

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Towards Implementing RSA-based CP- ABE Algorithm on Android System

A thesis presented in partial fulfilment of the requirements

for the degree of

Master of Information Sciences

at Massey University, Auckland, New Zealand

Jiaxin Xing

2019

Abstract

Cipher-text-Policy Attribute-Based Encryption (CP-ABE) algorithm has been proposed to encrypt and decrypt data based on the matching between attributes and an access policy placed over cipher-text. Using CP-ABE, data owner can encrypt data along with an access policy to enforce a fine-grained access control. To improve the efficiency of performance, this study chose a RSA-based CP-ABE algorithm with an access-tree structure while most existing CP-ABE has been implemented using ECC. This new RSA-based CP-ABE algorithm was implemented in the Linux system in another study while this thesis addresses an implementation strategy on an Android system. To achieve this goal, a simple encryption application was designed for users who want to encrypt and decrypt messages through their mobile devices. This study used Android Studio to create the encryption application. In this cipher program, users input the message they want to encrypt and get the encrypted data through the function button named “CIPHER”, and they also can decrypt the cipher-text in the same way.

There are four main algorithms involved in a CP-ABE scheme. They respectively are setup, key generation, encryption and decryption. During the setup process, the CP-ABE scheme uses the RSA algorithm to choose two prime numbers. These prime numbers are used to a master public key and a master private key. In the key generation algorithm, a secret key is generated for a set of attributes using the master private key. In the encryption step, it creates a cipher-text with an access tree. In the decryption algorithm, if and only if the attributes for the user's decryption key satisfies this access policy is able to decode the encrypted data. This algorithm uses the construction of lightweight no-pairing crypto-system based on RSA, and the construction supports an expressive monotone tree access structure to implement the complex access control as a more generic system. By using this algorithm, the encryption and decryption processes are more efficient and secure.

Acknowledgments

I would like to express my deep and sincere gratitude to my primary research supervisor Associate Professor Julian Jang-Jaccard of Massey University (New Zealand) for giving me the opportunity to do this research and providing invaluable guidance throughout this research and making the CP-ABE projects possible. Her extensive industry and academic experiences have been extremely valuable in contributing to the successful completion of my projects and thesis. It was a great privilege and honor to study under her guidance. I am extremely grateful for what she has offered me. I would also like to thank for her kindness and patience during the whole process.

I would also like to express my appreciation to my friends, Mrs. Ping Li, Yuanyuan Wei and Timothy Raymond McIntosh for their warm-hearted help and constant encouragement. I express my special thanks Mrs Ping Li for providing her RSA-based Access-Tree CP-ABE algorithm to support my thesis. I would also like to express my special thanks Yuanyuan Wei. She provided me a lot of help with this study, and supports me all the time.

I am extremely grateful to my parents for their love, caring and sacrifices for educating me for my future. Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

Table of Contents

CHAPTER 1. INTRODUCTION	1
1.1 Overview	1
1.2 Goal.....	2
1.3 Structure.....	3
CHAPTER 2. BACKGROUND TECHNOLOGY.....	4
2.1 Attributes-Based Encryption (ABE)	4
2.2 Access structure	5
2.3 CP-ABE.....	8
2.3.1 Five Fundamental Algorithms of CP-ABE	11
2.3.2 Security Mode for CP-ABE.....	15
2.4 Rivest-Shamir-Adleman (RSA)	16
2.4.1 RSA Algorithm	18
2.4.2 The Security of RSA	19
CHAPTER 3. LITERATURE REVIEW	21
3.1 CP-ABE for Mobile PHR System	21
3.2 CP-ABE for Mobile Devices in IoT	22
3.3 CP-ABE Based Privacy-Preserving User Profile Matching in Mobil Social Networks.....	23
CHAPTER 4. REVISIT RSA-based ACCESS-TREE CP-ABE SCHEME	24
4.1 Model.....	24
4.2 Algorithm Description	26
4.3 Security	31
CHAPTER 5. IMPLEMENTATION	33
5.1 Overview	33
5.2 Preparation for CP-ABE.....	33
5.2.1 Setup Environment	34
5.2.2 Java Native Interface (JNI).....	45
5.3 App on Android Studio	47
5.4 Implementation Algorithm	50
5.4.1 Setup	511
5.4.2 Key Generation	56
5.4.3 Encryption.....	59

5.4.4 Decryption	633
5.5 Running the App	69
5.6 App Demonstration	711
5.7 Performance Evaluation	742
5.8 Lessons Learned	74
<u>CHAPTER 6. CONCLUSION.....</u>	<u>77</u>
<u>REFERENCES</u>	<u>79</u>

List of Figures

Figure 1 . Attributes based encryption [7]	5
Figure 2 . Access control tree structure [32]	7
Figure 3 . A functional overview of CP-ABE [10]	9
Figure 4 . CP-ABE Implementation [10].....	10
Figure 5 . Flow Chart of CP-ABE Algorithm	12
Figure 6 . Asymmetric Algorithm	17
Figure 7 . An CP-ABE access control structure for PHR data [15].....	22
Figure 8 . Setup algorithm.....	27
Figure 9 . Key generation algorithm.....	28
Figure 10 . Encryption algorithm.....	29
Figure 11 . Decryption algorithm	30
Figure 12 . Install M4, bison and flex	35
Figure 13 . Configure the GMP library	36
Figure 14 . Build the GMP library.....	37
Figure 15 . Install the GMP library	37
Figure 16 . Extract compressed file.....	38
Figure 17 . Configure the PBC library	39
Figure 18 . Build the PBC library.....	39
Figure 19 . Install the PBC library	40
Figure 20 . Test code.....	40
Figure 21 . Compile file “foo” and run	40
Figure 22 . Configure the Libbswabe library	41
Figure 23 . Build the Libbswabe library.....	41
Figure 24 . Libbswabe sudo make install.....	42
Figure 25 . Install openssl and glib	42
Figure 26 . Build CP-ABE.....	43
Figure 27 . Test CP-ABE scheme	43
Figure 28 . Interface pointer [20]	46
Figure 29 . Interface design window.....	48
Figure 30 . Native CP-ABE.....	48
Figure 31 . Main activity widow	49
Figure 32 . Build Gradle(Module: app).....	49
Figure 33 . Setup	55
Figure 34 . Key generation for Sara	55
Figure 35 . Key generation for Kevin	58
Figure 36 . Encryption	62
Figure 37 . Steps for USB debugging	69
Figure 38 . Install Android adb tools package on terminal.....	70
Figure 39 . Select deployment target.....	70
Figure 40 . (a) Application interface (b) Hint for empty import (c) import text	71
Figure 41 . Key size	71

Chapter 1. Introduction

The first part of this chapter gives an overview of this study. It introduces the basic algorithms supported in the new CP-ABE system, and also briefly explains how CP-ABE scheme works. In the second part, it describes the main purpose of this project which is to implement the RSA-based CP-ABE scheme on an Android system. In the last part, it provides a brief description of the structure of this study.

1.1 Overview

The open source mobile Android system was developed by Google based on an Android kernel design. The Android operating system has become the basis of the recent applications that include smart phone, tablet computer, and smart TV etc. Android system supports a complete ecosystem that includes from development, installation and usage of applications. Due to its wide use, it is important to provide a secure mechanism to keep the data system and application that runs on an Android OS.

The Attributes-Based Encryption (ABE) algorithm was first proposed by Amit Sahai and Brent Waters in Fuzzy Identity-Based Encryption in 2005 [1]. One year later, Goyal et al., proposed the ABE for Fine-Grained Access Control of Encrypted Data [2]. Then the first construction of the Chiphertext-Policy Attributes-Based Encryption (CP-ABE) was provided by John Bethencourt, Amit Sahai, and Brent Waters in 2007 [3]. In their work, they presented the first construction of a scheme as CP-ABE, in which, the secret key of the user is combined with a set number of attributes. Data owners can define an access structure over a secret document in such a way that allows the users to decrypt only if the users' attributes satisfy the access policy. A more efficient CP-ABE scheme named RSA-based Access-Tree CP-ABE was presented by a master student Ping Li [36]. In her research, a no-pairing RSA algorithm was proposed to improve the efficiency of a CP-ABE scheme that was also based on the access tree structure. This scheme was implemented on Linux system. This study aims to provide an implement strategy for the new CP-ABE scheme on Android system.

Typically, there are four main algorithms of a CP-ABE scheme, which respectively are setup, key generation, encryption and decryption. In the first setup algorithm, two prime numbers are picked based on the concept of RSA algorithm, and new security parameters are produced step by step from these two primes. After getting all the security parameters, a master public

key as well as a master secret key are created by utilising the security parameters produced in the setup stage. In the second algorithm, it takes a set of attributes and the master private key as inputs and then it outputs a secret key associated with a set of attributes defined by each decoder by using a hash function for each attribute. In the third algorithm, a Lagrange polynomial is selected with an up-to-down manner for each node in an access structure. The algorithm randomly chooses a value from Z_N which is defined in setup algorithm and sets this value as the polynomial of the root node. Then this algorithm constructs a polynomial for each internal leaf node in an access tree, such that, an access tree structure is built during this process. Then, it encrypts the text with the access tree that associated with a set of attributes. In the last step, it checks the user's attributes associated with the secret key with the access policy by calling a recursive algorithm. The users will be able to access and decode the documents if their attributes meet the access policy.

This study implements the RSA-based CP-ABE scheme on an Android system by making an application for Android mobile devices. In order to build up this app, an Android Studio was used along with a series of tools, such as Java Native Interface (JNI). Many tools are all used to build a bridge that connects the native Java methods with the CP-ABE library written in C code.

1.2 Goal

There have been many algorithms to encrypt sensitive data as a way to provide data privacy. This study chose the RSA-based Access-Tree CP-ABE scheme that is considered as an efficient and lightweight encryption system that can run on resource constraint devices such as mobile phones. The original proposal of RSA-based CP-ABE system was written in C code and was implemented in the Linux system by another master student Ping Li. Extending from this, the goal of this study is to implement the RSA-based CP-ABE on the Android system so that it can be applied to android mobile devices.

Based on the new features of RSA-based CP-ABE scheme, this study made an encryption application that can work on an Android mobile device. This app uses the improved algorithm to encrypt and decrypt messages, and it is designed as a user interface with three basic functions: input text, encryption and decryption. In this encryption application, the users input messages that they want to encrypt in a text box, then click the "CIPHER" button to encrypt the message. The encrypted text is shown in the second text box that under the cipher button. After getting the encrypted text, a users can decrypt it by

using the “DECRYPT” button and get the original plain-text message.

To implement the RSA-based Access-Tree CP-ABE algorithm on Android system, there are a number of mechanisms involved. First of all, a setup of the development environment needed be built successfully. Some libraries that supports a CP-ABE algorithm need be installed, such as PBC, GMP and so on. After preparing those libraries, the CP-ABE algorithm should be compiled first, and then it is ready to be used in the system. Then, the Android Studio should be prepared to link the dynamic libraries. For example, the project should be built under the “C/C++ support” version. In addition, an Android.mk file is required to make .so libraries work, and other tools inside Android Studio are also needed to be prepared such as Native Development Kit (NDK) tools. Those tools can work as a bridge between C language and Java language. JNI is one of them, and it is a tool for calling functions from Java to another language. In Android Studio, it needs to import the CP-ABE libraries as well as to write Java code to link CP-ABE functions with Java command by JNI. In this way, the encryption application is able to implement the RSA-based Access-Tree CP-ABE scheme on the Android system.

1.3 Structure

In the beginning of this study, it describes the preliminary understanding of the encryption algorithm and discusses the advantages of a CP-ABE algorithm. In the next chapter, it describes the background knowledge required to better understand the proposed algorithm which includes the description of ABE algorithm, CP-ABE algorithm and RSA algorithm, which will all be involved in the RSA-based Access-Tree CP-ABE scheme. The related works are presented in chapter 3. It introduces the application of the CP-ABE algorithm in three different fields, and describes the basic working principle for each field. Chapter 4 revisits the new CP-ABE algorithm and provides in depth description of the new CP-ABE algorithm. Chapter 5 describes the implementation process of the new CP-ABE system. In particular, it introduces the setup environment that is used for preparing libraries and JNI function that is used to help the new algorithm run in the Android mobile devices. It also presents the steps of the application demonstration, and discusses the lessons learned. In the last chapter, it provides the conclusion.

Chapter 2. Background Technology

This chapter describes the background technology of the RSA-based CP-ABE scheme. This includes the description of the original ABE scheme where a CP-ABE is based on the description of a RSA algorithm, and the details of the original CP-ABE algorithm.

2.1 Attributes-Based Encryption (ABE)

In an ABE system, an encrypted data is associated with a set of attributes. A user's private key is associated with an access structure over a set of attributes. The user's private key reflects the user's access policy [6]. This implies that the user is allowed to decrypt if and only if the set of attributes of a user's private key satisfies the access policy.

Comparing with the original ABE scheme, the advanced ABE scheme improved its expressibility, which means that the user's private key is able to express any monotone access formula consisting of AND, OR, or threshold gates [6]. Moreover, when encrypting a message, a user may not be aware of the attributes. After creating the cipher-text, a new set of attributes may be used in the system. The core component of the current ABE systems is a secret-sharing scheme.

The ABE algorithm can encrypt the text from a data owner which correlates the attributes of the user [2]. The main characteristic of ABE is to reconsider the concept of a public key. Normally, a receiver decodes an encrypted message with a public key. In identity-based encryption (IBE) cryptography, the user's public key can be any string such as an email address [4]. Thus, the cipher-text can be decoded only if someone holds the key with the matching attributes. Generally, the user's key is issued by a third trusted party. ABE is basically a one-to-many algorithm that sends a message so that all legitimate users are able to decode [5]. By contrast, a one-to-one encryption algorithm has scalability issue as it can only send the message to a single recipient.

secure. The users who are authorized to access the resource are called the qualified groups of parties, which in turn are called qualified sets, and the set of all qualified sets is called the access structure of the system. In other words, the access structures are used to describe the necessary conditions that are needed to be able to access the information asset.

At the beginning stages of access structure applied in cryptography, the information asset was shared among the user's secretly [8]. Only the groups of users who are contained in the access structure are able to access the information asset. Normally, the information asset is a task that participants can solve together, such as decoding an encrypted message.

The background knowledge of an access structure is introduced below.

Gates

There are usually three types of gates in an access structure in a CP-ABE algorithm, which respectively are AND gates, OR gates and NOT gates:

- The AND gates allow passing if all the conditions are met.
- The OR gates allow passing if at least one of the conditions is met.
- The NOT gates allow passing if all the conditions are not met.

Threshold [1]

Both the user's private key and the cipher-text are each associated with a different set of attributes. If and only if the overlapping part between each set is at least the same as the globally defined threshold, then the decryption algorithm is able to work [31].

(n, n) -Threshold

The Shamirs Secret Sharing (SSS) with a variable threshold value t is used by the AND gates as a (t, n) -scheme. A single AND gates with n attributes is used, if all shares are required to be present. The (n, n) -Threshold access structure is the same as the AND gates method because the attributes can be multi-valued, thus it is called (n, n) -Threshold. However, in most cases, it is usually just called AND gates.

Tree [2]

This is a method that shares a secret resource in a KP-ABE or CP-ABE scheme. By using Lagrange interpolation, it can be reconstructed. It not only allows both AND gates and OR gates, but also permits any threshold gates with SSS for each node of the tree. The access tree can be easily built up from a textual representation (such as a boolean formula).

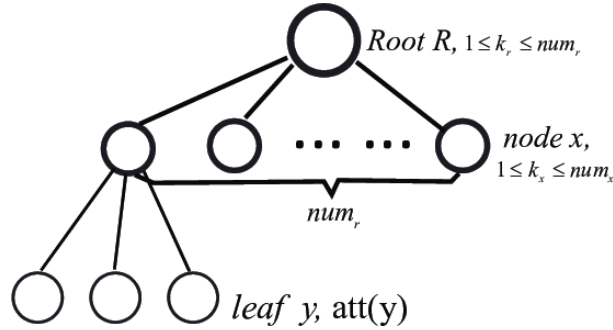


Figure 2. Access control tree structure [32]

The children of each internal-leaf node and a threshold value are used to describe a threshold gate represented by each internal-leaf in an access tree T which is illustrated in Figure 2. The number of child nodes of node x is presented as num_x , and the threshold value of the node x is $k_x \in [1, num_x]$. It indicates that node x performs the OR operation on all subsets of k_x children of x , and each subset supports an AND operation. As for the leaf-nodes, each of them is described by an attribute and a threshold value which is defined as $k_x=1$. When generating a data item, a tree is created for access control by defining the access rights with the associated attributes. This means that only the person who owns the attributes of the data item is able to decode the cipher-texts.

Linear Secret Sharing Scheme (LSSS)

LSSS stands for Linear Secret Sharing Scheme and it works on a matrix that signed rows with attributes of the policy. Based on this matrix, it creates shares from a secret element. The output of this scheme is the same as the access tree structure, although, in most situations, the LSSS approach can be replaced by the tree approach. However, due to the security games may depend on a specific approach, thus the security will not be guaranteed after exchanging [31].

2.3 CP-ABE

By using traditional methods, the difficulty of guaranteeing data security increases. When storing resources at several locations, the chances that one of them may become compromised increases dramatically [3]. A CP-ABE scheme allows sensitive data to be stored in an encrypted form regardless of whether or not a server is compromised. Moreover, it is able to efficiently handle more expressive types of encrypted access control [3].

Most existing CP-ABE schemes have a disadvantage of not being able to satisfy the flexibility and efficiency required by an enterprise's requirements of access control. A CP-ABE scheme is limited in some aspects, such as specifying policies and managing user attributes. In the CP-ABE construction, the secret key contains the set of attributes of the user. In other words, to satisfy policies, a user is only able to use possible combinations of attributes in a single set that is issued in their private keys [33]. The nature of a CP-ABE scheme influences itself. Due to the decryption privilege is shared by multiple users with the same attributes, this makes it hard to identify the original key owner from an exposed key. The commercial applications of CP-ABE are limited by this situation to some extent [34].

Here are the two main algorithms of the ABE scheme:

- KP-ABE, it combines a set of attributes with every encrypted message. Then an access structure is associated with each secret key of the user.
- CP-ABE, a set of attributes is associated with each user's private key. An access policy is defined over a message [9].

It is important in a CP-ABE scheme that the person who owns the secret data is able to determine the access policy. Moreover, the data owner can only determine the people who have the authority to access the data by describing their attributes or credentials rather than knowing the certain identities of the other users.

Generally, the information asset of the owner is in the custody of a trustworthy server. In order to make sure that only the accredited person is entitled to obtain the secret message, software checking is applied to control the access structure. However, identifying every potential recipient, acquiring and storing their public keys is difficult to do in a large-scale system. Thus, it is expected to encrypt the data with an incomplete list of intended recipients. The CP-ABE algorithm is an emerging approach to handle this situation and

ensures the authentication of the user by checking whether the user's attributes satisfy the access policy or not. CP-ABE can identify whether the user is legitimate when performing an operation on the stored data. Access control policies provide two significant functions, the first one is authorizing multiple user rights to access an information asset. The second one is making personal access rights more flexible. The data owner can encrypt the data without the information of all the receivers by using flexible access policy in a CP-ABE scheme.

Bethencourt, Sahai and Waters introduced BSW's scheme as the construction of a CP-ABE scheme. The function of BSW's scheme is to present more expressive models for access control. The private key is combined with strings that represent an arbitrary number of attributes. For those users who are able to decrypt the message, there is always a policy that is selectively specified by the encrypted data party. The only condition for a user to decode the cipher-texts is that the decryption key meets the access policy.

To provide privacy for the access of confidential data, the CP-ABE system is required. Here is a simple case of how CP-ABE works. In Figure 1, let's suppose that the data owner only wants the users with "MA" and "MSc" to access the resources and decrypt the documents. Therefore, only the group of people whose attributes all satisfy the policy is able to access the data. Any other users without the attributes which can satisfy the access policy are unable to access those data. For example, the user U2 in Figure1 has the attributes {"CS", "MSc"} which does not meet the access policy, thus user U2 will be unable to access the files.

