

國立交通大學

資訊科學與工程研究所

碩士論文

多授權中心的屬性加密及其實現

Attribute-Based Encryption with Multiple Authorities and Its
Implementation

研究生：蔡禮鼎

指導教授：陳榮傑 教授

中華民國一零一年七月

多授權中心的屬性加密及其實現

Attribute-Based Encryption with Multiple Authorities and Its Implementation

研究生：蔡禮鼎

指導教授：陳榮傑

Student : Li-Ting Tsai

Advisor : Rong-Jaye Chen

國立交通大學
資訊科學與工程研究所
碩士論文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2012

Hsinchu, Taiwan, Republic of China

中華民國 101 年 7 月

Attribute-Based Encryption with Multiple Authorities and Its Implementation

Student : Li-Ting Tsai

Advisors : Dr. Rong-Jaye Chen

Institute of Computer Science and Engineering

College of Computer Science,
National Chiao Tung University

ABSTRACT

The Attribute-based encryption (ABE) scheme provides a fine-grained access control mechanism which is better than traditional public-key encryption schemes such as RSA and ElGamal. In an ABE scheme, the encryptor can specify an access formula that controls which private keys have the ability to decrypt the ciphertext. In most ABE schemes, there is only one authority who issues all private keys and public keys. If there are many authorities who issues their own private keys and public keys in an ABE scheme, the scheme is called ABE with multiple authorities. The access control mechanism in ABE schemes is from secret sharing schemes. In this thesis, we proposed an algorithm which is used both in secret sharing schemes and ABE schemes. Our algorithm provides more expressiveness in the access formula. Compared to the previous algorithm, our algorithm can handle more types of access formulae. Also, the algorithm and an ABE scheme with multiple authorities are implemented. Finally, we discuss applications that are highly related to our implementation.

多授權中心的屬性加密及其實現

學生：蔡禮鼎

指導教授：陳榮傑 教授

國立交通大學資訊科學與工程研究所碩士班

摘要

屬性加密系統提供了細粒度的存取控制，這是傳統的公開金鑰加密系統，如 RSA 或 ElGamal 所做不到的。在屬性加密系統裡，加密者可以決定一個存取公式，這個存取公式控制了哪些私鑰可以解開密文。在大多數的屬性加密系統中，只有單一授權中心負責發放所有的公鑰和私鑰，假如有許多個授權中心可以發放自己的公鑰和私鑰，這樣的屬性加密系統稱為多授權中心的屬性加密系統。屬性加密系統裡的存取控制是來自於秘密分享機制。在這篇論文中，我們提出了一個用在屬性加密系統和秘密分享機制的演算法，這個演算法增加了存取公式的表達性，跟之前的演算法比較，我們的演算法可以處理更多種類的存取公式。我們並實現了上述的演算法和一個多授權中心的屬性加密系統。最後，我們討論了許多可行的應用，這些應用和我們的實現都有高度的相關。

誌 謝

這篇論文的完成，必須誠摯的感謝指導教授陳榮傑老師，老師的悉心指導使我獲益良多，不只是在課業方面，在日常生活上，老師也時時會關心我的狀況，老師在研究生涯給予的照顧，我將永遠感念在心。

感謝范俊逸教授、張仁俊教授和李金鳳教授能撥冗擔任口試委員，口試委員們的指點和建議，使論文能夠更完整，也帶給我更多研究的方向。

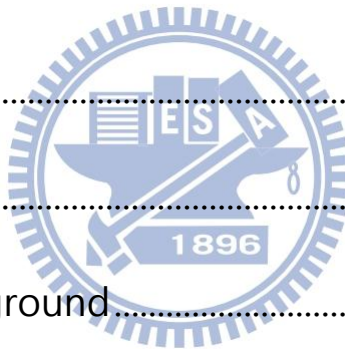
Cryptanalysis 實驗室志賢學長、輔國學長及用翔學長，感謝你們兩年的照顧，常常解答我的疑問，以及給予我報告上的建議。感謝同學嘉雯，不論是在修課、程式或論文中，都常和我討論、分享心得並一起努力完成。感謝你們，你們是我研究生活上，很重要的一部分。

也要感謝一路走來遇到的同學和朋友，你們的陪伴和幫忙，都是我能夠堅持下去的動力，十分感謝在研究生涯曾幫助過我的人。

最後更要謝謝我的家人，感謝父母在求學生涯的栽培，讓我能夠順利的完成學位，也要感謝姊姊不時的關心，家人的照顧都使我感到溫暖及窩心。謹以此文獻給我摯愛的家人。

Contents

Abstract in English.....	i
Abstract in Chinese.....	ii
Acknowledges.....	iii
Contents	iv
List of Tables.....	vi
List of Figures	vii
1. Introduction	1
2. Mathematical Background.....	4
2.1 Elliptic Curves	4
2.2 Rational Functions and Divisors	5
2.3 The Tate Pairing.....	7
2.4 Supersingular Curves and Distortion Maps.....	8
2.5 Complex Multiplication Method	10
3. Secret Sharing Schemes.....	12
3.1 Shamir' s Threshold Secret Sharing Scheme	13

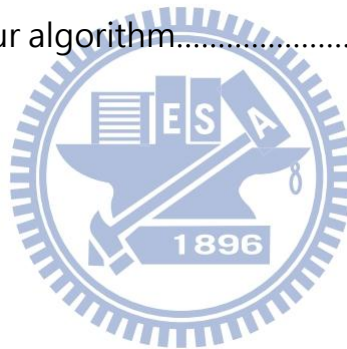


3.2	Monotone Span Program Construction.....	14
4.	Attribute-Based Encryption	17
4.1	Threshold ABE	17
4.2	Key-Policy ABE	19
4.3	Ciphertext-Policy ABE.....	23
4.4	Dual-Policy ABE	25
4.5	ABE with Multiple Authorities	26
5.	Implementation and Applications.....	28
5.1	Implementation of CP-ABE with Multiple Authorities	28
5.2	Converting from the Access Formula to the MSP	36
5.3	Applications	44
6.	Conclusion and Future Work.....	50
	Bibliography	51
	Appendix A: Source Code.....	56



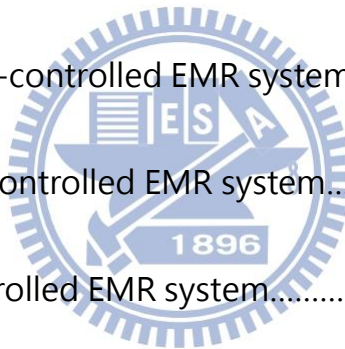
List of Tables

Table 2.1: Supersingular curves.....	9
Table 5.1: NIST recommended key sizes (bits).....	29
Table 5.2: Pairings in the PBC library.....	30
Table 5.3: Comparison of speed of different pairings.....	31
Table 5.4: Pseudo code of LW algorithm.....	38
Table 5.5: Pseudo code of our algorithm.....	43



List of Figures

Figure 4.1: An example of access tree.....	21
Figure 4.2: An example of access tree in Chase' s scheme.....	27
Figure 5.1: A demonstration of encryption in CP-ABE scheme.....	34
Figure 5.2: A demonstration of decryption in CP-ABE scheme.....	34
Figure 5.3: Encryption time.....	35
Figure 5.4: Decryption time.....	36
Figure 5.5: Common patient-controlled EMR system.....	46
Figure 5.6: Flexible patient-controlled EMR system.....	47
Figure 5.7: Our patient-controlled EMR system.....	49



Chapter 1

Introduction

Prior to the invention of public-key cryptography, encryption algorithm and decryption algorithm employ the same secret key. This means when two parties want to communicate securely, they must first find a way to share an mutual secret key. This inconvenience of symmetry makes people to think about the possibility of asymmetric cryptosystems. Asymmetric cryptosystems allow encryption and decryption algorithm employ different keys. And we can make the encryption key public and keep the decryption key secret to avoid the inconvenience of symmetric cryptosystems. Therefore, asymmetric cryptosystems are also referred to as public-key cryptosystems.

In 1977, Rivest, Shamir and Adleman proposed a public-key encryption scheme based on the factoring problem [30]. Since then many other public-key schemes have been developed, such as the ElGamal encryption [13] and the NTRU encryption [17]. Today public-key cryptography is ubiquitous. When we want to send confidential information such as the credit card number to a trusted service provider over the Internet, we often encrypt the credit card number under the service provider's public key. Many Internet protocols such as IPsec and TLS which provide communication security are based on the public-key cryptography.

Recently, cloud computing is consider as a new turning point in information technology. Cloud computing has become more and more popular due to its

characteristics such as low cost, reliability, scalability and performance. Many countries is investing cloud computing. In Taiwan, the government has budgeted NT\$ 24 billion for 5 years to develop cloud computing.

A lot of users upload their data to the cloud. They enjoy the benefits of cloud computing but they also care about privacy and security. For example, the government might want to outsource their information services to a cloud service provider. Therefore, the government can reduce their maintenance cost and have a better performance environment, so they can do their work more efficiently. However, the government does not want the cloud service provider to watch sensitive information. The data in cloud must be stored in encrypted form.

Traditional public-key cryptography has its limits in the cloud environment. If a party wants to share a file with other parties in the cloud, he must encrypt the file under each party's public key. Suppose that there are thousands of receivers in the cloud, and then there will thousands of encrypted files. This coarse-grained approach is not economical.

Many encryption schemes has been developed in recent years such as searchable encryptions [8, 10], broadcast encryptions [9, 22], and attribute-based encryptions [3, 6, 11, 12, 15, 16, 23, 28, 31, 34]. Those new encryptions can help us not only keep our data in secret but also enjoy the advantages from the cloud.

In this thesis, we will focus on the attribute-based encryptions (ABE). The ABE is first proposed by Sahai and Waters [31]. In subsequent research, the concept of key-policy ABE (KP-ABE) and cipher-policy ABE (CP-ABE) are developed in [6, 15, 16]. The CP-ABE scheme is described in the below.

In our daily life, we all have some attributes associated with us. For example, everyone has his or her age, his or her gender and his or her location. In a chatroom in the Internet, if we want to send a message to anyone who is below

30 and is male, we can write our access formula down “Age < 30” AND “Male.” In a CP-ABE scheme, we can encrypt our message under this access formula. And anyone who is male and is below 30 can decrypt our messages. The above example shows that we do not need to specify exactly who can read our message. We just describe the attributes the decryptor should have, and then the person who has the corresponding private keys can decrypt our message. In addition, we just produce one ciphertext. That is economical compared to the traditional public-key encryption.

The rest of this thesis is organized as follows. In Chapter 2, we first review some important background in elliptic curves and introduce the divisors on a curve, the Tate pairing, supersingular curves, and so on. The divisor plays an important role in defining the Tate pairing.

In Chapter 3, secret sharing schemes which have close relationships to ABE schemes are discussed. Shamir’s secret sharing scheme and monotone span program (MSP) construction are introduced. An important issue of the MSP is that how to convert from an access formula to an MSP matrix. We will design an algorithm in Chapter 5.

In Chapter 4, several prominent ABE schemes are discussed. We will discuss their construction, their relation to secret sharing schemes and how to avoid collusion attack.

In Chapter 5, our implementation of a CP-ABE scheme with multiple authorities is first discussed. Then we will give an algorithm of converting from an access formula to an MSP matrix. Finally, we discuss applications based on our implementation. The conclusion is given in Chapter 6.

Chapter 2

Mathematical Background

In this chapter, we review elliptic curves and bilinear pairings.

