementation of LSSS for Encryption

D.1.1 Algorithm

The following algorithm generates the LSSS matrix for encryption [5]. In each step of the algorithm, I have mentioned all the pyhton code (D.1.2) line numbers which are used to implement that algorithm step.

- **Input** : A threshold-tree-string $F_{\mathbb{A}}$ for a Threshold-gate access tree $\mathbb{A}$. (line-1)
- **Output** : A matrix M and a function ρ , which maps the i_{th} row of M to the i_{th} attribute in $F_{\mathbb{A}}$. (M, ρ) is the LSSS realizing $\mathbb{A}$. (from line-71 to line-74)
- **Convert** $(F_{\mathbb{A}})$: In the following, M is an $m \times d$ matrix over $\mathbb{Z}_p$, and $L = (L_1, L_2, ..., L_m)$ is a vector with m coordinates, where each coordinate is an attribute or a threshold-tree-string. The i_{th} coordinate of L labels the i_{th} row of M.
    1. Let matrix $M = (1)_{1\times 1}$, vector $L = (F_{\mathbb{A}})$, and $m = 1$, $d = 1$. (from line-2 to line-9)
    2. Repeat the following until all coordinates of L are attributes : (from line-10 to line-68)
        (a) Consider M to be an $m \times d$ matrix over $\mathbb{Z}_p$, and $L = (L_1, L_2, ..., L_m)$.
        (b) Scan the coordinates of L to find the first coordinate that is a threshold-tree-string rather than an attribute. Suppose the index of this coordinate is z. We have a threshold-tree-string $L_z = F_z = (F_{z,1}, F_{z,2}, ..., F_{z,m_2}, d_2)$. (from line-12 to line-17)
        Remark : If such a coordinate does not exist, it means that all the coordinates have been attributes and the algorithm should stop and output the matrix M.
        (c) Resolve F_z to obtain its m_2 children $F_{z,1}, F_{z,2}, ..., F_{z,m_2}$ and threshold value d_2. (from line-18 to line-27)
        (d) For this (d_2, m_2) threshold access structure, construct the corresponding LSSS matrix according to Equation(1), then execute “insertion" of this (d_2, m_2) LSSS matrix on the z_{th} row of M to obtain a new M with $m-1+m_2$ rows and $d+d_2-1$ columns. Set $L = (L_1, L_2, ..., L_{z-1}, F_{z,1}, F_{z,2}, ..., F_{z,m_2}, L_{z+1}, ..., L_m)$, and then set $m = m - 1 + m_2$ and $d = d - 1 + d_2$. (from line-46 to line-66)
    3. Return the matrix M and vector L. (from line-69 to line-74)

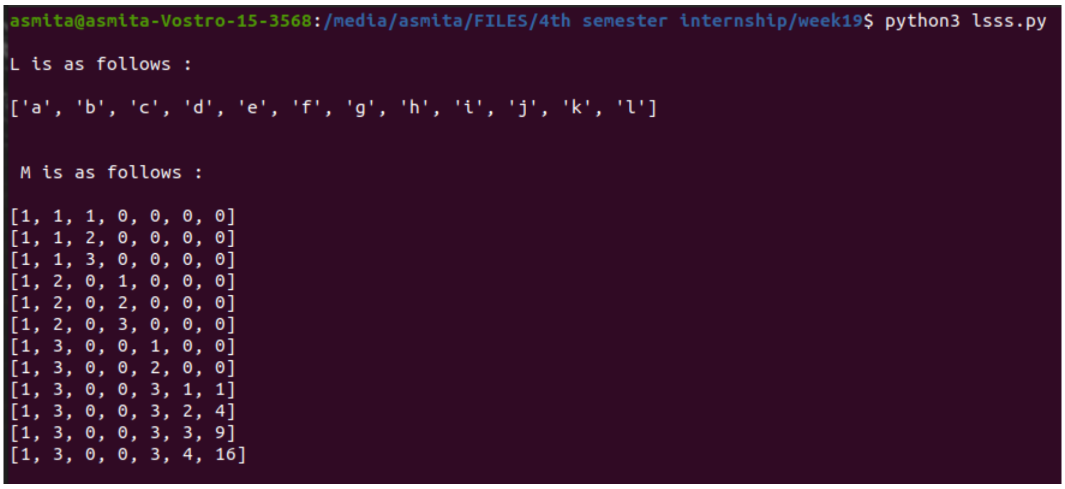
Equation(1) :
For a (t, n) threshold access structure $(P_1, P_2, ..., P_n, t)$, we can construct the corresponding LSSS over $\mathbb{Z}_p$, $p > n + 1$, as follows :