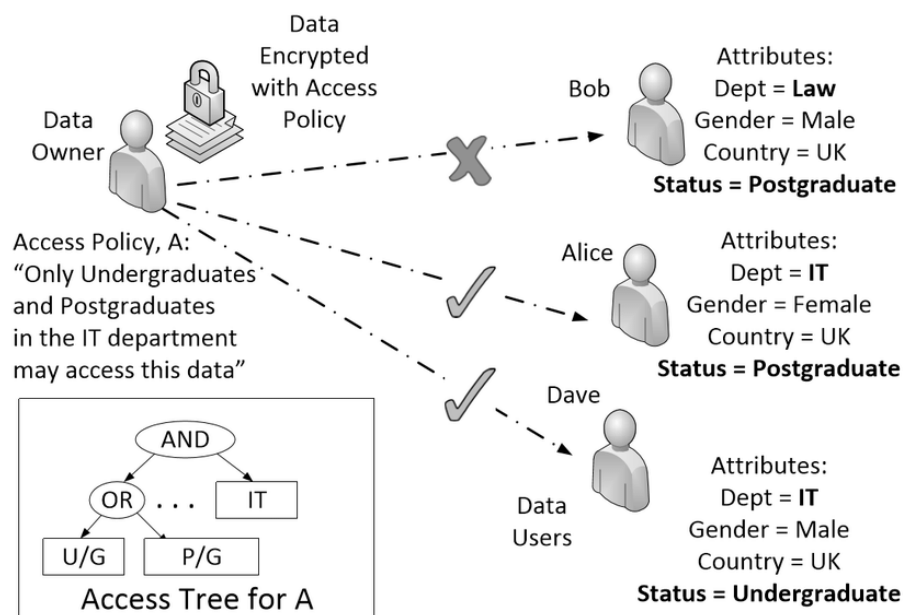


Figure 3. A functional overview of CP-ABE [10]

This graph gives a detailed explanation of the usage and the basic function of an access tree in CP-ABE. An access policy A is created by the data owner, which is “Only Undergraduates and Postgraduates in the IT department may access this data”. Then the data is encrypted with this access policy. There are three data users with their own attributes respectively. The attributes of Alice are $\{\text{Dept} = \text{IT}, \text{Gender} = \text{Female}, \text{Country} = \text{UK}, \text{Status} = \text{Postgraduate}\}$. In the attributes set of Alice, there shows IT and Postgraduate which meet the access policy. Therefore, Alice is authorized and she can access the data and the same goes for Dave. The attributes of Bob are $\{\text{Dept} = \text{Law}, \text{Gender} = \text{Male}, \text{Country} = \text{UK}, \text{Status} = \text{Postgraduate}\}$. However, as Bob is a Postgraduate student in the Department of Law, he only satisfies one condition. Therefore, Bob is unable to access the data.

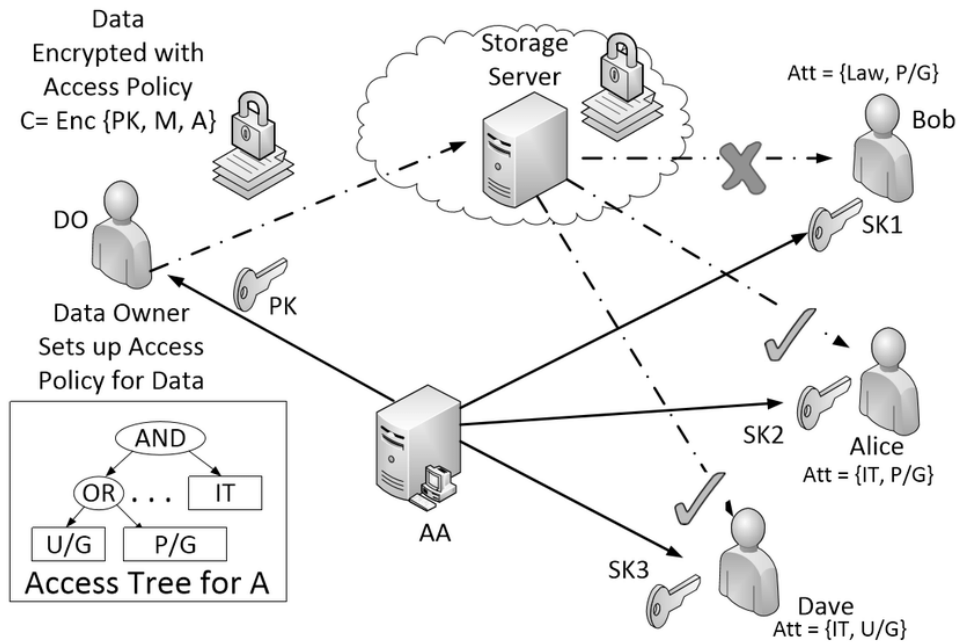


Figure 4. CP-ABE Implementation [10]

Figure 4 shows how CP-ABE can work in daily life. The data owner (DO) wants to share some documents, then the owner sets up an access policy for their information asset. When the data owner gets the public key (PK) from the attribute authority (AA), they can then encode the message (M) by using the public key (PK) and an access policy (A). Then an encrypted data is kept in a trusted storage server. The attribute authority (AA) also gives each user a different secret key, in the example of Figure 4, they are SK1, SK2 and SK3. As it is shown in Figure 4, there are three users, Bob, Alice and Dave, and their attributes respectively are $\{\text{Law}, \text{P/G}\}$, $\{\text{IT}, \text{P/G}\}$ and $\{\text{IT}, \text{U/G}\}$. In the access tree for attributes shows that the user should satisfy those conditions. The user should be in the group of $\{\text{IT}, \text{U/G}\}$ or $\{\text{IT}, \text{P/G}\}$ respectively. The access tree for the attributes shows what conditions the users need to meet in order to gain

access. In Figure 4, the user needs to be in the group of {IT, U/G} or {IT, P/G}. Thus, according to the parameters of Figure 4, Alice and Dave meet the conditions and they can decrypt the documents uploaded by the data owner respectively with their secret key SK2 and SK3 respectively. However, the attributes of Bob are {Law, P/G}, which does not meet the access attributes required. Therefore, due to his attributes, Bob cannot obtain the files from the data owner with his secret key SK1.

2.3.1 Five Fundamental Algorithms of CP-ABE

Here is the basic description of five fundamental algorithms in a CP-ABE scheme [3]:

- ① set up (input: universal attributes set, security parameter
output: public parameters PK, master secret key MK)

During this process, security parameter and universal attributes set are taken as the input of this algorithm. Then it outputs the public parameter PK and master key MK.

- ② key generate (input: MK, attributes S
output: private key SK)

The key generation algorithm inputs MK and a set of attributes S that describes the key. Then it outputs a private key SK.

- ③ Encrypt (input: PK, message M,
access structure A over the universe of attributes
output: cipher-text CT)

The inputs of the encryption algorithm are the PK, a message M, and an access structure A over the universe of attributes. Then the message M is encrypted into the cipher-text CT, and only the receiver whose attributes satisfy the access structure are able to decrypt the cipher-text.

- ④ Decrypt (input: PK, CT[A], SK[S]
output: M)

PK, CT with an access policy A, private key SK (a private key for a set of attributes S) are the inputs of decryption algorithm, and it decodes the cipher-text CT into the message M only if the set of attributes S satisfies the

access structure A.

⑤ Delegate (input: SK
output: SK, S^\sim)

If required, a delegate takes the secret key (SK) as an input and return a secret key (SK) for a given set of attributes (S^\sim).

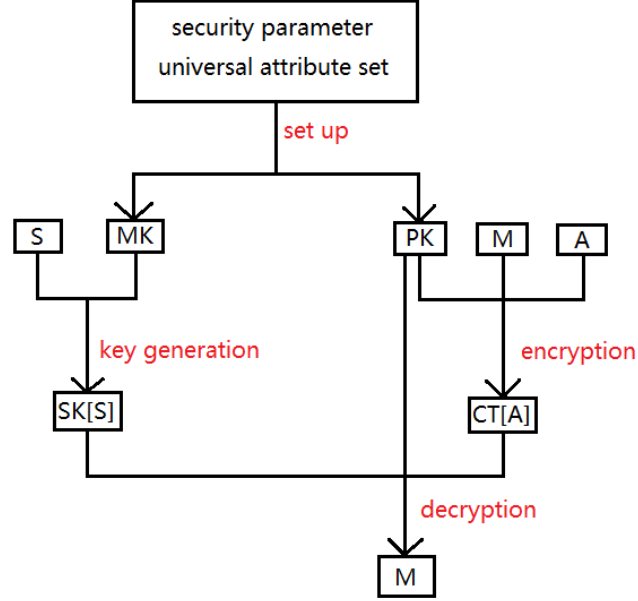


Figure 5. Flow Chart of CP-ABE Algorithm

Figure 5 shows the process of CP-ABE algorithm. Firstly, it uses a setup algorithm to create a master secret key MK and a public key PK from a universal attributes set. Then, the key generation algorithm takes a set of attributes and MK as input and output the assigned secret key SK with this set of attributes. Meanwhile, PK is used to encrypt the message M with an access policy A during the encryption algorithm, and it outputs the cipher-text with A. Finally, the message can be decoded with PK, the satisfied SK[S] and CT[A].

Here is the construction of these five algorithms, but first, it defines some elements which are used in those algorithms.

- G_0 : A bilinear group of prime order p
- g : A generator of G_0
- $e: G_0 \times G_0 \rightarrow G_1$: The bilinear map

- k : A security parameter which determine the size of the groups.
- $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$: The Lagrange coefficient $\Delta_{i,S}$ for $i \in Z_p$ and S which is a set of elements in Z_p .
- $H : \{0,1\}^* \rightarrow G_0$: A hash function H which models as a random oracle. This function maps any attribute described as a binary string to a random group element.

Here are the mathematical formula descriptions of these five algorithms [3]:

① Setup algorithm:

Choose: A bilinear group G of prime order p with a generator g ;

Two random exponents $\alpha, \beta \in Z_p$.

Then get the public key and the master key respectively. Note that f is only used for delegation procedure:

$$PK = G_0, g, h = g^\beta, f = g^{1/\beta}, e(g, g)^\alpha$$

$$MK = (\beta, g^\alpha)$$

② Key generate algorithm (master secret key MK and a set of attributes S):

Choose: a random $t \in Z_p$ and random $r_j \in Z_p$ for each attribute $j \in S$.

Then get the secret key SK as follow:

$$SK = (D = g^{(\alpha+t)/\beta}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j})$$

③ Encryption algorithm(public key PK , message M and access tree T):

Choose: a polynomial q_x for each node x in the access tree T ;

For the root node R choose:

a random $s \in Z_p$ and sets $q_R(0) = s$;

d_R other points of the polynomial q_R randomly.

For any other node x :

$$\text{set } q_x(0) = q_{\text{parent}(x)}(\text{index}(x));$$

choose d_x other points randomly to define q_x completely.

Let Y be the set of leaf nodes in T , then the cipher-text CT is computed as:

$$CT = (T, \tilde{C} = Me(g, g)^{as}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\text{att}(y))^{q_y(0)})$$

④ Delegate algorithm (secret key SK and a set of attributes $\tilde{S} \in S$)

Choose: random \tilde{r} such that $\tilde{r}_k \forall k \in \tilde{S}$.

Then a new secret key is computed as:

$$\tilde{S} K = (\tilde{D} = Df^{\tilde{r}}, \forall k \in \tilde{S} : \tilde{D}_k = D_k g^{\tilde{r}} H(k)^{\tilde{r}_k}, \tilde{D}'_k = D'_k g^{\tilde{r}_k})$$

⑤ Decryption algorithm (cipher-text CT and secret key SK):

This is a recursive algorithm, and it is defined as $DecryptNode(CT, SK, x)$. If the node x from access tree T is a leaf node, then set $i = \text{att}(x)$:

If $i \in S$, then

$$\begin{aligned} DecryptNode(CT, SK, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= \frac{e(g^{\tilde{r}} \cdot H(i)^{\tilde{r}_i}, h^{q_x(0)})}{e(g^{\tilde{r}_i}, H(i)^{q_x(0)})} \\ &= e(g, g)^{r q_x(0)} \end{aligned}$$

If $i \notin S$, then

$$DecryptNode(CT, SK, x) = \perp$$

If x is a non-leaf node, for nodes z which are the children of node x , it is defined as $DecryptNode(CT, SK, z)$. Let F_z be the output. Let S_x be the arbitrary k_x -sized set of child nodes z which satisfies that $F_z \neq \perp$, then

$$\begin{aligned}
F_z &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \text{ where } i = \text{index}(z), \text{ and } S'_x = \{\text{index}(z) : z \in S_x\} \\
&= \prod_{z \in S_x} (e(g, g)^{r \cdot q_x(0)})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} (e(g, g)^{r \cdot q_{\text{parent}(z)}(\text{index}(z))})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} e(g, g)^{r \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\
&= e(g, g)^{r \cdot q_x(0)}
\end{aligned}$$

Else return \perp .

After the function *DecryptNode* is defined, now the decryption algorithm can start with calling function *DecryptNode* on the root node R of the access tree T . If and only if this access tree T is satisfied by the set of attributes S , then set

$$A = \text{DecryptNode}(CT, SK, r) = e(g, g)^{r q_R(0)} = e(g, g)^{rs}$$

Then the original message is computed as:

$$\tilde{C} / (e(C, D) / A) = \tilde{C} / (e(h^s, g^{(\alpha+r)/\beta}) / e(g, g)^{rs}) = M$$

2.3.2 Security Mode for CP-ABE

① Setup

The challenger generates the public parameters PK by running the Setup algorithm, and the adversary receives the public parameters PK .

② Phase 1

The adversary makes repeated private keys corresponding to the attributes sets which are S_1, \dots, S_{q_1} .

③ Challenge

The adversary not only submits two messages M_0 and M_1 which have equal length, but also gives a challenge access structure A^* such that none

of the sets of attributes S_1, \dots, S_{q_1} from Phase 1 satisfy the access tree made by adversary. The challenger flips a random coin b , and encrypts M_b under the access tree structure A^* . Then the cipher-text CT^* is sent to the adversary.

④ Phase 2

Phase 1 is repeated with the restriction that none of sets of attributes S_{q_1+1}, \dots, S_q satisfy A^* corresponding to the challenge.

⑤ Guess

The adversary gives a guess b' of b .

In these processes, the advantage of an adversary is defined as $\Pr[b' = b] - \frac{1}{2}$.

Allowing for decryption queries in Phase 1 and Phase 2, the model can be extended to handle chosen-cipher-text attacks easily.

Therefore, if all polynomial time adversaries have at most a negligible advantage in the above procedures, then a CP-ABE scheme can be secure.

2.4 Rivest-Shamir-Adleman (RSA)

Rivest-Shamir-Adleman (RSA) is an asymmetric algorithm that uses key pairs (public keys & private keys) to deal with the message. Public keys can be exposed to anyone. On the contrary, only the owner of keys holds the private keys. In other words, there is no need to compromise security of public keys which can be distributed publicly, but the privacy of private keys.

In order to generate the key pairs safely, cryptographic algorithms based on mathematical problems to generate one-way functions are needed. It can be a notion of a trapdoor function which is a mathematical function that underpins the public key encryption system [23]. For example, the process of taking a given value A and using the trapdoor function to get another value B is very easy, however, it is intractable to use trapdoor function to get the value A from the value B . The reason is that it is easy to “add” points together and to “multiply” a point by an integer by using the “group law” (trapdoor function), but it is very difficult to work backward to “divide” a point by a number. In other words, assuming that it is intractable to factor a large integer composed of two or more large prime factors, the public key systems are secure. The greatest common divisor of two numbers can be found by using the Euclidean algorithm.

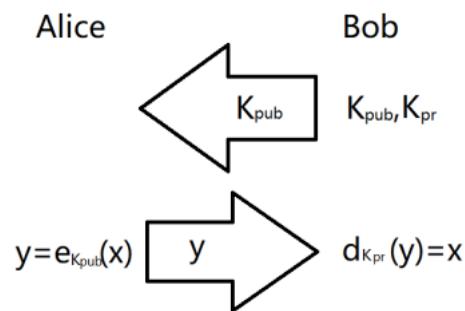


Figure 6. Asymmetric Algorithm

As it is shown in Figure 6, Alice wants to send message to Bob. In the beginning, Bob generates both his public key (K_{pub}) and the private key (K_{pr}). When Alice receives the public key from Bob, she can encrypt the message x with the public key. After encrypting, Alice sends the encrypted message y to Bob. By using the private key, Bob can decrypt the message y into the original message x . This figure explains how the asymmetric algorithm works. Alice uses the public key which is from Bob to encrypt the message x , and Bob uses the private key to decrypt the message y . The security coefficient is improved by using different keys during encryption and decryption process.

Due to the complex mathematics that RSA algorithm presents, it is safe and secure. The reason is that it refers to the factorization of prime numbers. These prime numbers are large enough that is hard to factorize. This trap door function guarantees the security of this algorithm. Moreover, another advantage of RSA algorithm is the public key which is public and easy to get by anyone. This allows distributing keys safely, which is the big issue in cryptography system.

The RSA algorithm can be slow when generating key pairs with large primes. Besides this, when encrypting large data in the same computer, the RSA algorithm can also be very slow. For further explanation, the main computational cost of the RSA algorithm is the modular exponentiation during the key generation, encryption and decryption process [35]. The reason is that this algorithm needs a third trusted party to identify the public keys. During the data transmission, it can be exposed to the middlemen who is able to temper with the public key system and the algorithm can be compromised. Thus, a secure implement is difficult due to the slow speed of signing and decryption. In addition, the RSA algorithm has weaknesses against certain attacks, such as Brute force as the capacity of supercomputer advances rapidly.

2.4.1 RSA Algorithm

RSA algorithm is the first algorithm that is applied for data encryption and digital signatures [24]. As it is mentioned before, RSA algorithm is a trap-door function and the security of this algorithm is based on the difficulty of the one way function which means dealing with the problem that decomposes a large number into two large prime numbers.

In the RSA algorithm, it has three processes: key generation algorithm, encryption algorithm and decryption algorithm, respectively. Here is the description of each process (see Figure 9):

① Key generation:

- a. Choose large prime numbers: p, q
- b. $n = p \cdot q$
- c. $\phi(n) = (p-1)(q-1)$
- d. Choose $e \in \{1, 2, \dots, \phi(n)-1\}$;
and $\gcd(e, \phi(n)) = 1$
- e. Compute d , and $d \cdot e \equiv 1 \pmod{\phi(n)}$

Then, the public key is computed as:

$$K_{pub} = (n, e)$$

and the private key is computed as:

$$K_{pr} = (n, d)$$

② Encryption:

- a. Obtain the public key $K_{pub} = (n, e)$
- b. Choose $x \in Z_n = \{0, 1, \dots, n-1\}$
- c. Compute $y = e_{K_{pub}}(x) \equiv x^e \pmod{n}$
- d. Send out the message y

③ Decryption:

- a. Obtain the cipher-text $y \in Z_n$
- b. Use $K_{pr} = (n, d)$ to compute $x = d_{K_{pr}}(y) \equiv y^d \pmod{n}$

Sometimes, the value x can be a short integer which is the output of some not injective compression function, such as a hash function. Thus, the value x can be defined as a message digest. It is based on that the exponentiation modulo n is a one-way permutation on Z_n when e is co-prime to $\phi(n)$. Here is a trapdoor function that the permutation can be efficiently inverted

by the private key (n, d) . Therefore, the RSA system can also be used as a digital signature algorithm which is listed below:

When sending a message, the sender:

- a. Create a message digest
- b. Represent this digest as an integer $x \in Z_n$
- c. Use private key $K_{pr} = (n, d)$ to compute the signature

$$s = x^d \pmod{n}$$
- d. Send signature s to the recipient

When receiving a message, the receiver:

- a. Obtain the public key $K_{pub} = (n, e)$ of the message owner
- b. Compute integer $v = s^e \pmod{n}$
- c. Independently computes the message digest x' of the information that has been signed
- d. Computes the expected representative integer v' by encoding the expected message digest x'
 If the verification equation $v = v'$ holds, the signature is valid, then
- e. the message is from the owner.

As it can be seen from the algorithms above, the decryption algorithm and signing algorithm both use the private key, those two algorithms are same from a mathematical viewpoint. So as the encryption and verification, they both use the identical algorithm with the public key, where

$$x = (x^e)^d = (x^d)^e \pmod{n}$$

2.4.2 The Security of RSA

The security of public key encryption is mainly based on decomposing large integers multiplied by two large prime numbers p and q . This kind computation is very difficult, because it is easy to multiply two large prime number together but hard to determine the initial number from the total. This builds up the security of RSA.

The public key encryption can be on risk if the large prime numbers p and q are found. Obviously, the public key (n, e) is known to the public. Thus anyone can compute p and q by factorizing n , and get $\phi(n)$ by computing $\phi(n) = (p-1)(q-1)$. Once $\phi(n)$ is known, the private key $K_{pr} = (n, d)$ can be computed by $d \cdot e \equiv 1 \pmod{\phi(n)}$. RSA algorithm can be secure only if the integer n is big enough so that the prime numbers p and q are not easy to be found.