2.1 Elliptic Curves

Suppose that $E: y^2 = x^3 + ax + b$ is an elliptic curve defined over a finite field F_q and q is power of a prime $p > 3$. E has $q + 1 - t$ points in F_q and $-2\sqrt{q} \leq t \leq 2\sqrt{q}$. These points plus \mathcal{O} , an imaginary identity point at infinity, become a group with addition structure. The group is denoted as $E(F_q)$. That is,

$$E(F_q) = \{(x, y) \cup \mathcal{O} \mid y^2 = x^3 + ax + b, a, b \in F_q, x, y \in F_q\}.$$

The group addition operation is defined as follows. Given points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on the curve, we first draw the line through P and Q . The line intersects the curve in $S' = (x_3, y_3')$. We then reflect S' over the x -axis to obtain $S = P + Q = (x_3, -y_3') = (x_3, y_3)$. Suppose that λ is the slope of the line through P and Q , then the coordinates of $P + Q = (x_3, y_3)$ are $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$, where

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P \neq Q \\ (3x_1^2 + a)/2y_1 & \text{if } P = Q \end{cases}$$

We also define $E(F_{q^d}) = \{(x, y) \cup \mathcal{O} \mid y^2 = x^3 + ax + b, a, b \in F_q, x, y \in F_{q^d}\}$. Suppose that \bar{F}_q is the algebraic closure of F_q , then $E(F_q) \subseteq E(F_{q^d}) \subseteq E(\bar{F}_q)$.

Suppose that $r \mid \#E(F_q) = q + 1 - t$, then we define $E[r] = \{P \in E(\bar{F}_q) \mid rP = \mathcal{O}\}$. $E[r]$ are called the r -torsion points. The r -torsion point plays an important role in pairing's definitions.

We can also find a smallest positive integer k such that $r \mid q^k - 1$. k is called the embedding degree. There are two important facts about the embedding degree. One is that $E[r] \subseteq E(F_{q^k})$ and then we can compute the r -torsion points in $E(F_{q^k})$ rather than in $E(\bar{F}_q)$. The other fact is that $\mu_r \subseteq F_{q^k}$ where $\mu_r = \{x \in \bar{F}_q \mid x^r = 1\}$.

2.2 Rational Functions and Divisors

$F_{q^k}[X, Y]$ represents the ring of polynomials in two variables X, Y with coefficients in F_{q^k} . A rational function $h = f/g$ where $f, g \in F_{q^k}[X, Y]$ and f is coprime to g .

Given an elliptic curve E and a rational function $h = f/g$, we consider the points that $f(x, y) = 0$ and $(x, y) \in E(F_{q^k})$. We call those points zeroes of h . We also consider the points that $g(x, y) = 0$ and $(x, y) \in E(F_{q^k})$. Those points are called poles of E .

The divisor is a useful tool for keeping track of the zeros and poles [33]. We

use divisors to indicate which points are zeros or poles and their orders for a rational function over an elliptic curve. A divisor D on an elliptic curve E is the finite linear combination of the formal symbols with integer coefficients:

$$D = \sum_{P \in E} n_P [P].$$

If $n_P > 0$, it indicates that P is a zero, and if $n_P < 0$, it indicates that P is a pole. We define $Div(E)$ as the group of divisors. For a divisor $D = \sum n_P [P]$, we define $supp(D) = \{P \in E \mid n_P \neq 0\}$ as the support of D , $deg(D) = \sum n_P$ as the degree of D , and $sum(D) = \sum n_P P$.

Now we consider only the set of divisors of degree zero. The set forms a subgroup $Div^0(E) \subset Div(E)$. Let f be a rational function. The evaluation of a rational function f on a divisor $D = \sum n_P [P]$ is defined by

$$f(D) = \prod_{P \in supp(D)} f(P)^{n_P}.$$

The divisor of a rational function f is defined as $div(f) = \sum n_{P,f}(P)$ where $n_{P,f}$ is the zero or pole order of point P on f . The degree of $div(f)$ must be zero [7]. A divisor $D \in Div^0(E)$ is principal if it is the divisor of a function. The following is an important fact.

Theorem 2.1 [33]. Let E be an elliptic curve and D be a divisor on E with $deg(D) = 0$. Then there is a function f on E with $div(f) = D$ if and only if

$$sum(D) = \mathcal{O}.$$

2.3 The Tate Pairing

The Tate pairing and the Weil pairing are two well-studied pairings. Under the same security level, The Tate pairing is generally considered more efficient than the Weil pairing.

Let E be an elliptic curve defined over a finite field F_q and q is power of a prime $p > 3$. Let G be a cyclic subgroup of $E(F_q)$ of order r which is coprime to q . The embedding degree is k such that $r \mid q^k - 1$. The Tate pairing is a map

$$\langle P, Q \rangle_r: E(F_{q^k})[r] \times E(F_{q^k})/rE(F_{q^k}) \rightarrow F_{q^k}^*/(F_{q^k}^*)^r.$$

$E(F_{q^k})[r]$ is defined as $E[r] \cap E(F_{q^k})$ and $rE(F_{q^k})$ is $\{rS \mid S \in E(F_{q^k})\}$ and $(F_{q^k}^*)^r$ is $\{\alpha^r \mid \alpha \in F_{q^k}^*\}$. The groups μ_r and $F_{q^k}^*/(F_{q^k}^*)^r$ are isomorphic.

Let $P \in E(F_{q^k})[r]$ and let $Q \in E(F_{q^k})$. Q represents a coset in $E(F_{q^k})/rE(F_{q^k})$. Let f be a rational function with divisors $(f) = r[P] - r[\mathcal{O}]$. Choose a $S \in E(F_{q^k})$ such that $S \neq P, (P - Q), -Q$ or \mathcal{O} . Let D be a divisor and $D = [Q + S] - [S]$. The Tate pairing is defined to be

$$\langle P, Q \rangle_r = f(D).$$

$f(D) \in F_{q^k}^*$ represents a coset in $F_{q^k}^*/(F_{q^k}^*)^r$. In fact, we often want to standardize the coset representative. Therefore the reduced Tate pairing is

defined to be

$$t_r = \langle P, Q \rangle_r^{q^k - 1/r}.$$

The Tate pairing has bilinearity property and other important properties. See Theorem 2.2.

Theorem 2.2 [7]. Let E be an elliptic curve defined over a finite field F_q and q is power of a prime $p > 3$. Let G be a cyclic subgroup of $E(F_q)$ of order r which is coprime to q . The embedding degree is k . The Tate pairing satisfies:

1. Bilinearity: For all $P, P_1, P_2 \in E(F_{q^k})[r]$ and $Q, Q_1, Q_2 \in E(F_{q^k})$,

$$t_r(P_1 + P_2, Q) = t_r(P_1, Q)t_r(P_2, Q) \text{ and}$$

$$t_r(P, Q_1 + Q_2) = t_r(P, Q_1)t_r(P, Q_2).$$

2. Non-degeneracy:

$$\forall P, t_r(P, Q) = 1 \text{ if and only if } Q = \mathcal{O} \text{ and}$$

$$\forall Q, t_r(P, Q) = 1 \text{ if and only if } P = \mathcal{O}.$$

To compute the Tate pairing, we need to evaluate a rational function f that $(f) = r[P] - r[\mathcal{O}]$. The Miller's algorithm [26] can help us find the function and compute the result of the Tate pairing.

2.4 Supersingular Curves and Distortion Maps

Suppose that $E: y^2 = x^3 + ax + b$ is an elliptic curve defined over a finite field F_q and q is power of a prime $p > 3$. E has $q + 1 - t$ points in F_q and $-2\sqrt{q} \leq t \leq 2\sqrt{q}$. If $p \mid t$, then E is said to be supersingular. Otherwise, E is said to be ordinary. An important property of supersingular curves is that their

embedding degrees are low. Their embedding degrees are from 1 to 6. Low embedding degree is crucial for the efficiency of computing a pairing. Another important property of supersingular curves is the existence of distortion maps.

A distortion map ϕ maps a point $P \in E(F_q)$ to a point $\phi(P) \in E(F_{q^k})$ such that P and $\phi(P)$ are linearly independent. If E is supersingular and $k > 1$, the distortion map exists. If E is ordinary and $k > 1$, then no distortion map exists [21]. By using the distortion map, we can define the modified Tate pairing.

Let E be a supersingular curve defined over a finite field F_q and q is power of a prime $p > 3$. Let G be a cyclic subgroup of $E(F_q)$ of order r which is coprime to q . The embedding degree is k such that $r \mid q^k - 1$. A distortion map ϕ exists. The modified Tate pairing is a map $\hat{t}_r: G \times G \rightarrow \mu_r$ and defined to be

$$\hat{t}_r(P, Q) = t_r(P, \phi(Q)).$$

We note that the first input and the second input of the modified Tate pairing are from the same group. Therefore we say the modified Tate pairing is symmetric.

Table 2.1 [7] contains some popular supersingular curves.

k	Elliptic curve data
2	$E: y^2 = x^3 + a$ over F_p , where p is a prime and $p \equiv 2 \pmod{3}$ E has $p + 1$ points Distortion map $(x, y) \mapsto (\zeta_3 x, y)$, where $\zeta_3^3 = 1$.
2	$E: y^2 = x^3 + x$ over F_p , where p is a prime and $p \equiv 3 \pmod{4}$ E has $p + 1$ points Distortion map $(x, y) \mapsto (-x, iy)$, where $i^2 = -1$.

Table 2.1: Supersingular curves

In the rest of this thesis, we usually treat pairings as “black boxes.” It can help us focus on the design of the encryption scheme. Therefore, we now give an abstract definition of the pairing.

Definition 2.3 (Pairing). Let G_1 and G_2 be two additive cyclic groups and G_T be a multiplicative cyclic group. G_1 , G_2 , and G_T are all of prime order p . Let P be a generator of G_1 and Q be a generator of G_2 . A pairing is a map: $e: G_1 \times G_2 \rightarrow G_T$. The map e has the following properties:

1. Bilinearity: $e(aP, bQ) = e(P, Q)^{ab} \forall a, b \in \mathbb{Z}_p$.
2. Non-degeneracy: $e(P, Q) \neq 1$.
3. Computability: $\exists A$, a polynomial-time algorithm, $\forall P \in G_1, Q \in G_2$
 A computes $e(P, Q)$ efficiently.

If $G_1 = G_2$, the pairing is called symmetric. Symmetric pairings are widely used in cryptography [6, 8, 11, 28, 31]. Otherwise the pairing is called asymmetric.

We note that symmetric pairing is usually realized as the modified Tate pairing and the asymmetric pairing is usually realized as the reduced Tate pairing.

2.5 Complex Multiplication Method

In the previous Section, we know that the embedding degrees of supersingular curves are small. But there are also ordinary curves with low embedding degrees. All known techniques for generating ordinary curves with low embedding degrees are based on the Complex Multiplication (CM) method.

Let q be a prime. The CM method is an algorithm for finding an elliptic curve E over F_q that E has $q + 1 - t$ points in F_q and $-2\sqrt{q} \leq t \leq 2\sqrt{q}$.

The CM equation is

$$DV^2 = 4q - t^2,$$

where the discriminant D is positive. We omit the details of the CM method here. The algorithm is described in [33].

Miyaji et al. construct a method for finding ordinary curves with embedding degree 3, 4 or 6 [27]. Freeman shows a method for finding ordinary curves with embedding degree 10 [14]. The above works are all based on the CM method.



Chapter 3

Secret Sharing Schemes

A secret sharing scheme is composite of a dealer, a set of n parties, and a collection A of subsets of parties. We call A the access structure. The dealer has a secret, and he distributes the shares of the secret to the parties. The scheme ensures two things. First any subset in A can reconstruct the secret from its shares. Secondly, any subset not in A cannot get any partial information on the secret. The first property is called correctness and the second is called perfect privacy.

Definition 3.1 (Monotone Access Structure). [5] Let $\{P_1, P_2, \dots, P_n\}$ be a set of parties. A collection $A \subseteq 2^{\{P_1, P_2, \dots, P_n\}}$ is monotone if $B \in A$ and $B \subseteq C$ then $C \in A$. A monotone access structure is a monotone collection A of non-empty subsets of $2^{\{P_1, P_2, \dots, P_n\}}$. The sets in A are called the authorized sets, and the sets not in A are called the unauthorized sets.