$$\rho(i) = P_i, \forall i \in \{1, 2, ..., n\} \qquad \text{and} \qquad M = \begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & 2 & 4 & ... & 2^{t-1} \\ 1 & 3 & 9 & ... & 3^{t-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & n & n^2 & ... & n^{t-1} \end{pmatrix}$$

D.1.2 My Python Code to Implement Above Algorithm

```
ap= [[[‘a’,‘b’,‘c’,2],[‘d’,‘e’,‘f’,2],[‘g’,‘h’,[‘i’,‘j’,‘k’,‘l’,3],2],2]]
M=[]
p=[]
p.append(1)
M.append(p)
l=[]
l=ap
m=1
d=1
z=0
while (z>=0) :
    z=(-1)
    i=0
    while (i<=m-1 and z==(-1)) :
        if (isinstance(l[i],list)) :
            z=i
        i=i+1
    if (z>=0) :
        fz=l[z]
        l2=[]
        m2=0
        for ele in fz :
            if (isinstance(ele, (list, str))) :
                m2 = m2 + 1
                l2.append(ele)
            if (isinstance(ele, int)) :
                d2 = ele
        m1=m
        d1=d
        M1=[]
        for i in range(0,m1,1) :
            p=[]
            for j in range(0,d1,1) :
                p.append(M[i][j])
            M1.append(p)
        l1=[]
        for i in l :
            l1.append(i)
        l=[]
        M=[]
        for i in range(0,m1+m2-1,1) :
            p=[]
            for j in range(0,d1+d2-1,1) :
                p.append(0)
            M.append(p)
        for i in range(0,(z),1) :
            l.append(l1[i])
            for j in range(0,(d1),1) :
                M[i][j]=M1[i][j]
            for j in range(d1,(d1+d2-1),1) :
                M[i][j]=0
        for i in range(z,(z+m2),1) :
            l.append(l2[i-z])
            for j in range(0,(d1),1) :
                M[i][j]=M1[z][j]
            a=i-(z-1)
            x=i-(z-1)
            for j in range(d1,(d1+d2-1),1) :
                M[i][j]=x
                x=x*a
        for i in range((z+m2),(m1+m2-1),1) :
            l.append(l1[i-m2+1])
            for j in range(0,(d1),1) :
                M[i][j]=M1[i-m2+1][j]
            for j in range(d1,(d1+d2-1),1) :
                M[i][j]=0
        m=m1+m2-1
        d=d1+d2-1
print(‘\n L is as follows :\n’)
print(l)
print(‘\n\n M is as follows :\n’)
for i in range(0,m,1) :
    print(M[i])
print(‘\n\n’)
```

This program gives me the following output :



```
asmita@asmita-Vostro-15-3568:/media/asmita/FILES/4th semester internship/week19$ python3 lsss.py

L is as follows :

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']


 M is as follows :

[1, 1, 1, 0, 0, 0, 0]
[1, 1, 2, 0, 0, 0, 0]
[1, 1, 3, 0, 0, 0, 0]
[1, 2, 0, 1, 0, 0, 0]
[1, 2, 0, 2, 0, 0, 0]
[1, 2, 0, 3, 0, 0, 0]
[1, 3, 0, 0, 1, 0, 0]
[1, 3, 0, 0, 2, 0, 0]
[1, 3, 0, 0, 3, 1, 1]
[1, 3, 0, 0, 3, 2, 4]
[1, 3, 0, 0, 3, 3, 9]
[1, 3, 0, 0, 3, 4, 16]
```

Figure D.1: Output of LSSS for Encryption

D.2 Implementation of LSSS for Decryption

D.2.1 Algorithm

The following algorithm generates the LSSS matrix for decryption [5]. I have highlighted all the steps which are different from the algorithm D.1.1 with bold letters. In each step of the algorithm, I have mentioned all the pyhton code (D.2.2) line numbers which are used to implement that algorithm step.