However, if the integer n is small, the prime numbers p and q are easy to know by testing all possible prime numbers in the range of $(1, n)$. Most attackers can't find the two prime numbers p and q , because it takes so long to decompose the product of these two prime numbers which are large enough. There is no capability or equipment for most attackers to do so other than the state actors that may have access to sufficient computing power.

Typically, the key length of an RSA key refers to the length of the modulus n in bits. More bits mean more secure but more CPU and power while encrypting and decoding, which can impact the performance of server. The recommended key length for a secure RSA transmission is 2048 bits long. According to experts, 2048 bits is more secure than many facilities are replacing key lengths of 1024 bits with the minimum of 2048 bits. However, RSA is unable to encrypt anything that is larger than its modulus n . Thus, the key length is usually defined as $(\log_{256}(n+1)) \times 8$.

Chapter 3. Literature Review

This chapter introduces the three related fields of applications that utilises CP-ABE scheme. The first field is described from the PHR system which is used for the patient and doctor in a hospital. The second field describes the application of IoT (internet of things) applying CP-ABE as data privacy mechanism. The last field the literature review is conducted is from the Privacy-Preserving User Profile Matching in Mobile Social Networks.

3.1 CP-ABE for Mobile PHR System

PHR system means Personal Health Record system [11], which is secure to a great extent. It only allows patients or health care providers to securely access the health information of the patient through the internet. This patient's centered system has massive private data in lots of aspects. For example, it contains the health conditions of the patient, the medical history of the user and much other private information. The PHR system is exposed under the open internet, which means the internet can be a threat to the personal information contained in this system. Therefore, the big issue of PHR system is to keep it as secure as possible. In order to keep the PHR system secure, an efficient mediated cipher-text-policy attribute-based encryption (M-CP-ABE) scheme is used to revoke attribute at once and support monotonic access structure by using Linear Secret-Sharing Schemes(LSSS). Thus, the chance that an unauthorized user accesses to the personal health information is reduced in a PHR system. In the other aspect, the developed mobile internet allows most users to access the PHR system more conveniently via their mobile devices, thus the CP-ABE based PHR system suits for the mobile internet.

CP-ABE scheme is suitable for access control in the PHR system, for it not only cuts the encryption cost for PHR data owner but provides agile self-centric data access management as well [12][13]. In this CP-ABE scheme, the access policy of the user is defined by a set of attributes. A user can access the encrypted data if and only if his or her attributes satisfy the access policy [14][15].

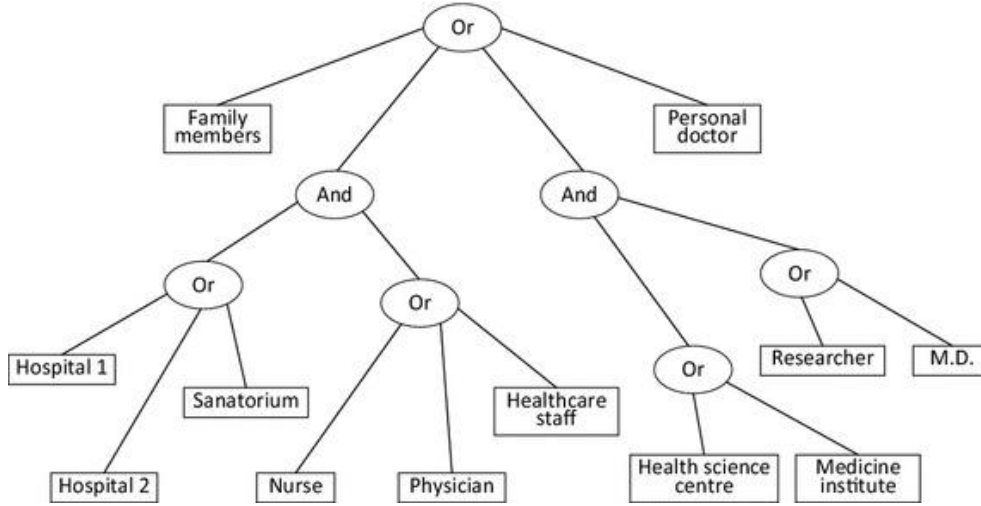


Figure 7. An CP-ABE access control structure for PHR data [15]

In Figure 7, it presents an access tree policy. The data owner can describe users who own the right to access the data by a set of attributes (such as personal doctor) without knowing the exact identity of users. For example, a user has the attributes of {Medicine institute, Researcher} is able to access the PHR data if his or her attributes satisfy with the policy in Figure 7.

PHR as well as other systems applied the attribute based encryption scheme to build the medical care system. However, the performance degradation is observed due to expensive bilinear pairing operations have to be run for a several times while decoding a message. The heavy computation increases the difficulty in the process of mobile PHR data sharing when a PHR user accesses to the personal information via a mobile device with limited computing resources such as cell phone and smart-watch.

3.2 CP-ABE for Mobile Devices in IoT

Lightweight CP-ABE schemes with constant size secret keys and constant-size cipher-texts have been proposed for battery-limited mobile devices such as cell phone [16]. More and more people using those mobile devices in their daily lives. Lightweight CP-ABE scheme is in demand in order to provide data privacy solution in such resource constraint devices.

An RSA-based AND-gate access structure CP-ABE scheme is presented to provide constant-size secret keys and cipher-texts with efficiently encrypt and decrypt process. In this scheme, a secret key is associated with an attribute set A , and a user is able to use the secret key with attribute A to decrypt cipher-texts with the access policy P if and only if $P \subseteq A$. This scheme offers constant-size secret keys and cipher-texts without using bilinear maps,

and it is suitable for practical deployments on battery-limited devices due to the underlying RSA architecture.

3.3 CP-ABE Based Privacy-Preserving User Profile Matching in Mobil Social Networks

Based on CP-ABE scheme, users are able to offer a preference-profile and find other users with matching-profile in decentralized mobile social networks while the preference-profile of no one is exposed, and then build a secure channel between matched users. This scheme helps to solve the privacy-preserving profile matching problem to some extent. It is because it offers verifiability with few interactions among users is required. In most CP-ABE scheme, when the number of attributes increases, then the size of cipher-text and decryption time will increase as well. This situation will influence communication and computation efficiency [17]. Therefore, a CP-ABE construction that provides receiver anonymity through hidden access policy with unchanged cipher-text size and decryption time is need to keep both the security and efficiency. This scheme is based on the prime order group and relies on asymmetric decision bilinear Diffie-Hellman problem.

The CP-ABE scheme needs to satisfy with the condition that the profile of the matched user must include all the attributes in the preference-profile. Thus, a reminder vector which is a data structure and a corresponding algorithm that is used to improve matching speed is used to overcome the barrier. This CP-ABE construction not only has the capability of privacy-preserving, but also provides verifiability that no one can cheat the initiator with the wrong matching result. In addition, it also builds a secure channel between matched users, and the unmatched user can be excluded soon. In addition, only a small amount of interaction is required between the initiator and the matching user, since the matching user can determine the matching result without the help of the initiator, which is important for reducing computing and communication costs.

Chapter 4. Revisit RSA-based

Access-Tree CP-ABE scheme

The first construction of a CP-ABE with a monotonic “access tree” was provided in the work of Bethencourt [3]. In the use of an access tree, the internal nodes of the access structure were consisted of threshold gates and leaves that associated with descriptive attributes. Most asymmetric encryption schemes allow for data encryption with a restrictive policy that are unable to efficiently enforce more expressive types of access control. By comparison, the access structure of the scheme provided by Bethencourt et al. supports AND gate, OR gate and the comparison of numerical attributes. For the AND gates can be built as the n-of-n threshold gates, the OR gates can be constructed as the 1-of-n threshold gates, and the comparison of numerical attributes that belong to more complex access control can be solved by converting them into small access tree.

For decryption, the attributes of the user are associated with the private keys. If the receiver wants to decrypt the information asset, then the attributes of his or her private key should satisfy the specified policy through an access tree structure. In other words, only if the secret key of the user contains all the attributes which are assigned to the leaves of the tree, then the user can decrypt the data.

4.1 Model

Access Structure Definition [3]

Let $\{P_1, P_2, \dots, P_n\}$ be a set of parties. A collection $A \subseteq 2^{\{P_1, \dots, P_n\}}$ is monotone if $\forall B, C : \text{if } B \in A \text{ and } B \subseteq C \text{ then } C \in A$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) A of nonempty subsets of $\{P_1, P_2, \dots, P_n\}$. The sets in A are called the authorized sets, and the sets not in A are called the unauthorized sets.

Access tree T [3]

An access structure is represented by T , and each internal node of T

expresses a threshold value gate which are described by it's children and a threshold value. If a non-leaf node x has $number_x$ children and the threshold value of node x is kx , then $0 < kx \leq number_x$. The threshold gate is an OR gate when kx equals to one while the threshold gate is an AND gate when kx equals to $number_x$. A set of attributes and a threshold value($kx=1$) are used to describe each leaf node x of the access tree.

Some functions are defined to make it easy to use access tree:

- ① The parent of the node x in the access tree is defined as $parent(x)$;
- ② When x is a leaf node, the attribute associated with x is defined as $att(x)$;
- ③ In an access tree, the children of each node are ordering from 1 to $number_x$;
- ④ The function $index(x)$ returns a number which is associated with the node, and the index values are assigned with nodes in the access tree uniquely for a given key in a random manner.

Satisfying an Access Tree T [3]

T is defined as an access tree with root r , and the subtree of T rooted at the node x is defined as T_x , thus T is same as T_r . If the access tree T_x is satisfied with a set of attributes γ , then it is defined as $T_x(\gamma)=1$. If x is an internal node, evaluate $T_{x'}(\gamma)$ for all children x' of node x . If and only of at least k_x children return 1, then $T_x(\gamma)=1$. If x is a leaf node, then $T_x(\gamma)=1$ if and only if $att(x) \in \gamma$.

Lagrange polynomials [18]

Lagrange polynomials are used for polynomial interpolation. For a given a set of k points $(x_0, y_0), \dots, (x_i, y_i), \dots, (x_k, y_k)$, where no two x_j values are equal, the Lagrange polynomial is the polynomial of lowest degree that assumes at each value x_j the corresponding value y_j . The interpolating polynomial of the least degree is unique. An n degree polynomial in the Lagrange form is a linear combination:

$$P(x) = \sum_{i=0}^n \left(\prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j} \right) y_j$$

According to the definition, define the Lagrange coefficient $\Delta_{i,S}$ for $i \in Z_n$ and a set S of elements in Z_n :

$$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x - j}{i - j}$$

Polynomial interpolation is the method to construct the Lagrange polynomial. The polynomial $P(x)$ is called Lagrange interpolation polynomial. Suppose that the polynomial $P(x)$ is in the form as below:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

Polynomial interpolation

Polynomial interpolation is to find a polynomial p of lowest possible degree with the property $P(x_i) = y_i$ for all $i \in \{0, 1, \dots, n\}$.

4.2 Algorithm Description

This RSA-based Access-Tree CP-ABE cryptosystem is constructed with no-pairing RSA, and works on a group Z_N and its subgroup written in multiplication notation as Z_N^* or $(Z/nZ)^*$, which is of congruence classes of integers modulo N where $N = p \cdot q$. The generation of key pair is based on the integer decomposition. This encryption scheme is also designed to be built on discrete logarithm problem.

① Set Up

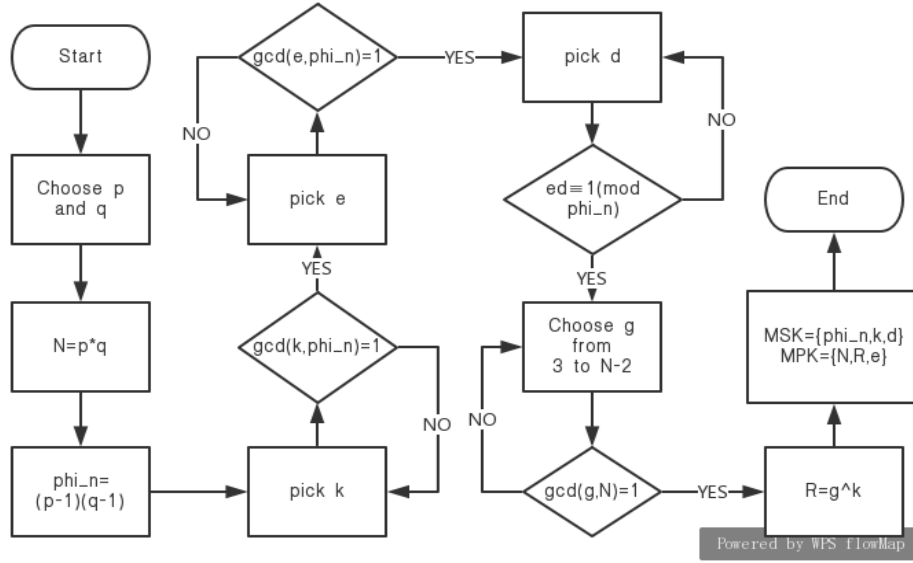


Figure 8. Setup algorithm

During this step, two primes p and q are chose via RSA algorithm. By multiplying these two primes, the value of N can be know. According to the value of N , $\varphi(N)$ can be computed as $\varphi(N) = (p-1)(q-1)$. Then choose system private key k such that $\gcd(k, \varphi(N)) = 1$ and RSA public exponent e_i such that $\gcd(e_i, \varphi(N)) = 1$. Thus, d_i can be computed by satisfying $e_i d_i \equiv 1 \pmod{\varphi(N)}$ corresponding to each attribute $A_i \in A, \forall i = 1, 2, \dots, n$. Last, choose a random integer g such that $\gcd(g, N) = 1$ and $g \in \{3, 4, \dots, N-2\}$, then compute the public parameter $R = g^k$. Then gain the master secret key MSK and master public key MPK respectively as:

$$MSK = \{\varphi(N), k, d_1, \dots, d_n\}$$

$$MPK = \{N, R, e_1, \dots, e_n\}$$

1) Choose:

- a. Two RSA primes p and q ($p \neq q$), then compute $N = p \cdot q$
- b. A system private key k such that $\gcd(k, \varphi(N)) = 1$
- c. The RSA public exponent e_i with $\gcd(e_i, \varphi(N)) = 1$, then compute d_i such that $e_i d_i \equiv 1 \pmod{\varphi(N)}$ with each attribute $A_i \in A, \forall i = 1, 2, \dots, n$.

- d. A random integer g such that $\gcd(g, N) = 1, 2 < g < N - 1$, then compute the public parameter $R = g^k$.

2) Generate:

- a. Master secret key MSK, $MSK = \{\varphi(N), k, d_1, \dots, d_n\}$.
- b. Master public key MPK, $MPK = \{N, R, e_1, \dots, e_n\}$.

② Key Generate

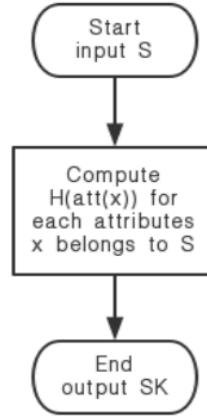


Figure 9. Key generation algorithm

During this algorithm, it output a secret key which identifies with a set of attributes S by computing $H(att(x))$ for each attribute $x \in S$. Thus, this secret key can be assigned as:

$$SK = \{k_x = H(att(x))/d_i, x \in S, i = \{order(x)\}\}$$

- 1) Input a set of attributes S
- 2) Compute $H(att(x))$ for each attribute $x \in S$
- 3) Output a key SK which identifies with the set of attributes S

$$SK = \{k_x = H(attr(x))/d_i, x \in S, i = \{order(x)\}\}$$

③ Encryption

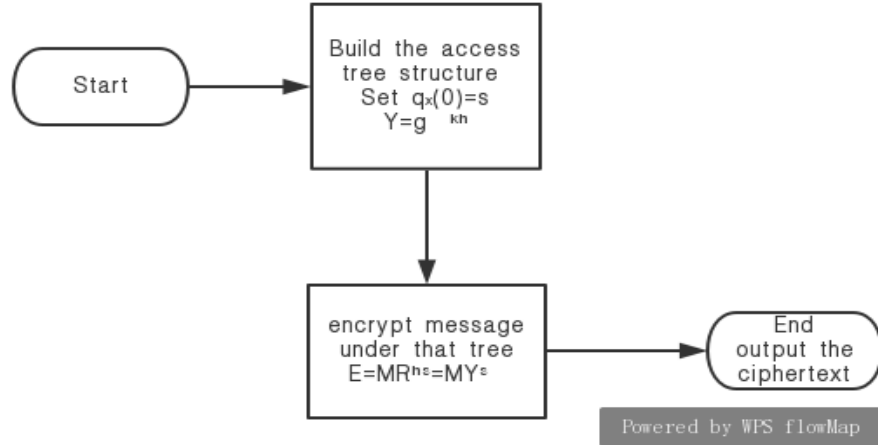


Figure 10. Encryption algorithm

This encryption algorithm uses the AES key to encrypt a message M under the access tree T . At the beginning, the encryption algorithm generates an access tree by choosing a Lagrange polynomial q_x for each non leaf node x in this access tree T , and choosing Y as the set of leaves in T . Then the algorithm chooses $h \in \mathbb{Z}_N$ and Y_m can be known by computing $Y_m = g^{kh}$. These polynomials are chosen in a top down manner starting from the root node R . For each node x in the access tree T , the threshold value k_x of the polynomial q_x is set to be one more than the degree d_x of that node, which is expressed as $d_x = k_x - 1$. For the root node R , the algorithm chooses a random $s \in \mathbb{Z}_N$ and sets $q_R(0) = s$. In order to define it completely, this algorithm randomly chooses other points d_R of the polynomial q_R . For any other node x , it sets $q_x(0) = q_{parent(x)}(index(x))$ and randomly chooses other points d_x to completely fix q_x . After ensuring the access structure, then the cipher-text CT is computed by giving the access tree T as follows:

- 1) Choose a random $s \in \mathbb{Z}_N$, and set $q_R(0) = s$, construct a polynomial q_x for each non-leaf node x in the tree T . Let Y be the set of leaves in T .
- 2) Choose a random $h \in \mathbb{Z}_N$, and compute:

$$Y_m = g^{kh}, CT: E = MR^{hs} = MY_m^s$$

$$\forall x \in Y: E_x = H(att(x)) \cdot q_x(0) \cdot e_i, \text{ where } i = \{order(x): x \in Y\}$$

3) Output the cipher-text:

$$CT = \{T, E, Y_m, \{E_x\}_{\forall x \in Y}\}$$

④ Decryption

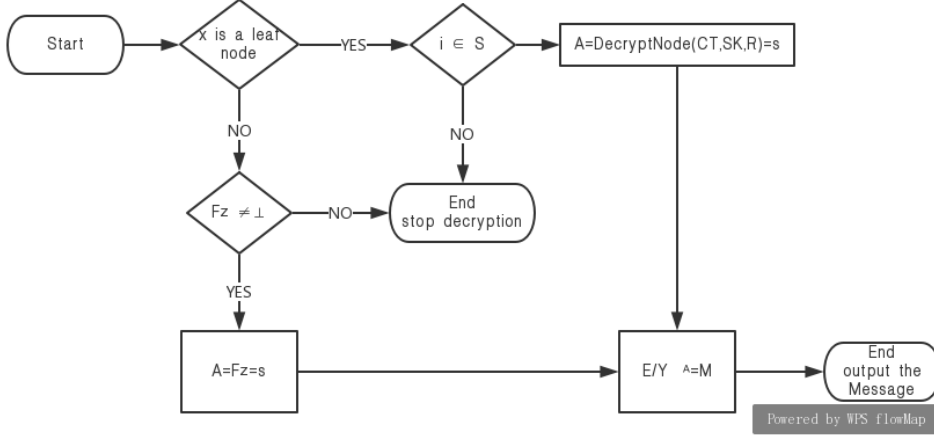


Figure 11. Decryption algorithm

If a set S of attributes satisfies the access tree T , then the decryption algorithm calls the function on the root node R of the access structure. This decryption algorithm specifies the decryption procedure as a recursive algorithm. The simple form of the decryption algorithm is presented below:

First define a function named $DecryptNode(CT, SK, x)$ which is a recursive algorithm and takes a cipher-text CT , a private key SK and a node x in access tree T as input. The cipher-text CT is defined as $CT = \{T, E, Y_m, \{E_x\}_{\forall x \in Y}\}$. The private key SK is associated with the set S of attributes.