Most well-known secret-sharing schemes are linear. In a linear secret sharing scheme, the secret is an element of a finite field and the generation of shares is done by a linear combination of the secret and some random numbers. In the following, we present two important linear secret sharing schemes.

3.1 Shamir's Threshold Secret Sharing Scheme

Consider the $t - out - of - n$ access structure $A_t = \{s \subseteq \{P_1, P_2, \dots, P_n\} \mid |s| < t\}$ where $s, t \in N$ and $1 \leq t \leq n$. If a secret sharing scheme's access structure is $t - out - of - n$, we define it as threshold secret sharing scheme. Shamir gave an elegant construction based on the well-known fact that it takes d points to define a polynomial of $d - 1$ degree where $d \in N$ and $d \geq 1$ [32].

Suppose the dealer has a secret $s \in F_q$ and he wants to use a $t - out - of - n$ access structure A_t to share the secret. The dealer chooses a random polynomial $p(x)$ of degree t and set $p(0) = s$. The shares of party P_j is $p(j)$. Any authorized sets can reconstruct the secret by using Lagrange's interpolation.

Theorem 3.2 (Lagrange interpolation). Given t distinct points (x_i, y_i) , where y_i is of the form $f(x_i)$ and $f(x)$ is a polynomial of degree less than t , then $f(x)$ is determined by

$$f(x) = \sum_{i=1}^t y_i \prod_{\substack{1 \leq j \leq t \\ i \neq j}} \frac{x - x_j}{x_i - x_j}.$$

And we call $\prod_{\substack{1 \leq j \leq t \\ i \neq j}} \frac{x - x_j}{x_i - x_j}$ Lagrange coefficient.

For example, the dealer wants a $3 - out - of - 5$ access structure to share the secret $100 \in F_{101}$. The dealer choose a random polynomial $p(x) = 23x^2 + 65x + 100 \pmod{101}$. P_1 will receive $p(1) \equiv 87$, P_2 will receive $p(2) \equiv 19$, P_3 will receive $p(3) \equiv 98$, P_4 will receive $p(4) \equiv 21$, and P_5 will receive

$p(5) \equiv 91$. If P_1 , P_3 , and P_4 want to reconstruct the secret, they will first compute the Lagrange coefficients:

$$l_1 = (0 - 3)(0 - 4)/(1 - 3)(1 - 4) \equiv 2 \pmod{101},$$

$$l_3 = (0 - 1)(0 - 4)/(3 - 1)(3 - 4) \equiv 99 \pmod{101}, \text{ and}$$

$$l_4 = (0 - 1)(0 - 3)/(4 - 1)(4 - 3) \equiv 1 \pmod{101}.$$

Then they get s after computing $\sum y_i l_i = 87 \times 2 + 98 \times 99 + 1 \times 21 = 9897 \equiv 101 \pmod{100}$.

Shamir's secret sharing scheme is a key technique to many attribute-based encryption schemes [16, 31]. In [16], Goyal et al. construct an access tree based on Shamir's secret sharing scheme. We will discuss their construction in Chapter 4.

3.2 Monotone Span Program Construction

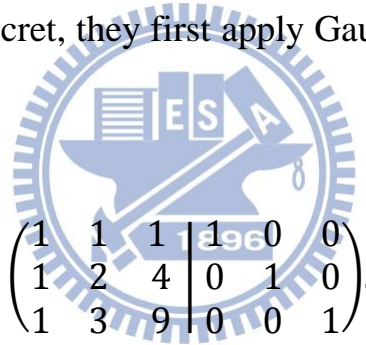
Definition 3.3 (Monotone Span Program) [20]. A monotone span program (MSP) is a quadruple $\mathcal{M} = (F, M, \pi, e)$, where F is a field and M is a $r \times c$ matrix over F . $\pi: \{1, \dots, r\} \rightarrow \{P_1, \dots, P_1\}$ maps each row of M to a party, and $e = (1, 0, 0, \dots) \in F^c$ is called target vector. The size of \mathcal{M} is the number r of rows and is denoted as $size(\mathcal{M})$. Let B be a set of parties, we denote M_B by restricting M to the rows labeled by parties in B . We define that M accepts B if rows of M_B span the vector e . We also define that M accepts an access structure A if M accepts a set B if and only if $B \in A$.

We now describe how to use the MSP to construct a secret sharing scheme [5]. First we have an MSP $\mathcal{M} = (F, M, \pi, e)$ corresponding to an access structure A . Then we consider the column vector $v = (s, r_2 \dots r_c)$, where s is the secret and $r_2 \dots r_c$ are randomly chosen. Then we compute Mv . Mv is the vector of r shares according to \mathcal{M} . The shares $(Mv)_i$ belongs to the party $\pi(i)$.

For example, consider the following MSP $\mathcal{M} = (F_{101}, M, \pi, (1,0,0))$

$$\text{where } M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{pmatrix}$$

and $\pi(1) = P_1$, $\pi(2) = P_2$, $\pi(3) = P_3$, $\pi(4) = P_4$, and $\pi(5) = P_5$. If the secret $s = 100$ and we set $v = (100, 5, 37)$. Then we compute $Mv = (41, 56, 44, 5, 40)^T$. After that, P_1 gets 41, P_2 gets 56, P_3 gets 44, P_4 gets 5, and P_5 gets 40. We note that it is a 3-out-of-5 threshold secret sharing, because any 3 rows of M can form a full rank matrix. If P_1 , P_2 , and P_3 wants reconstruct the secret, they first apply Gauss-Jordan elimination to the following matrix:



$$\left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 4 & 0 & 1 & 0 \\ 1 & 3 & 9 & 0 & 0 & 1 \end{array} \right).$$

And they will get

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 3 & 98 & 1 \\ 0 & 1 & 0 & 48 & 4 & 49 \\ 0 & 0 & 1 & 51 & 100 & 51 \end{array} \right).$$

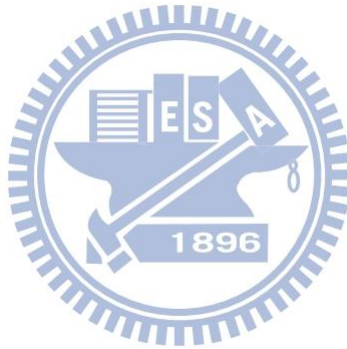
After the Gauss-Jordan elimination, they could compute the secret s by $3 \times 41 + 98 \times 56 + 1 \times 44 = 5655 \equiv 100 \pmod{101}$

The above example is a 3-out-of-5 threshold secret sharing. On the other hand, MSP is not limited to threshold secret sharing. It can describe a boolean formula directly. For example, consider the following MSP matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 3 \end{pmatrix}.$$

And we set $\pi(i) = P_i \forall i \leq 6, i \in N$. The MSP describes the boolean formula $(P_1 \text{ AND } P_2 \text{ AND } P_3) \text{ OR } 2 \text{ OF } (P_4, P_5, P_6)$. We can verify that by examining if the parties' corresponding rows can span $(1,0,0,0)$.

Although an MSP can describe a boolean formula, the conversion from a boolean formula to an MSP is not trivial. In Chapter 5, we give an algorithm which converts any monotone boolean formula to an MSP.

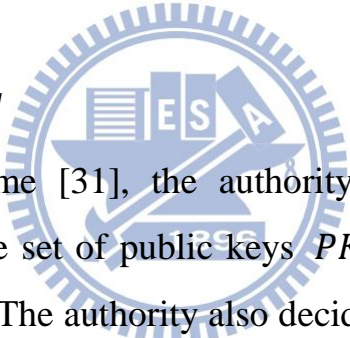


Chapter 4

Attribute-Based Encryption

In this chapter, we will review the variants of attribute-based encryption (ABE) in recent researches.

4.1 Threshold ABE



In a threshold ABE scheme [31], the authority will publish the set U of attributes it controls and the set of public keys PK in which every public key corresponds to an attribute. The authority also decides the threshold $t \in N$. The encryptor can choose a subset $\omega \subseteq U$ and encrypt the plaintext M under corresponding public keys. The decryptor will receive his private keys corresponding to a subset $\omega' \subseteq U$ from the authority. If $|\omega \cap \omega'| \geq t$, then the ciphertext can be recovered.

We will discuss their construction in details and the relationship between Shamir's secret sharing scheme and Threshold ABE.

Setup First, the authority chooses a symmetric pairing $e: G_1 \times G_1 \rightarrow G_T$. The order of G_1 is a prime r and g is a generator of G_1 .

Next, the authority decides the set U of attributes. Each attribute is labeled from 1 to $|U|$ and the number is used to index the attributes. The authority

then choose $t_1 \dots t_{|U|}$ uniformly at random from Z_r and choose y uniformly at random from Z_r .

Finally, the public parameters are:

$$T_1 = t_1 g, \dots, T_{|U|} = t_{|U|} g, Y = e(g, g)^y.$$

Each T_i represents an attribute's public key and each t_i represents an attribute's secret key. The authority keeps $t_1 \dots t_{|U|}$, y in secret.

Encryption If Alice wants to encrypt a message $M \in G_T$ under attribute set $\omega' \subseteq U$, she first chooses a random $s \in Z_r$. The ciphertext is

$$E = (\omega', MY^s, (sT_i)_{i \in \omega'}).$$

Key Generation If Bob wants apply his private keys for his attribute set $\omega \subseteq U$, the authority first chooses a $(t-1)$ degree polynomial p and sets $p(0) = y$. The authority then computes $p(j) \forall j \in \omega$. The private key consists of

$$(D_j)_{j \in \omega} = \frac{p(j)}{t_j} g \quad \forall j \in \omega.$$

We note that this procedure is similar to Shamir's threshold secret scheme as described in Section 3.1. The difference is that each share is binding to an attribute rather than a party and each share is represented as an element in G_1 .

Decryption If Bob wants to decrypt Alice's ciphertext $E = (\omega', MY^s, (sT_i)_{i \in \omega'})$ with his private keys associated with $\omega \subseteq U$.

Suppose that $S = \omega \cap \omega'$ and $|S| \geq t$. In the view of secret sharing, this means Bob has authorized sets. Therefore, Bob has the ability to decrypt the ciphertext. First he chooses an arbitrary subset $S' \subseteq S$ where $|S'| = t$ and computes Lagrange's coefficient $l_i \forall i \in S'$. Then he computes

$$\prod_{i \in S'} e(sT_i, D_i)^{l_i} = \prod_{i \in S'} e(st_i g, \frac{p(i)}{t_i} g)^{l_i} = e(g, g)^{s \sum_{i \in S'} p(i) l_i} = e(g, g)^{s y}.$$

Finally he gets M by computing $MY^s / e(g, g)^{s y} = M$.

Different users cannot collude to decrypt a ciphertext when neither of them can decrypt the ciphertext. In this scheme, the collusion is not allowed because each user's private keys corresponding to a different polynomials. For example, if Eve has private keys associated with attribute set $s = \{1,2\}$ and Alice has private keys associated with attribute set $s' = \{3\}$. There is a ciphertext associated with attribute set $\omega = \{1,2,3\}$ and the threshold t is 3. Suppose that Eve steals Alice's private keys, she cannot decrypt the ciphertext because Alice's keys are binding to another polynomial from Eve's. Therefore she cannot interpolate the secret $e(g, g)^{s y}$.

4.2 Key-Policy ABE

In a key-policy ABE (KP-ABE) [16] scheme, each ciphertext is associated with a set of attributes and each private key is associated with an access formula. The access formula describes which type of ciphertexts the key can decrypt.

For example, a stream video online may be encrypted with the attributes: "NBA", "Season2012", and "Playoffs". If Alice receives her private key for the

access formula “NBA” AND “Season2012” AND “Regular Season”, then Alice can only watch the regular season games of NBA 2012 but not the playoffs. Therefore she cannot decrypt the encrypted stream. On the other hand, If Bob receives his private key for the access formula “NBA” AND “Season=2012”, which means Bob can watch all NBA games in 2012 no matter regular season or playoffs.