- **Input** : A threshold-tree-string $F_{\mathbb{A}}$ for a Threshold-gate access tree $\mathbb{A}$ **and an attribute set** S. (line-1 and line-2)
- **Output** : A matrix M **and a vector** L **whose coordinates are the attributes in** S**. The** i_{th} **row of** M **is labeled by the** i_{th} **coordinate of** L**.** (from line-113 to line-118)
- **Convert** $(F_{\mathbb{A}})$: In the following, M is an $m \times d$ matrix over $\mathbb{Z}_p$, and $L = (L_1, L_2, ..., L_m)$ is a vector with m coordinates, where each coordinate is an attribute or a threshold-tree-string. The i_{th} coordinate of L labels the i_{th} row of M.
  1. Let matrix $M = (1)_{1\times 1}$, vector $L = (F_{\mathbb{A}})$, and $m = 1$, $d = 1$. (from line-3 to line-10)
  2. Repeat the following until all coordinates of L are attributes :
     (a) Consider M to be an $m \times d$ matrix over $\mathbb{Z}_p$, and $L = (L_1, L_2, ..., L_m)$.
     (b) Scan the coordinates of L to find the first coordinate that is a threshold-tree-string rather than an attribute. Suppose the index of this coordinate is z. We have a threshold-tree-string $L_z = F_z = (F_{z,1}, F_{z,2}, ..., F_{z,m_2}, d_2)$. (from line-14 to line-18)
     If L_z **does not contain any attribute in** S**, remove the** z_{th} **row of M and the** z_{th} **coordinate of** L **, set** $m = m - 1$**, and go to (a).** (from line-29 to line-66)

(c) Resolve F_z to obtain its m_2 children $F_{z,1}, F_{z,2}, ..., F_{z,m_2}$ and threshold value d_2. (from line-20 to line-28)

(d) For this (d_2, m_2) threshold access structure, construct the corresponding LSSS matrix according to Equation(1), then execute “insertion" of this (d_2, m_2) LSSS matrix on the z_{th} row of M to obtain a new M with $m-1+m_2$ rows and $d+d_2-1$ columns. Set $L = (L_1, L_2, ..., L_{z-1}, F_{z,1}, F_{z,2}, ..., F_{z,m_2}, L_{z+1}, ..., L_m)$, and then set $m = m - 1 + m_2$ and $d = d - 1 + d_2$. (from line-67 to line-95)

3. **Remove the coordinates (i.e. attributes) of L that do not appear in S, and remove the corresponding rows of M** (from line-107 to line-112), **then return the matrix M and vector L.** (from line-107 to line-118)

Equation(1) :
For a (t, n) threshold access structure $(P_1, P_2, ..., P_n, t)$, we can construct the corresponding LSSS over $\mathbb{Z}_p$, $p > n + 1$, as follows :

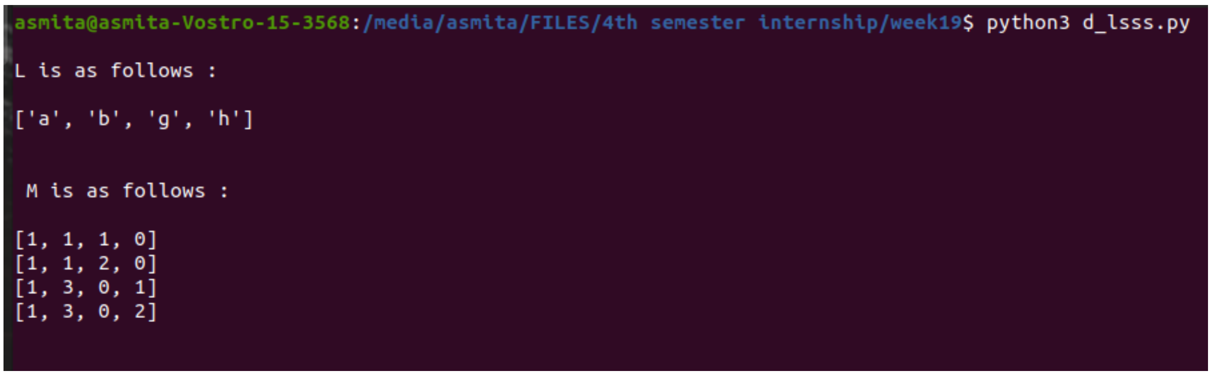
$$\rho(i) = P_i, \forall i \in \{1, 2, ..., n\} \qquad \text{and} \qquad M = \begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & 2 & 4 & ... & 2^{t-1} \\ 1 & 3 & 9 & ... & 3^{t-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & n & n^2 & ... & n^{t-1} \end{pmatrix}$$