1) Input a cipher-text $CT = \{T, E, Y_m, \{E_x\}_{\forall x \in Y}\}$

2) When x is a leaf node:

If $i \in S$, $i = \{order(x)\}$, then

$$DecryptNode(CT, SK, x) = \frac{H(att(x)) \cdot q_x(0) \cdot e_i}{H(att(x))/d_i} = q_x(0)$$

If $i \notin S$, then

$$\text{DecryptNode}(CT, SK, x) = \perp$$

3) When x is a non-leaf node, define the function $\text{DecryptNode}(CT, SK, z)$. It means for all nodes z that are children of node x calls that function and stores the output as F_z . Let s_x be a random k_x size set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns \perp , else:

$$\begin{aligned} F_z &= \sum_{z \in s_x} q_z(0) \cdot \Delta_{i, s_z}(0), \text{ where } i = \text{index}(z), s_z = \{\text{index}(z) : z \in s_x\} \\ &= \sum_{z \in s_x} q_{\text{parent}(z)}(\text{index}(z)) \cdot \Delta_{i, s_z}(0) \\ &= \sum_{z \in s_x} q_z(i) \cdot \Delta_{i, s_z}(0) \\ &= q_x(0) \\ &= s \end{aligned}$$

The last equation F_z is obtained by summing the polynomial interpolation. Therefore, Lagrange polynomial interpolation can be computed by doing multiplication in Z_N for each node $x \in S$, instead of exponentiating at each level.

Then the function DecryptNode is defined, and the decode algorithm can be implemented by calling DecryptNode function on the root node R of access tree T . If a set S of attributes satisfies the access tree T , then set $A = \text{DecryptNode}(CT, SK, R) = s$. Now the decryption algorithm decodes message by computing:

$$E / Y_m^A = E / g^{khs} = M$$

4.3 Security

This RSA-based Access-Tree CP-ABE scheme is proven to be secure under integer factorization and computational Diffie-Hellman(CDH) assumption. Integer factorization, also named as prime factorization, usually used in public-key encryption systems to keep the system secure. To use this function, a very large number is created by multiplying two prime numbers. This large number is used to secure the encryption system. It is easy to create the multiplication but not easy to find the prime factorization of the large number. This trap door function is used in many security systems. The computational Diffie-Hellman assumption is a computational hardness assumption about the

Diffie-Hellman problem which is a mathematical problem in cryptography. Similar to integer factorization, it uses mathematical operations that are easy to compute but difficult to reverse. Systems can be easily broken when it is easy to solve the Diffie-Hellman problem. As long as at least one of those two problems is hard to solve, the encryption system with very large size keys will be secure.

Chapter 5 Implementation

This chapter presents the implementation details of the RSA-based CP-ABE scheme. This includes the detailed implementation strategies for the four main algorithms of the RSA-based CP-ABE scheme performance evaluation and lessons learnt.

5.1 Overview

In the beginning, this chapter shows how to implement the RSA-based Access-Tree CP-ABE scheme into Android Studio, including the preparation and Android compilation environment. To fully compile the RSA-based Access-Tree CP-ABE to Android needs supports such as NDK-r9a, Android SDK and so on. Then the `android.mk` is required for compilation and required modification of the existing code, and connecting it to the JNI interface. Finally, compile of the RSA-based Access-Tree CP-ABE code along with dependent libraries is required to make it into the dynamic link library. The dynamic link library is now ready to be called by the Android system.

The four important algorithms that are mentioned before, such as set up, key generation, encryption and decryption, needs to be implemented. So next, it describes how those four algorithms are implemented and run in Android Studio. These algorithms are called by the main application program (named as `mainActivity`).

Once it finishes the coding part of this project, then it needs to be imported into an Android cell phone. This chapter also demonstrates how to install and run this encryption application on the hardware device, and presents how to use the functions in this application.

5.2 Preparation for CP-ABE

Implementing the RSA-based Access-Tree CP-ABE algorithm on the Android platform had a number of issues. The first issue is that this scheme is a program written in C language while it is required to be implemented on an Android platform. By using this external functional interface programming framework, Java code running in a Java virtual machine (VM) is able to call the native libraries that are written in other languages. Thus, this CP-ABE scheme written in C language can be used in the Android Studio after making it as a library. Then it can be called by any class through the defined name of

the native method.

To make this CP-ABE construction work, it needs the support of libraries such as PBC, GMP, M4, bison, flex, libbsswabe and so on. It is important step to prepare the libraries for running the CP-ABE scheme. In addition, there are some tools in Android Studio that also need to make the algorithm work such as Android SDK. Here is the list below.

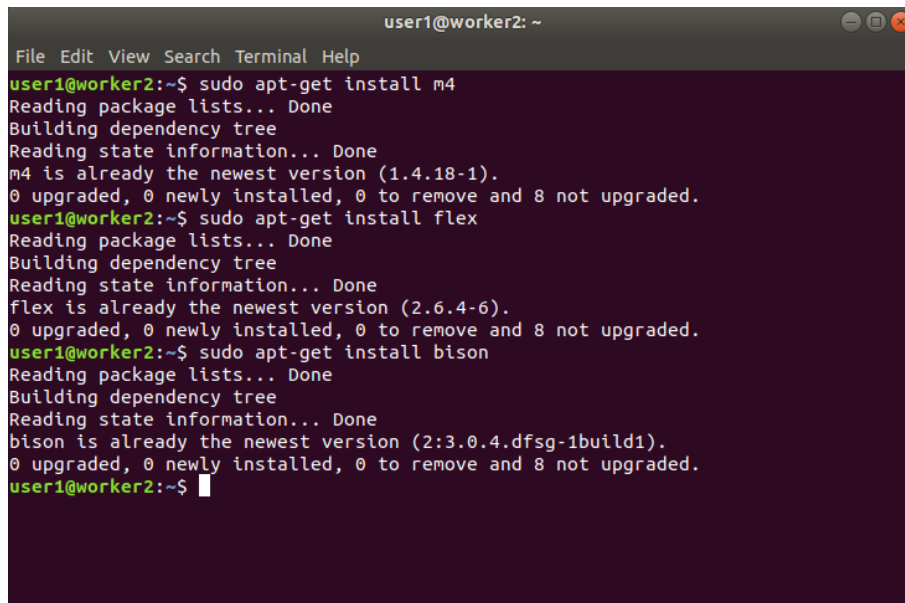
Download:

- M4, bison, flex (support GMP)
- GMP (supports PBC)
- PBC (supports CP-ABE)
- libbsswabe (supports CP-ABE)
- openssl, glib (support CP-ABE)
- NDK-r9a (allows C/C ++ code to be compiled into native code)

5.2.1 Setup Environment

It is an important step to build up the environment for the RSA-based Access-Tree CP-ABE scheme. Before using it, making sure the libraries are ready to use. First of all, a several libraries need to be installed to make sure the RSA-based Access-Tree CP-ABE algorithm can be run successfully. For example, the RSA-based Access-Tree CP-ABE crypto-system needs the PBC library while PBC requires GMP, and GMP is supported by other three libraries which are named M4, bison and flex respectively. After preparing those essential libraries, then the RSA-based Access-Tree CP-ABE scheme can be installed and run under this environment.

M4, Bison and Flex¹



```

user1@worker2: ~
File Edit View Search Terminal Help
user1@worker2:~$ sudo apt-get install m4
Reading package lists... Done
Building dependency tree
Reading state information... Done
m4 is already the newest version (1.4.18-1).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
user1@worker2:~$ sudo apt-get install flex
Reading package lists... Done
Building dependency tree
Reading state information... Done
flex is already the newest version (2.6.4-6).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
user1@worker2:~$ sudo apt-get install bison
Reading package lists... Done
Building dependency tree
Reading state information... Done
bison is already the newest version (2:3.0.4.dfsg-1build1).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
user1@worker2:~$

```

Figure 12. Install M4, bison and flex

M4 is an implementation of the traditional Unix macro processor [21]. Although it has some extensions, it is primarily compatible with SVR4. M4 also has built-in functions for performing integer operations, manipulating text in various ways, recursion, running UNIX commands, including named files, etc. Bison is an alternative to yacc, and it is a parser generator that generates a program to analyze the structure of text files [22]. Flex is an automatic lexical analyzer that is often used with Bison to mark input data and provide tokens for Bison [22]. Figure 12 shows how to install m4, flex and bison. First of all, using command “sudo apt-get install” to install M4, bison and flex in terminal, and these three libraries are installed successfully. Then the environment is ready for the GMP library.

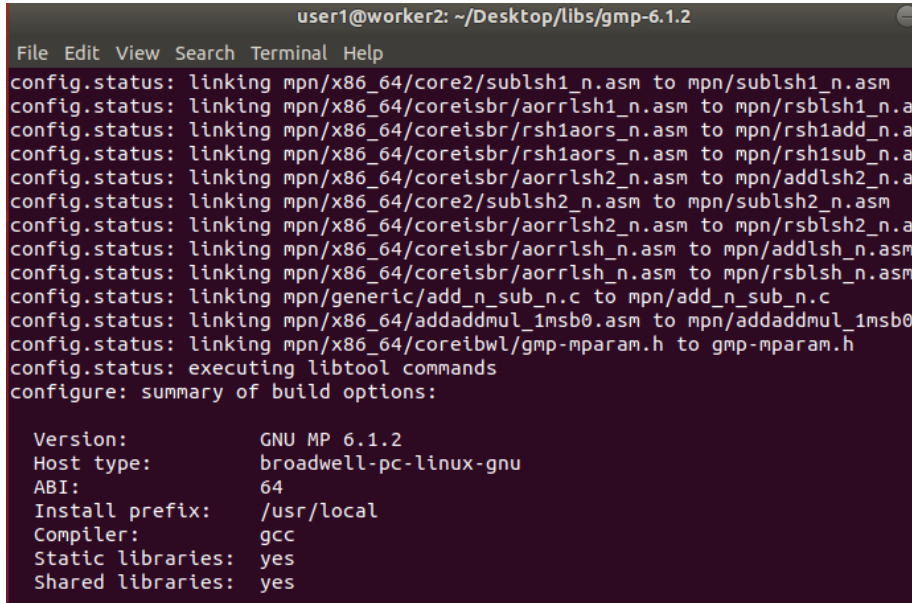
GMP²

GMP stands for GNU Multiple Precision which is a portable library written in C for arbitrary precision arithmetic on integers, rational numbers, and floating-point numbers [25]. The goal of GMP is to provide the fastest algorithms possible for all applications that require more precision than the basic C type directly supports. When operands are larger than the kernel, Maple uses the GMP library for integer operations. The GMP library is useful for fast multiple precision operations. The speed of GMP is achieved by using full words as a basic arithmetic type and complex algorithm, and including

¹ Download website: <http://ftp.gnu.org/gnu/m4/>; <http://ftp.gnu.org/gnu/bison/>; <http://ftp.gnu.org/gnu/flex/>

² Download website: <https://gmplib.org/>

assembly code optimized for the most common internal loops of many different CPUs. Maple is much faster when using the GMP library for long integer operations. The setup of GMP requires three steps which respectively are configure, make and install, and these three steps are listed below.



```

user1@worker2: ~/Desktop/libs/gmp-6.1.2
File Edit View Search Terminal Help
config.status: linking mpn/x86_64/core2/sublsh1_n.asm to mpn/sublsh1_n.asm
config.status: linking mpn/x86_64/coreisbr/aorrlsh1_n.asm to mpn/rsblsh1_n.a
config.status: linking mpn/x86_64/coreisbr/rsh1aors_n.asm to mpn/rsh1add_n.a
config.status: linking mpn/x86_64/coreisbr/rsh1aors_n.asm to mpn/rsh1sub_n.a
config.status: linking mpn/x86_64/coreisbr/aorrlsh2_n.asm to mpn/addlsh2_n.a
config.status: linking mpn/x86_64/core2/sublsh2_n.asm to mpn/sublsh2_n.asm
config.status: linking mpn/x86_64/coreisbr/aorrlsh2_n.asm to mpn/rsblsh2_n.a
config.status: linking mpn/x86_64/coreisbr/aorrlsh_n.asm to mpn/addlsh_n.asm
config.status: linking mpn/x86_64/coreisbr/aorrlsh_n.asm to mpn/rsblsh_n.asm
config.status: linking mpn/generic/add_n_sub_n.c to mpn/add_n_sub_n.c
config.status: linking mpn/x86_64/addaddmul_1msb0.asm to mpn/addaddmul_1msb0
config.status: linking mpn/x86_64/coreibwl/gmp-mparam.h to gmp-mparam.h
config.status: executing libtool commands
configure: summary of build options:

Version:          GNU MP 6.1.2
Host type:        broadwell-pc-linux-gnu
ABI:              64
Install prefix:   /usr/local
Compiler:         gcc
Static libraries: yes
Shared libraries: yes

```

Figure 13. Configure the GMP library

The first step of the setup GMP library is a configuration. The GMP library needs to be configured by executing “./configure” in the terminal, and the result is shown in Figure 13. Configure is a script that is usually supplied with the source code. It contains the code that patches and localizes the source distribution in order to compile and load on the system. In other words, configure is responsible for preparing the preparatory work of building the software on the system.

```

user1@worker2: ~/Desktop/libs/gmp-6.1.2
File Edit View Search Terminal Help
r.o mpn/sec_pi1_div_qr.o mpn/sec_pi1_div_r.o mpn/sec_add_1.o mpn/sec_sub_1.o mpn/
/sec_invert.o mpn/trialdiv.o .libs/libgmp.lax/lt105-remove.o mpn/and_n.o mpn/and
_n.o mpn/nand_n.o mpn/ior_n.o mpn/iorn_n.o mpn/nior_n.o mpn/xor_n.o mpn/xnor_n.
o mpn/copyi.o mpn/copyd.o mpn/zero.o mpn/sec_tabselect.o mpn/comb_tables.o mpn/i
nvert_limb.o mpn/sqr_diag_addlsh1.o mpn/mul_2.o mpn/addmul_2.o mpn/addlsh1_n.o m
pn/sublsh1_n.o mpn/rsblsh1_n.o mpn/rsh1add_n.o mpn/rsh1sub_n.o mpn/addlsh2_n.o m
pn/sublsh2_n.o mpn/rsblsh2_n.o mpn/addlsh_n.o mpn/rsblsh_n.o mpn/add_n_sub_n.o m
pn/addaddmul_1msb0.o printf/asprintf.o printf/asprintf.o printf/asprintf.o prin
tf/doprntf.o printf/doprnti.o printf/fprintf.o printf/obprintf.o printf/obvprin
t.o printf/obprntffuns.o printf/printf.o printf/printffuns.o printf/snprintf.o p
rintf/snprntffuns.o printf/sprintf.o printf/sprintf.o printf/sprintf.o printf/v
printf.o printf/vprintf.o printf/vsnprintf.o printf/vsprintf.o printf/repl
-vsnprintf.o scanf/doscan.o scanf/fscanf.o scanf/fscanf.o scanf/scanf.o scan
f/sscanf.o scanf/sscanf.o scanf/vfscanf.o scanf/vscanf.o scanf/vscanf.o ran
d/rand.o rand/randclr.o rand/randdef.o rand/randiset.o rand/randlc2s.o rand/rand
lc2x.o rand/randmt.o rand/randmts.o rand/rands.o rand/randsd.o rand/randsdui.o r
and/randbui.o rand/randmui.o
libtool: link: ranlib .libs/libgmp.a
libtool: link: rm -fr .libs/libgmp.lax
libtool: link: ( cd ".libs" && rm -f "libgmp.la" && ln -s "../libgmp.la" "libgmp
.la" )
make[2]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
make[1]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
user1@worker2:~/Desktop/libs/gmp-6.1.2$

```

Figure 14. Build the GMP library

After configuring the GMP library, “make” can be invoked to build the RSA-based Access-Tree CP-ABE scheme. In Figure 14, it shows the second step of the setup the GMP library that is using “sudo make” command in the terminal to build the library. The make utility is designed to automatically build the finished program from its source code according to a series of tasks defined in a “Makefile”.

```

user1@worker2: ~/Desktop/libs/gmp-6.1.2
File Edit View Search Terminal Help
-----
/bin/mkdir -p '/usr/local/include'
/usr/bin/install -c -m 644 gmp.h '/usr/local/include'
make install-data-hook
make[4]: Entering directory '/home/user1/Desktop/libs/gmp-6.1.2'

+-----+
| CAUTION:                                     |
| If you have not already run "make check", then we strongly |
| recommend you do so.                         |
|                                              |
| GMP has been carefully tested by its authors, but compilers |
| are all too often released with serious bugs.  GMP tends to |
| explore interesting corners in compilers and has hit bugs   |
| on quite a few occasions.                         |
+-----+

make[4]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
make[3]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
make[2]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
make[1]: Leaving directory '/home/user1/Desktop/libs/gmp-6.1.2'
user1@worker2:~/Desktop/libs/gmp-6.1.2$

```

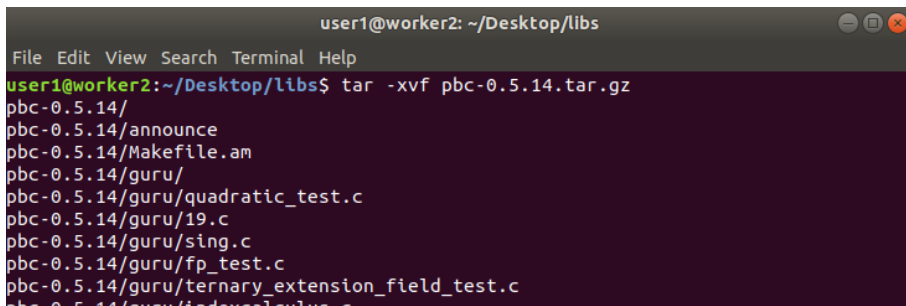
Figure 15. Install the GMP library

In Figure 15 shows the last step of the setup the GMP library in the local system that is using “sudo make install” command in the terminal to install

the GMP library. The make install command copies the built program, including its libraries and documentation, to the right locations.

PBC³

After installing GMP, the next step is to install the PBC library (Pairing-Based Cryptography). The PBC library is a GMP-based free C library that performs the mathematical operations underlying pairing-based crypto-system [26]. It is designed to be the core of implementations of pairing-based crypto-system. Therefore, speed and portability are significant goals. For example, some routines such as elliptic curve generation, elliptic curve arithmetic and pairing computation are provided by the PBC library. The pairings times are reasonable due to the GMP library without being written in C. In addition, the PBC library can also be used to build conventional crypto-system. The process of setup the PBC library is similar to GMP library. First of all, download this library and extract it, this is presented in Figure 16.



```

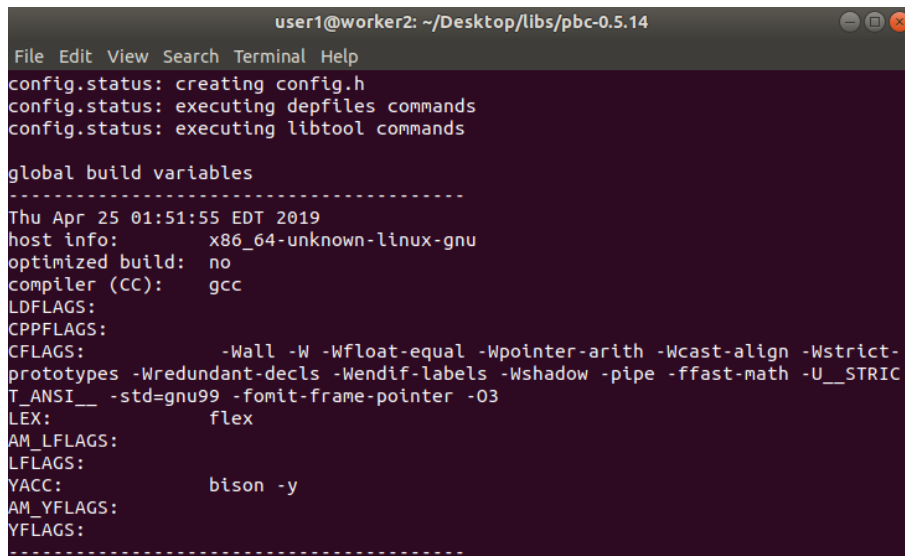
user1@worker2: ~/Desktop/libs
File Edit View Search Terminal Help
user1@worker2:~/Desktop/libs$ tar -xvf pbc-0.5.14.tar.gz
pbc-0.5.14/
pbc-0.5.14/announce
pbc-0.5.14/Makefile.am
pbc-0.5.14/guru/
pbc-0.5.14/guru/quadratic_test.c
pbc-0.5.14/guru/19.c
pbc-0.5.14/guru/sing.c
pbc-0.5.14/guru/fp_test.c
pbc-0.5.14/guru/ternary_extension_field_test.c
pbc-0.5.14/guru/ternary_test.c

```

Figure 16. Extract compressed file

In Figure 16, it shows the process of extracting the compressed PBC library from the terminal.