Goyal et al. (GPSW) constructs the scheme based on the access tree technique. Given an access formula, we can easily construct an access tree represents the access formula. Every non-leaf node of the tree represents a $t - out - of - n$ threshold gate. We note that when $t = 1$ and $n = 2$, the threshold gate is an *OR* gate. And when $t = 2$ and $n = 2$, the threshold gate is an *AND* gate. Each leaf node x of the tree represents an attribute. We define the function $parent(x)$ as returning the parent of the node x . The function $attr(x)$ is defined only if x is a leaf node and returns the attribute the leaf node represents.

We also define an ordering between the children of every node, i.e., the children of a node are numbered from 1 to n . Therefore, the function $index(x)$ is defined as returning the number associated with the node x .

Figure 4.1 is an example of access tree T . In this example, T describes an access formula: $(A_1 OR A_2) AND 2 OF (A_3, A_4, A_5)$. The access formula and the access tree can be easily transformed to each other. The access structure is $\{ (A_1, A_3, A_4), (A_1, A_3, A_5), (A_1, A_4, A_5), (A_2, A_3, A_4), (A_2, A_3, A_5), (A_2, A_4, A_5), (A_1, A_2, A_3, A_4), (A_1, A_2, A_3, A_5), (A_1, A_2, A_4, A_5), (A_1, A_3, A_4, A_5), (A_2, A_3, A_4, A_5), (A_1, A_2, A_3, A_4, A_5) \}$.

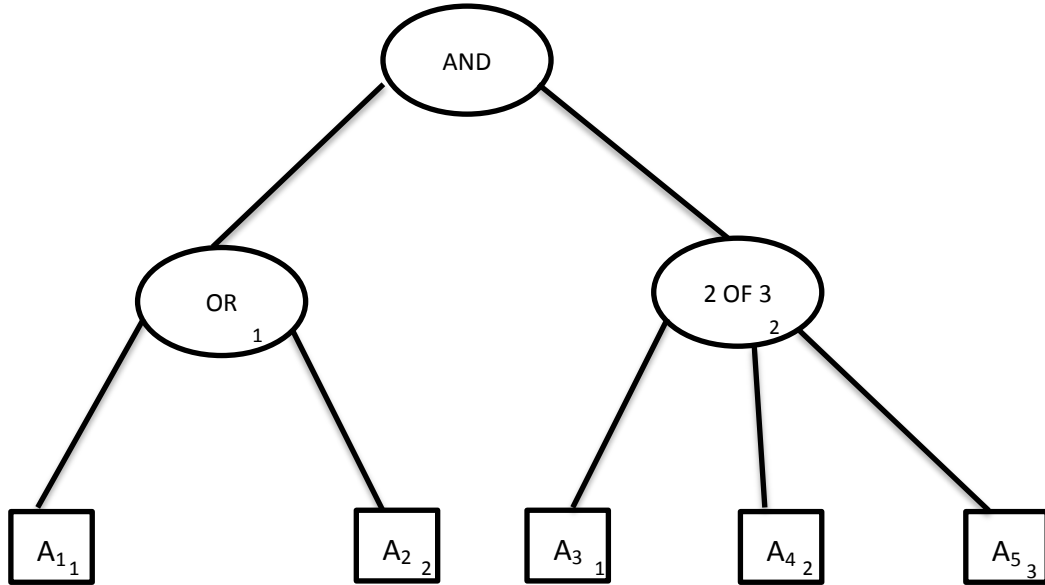


Figure 4.1: An example of access tree

We now give GPSW construction in the below.

Setup First, the authority chooses a symmetric pairing $e: G_1 \times G_1 \rightarrow G_T$. The order of G_1 is a prime r and g is a generator of G_1 .

Next, the authority decides the set U of attributes. Each attribute is a string and labeled from 1 to $|U|$ and the number is used to index the attributes. The authority then choose $t_1 \dots t_{|U|}$ uniformly at random from Z_r and choose y uniformly at random from Z_r .

Finally, the public parameters are:

$$T_1 = t_1 g, \dots, T_{|U|} = t_{|U|} g, Y = e(g, g)^y.$$

The authority keeps $t_1 \dots t_{|U|}$, y in secret.

Encryption If Alice wants to encrypt a message $M \in G_T$ under attribute set $\omega' \subseteq U$, she first chooses a random $s \in Z_r$. The ciphertext is

$$E = (\omega', MY^s, (sT_i)_{i \in \omega'}).$$

Key Generation If Bob wants apply his private keys for his access formula, the authority first covert the formula to an access tree T .

Then for each node x in the tree T , the authority chooses a random polynomial p_x and the degree d_x of p_x is $t - 1$. For the root node k , the authority sets $p_k(0) = s$ and for any other node x , he sets $p_x(0) = p_{parent(x)}(index(x))$.

Finally, for each leaf node x , the authority gives the following private key to Bob:

$$D_x = \frac{p_x(0)}{t_j} g \text{ where } j = attr(x).$$

Decryption If Bob wants to decrypt Alice's ciphertext $E = (\omega', MY^s, (sT_i)_{i \in \omega'})$ with his private key associated with access tree T . Suppose that ω' can satisfy the access tree T , then Bob has the ability to decrypt the ciphertext. For each leaf node x , he computes

$$e(D_x, sT_i) = e\left(\frac{p_x(0)}{t_i} g, st_i g\right) = e(g, g)^{sp_x(0)} \text{ for } attr(x) = i.$$

Because $p_x(0) = p_{parent(x)}(index(x))$. As a result, Bob can then interpolate $e(g, g)^{sp_{parent(x)}(0)}$ for node x . Bob repeats this approach until he meets root node k . Therefore, $e(g, g)^{sp_r(0)} = e(g, g)^{sy}$ is computed and he gets $MY^s / e(g, g)^{sy} = M$.

We can view threshold ABE as a special case of KP-ABE, because in a

Threshold ABE scheme the private key is associated with an access tree having just one threshold gate.

We also note that the access tree can be replaced with an MSP. If the MSP is used, we only need to apply Gauss-Jordan elimination one time rather than interpolate the polynomials multiple times. Therefore, in subsequent research [23, 34], the MSP is used in place of the access tree.

4.3 Ciphertext-Policy ABE

The concept of ciphertext-policy ABE (CP-ABE) is introduced in [16] and Bethencourt et al. (BSW) give the first construction [6]. The roles of ciphertexts and keys are reversed. Each ciphertext is associated with an access formula and each private key is associated with a set of attributes.

For example, a job posting may be encrypted under the access formula: “master degree” and “two years’ work experience”. Suppose Alice has two private keys. One is for attribute “master degree”. The other is for attribute “two years’ work experience”. Therefore Alice can decrypt the job posting.

The security of BSW construction is argued in the generic group model. Subsequently Waters [34] proposed ciphertext-policy ABE constructions in the standard model. We now give Water’s construction in the below.

Setup First, the authority chooses a symmetric pairing $e: G_1 \times G_1 \rightarrow G_T$. The order of G_1 is a prime r and g is a generator of G_1 . Next, the authority decides the set U of attributes. Each attribute is a string and labeled from 1 to $|U|$ and the number is used to index the attributes. The authority then choose $h_1 \dots h_{|U|}$ uniformly at random from G_1 . In addition he chooses random exponents $\alpha, a \in Z_r$.

Finally, the public parameters are:

$$h_1, \dots, h_{|U|}, g, e(g, g)^\alpha, ag.$$

The authority keep ag in secret.

Encryption If Alice wants to encrypt a message $M \in G_T$ associated with an access formula, she first converts the access formula to the MSP \mathcal{M} . Assume that the MSP matrix N of \mathcal{M} is a $n \times c$ matrix and π mapping its rows to attributes.

She then choose a random $s \in Z_r$, and computes the shares of s by using the MSP \mathcal{M} . The shares of s is the set $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$. In addition, she chooses a set of n random numbers $\{r_1, r_2, \dots, r_n\}$ where each $r_i \in Z_r$.

For each $\{\lambda_i, r_i\}$, Alice computes $C_{1,i} = \lambda_i ag + (-r_i h_{\pi(i)})$, $C_{2,i} = r_i g$. We note that ag and $h_{\pi(i)}$ are public keys.

Finally, the ciphertext is:

$$C = \{C_0 = Me(g, g)^{\alpha s}, C' = sg, \{C_{1,i}, C_{2,i} \forall i\}, \mathcal{M}\}.$$

Key Generation If Bob wants to apply his private keys for his attribute set $\omega \subseteq U$, then the authority first chooses a random $t \in Z_r$. Next, it creates the private key as

$$\{K = ag + atg, L = tg, \{K_j = th_j\} \forall j \in \omega\}.$$

Decryption Suppose that Bob has enough private keys to decrypt C . That is, he has the private keys $\{K_{\pi(i)}\}$ for a subset of rows N_i of N such that $(1, 0, \dots, 0)$ can be spanned by these rows. Recall that λ_i 's are shares of secret s .

Therefore, Bob can compute a set of c_i 's $\in Z_r$ by applying Gauss-Jordan elimination such that $\sum c_i \lambda_i = s$. Bob first computes:

$$\begin{aligned} & e(C', K) / \prod_i \left(e(C_{1,i}, L) e(C_{2,i}, K_{\pi(i)}) \right)^{c_i} = \\ & e(g, g)^{\alpha s} e(g, g)^{\alpha s t} / \prod_i \left(e(g, g)^{t \alpha \lambda_i} \right)^{c_i} = e(g, g)^{\alpha s}. \end{aligned}$$

Then the plaintext M can be obtained as $M = C_0 / e(g, g)^{\alpha s}$.

In a CP-ABE scheme, the key technique to avoid collusion attack is that each user's private keys are binding to a random exponent t . Therefore different users cannot collude. This is different from the threshold ABE scheme. In the threshold ABE scheme, the technique to avoid collusion is that each user's key is binding to different random numbers in the linear secret sharing scheme. So each user actually solves different secret sharing problems. But in the CP-ABE scheme, the random numbers in the secret sharing scheme are decided by the encryptor and those random numbers are binding to the ciphertext, so each user tries to solve the same secret sharing problem. Therefore we must choose another random number t to "personalize" the private keys.

4.4 Dual-Policy ABE

Dual-policy ABE (DP-ABE) conjunctively combines KP-ABE with CP-ABE. The ciphertext is associated with an access formula and a set of attributes simultaneously. The private key is also associated with an access formula and a set of attributes. The decryption can be done if and only if the ciphertext's attributes satisfies the private key's access formula and the private key's attributes satisfies the ciphertext's access formula in the same time.

The trivial construction is that we encrypt the plaintext twice. First we encrypt the plaintext using CP-ABE, and then we encrypt the previous result again using KP-ABE. The decryption is also two-step. First we decrypt the ciphertext under KP-ABE private keys, and then we decrypt again under CP-ABE private keys.

Attrapadung et al. [3] proposed a non-trivial DP-ABE construction. Their construction combines [16] with [34]. In their construction, the encryption and the decryption can be done in one step.

4.5 ABE with Multiple Authorities

[16, 28, 31, 34] are all single authority constructions. However, in real world, there are many authorities who control their own attributes. For example, two universities may have a joint research project. But the two universities do not want a third single authority is responsible for their attributes and issues private keys for them. They want to control their attributes themselves. Therefore, they need a multi-authority scheme.

Chase [11] presents a KP-ABE scheme with multi authorities. In Chase's scheme, global identifiers (GID) are introduced and every user is binding to a unique GID. The GID is used to "link" private keys from different authorities together. There is also a central authority (CA) in Chase's scheme. The CA is responsible for choosing the system master key and controlling all the other authorities' authority secret key. Therefore, in Chase's system, we still need to trust a single authority. In addition, the number of authorities is decided in CA's setup procedure. After CA's setup, no more authorities can join the system. Moreover, her system is restricted to express a conjunctive access formula across the set of authorities. That is, the access tree of Chase's scheme will be

like Figure 4.2. Each subtree represents an authority's access tree. The decryption is permitted if and only if all subtrees are satisfied. We also note that one authority's attributes can just appear in its own subtree.