D.2.2 My Python Code to Implement Above Algorithm

```
1   ap= [[[‘a’,‘b’,‘c’,2],[‘d’,‘e’,‘f’,2],[‘g’,‘h’,[‘i’,‘j’,‘k’,‘l’,3],2],2]]
2   S=[‘a’,‘b’,‘g’,‘h’]
3   M=[]
4   p=[]
5   p.append(1)
6   M.append(p)
7   l=[]
8   l=ap
9   m=1
10  d=1
11  z=0
12  while (z>=0) :
13      z=(-1)
14      i=0
15      while (i<=m-1 and z==(-1)) :
16          if (isinstance(l[i],list,)) :
17              z=i
18          i=i+1
19      if (z>=0) :
20          fz=l[z]
21          l2=[]
22          m2=0
```

```
        for ele in fz :
            if (isinstance(ele, (list, str))) :
                m2 = m2 + 1
                l2.append(ele)
            if (isinstance(ele, int)) :
                d2 = ele
        k=0
        for ele in l2 :
            if (isinstance(ele, str)) and ele in S :
                k=k+1
            ch=1
            while (isinstance(ele, list)) and (ch!=0) :
                ch=0
                t1=[]
                for el in ele :
                    if (isinstance(el, str)) and el in S :
                        k=k+1
                    if (isinstance(el, list)) :
                        ch=ch+1
                        for elm in el :
                            t1.append(elm)
                ele=[]
                for el in t1 :
                    ele.append(el)
        m1=m
        d1=d
        M1=[]
        for i in range(0,m1,1) :
            p=[]
            for j in range(0,d1,1) :
                p.append(M[i][j])
            M1.append(p)
        l1=[]
        for i in l :
            l1.append(i)
        l=[]
        M=[]
        if (k==0) :
            for i in range(0,m1,1) :
                if (i!=z) :
                    l.append(l1[i])
                    M.append(M1[i])
            m=m1-1
            d=d1
        else :
            for i in range(0,m1+m2-1,1) :
                p=[]
                for j in range(0,d1+d2-1,1) :
                    p.append(0)
                M.append(p)
            for i in range(0,(z),1) :
                l.append(l1[i])
                for j in range(0,(d1),1) :
                    M[i][j]=M1[i][j]
                for j in range(d1,(d1+d2-1),1) :
                    M[i][j]=0
            for i in range(z,(z+m2),1) :
                l.append(l2[i-z])
                for j in range(0,(d1),1) :
                    M[i][j]=M1[z][j]
                a=i-(z-1)
                x=i-(z-1)
                for j in range(d1,(d1+d2-1),1) :
                    M[i][j]=x
                    x=x*a
            for i in range((z+m2),(m1+m2-1),1) :
                l.append(l1[i-m2+1])
                for j in range(0,(d1),1) :
                    M[i][j]=M1[i-m2+1][j]
                for j in range(d1,(d1+d2-1),1) :
                    M[i][j]=0
            m=m1+m2-1
            d=d1+d2-1
M1=[]
for i in range(0,m,1) :
    p=[]
    for j in range(0,d,1) :
        p.append(M[i][j])
    M1.append(p)
l1=[]
for i in l :
    l1.append(i)
l=[]
M=[]
k=0
for i in range(0,m,1) :
    if l1[i] in S :
        l.append(l1[i])
        M.append(M1[i])
        k=k+1
print('\n L is as follows :\n')
print(l)
print('\n\n M is as follows :\n')
for i in range(0,k,1) :
    print(M[i])
print('\n\n')
```

This program gives me the following output :



```
asmita@asmita-Vostro-15-3568:/media/asmita/FILES/4th semester internship/week19$ python3 d_lsss.py

L is as follows :

['a', 'b', 'g', 'h']


 M is as follows :

[1, 1, 1, 0]
[1, 1, 2, 0]
[1, 3, 0, 1]
[1, 3, 0, 2]
```

Figure D.2: Output of LSSS for Decryption