³ Download website: <https://crypto.stanford.edu/pbc/download.html>



```

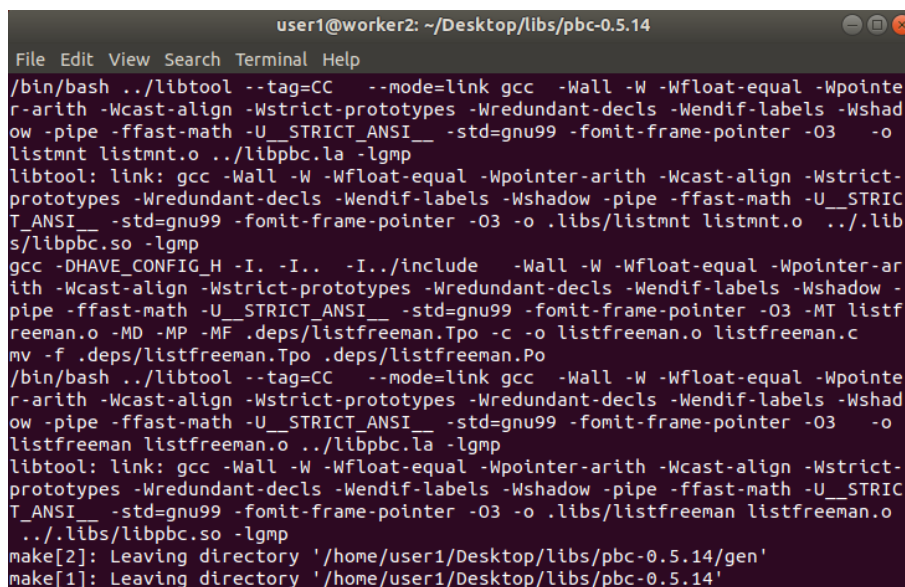
user1@worker2: ~/Desktop/libs/pbc-0.5.14
File Edit View Search Terminal Help
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands

global build variables
-----
Thu Apr 25 01:51:55 EDT 2019
host info:      x86_64-unknown-linux-gnu
optimized build: no
compiler (CC):  gcc
LDFLAGS:
CPPFLAGS:
CFLAGS:        -Wall -W -Wfloat-equal -Wpointer-arith -Wcast-align -Wstrict-
prototypes -Wredundant-decls -Wendif-labels -Wshadow -pipe -ffast-math -U__STRIC
T_ANSI__ -std=gnu99 -fomit-frame-pointer -O3
LEX:           flex
AM_LFLAGS:
LFLAGS:
YACC:          bison -y
AM_YFLAGS:
YFLAGS:
-----

```

Figure 17. Configure the PBC library

In Figure 17, it shows the results after running command “./configure”, and the PBC library is configured successfully.



```

user1@worker2: ~/Desktop/libs/pbc-0.5.14
File Edit View Search Terminal Help
/bin/bash ../libtool --tag=CC --mode=link gcc -Wall -W -Wfloat-equal -Wpointe
r-arith -Wcast-align -Wstrict-prototypes -Wredundant-decls -Wendif-labels -Wshad
ow -pipe -ffast-math -U__STRICT_ANSI__ -std=gnu99 -fomit-frame-pointer -O3 -o
listmnt listmnt.o ../libpbc.la -lgmp
libtool: link: gcc -Wall -W -Wfloat-equal -Wpointer-arith -Wcast-align -Wstrict-
prototypes -Wredundant-decls -Wendif-labels -Wshadow -pipe -ffast-math -U__STRIC
T_ANSI__ -std=gnu99 -fomit-frame-pointer -O3 -o .libs/listmnt listmnt.o ../lib
s/libpbc.so -lgmp
gcc -DHAVE_CONFIG_H -I. -I.. -I../include -Wall -W -Wfloat-equal -Wpointer-ar
ith -Wcast-align -Wstrict-prototypes -Wredundant-decls -Wendif-labels -Wshadow -
pipe -ffast-math -U__STRICT_ANSI__ -std=gnu99 -fomit-frame-pointer -O3 -MT listf
reeman.o -MD -MP -MF .deps/listfreeman.Tpo -c -o listfreeman.o listfreeman.c
mv -f .deps/listfreeman.Tpo .deps/listfreeman.Po
/bin/bash ../libtool --tag=CC --mode=link gcc -Wall -W -Wfloat-equal -Wpointe
r-arith -Wcast-align -Wstrict-prototypes -Wredundant-decls -Wendif-labels -Wshad
ow -pipe -ffast-math -U__STRICT_ANSI__ -std=gnu99 -fomit-frame-pointer -O3 -o
listfreeman listfreeman.o ../libpbc.la -lgmp
libtool: link: gcc -Wall -W -Wfloat-equal -Wpointer-arith -Wcast-align -Wstrict-
prototypes -Wredundant-decls -Wendif-labels -Wshadow -pipe -ffast-math -U__STRIC
T_ANSI__ -std=gnu99 -fomit-frame-pointer -O3 -o .libs/listfreeman listfreeman.o
../.libs/libpbc.so -lgmp
make[2]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14/gen'
make[1]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14'

```

Figure 18. Build the PBC library

Figure 18 presents the process of building this library.


```

user1@worker2: ~/Desktop/libs/pbc-0.5.14
File Edit View Search Terminal Help
./pbc_curve.h include/pbc_d_param.h include/pbc_e_param.h include/pbc_field.h include/pbc_multiz.h include/pbc_z.h include/pbc_fieldquadratic.h include/pbc_f_param.h include/pbc_g_param.h include/pbc_i_param.h include/pbc_fp.h include/pbc_ternary_extension_field.h include/pbc.h include/pbc_hilbert.h include/pbc_memory.h include/pbc_mnt.h include/pbc_pairing.h include/pbc_param.h include/pbc_poly.h include/pbc_random.h include/pbc_singular.h include/pbc_test.h include/pbc_utils.h /usr/local/include/pbc
make[2]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14'
make[1]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14'
Making install in example
make[1]: Entering directory '/home/user1/Desktop/libs/pbc-0.5.14/example'
make[2]: Entering directory '/home/user1/Desktop/libs/pbc-0.5.14/example'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14/example'
make[1]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14/example'
Making install in gen
make[1]: Entering directory '/home/user1/Desktop/libs/pbc-0.5.14/gen'
make[2]: Entering directory '/home/user1/Desktop/libs/pbc-0.5.14/gen'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14/gen'
make[1]: Leaving directory '/home/user1/Desktop/libs/pbc-0.5.14/gen'

```

Figure 19. Install the PBC library

The result of installing this library is shown in Figure 19. At this point, the PBC environment configuration is completed.

A test for whether the setup environment is built successfully is followed. First, there are some cases in PBC-0.5.14 - example directory, then copy a “.c” file and customize the name “foo.c”. Then change the contents into the content of Figure 20 below:

```

Open  foo.c  Save
~/Desktop/libs/pbc-0.5.14/example
#include "pbc.h"
int main(void)
{
    printf("this is a test\n");
    return 0;
}

```

Figure 20. Test code

Compile the file “foo” by using command which is contained in Figure 21 below:

```

user1@worker2:~/Desktop/libs/pbc-0.5.14/example$ gcc -o foo foo.c -I ~/Desktop/libs/pbc-0.5.14/include -L ~/Desktop/libs/pbc-0.5.14/pbc -Wl,-R ~/Desktop/libs/pbc-0.5.14/pbc -l pbc
user1@worker2:~/Desktop/libs/pbc-0.5.14/example$ ./foo
this is a test

```

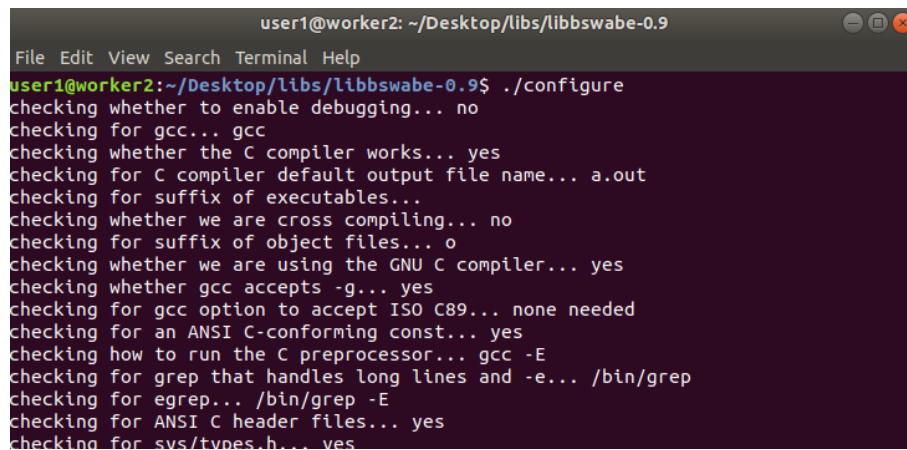
Figure 21. Compile file “foo” and run

When it is compiled successfully, run the file “foo” by using command “./foo” which is shown in Figure 21 as well. Then it shows the result “this is a test”,

and this means that the PBC environment configuration succeeded.

Libbswabe⁴

Libbswabe is a library that implements the core crypto operations. To install the Libbswabe library, there need three steps: configure, build and install, which are totally same as GMP library. Those steps are shown below:



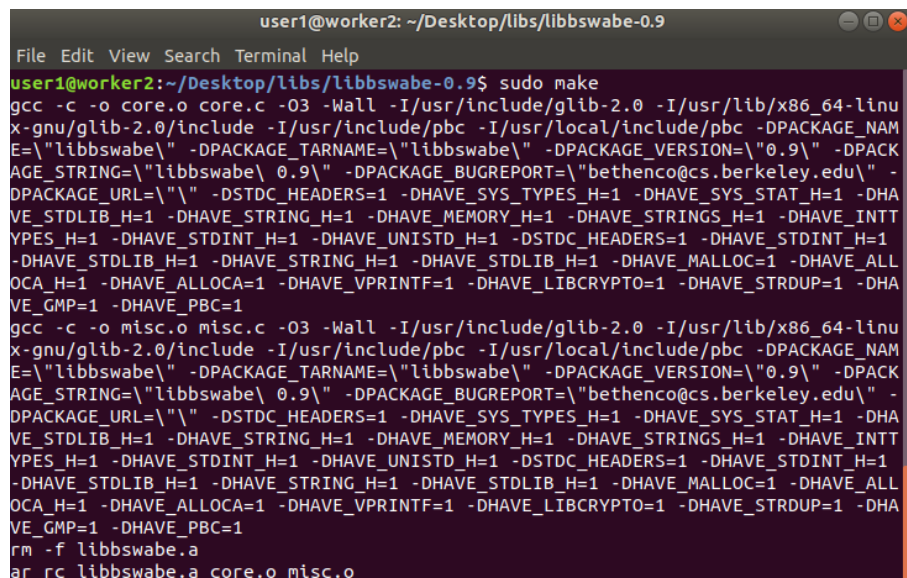
```

user1@worker2: ~/Desktop/libs/libbswabe-0.9
File Edit View Search Terminal Help
user1@worker2:~/Desktop/libs/libbswabe-0.9$ ./configure
checking whether to enable debugging... no
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for an ANSI C-conforming const... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes

```

Figure 22. Configure the Libbswabe library

Figure 22 shows the result after executing the command “./configure”.



```

user1@worker2: ~/Desktop/libs/libbswabe-0.9
File Edit View Search Terminal Help
user1@worker2:~/Desktop/libs/libbswabe-0.9$ sudo make
gcc -c -o core.o core.c -O3 -Wall -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -I/usr/include/pbc -I/usr/local/include/pbc -DPACKAGE_NAME="libbswabe" -DPACKAGE_TARNAME="libbswabe" -DPACKAGE_VERSION="0.9" -DPACKAGE_STRING="libbswabe 0.9" -DPACKAGE_BUGREPORT="bethenco@cs.berkeley.edu" -DPACKAGE_URL="" -DSTDC_HEADERS=1 -DHAVE_SYS_TYPES_H=1 -DHAVE_SYS_STAT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_MEMORY_H=1 -DHAVE_STRINGS_H=1 -DHAVE_INTTYPES_H=1 -DHAVE_STDINT_H=1 -DHAVE_UNISTD_H=1 -DSTDC_HEADERS=1 -DHAVE_STDINT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MALLOC=1 -DHAVE_ALL_OCA_H=1 -DHAVE_ALLOCA=1 -DHAVE_VPRINTF=1 -DHAVE_LIBCRYPTO=1 -DHAVE_STRDUP=1 -DHAVE_GMP=1 -DHAVE_PBC=1
gcc -c -o misc.o misc.c -O3 -Wall -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -I/usr/include/pbc -I/usr/local/include/pbc -DPACKAGE_NAME="libbswabe" -DPACKAGE_TARNAME="libbswabe" -DPACKAGE_VERSION="0.9" -DPACKAGE_STRING="libbswabe 0.9" -DPACKAGE_BUGREPORT="bethenco@cs.berkeley.edu" -DPACKAGE_URL="" -DSTDC_HEADERS=1 -DHAVE_SYS_TYPES_H=1 -DHAVE_SYS_STAT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_MEMORY_H=1 -DHAVE_STRINGS_H=1 -DHAVE_INTTYPES_H=1 -DHAVE_STDINT_H=1 -DHAVE_UNISTD_H=1 -DSTDC_HEADERS=1 -DHAVE_STDINT_H=1 -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MALLOC=1 -DHAVE_ALL_OCA_H=1 -DHAVE_ALLOCA=1 -DHAVE_VPRINTF=1 -DHAVE_LIBCRYPTO=1 -DHAVE_STRDUP=1 -DHAVE_GMP=1 -DHAVE_PBC=1
rm -f libbswabe.a
ar rc libbswabe.a core.o misc.o

```

Figure 23. Build the Libbswabe library

Then run command “sudo make” to build the Libbswabe library, and the result of this step is shown in Figure 23.

⁴ Download website: http://www.verysource.com/code/23755648_1/bswabe.h.html

```

user1@worker2:~/Desktop/libs/libbswabe-0.9$ sudo make install
./mkinstalldirs -m 755 /usr/local/lib
./mkinstalldirs -m 755 /usr/local/include
./install-sh -m 755 libbswabe.a /usr/local/lib
./install-sh -m 644 bswabe.h /usr/local/include

```

Figure 24. Libbswabe sudo make install

The last step which is shown in Figure 24 is to install the Libbswabe library by executing the command “sudo make install”. Then the Libbswabe library is ready to be used.

Openssl and Glib⁵

Before installing CP-ABE, the libraries openssl and glib are required. OpenSSL is a robust, commercial, and fully functional toolkit for transport layer security (TLS) and secure sockets layer (SSL) protocols [27]. It is also a universal cryptography library. Glib is a bundle of three low-level system libraries. It provides advanced data structures, such as memory chunks, and implements functions that provide threads, thread programming and related facilities [28]. Glib also includes message passing facilities such as byte order conversion.

```

user1@worker2:~/Desktop/libs$ sudo apt-get install libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libssl-dev is already the newest version (1.1.0g-2ubuntu4.3).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
user1@worker2:~/Desktop/libs$ sudo apt-get install libglib2.0-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libglib2.0-dev is already the newest version (2.56.3-0ubuntu0.18.04.1).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
user1@worker2:~/Desktop/libs$

```

Figure 25. Install openssl and glib

Figure 25 shows how to install both libraries openssl and glib by using command “sudo apt-get install”.

RSA-based CP-ABE

After preparing all libraries, the preparation of the RSA-based Access-Tree CP-ABE scheme is ready, now the setup of this scheme runs. It can be done by executing “sudo make” as the following figure.

⁵ Download website: <https://openssl.en.softonic.com/>; <https://sourceforge.net/projects/glib/>

```

user1@worker2: ~/Desktop/libs/cpabe
File Edit View Search Terminal Help
dec.c:362:19: warning: unused variable 'attrbit' [-Wunused-variable]
    int file_len, i, attrbit, ai_bi;
                        ^
dec.c:362:16: warning: unused variable 'i' [-Wunused-variable]
    int file_len, i, attrbit, ai_bi;
                   ^
dec.c:356:19: warning: variable 'pub' set but not used [-Wunused-but-set-variable]
    treecpabe_pub_t *pub;
In file included from dec.c:5:0:
cpabe_rsa.h: In function '_suck_file':
cpabe_rsa.h:121:2: warning: ignoring return value of 'fread', declared with attribute warn_unused_result [-Wunused-result]
    fread(in_buf,1,s.st_size,f);
    ^
At top level:
cpabe_rsa.h:93:13: warning: 'squeeze' defined but not used [-Wunused-function]
    static void squeeze(char s[], int c)
gcc -o dec dec.o common.o -O3 -Wall -lglib-2.0 -L/usr/local/lib -lbswabe -lgmp -lcrypto -lpbc

```

Figure 26. Build CP-ABE

Figure 26 shows the result after building the RSA-based Access-Tree CP-ABE scheme. In order to test whether this algorithm can work or not under the environment, run the command “cpabe-setup -h” in the terminal.

```

user1@worker2:~/Desktop/libs/cpabe$ cpabe-setup -h
Usage: cpabe-setup [OPTION ...]

Generate system parameters, a public key, and a master secret key
for use with cpabe-keygen, cpabe-enc, and cpabe-dec.

Output will be written to the files "pub_key" and "master_key"
unless the --output-public-key or --output-master-key options are
used.

Mandatory arguments to long options are mandatory for short options too.

  -h, --help                print this message
  -v, --version              print version information
  -p, --output-public-key FILE write public key to FILE
  -m, --output-master-key FILE write master secret key to FILE
  -d, --deterministic        use deterministic "random" numbers
                             (only for debugging)

```

Figure 27. Test CP-ABE scheme

Figure 27 shows the process of setup algorithm in this RSA-based Access-Tree CP-ABE scheme. It not only presents the inputs and outputs during this algorithm, but some toolkit functions as well. This means that the CP-ABE algorithm is running well, and the environment is built successfully.

Android NDK

The Android NDK is a set of tools that allows developers to get the most performance out of devices [29]. For example, NDK enables developers to call functions in C/C++ code in Android system, and it offers platform libraries for users to manage native activities and access physical device components, such as sensors and touch input [30].

The core of the Android NDK is the ndk-build script, and it is responsible for:

- Automatically browsing the project.
- Determining what to build.
- Generating binaries.
- Copying generated binaries on to an apps project path.

The NDK also is responsible for linking native shared libraries (.so) against other libraries. The native shared libraries are built from the native source code and Native static libraries (.a). By using these generated .so files, the Application Binary Interface (ABI) is able to understand both when to run the application and how the application machine code works with the system. ARMABI, MIPS, and x86 are supported by NDK in default. Java Native Interface (JNI) that connects between Java and C/C++ is introduced here.

Android.mk and Application.mk are the two primary files that used to build the ndk-build script. Respectively, the Android.mk file needs to go in JNI folder and defines:

- the module including it's name.
- the build flags (which libraries link to).
- what source files need to be compiled.

As for the Application.mk file, it is the same as the Android.mk that they all go in the jni directory. The difference is that it describes the native modules required by the application.

To make the NDK work, download LLDB (a debugger that Android Studio uses to debug native code) and NDK from the SDK manager. By providing a path to the ndk-build script file, consequently, Gradle is linked to the native library

(.so). Then Gradle imports source code into the Android Studio project and package the native library (.so) into the APK by using the build script [30].

Android Studio

Build “com.example.cpabe.NativeCPABE”, then load libraries and define native method. Put all the libraries generated above into the “libs/armeabi” directory of the project. Now the environment is ready, and it can be called in MainActivity.

It is found that three secret key files are generated under the SD card directory, indicating that the library is loaded normally and the native code also runs successfully.

The core of CP-ABE scheme is encrypting files, therefore, when clicking the button, it writes the content in the original input box into “/sdcard/to_enc.txt”, and then calls the enc method of the native layer to get the encrypted file. After obtaining the encoded text, use binary to read it into the second text box. Then click the “DECRYPT” button, and the dec function is called to get the decoded file, at the same time, the content is read into the third text box.