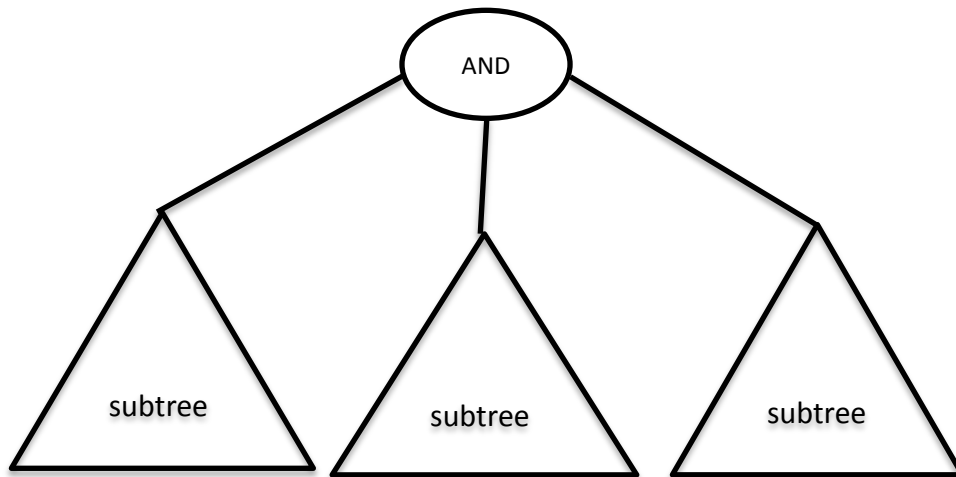


Figure 4.2: An example of access tree in Chase's scheme

Chase and Chow [12] modified the above construction. They remove the CA by using distributed pseudo random functions. But the restriction of expressiveness and the pre-determined set of authorities still remained.

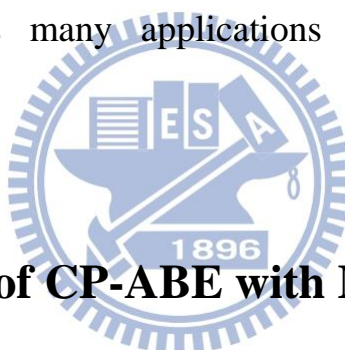
Lewko and Waters (LW) [23] propose a multi-authority CP-ABE scheme. Their system does not require any CAs. In addition, any party can become an authority after the system's setup. Moreover, the plaintext can be encrypted under any monotone boolean formulas over attributes.

LW construction requires global identifiers (GID) as [11] suggested. We will show our implementation of LW scheme in Chapter 5.

Chapter 5

Implementation and Applications

In this chapter, we first discuss our implementation of CP-ABE with multiple authorities. Next we give an algorithm that can convert an access formula to an MSP. Finally we discuss many applications of CP-ABE with multiple authorities.



5.1 Implementation of CP-ABE with Multiple Authorities

The implementation is based on the scheme described in [23]. We will first discuss the library we use and then describe the algorithms in the scheme in detail. And the experiment result will be given in the end.

5.1.1 The Pairing-based Cryptography Library

The pairing-based cryptography (PBC) library [25] is an open source library that is released under the GNU Lesser General Public License. The PBC library is written in C and provides routines such as elliptic curve generation, elliptic curve arithmetic and pairing computation. Many projects are based on the PBC library, such as [1, 2, 28].

We have tested the speed of the PBC library. We performed our experiments on a 2.4 GHz Intel Xeon E5620 processor running Ubuntu 11.10. The security level we choose is 128-bit. Table 5.1 is the key size comparison under different security levels [36].

Date	Minimum of Strength	Symmetric Key	RSA and DH	Elliptic Curve
2010	80	80	1024	160
2011-2030	112	112	2048	224
> 2030	128	128	3072	256

Table 5.1: NIST recommended key sizes (bits)

There are seven types of pairings defined in the PBC library. The seven types are type A, type B, type C, type D, type E, type F and type G. Type A, type B and Type C are supersingular curves. Type D, type E, type F and type G are based on the complex multiplication (CM) method. However, type B and type C are not implemented yet.

Type A pairings are constructed on the curve $E: y^2 = x^3 + x$ over F_q , where q is a prime and $q \equiv 3 \pmod{4}$. E is a supersingular curve, so this pairing is a symmetric pairing $e: G_1 \times G_1 \rightarrow G_T$. G_T is a subgroup of F_{q^2} because the embedding degree is 2. Therefore we choose the group order r to be 256-bit long and q to be 1536-bit long, because q^2 must be 3072-bit long to achieve the same security level as 256-bit long in elliptic curve.

Type D pairings are constructed on the MNT curves of embedding degree 6 [27]. This pairing is an asymmetric pairing $e: G_1 \times G_2 \rightarrow G_T$. G_T is a subgroup of F_{q^6} because the embedding degree is 6. Given different discriminant in the CM equation, the bits in q and the bits in r are determined.

Therefore we choose two suitable type D pairings. One is that the discriminant is 31387, q is 522-bit long and r is 514-bit long. The other is that discriminant is 873867, q is 486-bit long and r is 442-bit long

Type E pairings are constructed on the curves of embedding 1 [21]. The pairing is a symmetric pairing $e: G_1 \times G_1 \rightarrow G_T$. G_T is a subgroup of F_q because the embedding degree is 1. Therefore we choose the group order r to be 256-bit long and q to be 3072-bit long, because q must be 3072-bit long to achieve the same security level as 256-bit long in elliptic curve.

Type F pairings are constructed on the curves of embedding 12 [4]. This pairing is an asymmetric pairing $e: G_1 \times G_2 \rightarrow G_T$. G_T is a subgroup of $F_{q^{12}}$ because the embedding degree is 12. Therefore we choose the group order r to be 256-bit long and q to be 256-bit long.

Type G pairings are constructed on the curves of embedding 10 which Freeman suggests [14]. Given different discriminant in the CM equation, the bits in q and the bits in r are determined. Therefore we choose one suitable type G pairings. The curve is that the discriminant is 35707, q is 301-bit long and r is 279-bit long. Table 5.2 is a comparison of the pairings in the PBC library.

	Embedding Degree	Symmetric Pairing	Supersingular
Type A	2	yes	yes
Type D	6	no	no
Type E	1	yes	no
Type F	12	no	no
Type G	10	no	no

Table 5.2: Pairings in the PBC library

	Pairing Time (ms)	Multiplication Time in G_T (ms)	Addition Time in G_1 (ms)	Addition Time in G_2 (ms)
Type A	38	0.009	0.042	0.042
Type D-311387	48	0.023	0.011	0.078
Type D-873867	35	0.020	0.010	0.068
Type E	87	0.009	0.108	0.108
Type F	49	0.037	0.006	0.009
Type G-35707	45	0.036	0.006	0.090

Table 5.3: Comparison of speed of different pairings

For each type, we choose 10 random inputs of the pairing function and compute the average time. We also choose 100 random elements for G_1 , G_2 and G_3 for each type and compute the average time of an addition or an multiplication. The result of our test is shown in Table 5.3. We note that in our encryption scheme, we need a symmetric pairing. Therefore, in our implementation, we choose the type A pairing, because the type E pairing is the slowest pairing.

5.1.2 Implementation

We first proposed the CP-ABE with multiple authorities scheme described in [23] and discuss our implementation later.

Global Setup In the global setup, a pairing $e: G_1 \times G_1 \rightarrow G_T$ is chosen. The order of G_1 is a prime r and g is a generator of G_1 . A hash function

$H: \{0,1\}^* \rightarrow G_1$ is also decided. Pairing e , generator g and H are all public after the global setup.

Authority Setup If a party wants to be an authority, for each attribute the party controls, the party chooses two random numbers $\alpha_i, y_i \in Z_r$. The authority's secret key is the set $\{\alpha_i, y_i \forall i\}$ and the authority's public key is the set $\{e(g, g)^{\alpha_i}, y_i g \forall i\}$

Encryption If Alice wants to encrypt a plaintext $M \in G_T$ under an access formula A . She first converts A to an MSP \mathcal{M} . Suppose that the MSP matrix N of \mathcal{M} is a $n \times c$ matrix with π mapping its rows to attributes.

Alice then chooses a random $s \in Z_r$ and computes the shares of s and the shares of 0 by using the MSP \mathcal{M} . The shares of s is the set $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ and the shares of 0 is the set $\{w_1, w_2, \dots, w_n\}$. In addition, she chooses a set of n random numbers $\{r_1, r_2, \dots, r_n\}$ and each $r_i \in Z_r$.

For each $\{\lambda_i, w_i, r_i\}$, Alice computes $C_{1,i} = e(g, g)^{\lambda_i} e(g, g)^{\alpha_{\pi(i)} r_i}$, $C_{2,i} = r_i g$ and $C_{3,i} = r_i y_{\pi(i)} g + w_i g$. We note that $e(g, g)^{\alpha_{\pi(i)}}$ and $y_{\pi(i)} g$ are public keys.

Finally, the ciphertext is:

$$C = \{C_0 = Me(g, g)^s, \{C_{1,i}, C_{2,i}, C_{3,i} \forall i\}, \mathcal{M}\}.$$

Key Generation Every user is binding to a unique GID . Suppose that Bob's GID is GID_{Bob} . To issue a private for Bob for attribute x , the authority responsible for x computes:

$$K_{x, GID_{Bob}} = \alpha_x g + y_x H(GID_{Bob}).$$

Decryption Suppose that Bob has enough private keys to decrypt C . That is, he has the private keys $\{K_{\pi(i),GID_{Bob}}\}$ for a subset of rows N_i of N such that $(1,0, \dots, 0)$ can be spanned by these rows. For each i , Bob computes:

$$C_{1,i} \cdot e(H(GID_{Bob}), C_{3,i}) / e(K_{\pi(i),GID_{Bob}}, C_{2,i}) = e(g, g)^{\lambda_i} e(H(GID_{Bob}), g)^{w_i}.$$

Recall that λ_i 's are shares of secret s and w_i 's are shares of secret 0 . Therefore, Bob can compute a set of c_i 's $\in Z_r$ by applying Gauss-Jordan elimination such that $\sum c_i \lambda_i = s$ and $\sum c_i w_i = 0$. Bob then computes

$$\prod_i (e(g, g)^{\lambda_i} e(H(GID_{Bob}), g)^{w_i})^{c_i} = e(g, g)^s.$$

And the plaintext M can be obtained as $M = C_0 / e(g, g)^s$.

We have implemented the above scheme. The implementation contains six programs.

global-setup

Generates a pairing e and the generator g .

user-register

Given a user's name, generates a GID for the user.

authority-setup

Given the list of attributes the authority controls, generates the public key and the secret key for each attribute.

authority-keygen

Given the user's GID and the list of user's attributes the authority controls, generates the private keys for the GID for each attribute.

encrypt

Given an access formula and a file, generates the encrypted file under the access formula.

decrypt

Given an encrypted file and private keys, decrypts the file if satisfying the access formula.

We also construct a graphical user interface program to demonstrate our implementation. See Figure 5.1 and Figure 5.2.

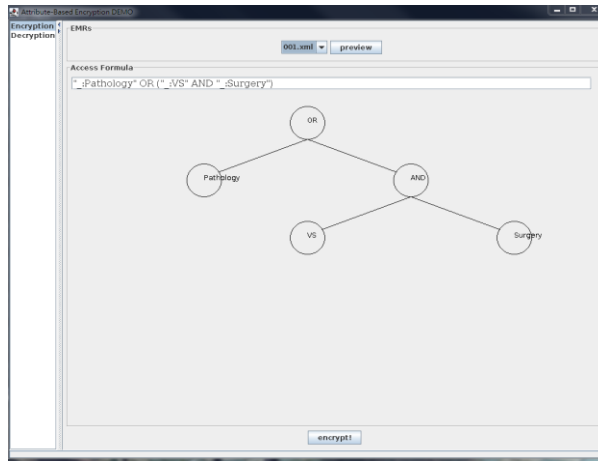


Figure 5.1: A demonstration of encryption in CP-ABE scheme

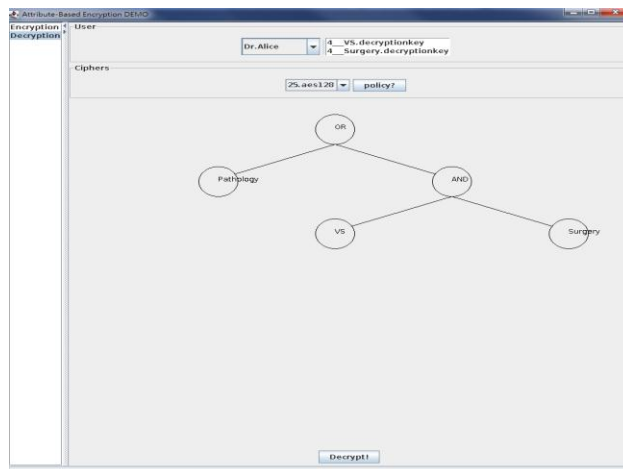


Figure 5.2: A demonstration of decryption in CP-ABE scheme

5.1.3 Evaluation

We have tested the speed of the encryption and decryption of our implementation. We performed our experiments on a 2.4 GHz Intel Xeon E5620 processor running Ubuntu 11.10. The security level we choose is 128-bit. We use the type A pairing in which r is 256-bit long and q is 1536-bit long. Our plaintext is 3072-bit long.

Figure 5.3 and Figure 5.4 summarize our measurements. In each figure the x-axis represents the number of rows in the MSP matrix and y-axis shows the time required to complete an encryption or a decryption. The figures show that the processing time and the complexity of the access formula are in direct proportion. In addition, the processing time shows that the scheme is practical on a workstation or a personal computer.

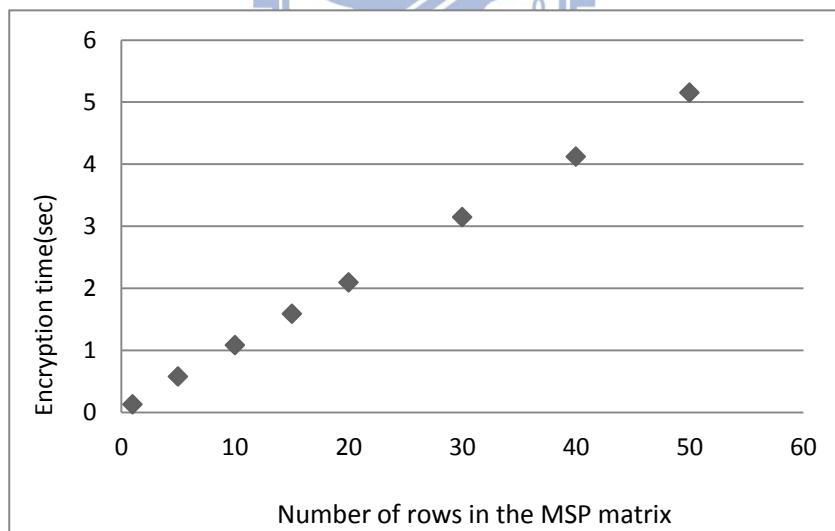


Figure 5.3: Encryption time

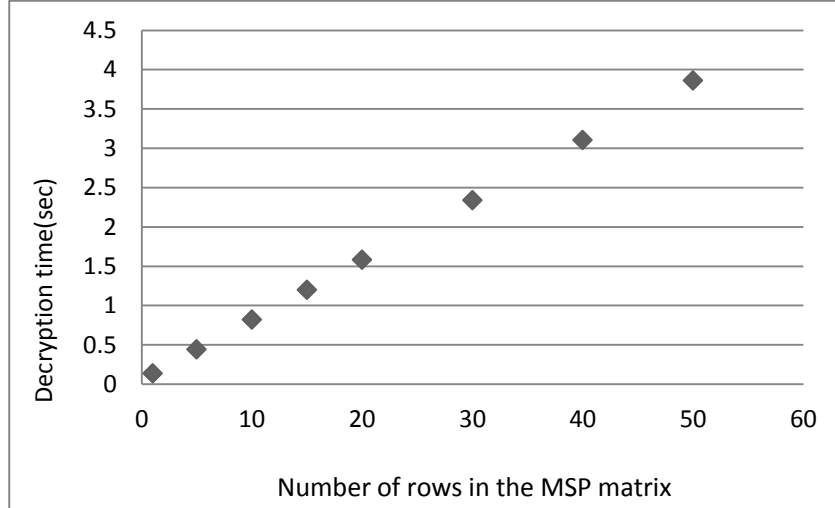


Figure 5.4: Decryption time

5.2 Converting form the Access Formula to the MSP

Lewko and Waters (LW) describe an algorithm in [24]. The algorithm's input is an access tree and the output is an MSP \mathcal{M} where $\text{size}(\mathcal{M})$ equals the number of leaf nodes. However, the algorithm cannot handle threshold gates. The all nodes of the input access tree must be *AND* gates and *OR* gates and leaf node gates but not threshold gates.

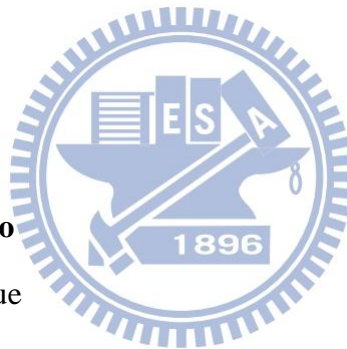
The idea of LW algorithm is that we first convert the access formula to an access tree and then we label each node with a vector determined by the vector assigned to its parent node. We first set an initial vector (1) and label the root node with (1) and set counter $c = 1$. Then we use queue-based level order traversal from the root. If the node is an *OR* gate labeled with the vector v , we also label its two children with v . If the node is an *AND* gate, we label one of its two children with $(v|1)$ where $|$ denotes concatenation and we label the other with the vector $(0, \dots, 0| - 1)$ where $(0, \dots, 0)$ denotes the zero vector which has length c . In addition we set $c = c + 1$. Therefore, the two vectors sum to $(v|0)$. We have implemented the above algorithm. Table 5.4 is the pseudo code.

LW Algorithm

Input: an access tree T which has no threshold gates

Output: An MSP \mathcal{M}

```
1.   r = number of leaf nodes of T
2.   l = (number of AND gates of T) + 1
3.   M = a  $r \times l$  matrix in which all entries are 0s.
4.   v = (1, 0, 0, ...)  $\in Z^l$ 
5.    $\mathcal{M} = (F, M, \pi, e)$  in which  $\pi$  is empty
6.   count = 1
7.   node = T's root
8.   q = empty queue
9.   q.enqueue((node, v))
10.
11.  while q is not empty do
12.      (node, v) = q.dequeue
13.      if node is an OR gate then
14.          queue.enqueue((node's left child, v))
15.          queue.enqueue((node's right child, v))
16.      else if node is an AND gate then
17.          l_v = v
18.          set l_v[count+1] = 1
19.          r_v = v - l_v
20.          count = count + 1
21.          queue.enqueue((node's left child, l_v))
22.          queue.enqueue((node's right child, r_v))
23.      else if node is a leaf node then
```



```

24      $M_i$  = find a row whose entries are all zeros from  $M$ 
25     set  $M_i = v$ 
26     set  $\pi(i) = \text{party}(\text{node})$ 
27     end if
28     end while
29     return  $\mathcal{M}$ 

```

Table 5.4: Pseudo code of LW algorithm

For example, consider the access tree T represents the formula $P_1 \text{ AND } (P_2 \text{ AND } (P_3 \text{ OR } P_4))$. We will first construct the 4×3 matrix M which is zero-filled. M is 4×3 because there are 4 parties and 2 *AND* gates. The vector v is $(1,0,0)$. Then we meet the root node which is an *AND* gate. We split v into $(1,1,0)$ and $(0,-1,0)$ and set $M_1 = (1,1,0)$ and $\pi(1) = P_1$. The $(0,-1,0)$ goes to the other subtree. Because we meet an *AND* gate in the subtree, we split the $(0,-1,0)$ into $(0,-1,1)$ and $(0,0,-1)$. We then set $M_2 = (0,-1,1)$ and $\pi(2) = P_2$. The $(0,0,-1)$ goes to the other subtree. Because we meet an *OR* gate there, we set $M_3 = (0,0,-1)$, $\pi(3) = P_3$, $M_4 = (0,0,-1)$ and $\pi(4) = P_4$. Therefore

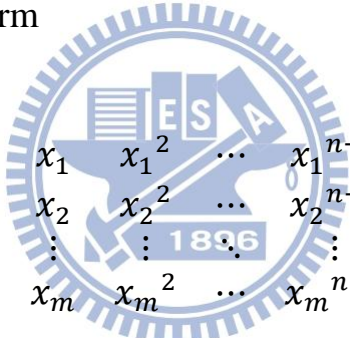
$$M = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & -1 \end{pmatrix}.$$

We now use M to share a secret s . That is, construct a vector $u = (s, r_1, r_2)$ where r_1, r_2 are randomly chosen. And compute $Mu = (s + r_1, -r_1 + r_2, -r_2, -r_2)$. P_1 will receive $s + r_1$, P_2 will receive $-r_1 + r_2$ and P_3 and P_4 will both receive $-r_2$.

LW algorithm has advantages when reconstructing the secret. When we want to reconstruct the secret, we just sum the shares we have. We do not need to multiply each share's coefficient first because each share's coefficient is always one. But the limitation of the algorithm is the disability of handling threshold gates. Therefore, we design an algorithm that can handle threshold gates and have the advantages of LW algorithm when reconstructing the secret.

5.2.1 Our Algorithm

The idea of our algorithm is from the Vandermonde matrix. A Vandermonde matrix is an matrix of the form



$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{n-1} \end{pmatrix},$$

where x_1, x_2, \dots, x_m are all distinct. The row vectors of the Vandermonde matrix are all linearly independent. If $m \geq n$, any n row vectors of the matrix can span $(1 \ 0 \ \dots \ 0) \in Z^n$.

We extend the concept by defining an matrix of the form

$$\begin{pmatrix} v & x_1 & x_1^2 & \dots & x_1^{n-1} \\ v & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v & x_m & x_m^2 & \dots & x_m^{n-1} \end{pmatrix}$$

where x_1, x_2, \dots, x_m are all distinct and v is a vector. If $m \geq n$, any n row

vectors of the matrix can span $v \mid (0 \dots 0)$.

Now back to our algorithm of converting from the access formula to the MSP. We first convert the access formula to an access tree and we model the *AND* gate as a 2 – out – of – 2 gate and model the *OR* gate as a 1 – out – of – 2 gate. Then we label each node with a vector determined by the vector assigned to its parent node. We first set an initial vector (1) and label the root node with (1) and set counter $c = 1$. Then we use queue-based level order traversal from the root. Given a t – out – of – n threshold gate labeled by v . We construct the following matrix

$$\left(\begin{array}{c|cccccc} v & 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ v & 0 & \dots & 0 & 2 & 2^2 & \dots & 2^{t-1} \\ \vdots & 0 & \dots & 0 & \vdots & \vdots & \ddots & \vdots \\ v & 0 & \dots & 0 & n & n^2 & \dots & n^{t-1} \end{array} \right)$$

We can now label the child nodes with the row vectors one by one. The $(0, \dots, 0)$ denotes the zero vector has size $(c - v's \text{ length})$. Adding the $(0, \dots, 0)$ is to make sure that the vectors of same level nodes are linearly independent. In addition we set $c = c + (t - 1)$.