5.2.2 Java Native Interface (JNI)

Java Native Interface(JNI) provides a solution for the byte code that Android compiles from managed code written in Java to interact with native code written in C or C++ [20]. When an application is unable to be written entirely in the Java programming language, the JNI allows the programmer to write native methods to handle situations.

In the situation of this thesis where the RSA-based Access-Tree CP-ABE written in C programming language can be used in Android Studio by modifying it to be accessible to Java application. Based on JNI, many library classes are able to provide functions in a safe and platform-independent manner to the user. JNI allows a native method to use Java objects just the same as how Java code uses those objects. Java objects can be created and be used to perform tasks by a native method. Java application code also can create objects, and those objects can be inspected and be used by a native method as well. JNI only can be invoked by applications and signed applets. JNI allows using dynamic shared libraries to load code, and this is an efficient way.

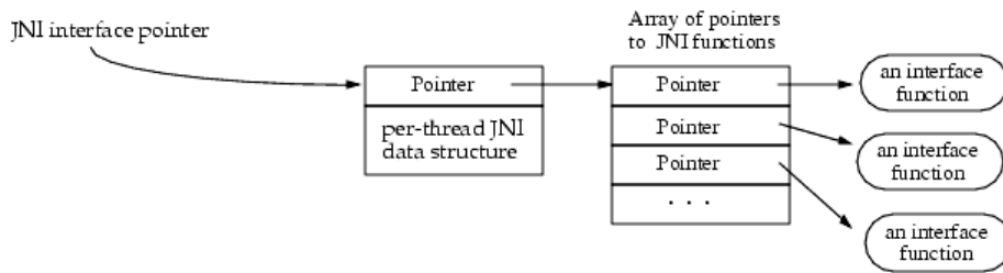


Figure 28. Interface pointer [20]

The Interface pointer is the basic of JNI function. Figure 20 presents that the interface pointer is a pointer to an array of pointers. Inside this array, each pointer points to an interface function which is predefined offset, JNI namespace is separated from native code, it means that the virtual machine (VM) is able to provide multiple versions of the JNI menu easily. That is the superiority of using interface tables.

“System.loadLibrary” is the method that loads native methods [20]. For example, in a class “NativeCPABE”, it defines the native method “setup”, and the class initialization method also loads the platform-specific native library “cpabe”. The code of this example is listed below:

```

package pkg;
public class NativeCPABE {
    static {
        System.loadLibrary("cpabe");
    }
    public native double setup(
        String pubFile,
        String mskFile,
        int parameters_type);
}

```

As it can be seen in the above example, the library name is defined arbitrarily when loading a library. Nevertheless, there is a standard for the system. It helps the system to convert the library name to a native library name. With the classes loaded in a same class loader, all the native methods called by any number of classes can be stored in a single library. The list of loaded native libraries for each class loader is maintained by the VM internally [20]. Thus, it is better to use the native library names that avoid the name clashes. There is one situation that the VM calls the “System.loadLibrary” with loading no library. That is the dynamic linking is not supported by the underlying operating system. To solve this problem, it should prelink all native methods

with the VM.

The dynamic linker parses an entry based on its name. The native method name is composed of the following components:

- The prefix `Java_`.
- A mangled fully-qualified class name.
- An underscore (`_`) separator.
- A mangled method name.
- Using two underscores (`__`) followed by the mangled argument signature for overloaded native methods.

JNI offers a rich set of accessor functions on global and local references which means that the programmer can implement Java objects using the same native methods regardless of how they are represented internally by the VM. This is the significant reason why many VM implementations support JNI. Using accessor functions through opaque references is more expensive than accessing C data structures directly.

Native code is allowed to access the fields and to call the methods of Java objects through JNI function. JNI function identifies methods and fields by their symbolic names and type signatures. The two-step process removes the cost of locating a field or method from the field name and signature.

5.3 App on Android Studio

① Overview

There are three main parts of this program: class `“NativeCPABE”`, class `“mainActivity”`, and `“activity_main.xml”`. These three parts build up the main body of this app. Because of the code of the RSA-based Access-Tree CP-ABE algorithm is not written in Java, and it could not be used in Android Studio directly, therefore, JNI is applied to this program to connect C and Java. Under the `“jniLibs\armeabi”` path, the main native library `cpabe` and its supporting libraries are stored. These libraries are already been compiled. Thus, the native library `cpabe` is ready to be called.

② Interface Design Window

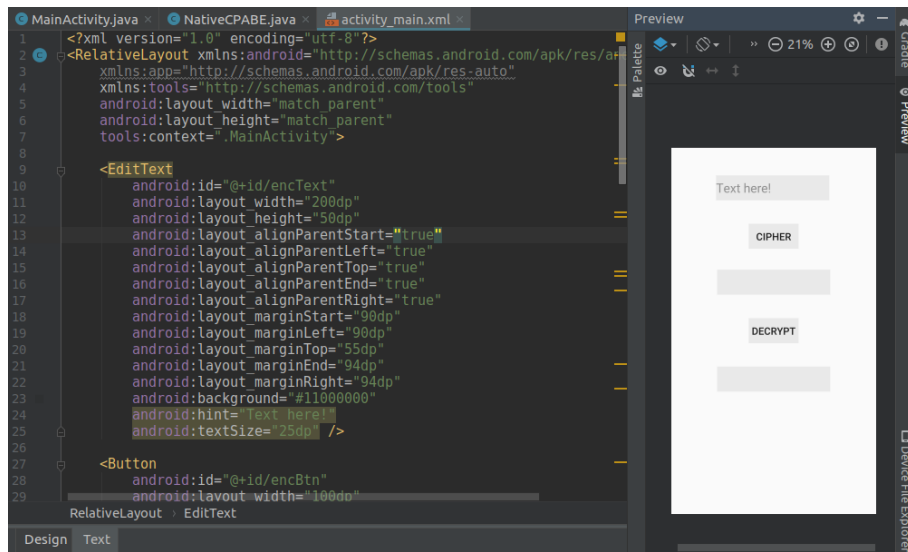


Figure 29. Interface design window

The interface of this app is shown in “activity_main.xml”, and the text boxes and buttons are stipulated by the code on the left of Figure 29. Not only the size, position and background, but also the ID of each text boxes and buttons are defined. The ID is used in the “mainActivity” package to call a text box or a button.

③ native CP-ABE package

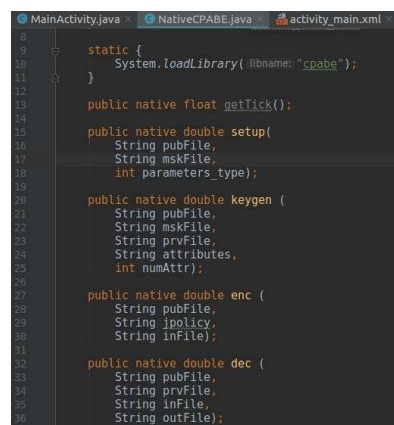
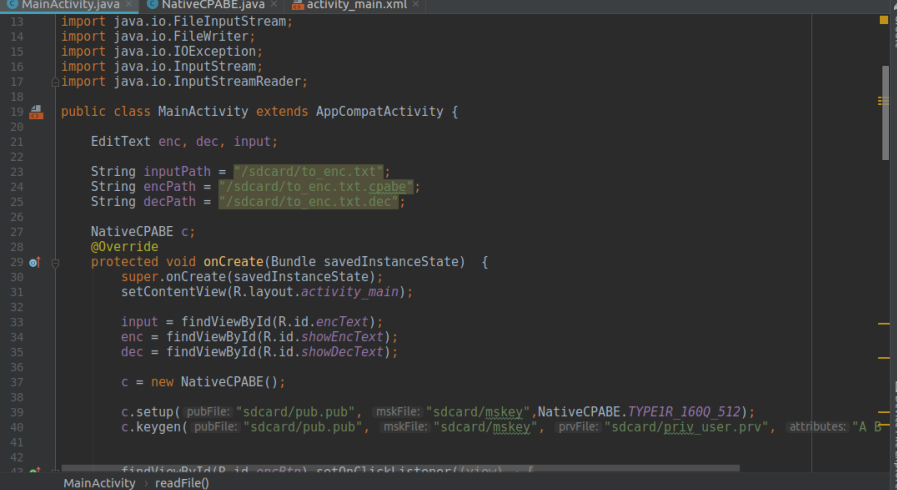


Figure 30. Native CP-ABE

The Native CP-ABE package showed in Figure is used to load the CP-ABE library and build up the four main functions of this algorithm: set up, key generation, encryption and decryption.

④ MainActivity



```

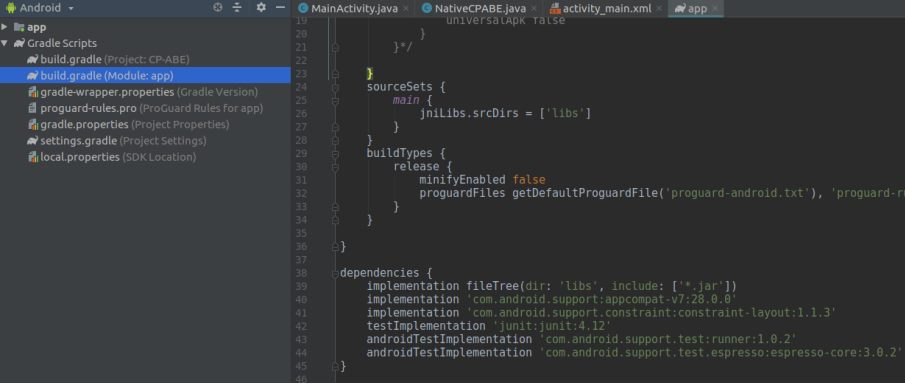
13 import java.io.FileInputStream;
14 import java.io.FileOutputStream;
15 import java.io.IOException;
16 import java.io.InputStream;
17 import java.io.InputStreamReader;
18
19 public class MainActivity extends AppCompatActivity {
20
21     EditText enc, dec, input;
22
23     String inputPath = "/sdcard/to_enc.txt";
24     String encPath = "/sdcard/to_enc.txt.cpabe";
25     String decPath = "/sdcard/to_enc.txt.dec";
26
27     NativeCPABE c;
28     @Override
29     protected void onCreate(Bundle savedInstanceState) {
30         super.onCreate(savedInstanceState);
31         setContentView(R.layout.activity_main);
32
33         input = findViewById(R.id.encText);
34         enc = findViewById(R.id.showEncText);
35         dec = findViewById(R.id.showDecText);
36
37         c = new NativeCPABE();
38
39         c.setup( pubFile: "sdcard/pub.pub", mskFile: "sdcard/mskey", NativeCPABE.TYPEIR_1600_512);
40         c.keygen( pubFile: "sdcard/pub.pub", mskFile: "sdcard/mskey", prvFile: "sdcard/priv_user.prv", attributes: "A B");
41
42         MainActivity.readFile()

```

Figure 31. Main activity widow

In the mainActivity window, the main functions are combined together to make the project working. In this class, the functions of three text boxes are built, and link to the text boxes ID made in “activity_main.xml”, such as encText, showEncText and showDecText. Two functions of two buttons are defined respectively as well. The encryption button is connected with the encryption algorithm in nativeCPABE, so that this “CHIPHER” button can call the encrypt function when it is clicked. The decryption button is the same, but to call the decrypt function.

⑤ The code of build gradle application module



```

19
20
21 */
22
23 }
24 sourceSets {
25     main {
26         jniLibs.srcDirs = ['libs']
27     }
28 }
29 buildTypes {
30     release {
31         minifyEnabled false
32         proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
33     }
34 }
35
36
37
38 dependencies {
39     implementation fileTree(dir: 'libs', include: ['*.jar'])
40     implementation 'com.android.support:appcompat-v7:28.0.0'
41     implementation 'com.android.support.constraint:constraint-layout:1.1.3'
42     testImplementation 'junit:junit:4.12'
43     androidTestImplementation 'com.android.support.test:runner:1.0.2'
44     androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
45 }

```

Figure 32. Build Gradle(Module: app)

In build.gradle, the library “libs” is connected by defining JNI connected it with native JAVA code.

5.4 Implementation Algorithm

Before utilizing for the four main algorithms, there is some preparation to do. First of all, JNI is required to connect Java and C language. These two languages are connected by JNI in “build.gradle” where to set the path to JNI, and it is presented in Figure 44. To define the path as “libs” by using the code below:

```
sourceSets {
    main {
        jniLibs.srcDirs = ['libs']
    }
}
```

Then the RSA-based Access-Tree CP-ABE library is called by the system in the NativeCPABE class by using “loadLibrary” function. In this class, it also defines the native methods that need by the other class.

```
static {
    System.loadLibrary("cpabe");
}
```

In MainActivity, the paths that save the input file, the encryption file and the decryption file are defined as below. This means the input text is stored in “to_enc” file inside the sdcard, the encrypted text is stored in the file named “to_enc.txt.cpabe”, and the decoded file is stored in the “to_enc.txt.doc” file. In addition, in this part of code, it also defines the NativeCPABE as c which is used later.

```
String inputPath = "/sdcard/to_enc.txt";
String encPath = "/sdcard/to_enc.txt.cpabe";
String decPath = "/sdcard/to_enc.txt.doc";
NativeCPABE c;
```

The text boxes are built on the interface of this application, and these boxes are also defined in MainActivity by finding the ID of each box from activityMain window, and the Native CP-ABE function is called as well.

```
input = findViewById(R.id.encText);
enc = findViewById(R.id.showEncText);
dec = findViewById(R.id.showDecText);
c = new NativeCPABE();
```

After finishing these steps above, the path of JNI libraries is defined, and the CPABE scheme written in C is loaded in the system and is ready to be called. Besides these, the three text boxes where to show the message, the cipher-text and decoded message respectively are defined as well by connecting the ID name in the interface design window. The preparation of those four significant algorithms is done, then they can be used later simply by using calling function.

5.4.1 Setup

① Define native function setup in the NativeCPABE class:

```
public native double setup(
    String pubFile,
    String mskFile,
    int parameters_type);
```

In this setup process, it has three contents. Two of them are strings, these strings respectively are “pubfile” which stores the public key and “mskfile” which stores the master key. The other is an integer that presents the security parameter. This native method defines the function name which is called by the MainActivity class later.

This native method is defined in the NativeCPABE class where the native library is loaded. This method is called by the MainActivity class to run the setup function. To make the function work, this NativeCPABE class is used to connect the native library named “cpabe” and the native method “setup”. There is where JNI works. In other words, after loading the native library and defining the method name, native code is allowed to access the fields and to call the methods of Java objects through JNI function.

② Call setup function in MainActivity:

```
c.setup("sdcard/pub.pub", "sdcard/mskey", K);
```

After connecting the native code with the native library, then the native method can be called in the MainActivity class. In this calling function, the setup algorithm contains the path of public key and master key which are stored in "sdcard/pub.pub" and "sdcard/mskey" severally into a cell phone. "K" is a kind of security parameter which usually is the length of key.

Let's define the security parameter as k, and the performance of attacks is

set as $O(\frac{1}{2^k})$. It is easy to see, the bigger the security parameter, the smaller the chance for an attack. The size of k is usually defined at least 1024 bits. In this RSA-based Access-Tree CP-ABE algorithm, it chooses a half value of k to be the size of two prime numbers p and q which is 512 bits for each prime. After multiplying these two numbers, the size of their product is 1024 bits which is considered as the lowest key size for the security, and 2048 bits is recommended.

When calling this function, the Android system calls the native function setup from the native library “cpabe”. When running this method, according to the CP-ABE construction written by Li, it generates the master secret key and public key by using RSA algorithm as well as a series of mathematical operations based on a chosen secret parameter which is 1024 in this situation. Then, these outputs are stored into the phone memory according to the specified path. Then calling the setup method is completed.

③Algorithm description:

This algorithm mainly based on the RSA key generation algorithm is introduced bellow in details:

Algorithm 1 RSA Key Generation Algorithm	
1: Procedure KEY GENERATION	
Output: public key, private key	
2: RSA_keygeneration (pub_key,priv_key)	
3: Select a random prime number p with size $N/2$;	
4: Select a random prime number q with size $N/2$;	
5: $\phi = (p-1)(q-1)$;	
6: modulus = pq ;	
7: Select a random e from 1 to ϕ ;	
8: if $\gcd(e, \phi) = 1$ then	
9: pub_key = e ;	
10: Select a random d ;	
11: if $d \equiv e^{-1} \pmod{\phi}$ then	
12: priv_key = d ;	
13: return pub_key, priv_key;	

Table 1. RSA pseudo-code [19]

The input of this setup process is security parameter and universal attribute set. It takes null or document path of attribute universe as the input argument vector. By using PBC library, randomly choose two numbers p and q of $k/2$ bits length:

```
PBC_mpz_randomb(p, 512);
PBC_mpz_randomb(q, 512);
```

This step follows the first process of RSA key generation algorithm, and creates two required prime numbers.

```
mpz_nextprime(p, p);
mpz_nextprime(q, q);
```

Set the previous p to the next prime greater than the following p, as well as q.

```
mpz_mul(N, p, q);
```

By multiplying these two prime numbers p and q, the value of N can be known as below, and N is 1024 bits length. This step follows the RSA key generation algorithm to generate the modulu N.

```
mpz_sub_ui(p, p, 1);
mpz_sub_ui(q, q, 1);
mpz_mul(phi_n, p, q);
```

Set the new p as old p minus one, and compute phi_n as new p multiplies new q which is respectively 1 less than before. This step computes the value of phi_n which follows the process of RSA key generation algorithm.

```
mpz_set(msk->phi_n, phi_n);
mpz_set(pub->N, N);
```

Set phi_n into master key, and set N into public key.

In order to compute R, g needs to be found as following steps:

```
do {
    element_random(g);
    element_to_mpz(mpz_g, g);
    mpz_gcd(p, mpz_g, N);
} while (mpz_cmp_d(p, 1));
```

Randomly pick an element g and convert it into multi-precision such as mpz_g. Find the greatest common factor between mpz_g and N and set it as p, if p is equal to 1, it means the g which is co-prime with N is found and stops the loop.

Similar to the process above, select random x with gcd(x, phi_n)=1.

```

do {
    PBC_mpz_random(msk->x, N);
    mpz_gcd(q, msk->x, phi_n);
}while (mpz_cmp_d(q, 1));

```

Find the greatest common factor between `msk->x` and `phi_n` and set it as `q`, if `q` is equal to 1, it means `x` is co-prime with `phi_n` and stop the loop.

Compute $R = gx = g^x$:

```

element_pow_mpz(pub->gx, g, msk->x);

```

Set `x` into the master key and `R` into the public key;

Parse attributes stored in the string buffer as follows;

```

int parse_attribute_list(char * attrs_str){
    squeeze(attrs_str, '\n');
    if ((substr = strtok(attrs_str, BLANK)) == NULL)
        parse_attribute(&attrlist, substr);
    while(substr){
        parse_attribute(&attrlist, substr);
        substr = strtok(NULL, BLANK);
    }
    attrlist = g_slist_sort(attrlist, comp_string);
    n = g_slist_length(attrlist);
    attrs = malloc((n + 1) * sizeof(char*));
    for( ap = attrlist; ap; ap = ap->next )
        attrs[i++] = ap->data;
    attrs[i] = 0;
}

```

First of all, formalize all strings by squeezing out the “\n” and “blank”. Then store the strings in the list by string comparisons. In the end, get a fixed-length array of string where the attribute strings are placed in order.

There are two parts of this code, first part is to pick the RSA exponent e , and the second part is to compute the multiplication inverse d .

```
for (i=0; i< num_attrs; i++) {
    do {
        PBC_mpz_random(pub->ei[i], N);
        mpz_gcd(divisor, pub->ei[i], phi_n);
    }while (mpz_cmp_d(divisor, 1));
    mpz_invert(msk->di[i],
        pub->ei[i], msk->phi_n);
}
```

The process of finding e is similar like g and x . Randomly choose e such that each of e satisfied $\gcd(e, \phi_n)=1$. The difference is that attributes are added into each e . Once the e is know, it is contained into public key, and d can be computed by $e * d \bmod \phi(n) = 1$.

[illegible]

Figure 33. Setup

In setup process, it randomly picks the value of p and q and then gets the rest of elements from these two values. Those elements and their values are shown in Figure 33 above. As seen in Figure 33, the length of public key “pub” is 528, and the size of master key is defined as 264.

5.4.2 Key Generation

① Define native function keygen in the NativeCPABE class:

```
public native double keygen (
    String pubFile,
    String mskFile,
    String prvFile,
    String attributes);
```

In the key generation process, there are four strings involved. They are a public key, a master key, and a private key respectively. In addition, this method also has the set of attributes. The content that it has is defined in the string name. For example, the string “pubFile” contains the public key file, the rest can be done in the same manner. In addition, the “pubFile”, “mskFile” and “attributes” these three files are input files, and the “prvFile” is the output file.

It is same as the setup method, the native library is connected and the native method name which is just defined. Therefore, the “keygen” method is ready to be called after the connection between the native library and the native code is built by JNI.

② Call key generation function in mainActivity:

```
c.keygen (
    "sdcard/pub.pub",
    "sdcard/mskey",
    "sdcard/priv_user.prv",
    "Name Gender Title");
```

In “keygen” algorithm, it uses three paths which are “sdcard/pub.pub”, “sdcard/mskey” and “sdcard/priv_user.prv”, respectively, inside a mobile device to call or store these key files. Besides these, it requires a set of attributes which can be several attributes such as “Name”, “Gender” and “Titles”. For example, this function can define a user Alice who is a female and belongs to the business team. Thus “Alice”, “female” and “business team” are the attributes of this user, and a specific private key for Alice is generated, and her attributes are signed with this private key.

By running this key generation function, a secret key associated with a set of attributes is built. According to the RAS-based access-tree CP-ABE algorithm presented by Li, this secret key with a set of attributes is created with the

help of the Secure Hash Algorithm 1 (SHA1).

③Algorithm description:

Deserialize public key and master secret key

```
parse_args(argc,argv);
pub = treecpabe_pub_unserialize(suck_file(pub_file),1);
msk = treecpabe_msk_unserialize(suck_file(msk_file),1);
```

Allocate and initialize user private key structure.

```
prv = init_prv_params(pub);
```

Compute private key

```
while (*user_attrs) {
    treecpabe_prv_comp_t c;
    element_t h_attr, inv;
    c.attr = *(user_attrs ++);
    element_init(c.dp, Zn);
    element_init(h_attr, Zn);
    element_init(inv, Zn);
    element_from_string(h_attr, c.attr);
    element_invert(inv, h_attr);
    element_set(c.dp, inv);
    element_clear(h_attr);
    element_clear(inv);
    g_array_append_val(prv->comps, c);
}
```

Obtain each user attributes in “c.attr”. Initial elements “c.dp”, “h_attr” and “inv” in the field Z_n . Mapping each user attribute string “c.attr” to one element hash attribute “h_attr” of the finite group, and invert “h_attr” to “inv”. Set the value of “inv” into “c.dp”, and clear the elements “h_attr” and “inv”. Finally, append the generated elements to bytes of array in the private key structure.

Serialize private key and write into the output file

```
spit_file(out_file, treecpabe_prv_serialize(prv),1);
```


5.4.3 Encryption

① Define native function enc in the NativeCPABE class:

```
public native double enc (
                        String pubFile,
                        String jpolicy,
                        String inFile);
```

In the NativeCPABE class, it defines the basic encryption algorithm structure which is three files for this process, such as a public key, access tree policy and encrypted message. One thing to point out it that the input message is not contained here. It is typed by the user through their mobile phone whenever they want to encode a message. Thus, the original message encrypted is defined in the mainActivity class and is used when calling encryption function.

Similar to the algorithms described before, this function is defined with the native library. The name of this method is defined and the native library is loaded, so that it is connected with the native code and it can be called successfully by other class.

② Set “on click” function to the button “CHIPHER” with encryption method in mainActivity window:

```
findViewById(R.id.encBtn).setOnClickListener
    (new View.OnClickListener() {
        @Override
        public void onClick(View view) { try {
            if(input.getText().toString().
                trim().equals("")) {
                Toast.makeText(MainActivity.this,
                    "please type something",
                    Toast.LENGTH_SHORT).show();
            } else {
                writeFile(inputPath, input.getText().toString().trim());
                c.enc("sdcard/pub.pub", "Name Gender Title",
                    inputPath, encPath);
                enc.setText(readFileBinary(encPath));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
```

First of all, the functional button is linked with the ID of encrypt button named “encBtn” which is defined in the activityMain window. Normally, for a user, the behavior to activate the encryption function is to click this button “CHIPHER”. For the users to input the text on the application, it needs IOException. When the application performs certain tasks to access the files, at the same time if there are any input or output file operation issues, then the IOExceptions are thrown. The developer has to handle the exception.

During this part, two functions are defined when the user clicks the encryption button. If there is no input, it shows the message “please type something” by using the “Toast.makeText” function in Android Studio. If the user inputs some message, then this message is read from a specific file named “inputPath” from the input path and turned into strings.

The encryption function uses a public key, a set of attributes and the input text to generate the cipher-text saved in the “encPath” later. Finally, it shows the encrypted text in the text box by using “setText”. Refer to CP-ABE algorithm, an access tree policy is built via encryption function. In this case, it could be an arbitrary name, male or female and the title of position. These defined attributes limit the user scope or specify a user identity. In addition, the input text is be encrypted under those access policy.

③Algorithm description:

A. Fill the policy(treecpabe_policy_t* p, treecpabe_pub_t* pub and element_t e):

```
p->q = _rand_poly(p->k - 1, e);
```

Call function “_rand_poly(deg, zero_val)” to set up a polynomial of $k_x - 1$ degree for the node x. This function sets $q_x(0) = s$ and sets Lagrange coefficient of rest of the points at random to completely define q_x .

For leaf node:

```
if( p->children->len == 0 ){
    element_init(p->cp, Zn);
    element_from_string(h, p->attr);
    element_mul(p->cp, h, p->q->coef[0]);
}
```

Call function “element_from_string” to map the corresponding attribute “attr” to an element of the group “h”. It produces the hash values corresponding to the attribute strings by using SHA1 hash algorithm. Each attribute string is mapped to a 160-bit message digest which is stored in the data buffer.

For non-leaf node:

```
for( i = 0; i < p->children->len; i++ ){
    element_set_si(r, i + 1);
    _eval_poly(t, p->q, r);
    _fill_policy(g_ptr_array_index(p->children, i), pub, t);
}
```

Call function “_eval_poly($T_x, q_x(index(x))$)” to set polynomials for each child node recursively.

B. Run the encryption algorithm:

```
parse_args(argc, argv);
pub = treecpabe_pub_unserialize(suck_file(pub_file), 1);
cph = init_cph_params(pub);
```

Read public key from file, and initial the cipher-text structure.

```
element_init(m, Zn);
element_init(h, Zn);
element_init(s, Zn);
element_random(m);
element_random(h);
element_random(s);
element_pow_zn(cph->cs, pub->gx, s);
element_pow_zn(cph->cs, cph->cs, h);
element_mul(cph->cs, cph->cs, m);
```

Initialize elements and compute $E = M \cdot Y_m^s = M \cdot g^{khs} = M \cdot R^{hs}$, and store the value of E “cs” in array “cph”.

```
element_pow_zn(cph->c, pub->gx, h);
```

Compute $Y_m = g^{kh}$, and store the value of Y_m “c” in array “cph”.

5.4.4 Decryption

① Define native function dec in the NativeCPABE class:

```
public native double dec (
    String pubFile,
    String prvFile,
    String inFile,
    String outFile);
```

The structure of decryption function is defined in the NativeCPABE class. It defines the four strings of this algorithm. The first one is “pubFile” which contains the public key. The second one named “prvFile” stores the private key. The third one called “inFile” presents the cipher-text. The last one is “outFile” which stores the output of decryption algorithm, and it is the decoded message. In this method, the “outFile” is the output, and the rest of them are inputs.

② Set “on click” function to button “DECRYPT” with decryption method in MainActivity:

```
findViewById(R.id.decBtn).setOnClickListener
    (new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            try {
                if (input.getText().toString().
                    trim().equals("")) {
                    Toast.makeText(MainActivity.this,
                        "please type something",
                        Toast.LENGTH_SHORT).show();
                } else {
                    writeFile(inputPath,
                        input.getText().toString().trim());
                    c.dec("sdcard/pub.pub",
                        "sdcard/priv_user.prv",
                        encPath, decPath);
                    dec.setText(readFile(decPath));
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
```


After the native library is loaded and the native method is defined, then the native code is able to call the decryption from the native library. In the decryption method, it is similar to the encryption function. The main function named decryption is called when the user clicks the decryption button. If the user wants to decrypt some message saved in “encPath”, and this file is read by decryption function and it puts the decoded text into “decPath”. Finally, it shows the decoded text in the last text box by using “setText” function.

The access tree structure is used to check whether the user is able or not to access the original text. According to the original RSA-based CP-ABE scheme proposed by Li [36], the decryption algorithm first checks if the access tree of the user is satisfied with the access tree policy defined by encryption algorithm. Then it finds the minimal leaves of the access tree to decryption calculation efficiently. Separately decoding the leaf node and internal node, and get the decoded text.

③Algorithm description:

There are four processes during this algorithm. The first part is to check if the secret key satisfies the policy. In order to effective the decryption process, the second part is to find out k (a threshold value) sets of the smallest size satisfies the access policy. The third part is the recursive function which is called by the decryption algorithm. The last part is the decryption process.

A. Check the satisfaction

For leaf node:

```

if( p->children->len == 0 ){
    for( i = 0; i < prv->comps->len; i++ )
        if( !strcmp(g_array_index(prv->comps,
                                treecpabe_prv_comp_t, i).attr,
                                p->attr) )
        {
            p->satisfiable = 1;
            p->attri = i;
            break;
        }
}

```

Use “strcmp” function to compare the string in the decryption key with the access policy to see whether they are equal or not. If the attribute of private key is satisfied with the policy, then set the satisfiability as 1 for the leaf node.

For non-leaf node:

```

for( i = 0; i < p->children->len; i++ )
    _check_sat(g_ptr_array_index
              (p->children, i), prv);

l = 0;
for( i = 0; i < p->children->len; i++ )
    if(((treecpabe_policy_t*)g_ptr_array_index
      (p->children, i))->satisfiable )
        l++;
if( l >= p->k )
    p->satisfiable = 1;

```

Compare the value of each child node with the content of a private key, statistic the satisfiable number of the nodes and set satisfiability as 1 only if at least the same number as threshold value of children are satisfied. Else, the decryption function will stop.

B. Find minimal satisfied leaves

For leaf node:

```

if( p->children->len == 0 ){
    p->min_leaves = 1;}

```

Define a set S_x for each leaf node x such that $S_x = \{x\}$.

For non-leaf node:

```

for( i = 0; i < p->children->len; i++ )
    if(((treecpabe_policy_t*)g_ptr_array_index
      (p->children, i))->satisfiable )
        _pick_sat_min_leaves(g_ptr_array_index
                              (p->children, i), prv);

c = alloca(sizeof(int) * p->children->len);
for( i = 0; i < p->children->len; i++ )
    c[i] = i;
cur_comp_pol = p;
qsort(c, p->children->len, sizeof(int), leaves_cmp_int);
p->satl = g_array_new(0, 0, sizeof(int));
p->min_leaves = 0;
l = 0;

```

```

for( i = 0; i < p->children->len && l < p->k; i++ )
    if(((treecpabe_policy_t*)g_ptr_array_index
        (p->children, c[i]))->satisfiable )
    {
        l++;
        p->min_leaves += ((treecpabe_policy_t*)
            g_ptr_array_index(p->children, c[i]))
            ->min_leaves;
        k = c[i] + 1;
        g_array_append_val(p->satl, k);
    }
assert(l == p->k);

```

Define a set S_x for each node x , and let k be the threshold value of each non-leaf node x . Choose k nodes x_1, x_2, \dots, x_k from the child nodes of x such that S_{x_i} (for $i = 1, 2, \dots, k$) are the first k sets of the smallest size, then $S_x = \{x' : x' \in S_{x_i}, i = 1, 2, \dots, k\}$. For root node r , define a set S_r which denotes the set of leaf nodes that are used in order to minimize the number of computations.

This function inputs a policy that includes an access tree with root and a set of attributes that are satisfied, then find a set S which is the subset of the nodes in an access tree such that minimized the number of leaves. It means non-leaf node has children less than k which is the threshold value of that non-leaf node. This function uses a recursive algorithm that makes a single traversal of the access tree.

C. DecryptNode function:

```

_dec_node_flatten( element_t r, element_t exp,
    treecpabe_policy_t* p,
    treecpabe_prv_t* prv)
{
    assert(p->satisfiable);
    if( p->children->len == 0 )
        _dec_leaf_flatten(r, exp, p, prv);
    else
        _dec_internal_flatten(r, exp, p, prv);
}

```

This situation is divided into two types: leaf nodes and non-leaf nodes.

For leaf node:

```

_dec_leaf_flatten( element_t r, element_t exp,
                  treecpabe_policy_t* p,
                  treecpabe_prv_t* prv)
{
    c = &(g_array_index(prv->comps,
                        treecpabe_prv_comp_t,
                        p->attri));
    element_mul(s, p->cp, c->dp);
    element_mul(s, s, exp);
    element_add(r, r, s);
}

```

This function is used to compute polynomial interpolation and add the results together. By using polynomial interpolation, this function returns the sum which is the value of random element s .

For non-leaf node:

```

_dec_internal_flatten( element_t r,
                     element_t exp,
                     treecpabe_policy_t* p,
                     treecpabe_prv_t* prv )
{
    for( i = 0; i < p->satl->len; i++ )
    {
        lagrange_coef(t, p->satl, g_array_index
                     (p->satl, int, i));
        element_mul(expnew, exp, t);
        _dec_node_flatten(r, expnew,
                        g_ptr_array_index
                        (p->children,
                         g_array_index(p->satl,
                                         int, i) - 1), prv);
    }
}

```

For all nodes z that are children of x , computing the Lagrange coefficient $\Delta_{i,s}$ for $i \in Z_n$ and a set s of elements in Z_n as

$\Delta_{i,s}(x) = \prod_{j \in s, j \neq i} \frac{x-j}{i-j}$. Then obtain $\Delta_{i,s_z}(0) = \prod_{j \in s, j \neq i} \frac{-j}{i-j}$. Then compute

“expnew” by multiplying “exp” and “t”, and recursively call DecryptNode function to pass through all children node.

D. Decryption process:

```
pub = treecpabe_pub_unserialize(suck_file(pub_file), 1);
prv = treecpabe_prv_unserialize(suck_file(prv_file), 1);
read_cpabe_file(in_file, &cph_buf, &file_len, &aes_buf);
cph = treecpabe_cph_unserialize(cph_buf, 1);
```

Obtain each files.

```
_check_sat(cph->p, prv);
if( !cph->p->satisfiable ){
    return 0;}
```

Check the satification.

```
element_init(t, Zn);
_pick_sat_min_leaves(cph->p, prv);
_dec_flatten(t, cph->p, prv);
```

Find the minimized nodes and call function for decryption of the nodes.

```
element_init(m, Zn);
element_pow_zn(m, cph->c, t);
element_invert(m, m);
element_mul(m, m, cph->cs);
```

Compute $M = E \cdot \frac{1}{g^{khs}}$

```
treecpabe_cph_free(cph);
plt = aes_128_cbc_decrypt(aes_buf, m);
g_byte_array_set_size(plt, file_len);
g_byte_array_free(aes_buf, 1);
```

Use AES decryption algorithm to decrypt the message.

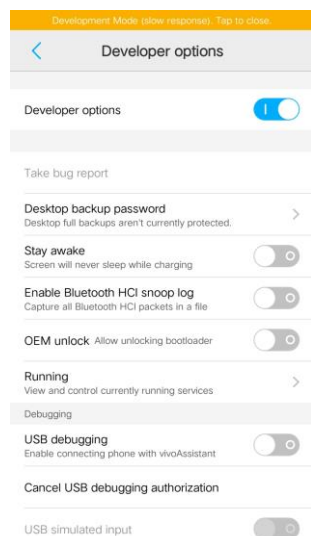
```
spit_file(out_file, plt, 1);
```

Output the decoded file.

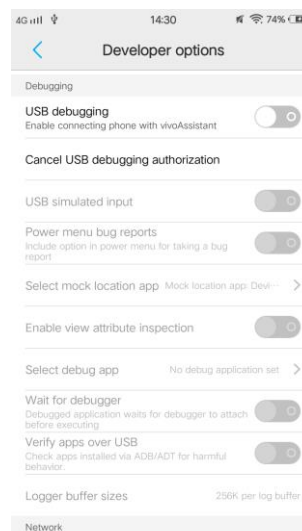
5.5 Running the App

To run the completed application on a mobile device needs three steps:

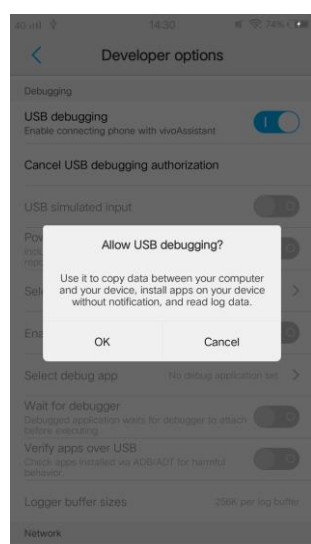
- ① In Android Studio, click the app module in the Project window and then select Run > Run (or click Run in the toolbar).
- ② Set up a device for development



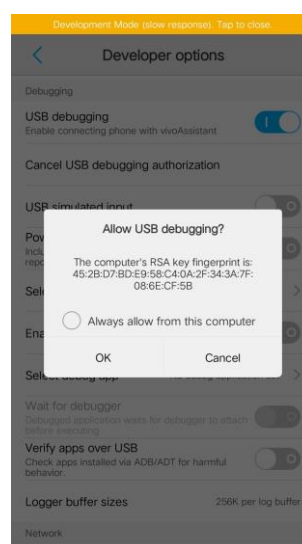
(a)



(b)



(c)

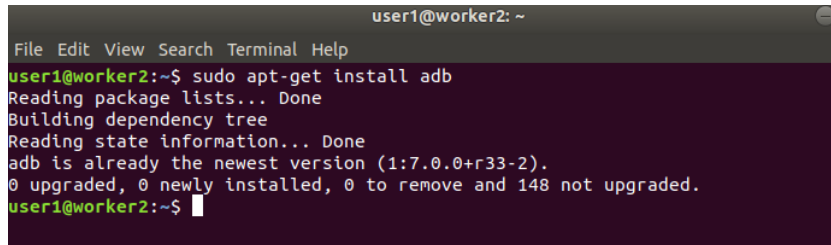


(d)

Figure 37. (a) Developer options (b)(c)(d) Three steps for USB debugging

As it is presented in Figure 37 (a), this process is to open the settings on the device, and select Developer options. In the debugging part of Developer options showed in Figure 37 (b)(c)(d), allow USB debugging, and authorize the computer.

Use apt-get install adb to install the Android adb tools package.

A terminal window titled 'user1@worker2: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The command 'sudo apt-get install adb' has been executed. The output shows: 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', 'adb is already the newest version (1:7.0.0+r33-2).', and '0 upgraded, 0 newly installed, 0 to remove and 148 not upgraded.' The prompt 'user1@worker2:~\$' is shown at the bottom.

```
user1@worker2: ~  
File Edit View Search Terminal Help  
user1@worker2:~$ sudo apt-get install adb  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
adb is already the newest version (1:7.0.0+r33-2).  
0 upgraded, 0 newly installed, 0 to remove and 148 not upgraded.  
user1@worker2:~$
```

Figure 38. Install Android adb tools package on terminal

③In the Select Deployment Target window, select the device, and click OK.

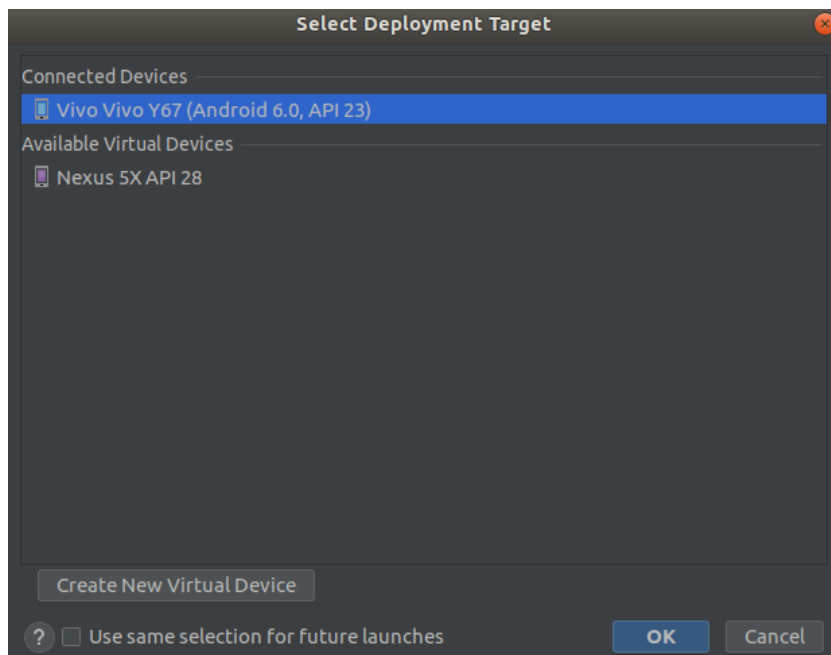


Figure 39. Select deployment target

Android Studio installs the app on the connected device and starts it.