We can modify the above procedure when we meet a 1 – out – of – n threshold gate or a n – out – of – n threshold gate. When we meet a 1 – out – of – n gate labeled with v , we just label the gate's child nodes all with v . When we meet a n – out – of – n gate labeled with v , we construct the following matrix.

$$\left(\begin{array}{c|cccccc} v & 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ \mathbf{0} & 0 & \dots & 0 & -1 & 0 & \dots & 0 \\ \mathbf{0} & 0 & \dots & 0 & 0 & -1 & \dots & 0 \\ \vdots & 0 & \dots & 0 & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & 0 & \dots & 0 & 0 & 0 & \dots & -1 \end{array} \right).$$

$\mathbf{0}$ represents the zero vector of the same length of v . We note that all rows of the matrix sum to $(v|0, \dots, 0)$. The pseudo code is given in Table 5.5.

For example, consider the access tree T represents the formula $3 \text{ OF } (A, B, 3 \text{ OF } (C, D, E), 1 \text{ OF } (F, G, H))$. We will first construct the 8×5 matrix M which is zero-filled. M is 8×5 because there are 8 parties and $\sum(t_i - 1) + 1 = 5$ for each threshold t_i . We also set $c = 1$. The vector v is $(1, 0, 0, 0, 0)$. Then we meet the root node which is a $3 - \text{out} - \text{of} - 4$ gate. We generate $(1, 1, 1, 0, 0)$, $(1, 2, 4, 0, 0)$, $(1, 3, 9, 0, 0)$ and $(1, 4, 16, 0, 0)$ and set $c = c + 2 = 3$. Now we set $M_1 = (1, 1, 1, 0, 0)$ and $\pi(1) = A$ and set $M_2 = (1, 2, 4, 0, 0)$ and $\pi(2) = B$. The $(1, 3, 9, 0, 0)$ goes to the $3 - \text{out} - \text{of} - 3$ gate, then we generate $(1, 3, 9, 1, 1)$, $(0, 0, 0, -1, 0)$ and $(0, 0, 0, 0, -1)$. Now we set $M_3 = (1, 3, 9, 1, 1)$, $\pi(3) = C$ and set $M_4 = (0, 0, 0, -1, 0)$, $\pi(4) = D$ and set $M_5 = (0, 0, 0, 0, -1)$, $\pi(5) = E$. Finally the $(1, 4, 16, 0, 0)$ goes to the $1 - \text{out} - \text{of} - 3$ gate. Now we set $M_6 = M_7 = M_8 = (1, 4, 16, 0, 0)$ and $\pi(6) = F$, $\pi(7) = G$, $\pi(8) = H$. Therefore

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 4 & 0 & 0 \\ 1 & 3 & 9 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 1 & 4 & 16 & 0 & 0 \\ 1 & 4 & 16 & 0 & 0 \\ 1 & 4 & 16 & 0 & 0 \end{pmatrix}.$$

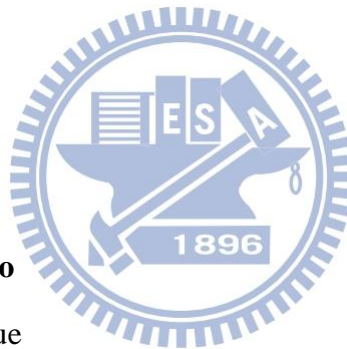
Our algorithm can handle the threshold gate. In addition, because we treat the $n - \text{out} - \text{of} - n$ gate as a special case, we can reconstruct the vector v the gate labeled with by just sum the vectors its child nodes labeled with. It is efficient because we do not need to multiply the vectors some coefficients first.

Our MSP Algorithm

Input: an access tree T

Output: An MSP \mathcal{M}

```
1      r = number of leaf nodes of T
2      c =  $\sum(t_i - 1) + 1$    (  $t_i$  is each non-leaf node's threshold value)
3      M = a  $r \times c$  matrix in which all entries are 0s.
4      v = (1, 0, 0, ...)  $\in Z^c$ 
5       $\mathcal{M} = (F, M, \pi, e)$  in which  $\pi$  is empty
6      count = 1
7      node = T's root
8      q = empty queue
9      q.enqueue((node, v))
10
11     while q is not empty do
12         (node, v) = q.dequeue
13         if node is a 1 – out – of – n gate then
14             for each child  $i$  of node do
15                 queue.enqueue((i, v))
16             end for
17         else if node is a n – out – of – n gate then
18             child_node = the first child of node
19             l_v = v
20             for  $i = 1$  to  $n - 1$  do
21                 set  $l\_v[\text{count}+i] = 1$ 
22             end for
23             queue.enqueue((child_node, l_v))
```



```

24     for  $i = 2$  to  $n$  do
25         child_node = the  $i_{\text{th}}$  child of node
26          $r_v = (0, 0, 0, \dots) \in F^c$ 
27         set  $r_v[\text{count}+i-1] = -1$ 
28         queue.enqueue((child_node,  $r_v$ ))
29     end for
30     count = count + ( $n - 1$ )
31 else if node is a  $t - \text{out} - \text{of} - n$  gate then
32     for  $i = 1$  to  $n$  do
33         child_node = the  $i_{\text{th}}$  child of node
34          $l_v = v$ 
35         for  $j = 1$  to  $t - 1$  do
36             set  $l_v[\text{count}+i] = j$ 
37         end
38         queue.enqueue((child_node,  $l_v$ ))
39     end
40 else if node is a leaf node then
41      $M_i =$  find a row whose entries are all zeros from  $M$ 
42     set  $M_i = v$ 
43     set  $\pi(i) = \text{party}(\text{node})$ 
44 end if
45 end while
46 return  $\mathcal{M}$ 

```

Table 5.5: Pseudo code of our algorithm

5.3 Applications

Many applications can be designed based on ABE. Here we first briefly show some of them, and then give a design of patient-controlled EMR system.

We can apply the KP-ABE scheme to the pay-tv system [16]. A broadcaster broadcasts some ciphertexts, each one labeled with a set of attributes. Each user has a private key associated with an access formula. For instance, a television broadcaster labels its programs with the name of the program, the date of the program and the season of the program. And each subscriber has a private key with an access formula and she can use the key to decrypt the programs she subscribed to.

The CP-ABE scheme with multiple authorities is often applied to an online social network [19]. Suppose that Alice has some photos she wants to share with her friends. She can encrypt the photos under an access policy: “Alice” AND “Alice’s friends.” Her friends can get their private keys from Alice and then decrypt the encrypted photos. Our implementation can be backbone to this application.

Recently, some electronic medical record (EMR) systems have been designed using the ABE [1, 18]. We will review them and design an EMR system in the next section.

5.3.1 EMR systems using ABE

In Taiwan, the Department of Health has promoted electronic medical record (EMR) in recent years. The government has budgeted NT\$ 60.4 hundred millions from 2010 to 2012 due to its advantages over the traditional paper-based records. Moving to EHRs is an important national goal.

The cost of sharing paper-based records is expensive. If a patient wants to

transfer to another health provider or she wants a copy of her own health record for health insurance purpose. She is likely to go to her health provider in person and runs the procedure of applying a patient's own health record. The procedure consumes a lot of manpower and material resource. Imagine that if the EMR is used, the cost due to paper-based record can be eliminated. The Taiwan Government estimated that cost of healthcare can be reduced by 10%. Therefore, the health provider reduces its total cost and it can promote healthcare quality. On the patient's side, the convenience of transfer to another healthcare provider increases the chance of receiving better treatment.

For researchers, more patients' information can be recorded, analyzed, and shared by digitalized health records. Researchers can mine EHRs for the information they are interested. This can speed up the application of new medical research. Therefore, all human beings can benefit from sharing their own health records.

However, people care about privacy. If a patient's personal health record contains sensitive information, e.g., AIDS, this information could result in employment discrimination. Recently in Taiwan, the personal health record of candidate was exposed as blackmail in elections. That paper-based record was revealed by medical staff. As a result, people want more security control when they digitalize their records and upload all data to the medical cloud.

Consequently, patient-controlled EMR systems have been designed in recent years [29]. In a patient-controlled EMR system, the patient decides who can access his medical data. Most approaches employ techniques based on smartcards ("chip" and "pin"). The record is produced or edited by some health professionals and is encrypted by the patients' secret key which is in the smartcard. In addition, the encrypted records are stored in a medical cloud. When a health professional wants to read a patient's medical data, the patient use his smartcard to decrypt the encrypted record in the cloud. The approach is

described in Figure 5.5.

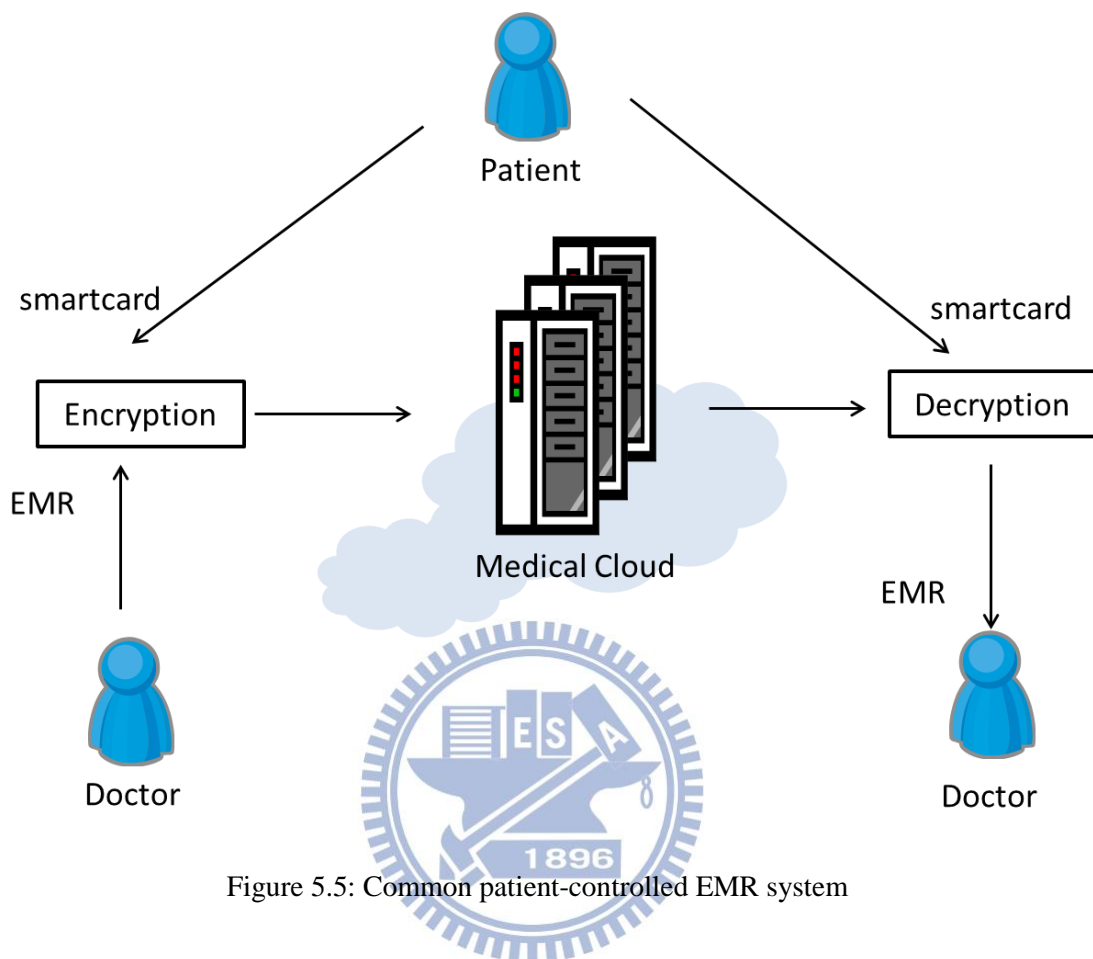


Figure 5.5: Common patient-controlled EMR system

This approach ensures strong privacy. However it has two drawbacks. First, the access control is coarse-grained. The patient needs to decrypt many times for anyone is authorized to read the record. Secondly, in emergency cases, the record needs to be decrypted as soon as possible. We need some mechanism to break the glass in emergency cases.