After following these three steps mentioned above, then the application is ready to be used in a mobile phone.

5.6 App Demonstration

This application is designed as a simple interface that contains an input message text-box, an output cipher message text-box and an output decoded message text-box. Here are the interfaces of the app below:

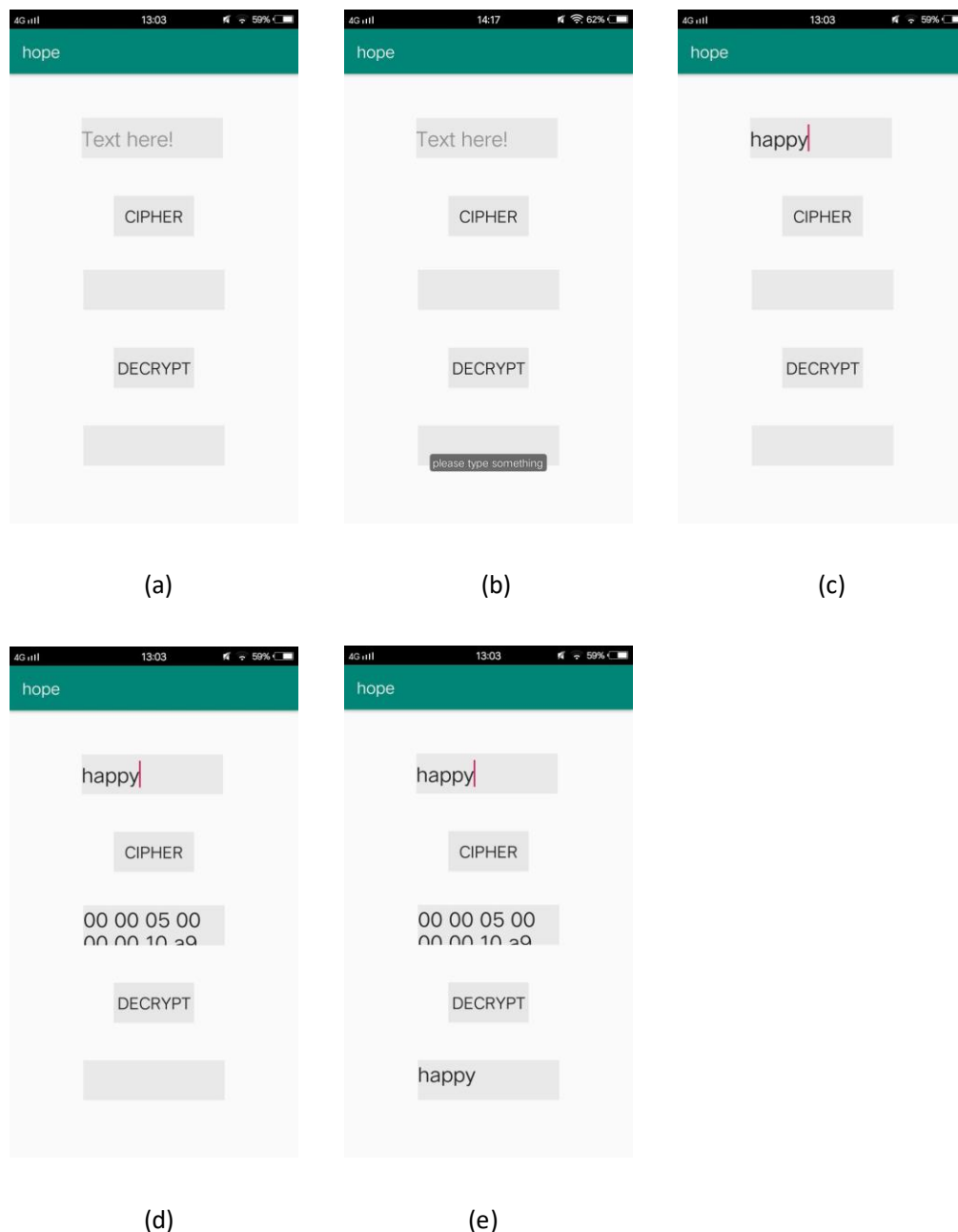


Figure 40. (a) Application interface (b) Hint for empty import (c) import text (d) Encryption (e) Decryption

The Figure 40 (a) shows the initial interface of the encryption application. On the top text-box shows “text here!”. The user can input the message that he

or she wants to encrypt in that text-box. In the Figure 40 (b), it shows a hint when the import of the data is empty. The Figure 40 (c) is demonstrating a user importing a message “happy” in the input text-box. The Figure 40 (d) shows the encryption process. The user presses the “CIPHER” button to encrypt the message, and the encrypted text is shown in the second text-box. In the decryption process showed in Figure 40 (e), the user uses the “DECRYPT” button to decrypt the cipher-text and the decoded text “happy” is shown in the last text-box which is showing the same message as the original one.

5.7 Performance Evaluation

This thesis evaluates the performance of the RSA-based CP-ABE scheme on Android platform in two aspects. The one is the performance of varying key sizes. The other is the performance involved with the number of attributes. For the first performance, it is assumed that larger the key size will take more execution time on key generation, encryption and decryption algorithms. The comparison of the execution time and the key size is analyzed in Table 2 below. In order to get more reliable results, each performance test ran five times then the average results are used.

Key Size (bit)		512	1024	2048
Time (ms)	Key Generation	11	18	39
	Encryption	8	17	35
	Decryption	5	11	18

Table 2. Comparison execution time with key size

In Table 2, it lists the three main algorithms of the CP-ABE scheme; key generation, encryption and decryption, respectively. The execution times of these three algorithms are compared for the different key sizes such as 512 bits, 1024 bits and 2048 bits. It can be noticed that the large key size needs longer execution time. Figure 41 shows the results as a line graph.

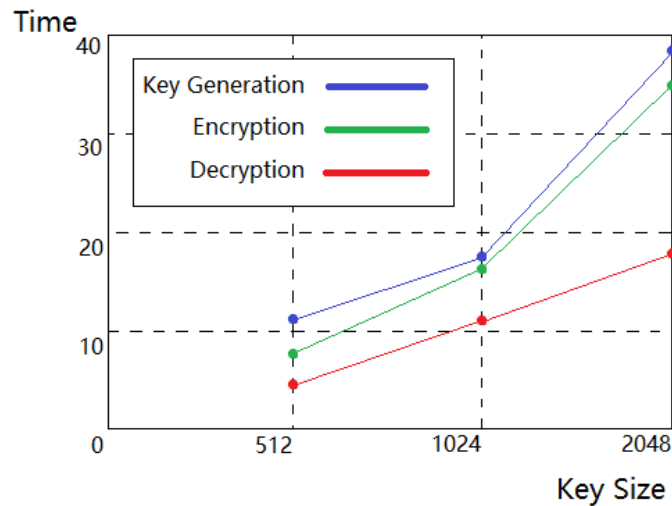


Figure 41. Key size

Figure 41 shows that there is a positive linear relationship between the execution time and the key size. The larger the key size is, the longer time it takes to run each algorithm. Note that the execution times for both the key generation and the encryption are similar which are both increased sharply with the key size 1024. By comparison, the execution time of the decryption algorithm is smooth in all different key sizes.

The other reason that affects the execution time is associated with the number of attributes. Assuming that the more attributes, the longer the execution time it takes. The results are listed below in Table 3.

Number of Attributes		1	3	6	9	12
Time (ms)	Key Generation	18	14	19	17	16
	Encryption	17	19	20	22	25
	Decryption	11	12	13	15	17

Table 3. Comparison execution time with number of attributes

The Table 3 compares the execution time with the number of attributes during key generation, encryption and decryption processes. It chooses the five different numbers of attributes which contains 1, 3, 6, 9 and 12 attributes. The result based on a line graph is shown in Figure 42 below.

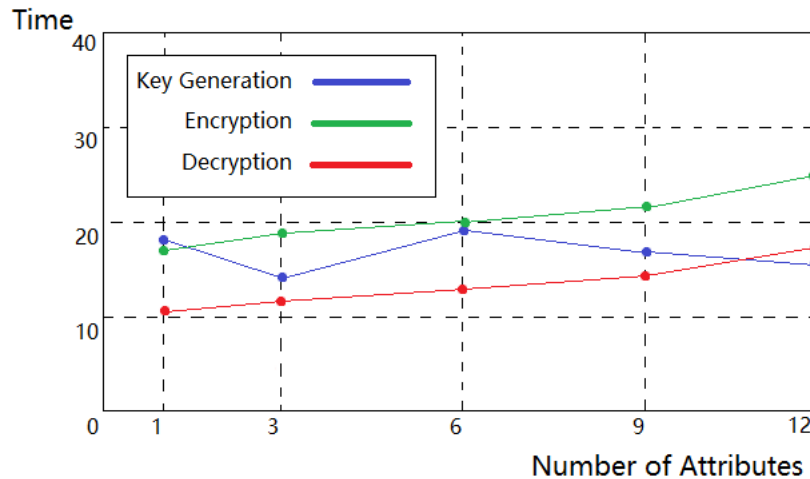


Figure 42. Number of attributes

It can be seen clearly that the execution time of encryption and decryption algorithms are smoother according to the increasing number of attributes. Nevertheless, the execution time of the key generation algorithm was not affected as much by the number of attributes. The reason appears that the user's attributes are set during the encryption process, thus it may not affect the execution time.

5.8 Lessons Learned

During the implementation part, there were a several problems that prevented from progressing with the thesis. This includes setting devices, preparing and compiling libraries, linking libraries with native methods and so on. Starting with the setting issues, it took some effort to make a simulator work. After downloading a phone simulator, the computer is still unable to open it. The reason was that the internal setting of the computer about the virtual device was not set properly. For Intel processor devices, this problem was solved by enabling the Intel Virtual Technology. This function can be found in BIOS by pressing key F2 when power the computer on. After enabling the setting of Intel Virtual Technology, then the simulator was allowed to work on the development computer. Nevertheless, every time when Android Studio calling the simulator, errors occurred as following:

```
Emulator: Process finished with exit code 1!
Emulator: emulator: ERROR: x86 emulation currently requires hardware acceleration!
```

There were three ways to solve this problem. The first method was to download HAXM drivers. The second one was to set the CPU's hardware

acceleration switch which allowed the quick start of the device. The last one was to use a lower version of a virtual device. However, the use of a simulator did not affect the performance of the application on the mobile device. So it is recommended that any developers in the similar situation to choose the right version of the simulator.

Moreover, this cipher app uses C code in Android Studio, thus at the beginning, the “Include C++ Support” option needs to be selected on a new project page. Thus the Android Studio could install C code compiler and libraries. Additional, tools that support the C code was also needed, such as LLDB (a debugger that Android Studio uses to debug native code) and NDK (this toolkit allows users to use C and C++ code for Android).

Compiling the CP-ABE library was had an issue on its own. The CP-ABE algorithm needs other libraries to run, such as PBC, GMP and so on. A series of libraries were needed to be installed through the terminal. A order of the libraries to setup the environment for CP-ABE scheme was important, for example, PBC required GMP, when GMP was supported by M4, bison and flex. After preparing the environment, the CP-ABE scheme was able to be called. However, it was still unable to be called by the Android system. Eventually it was figured that the .c files and .h files were needed to be compiled into a dynamic library called .so file. But, this dynamic library was not ready to be called by the Java code. By using the Android.mk file and calling the NDK-build command to generate the .so file, then the compiled libraries were ready to be used in the Android system, and then the preparation job was done.

When the libraries were ready, they were needed to be connected with the native methods. Thus, JNI function was used to solve this problem. To use JNI function, the native method was needed to be written as below:

```
public class NativeClass {
    static {
        System.loadLibrary("library name");
    }
    public native method(content);
}
```

System.loadlibrary(“library name”) function was used to call the library and then connect the library with the native methods by defining the native methods. Then the bridge that connects library and methods was built. Apart from this, a specified Android.mk file and a generated .so file were needed in a given path inside the “build.gradle” configuration. Then the dynamic libraries were able to be loaded in the static block, and the connection

between C and Java code was completed.

There are limitations of the implementation provided by this thesis. The cipher method could only be achieved inside this app. Another limitation of this project is that it could only encrypt and decrypt text in the same interface at the same time. In addition, this application assumes that the attributes of each user satisfies the access policy all the time. In the future, it would be necessary to build a database that stores the user's information and needs more user interfaces such as login to allow only authorized users can use the app. In addition, instead of using an app to offer a functionality for an encryption, it would be better to provide a CP-ABE based encryption at the system level to protect the data, app and the mobile device it self.

Chapter 6 Conclusion

CP-ABE scheme has been emerged as a flexible data privacy mechanism by offering an embedded access control with encryption. As mobile devices are not suited for complex and high resource demanding computations, a more efficient and lightweight encryption solution has been demanded. Lightweight CP-ABE schemes with constant key sizes and constant cipher-texts regardless of the number of attributes have been hailed as suitable solution for resource constrained devices such as mobile phones.

In this thesis, we offer an implementation strategy for a lightweight RSA-based CP-ABE scheme for an Android system. This thesis provides the implementation details for the four main algorithms of RSA-based CP-ABE which include setup, key generation, encryption, and decryption, respectively. In addition, the implementation offers a strategy to integrate a CP-ABE system written in C language [36] to work with a Java implementation. Our solution is provided by using the JNI function to connect the C language and Java. In other words, the functions written in C code can be called in an Android system written in Java code. Therefore, JNI plays an essential role in our implementation strategies as it is also used to link the dynamic libraries with the native method. This allows the RSA-based CP-ABE library to be implemented for the Android devices successfully. We demonstrate a mobile app that allows users to encrypt and decrypt data efficiently based on RSA-based CP-ABE approach.

References

- [1] Sahai, A., & Waters, B. (2005, May). Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 457-473). Springer, Berlin, Heidelberg.
- [2] Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2006, October). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security* (pp. 89-98). Acm.
- [3] Bethencourt, J., Sahai, A., & Waters, B. (2007, May). cipher-text-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)* (pp. 321-334). IEEE.
- [4] Cheung, B. (2019). *Attribute-based Encryption for Healthcare Blockchain*. Retrieved from <http://bennycheung.github.io/attribute-based-encryption-for-healthcare-blockchain>
- [5] Yu, S., & Shi, L. (2016). *cipher-text: Trust Establishment in Wireless Body Area Networks*. Retrieved from <https://www.sciencedirect.com/topics/engineering/cipher-text>
- [6] Ostrovsky, R., Sahai, A., & Waters, B. (2007, October). Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 195-203). ACM.
- [7] Internetwork Security. (2017, Mar 29). *APA referencing: Attribute based Encryption (ABE)* [Video file]. Retrieved from <https://www.youtube.com/watch?v=ZogQMKzoQdw>
- [8] Daza, V., Herranz, J., Morillo, P., & Ràfols, C. (2010). Extensions of access structures and their cryptographic applications. *Applicable Algebra in Engineering, Communication and Computing*, 21(4), 257-284.
- [9] Lai, J., Deng, R. H., & Li, Y. (2011, May). Fully secure cipertext-policy hiding CP-ABE. In *International conference on information security practice and experience* (pp. 24-39). Springer, Berlin, Heidelberg.
- [10] Moffat, S., Hammoudeh, M., & Hegarty, R. (2017, July). A survey on

cipher-text-policy attribute-based encryption (cp-abe) approaches to data security on mobile devices and its application to iot. In *Proceedings of the International Conference on Future Networks and Distributed Systems* (p. 34). ACM.

[11] Zuckerman, A. E., & Kim, G. R. (2009). Personal health records. In *Pediatric Informatics* (pp. 293-301). Springer, New York, NY.

[12] Hong, H., Chen, D., & Sun, Z. (2016). A practical application of CP-ABE for mobile PHR system: a study on the user accountability. *SpringerPlus*, 5(1), 1320.

[13] Abbott, A. A., Fuji, K. T., Galt, K. A., & Paschal, K. A. (2012). How baccalaureate nursing students value an interprofessional patient safety course for professional development. *ISRN nursing*, 2012.

[14] Price, M., Bellwood, P., Kitson, N., Davies, I., Weber, J., & Lau, F. (2015). Conditions potentially sensitive to a personal health record (PHR) intervention, a systematic review. *BMC medical informatics and decision making*, 15(1), 32.

[15] Hong, H., Chen, D., & Sun, Z. (2016). A practical application of CP-ABE for mobile PHR system: a study on the user accountability. *SpringerPlus*, 5(1), 1320.

[16] Yao, X., Chen, Z., & Tian, Y. (2015). A lightweight attribute-based encryption scheme for the Internet of Things. *Future Generation Computer Systems*, 49, 104-112.

[17] Cui, W., Du, C., & Chen, J. (2016). CP-ABE Based Privacy-Preserving User Profile Matching in Mobile Social Networks. *PloS one*, 11(6), e0157933.

[18] Lagrange polynomial. (2019). In *Wikipedia, The Free Encyclopedia*. Retrieved May 25, 2019, from https://en.wikipedia.org/wiki/Lagrange_polynomial

[19] Ireland, D. (2018, June 9). RSA Algorithm [Online forum contents]. Retrieved from https://www.di-mgt.com.au/rsa_alg.html#theauthor

[20] Design Overview (2018). In *Oracle, Java Native Interface Specification*. Retrieved June 8, 2019, from <https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/design.html#wp16696>

- [21] Vaughan, G. V., & Blake, E. (2016). *GNU M4*. Retrieved from <https://www.gnu.org/software/m4/>
- [22] Levine, J. (2009). *Flex & Bison: Text Processing Tools*. " O'Reilly Media, Inc."
- [23] Kenekayoro, P. T. (2011). One way functions and public key cryptography. *African Journal of Mathematics and Computer Science Research*, 3(6), 213-216.
- [24] Zhou, X., & Tang, X. (2011, August). Research and implementation of RSA algorithm for encryption and decryption. In *Proceedings of 2011 6th International Forum on Strategic Technology* (Vol. 2, pp. 1118-1121). IEEE.
- [25] The GNU Multiple Precision Arithmetic Library. (2018). *gmp*. Retrieved from <https://gmplib.org/>
- [26] Lynn, B. (2007). *PBC Library*. Retrieved from <https://crypto.stanford.edu/pbc/>
- [27] Cryptography and SSL/TLS Toolkit. (2019). *OpenSSL*. Retrieved from <https://www.openssl.org/>
- [28] Glib. (2019). In *Wikipedia, The Free Encyclopedia*. Retrieved June 9, 2019, from <https://en.wikipedia.org/wiki/GLib>
- [29] Mullis, A. (2017). *Android NDK—Everything you need to know*. Retrieved from <https://www.androidauthority.com/android-ndk-everything-need-know-677642/>
- [30] Getting Started with the NDK. (2019). *Developers*. Retrieved from <https://developer.android.com/ndk/guides>
- [31] Artjom B. (2016, February 2). ABE Schemes - Access Structures & Performance [Online forum comment]. Retrieved from <https://crypto.stackexchange.com/questions/32410/abe-schemes-access-structures-performance>
- [32] Hu, C., Li, H., Huo, Y., Xiang, T., & Liao, X. (2016). Secure and efficient data communication protocol for wireless body area networks. *IEEE Transactions on Multi-Scale Computing Systems*, 2(2), 94-107.
- [33] Lakshmi, R. N., Laavanya, R., Meenakshi, M., & Dhas, C. S. G. (2015).

Analysis of attribute based encryption schemes. *Int. J. Comput. Sci. Eng.*, 3(3), 1076-1081.

[34] Ning, J., Cao, Z., Dong, X., Wei, L., & Lin, X. (2014, September). Large universe cipher-text-policy attribute-based encryption with white-box traceability. In *European Symposium on Research in Computer Security* (pp. 55-72). Springer, Cham.

[35] Hemalatha, S., & Manickachezian, R. (2014). Dynamic auditing protocol using improved RSA and CBDH for cloud data storage. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(1).

[36] Ping Li. (2018). *Novel Lightweight Ciphertext-Policy Attribute-Based Encryption for IoT Applications*(Doctoral dissertation). Massey University, Auckland, New Zealand.