Akinyele et al. proposed a construction using the CP-ABE [1]. They implement the scheme described in [22] and [34]. However their construction is not patient-controlled. The records are encrypted by the hospital rather than by the patients. In addition, all patients' private keys are issued by one authority. If the authority is compromised, all records are exposed to danger.

Hupperich et al. construct a flexible patient-controlled EMR system using the

CP-ABE [18]. In their construction, the medical record is encrypted under the patient's ID (P_{ID}) and a transaction code TAC_1 . The access formula is " P_{ID} " OR " TAC_1 ." If a doctor wants to read the patient's record, the patient give the doctor a private key for attribute TAC_1 . Therefore, the doctor can use this key to decrypt the patient's record. Suppose that the doctor edit something in the record and she wants to submit the changes. The patient will ask the TAC server to generate another transaction code TAC_2 and the TAC server will give a public key for TAC_2 and a private key for TAC_2 to the patient. The patient now sends the public key to the doctor. And the doctor can encrypt the record under the access formula " P_{ID} " OR " TAC_2 ." Figure 5.6 summarizes their approach.

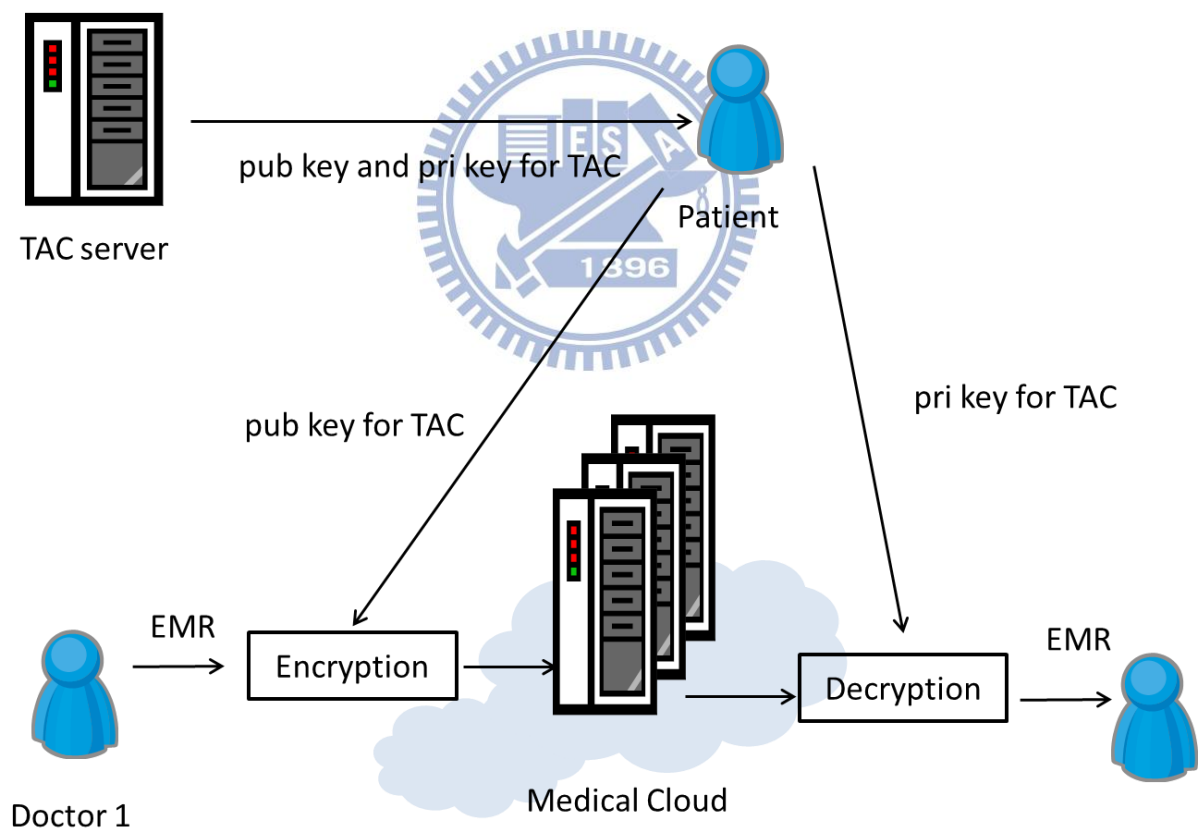


Figure 5.6: Flexible patient-controlled EMR system

However, in their construction, the TAC server is the single authority. Therefore all patients must trust the TAC server. Moreover, records are

encrypted by medical professionals rather than by patients. It may be dangerous, because we cannot make sure what access formula the professional choose. For instance, the professional can replace TAC with an attribute of her own. The access formula now provides a trapdoor that allows the professional decrypt the record.

Here we design a patient-controlled EMR system based on our implementation. There are many authorities in the systems such as the hospitals, the research institutes, and the patients themselves. The authorities will publish their public keys associated with the attributes they control. The patient can then encrypt his record under the access formula he chooses. The doctor gets her private keys from the authorities. After the doctor finishes her editing, she sends back the record encrypted under the patient's RSA public key. The patient then decrypts it and encrypts the record again under an access formula. We note that the patient is the only one allowed to upload his record to the medical cloud. And he can change his access formula by encrypting his record under a different access formula and re-upload it to the cloud.

Our system is patient-controlled because the patient can encrypt his record under the access formula he chooses. And the patient can generate the public keys and private keys himself. If he does not trust any other authorities in the system, he can encrypt his record only under the attribute he controls and give the corresponding private keys to the medical professional.

Another application is that the patient can generate attributes for his medical agent. By giving the private key of the attributes to his medical agent, his medical agent can now have the ability to read his records.

Our access control is fine-grained naturally. And we can handle the emergency case by adding an attribute named "emergency." Every record is encrypted under the access formula: ...OR "emergency." Then the medical professionals can decrypt the recodes in the emergency case. The attribute

“emergency” is controlled by a trusted party and the use of the private key of the attribute “emergency” must be strictly recorded to avoid abuse. Figure 5.7 summarizes our approach

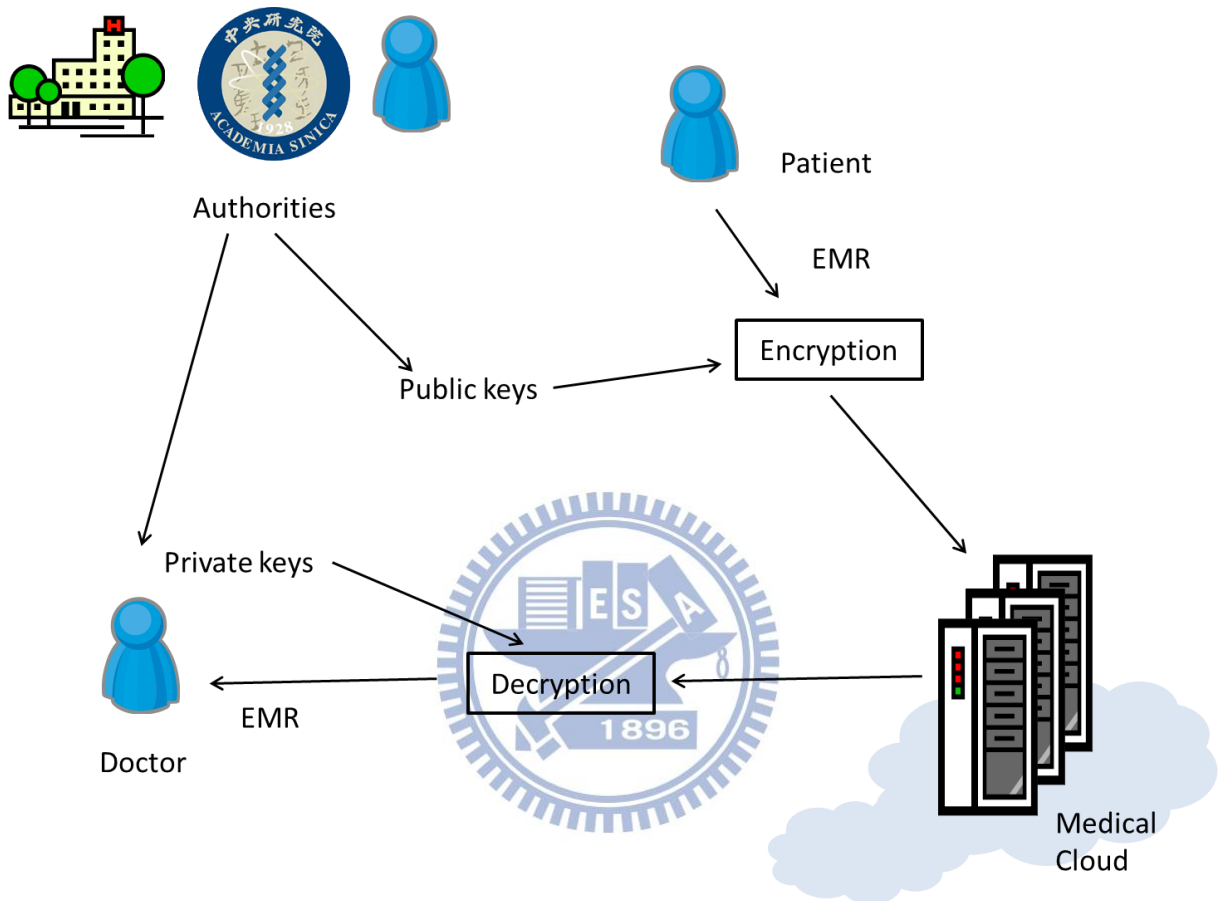


Figure 5.7: Our patient-controlled EMR system

Chapter 6

Conclusion and Future Work

In this thesis, we propose an algorithm that converts a monotone boolean formula to a monotone span program. The monotone span program is used in the secret sharing scheme and the ABE scheme. The algorithm can handle the threshold gate which is unable to handle in the previous algorithm. Therefore we expand the expressiveness of the access formula.

A scheme of CP-ABE with multiple authorities is implemented and some applications can be built upon our implementation are discussed in this thesis. According to the result of the experiment, the implemented scheme is practical run in an Intel-based computer. Our implementation can be backbone to those applications based on the CP-ABE scheme.

There are some issues we can improve in the ABE scheme. An existing issue of the ABE scheme is that we cannot verify that the ciphertext is really encrypted under the access formula it claims. It becomes a problem when we want others to encrypt for us. If this problem can be overcome, more applications can be designed based on the ABE. Another issue is that sometimes we want a non-monotonic access structure in the CP-ABE scheme. [22] is a KP-ABE with non-monotonic access structure. But researchers are still looking for a practical solution to CP-ABE scheme with non-monotonic access structure.

In the future, we would like to port our programs to smartphone platforms. Also we will research on the issues described in this Section.

Bibliography

[1] J. A. Akinyele, M. W. Pagano, M. D. Green, C. U. Lehmann, Z. N. J. Peterson, and A. D. Rubin, “Securing electronic medical records using attribute-based encryption on mobile devices,” in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 75–86.

[2] J. A. Akinyele, M. D. Green, and A. D. Rubin, “Charm: a framework for rapidly prototyping cryptosystems,” <http://eprint.iacr.org/2011/617>, 2011.

[3] N. Attrapadung and H. Imai, “Dual-policy attribute based encryption,” in *Applied Cryptography and Network Security*, 2009, pp. 168–185.

[4] P. Barreto and M. Naehrig, “Pairing-friendly elliptic curves of prime order,” in *Selected Areas in Cryptography*, vol. 3897, B. Preneel and S. Tavares, Eds. Springer Berlin / Heidelberg, 2006, pp. 319–331.

[5] A. Beimel, “Secret-sharing schemes: a survey,” *Coding and Cryptology*, pp. 11–46, 2011.

[6] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, 2007, pp. 321–334.

[7] I. F. Blake, G. Seroussi, and N. P. Smart, *Advances in elliptic curve cryptography*. Cambridge Univ Pr, 2005.

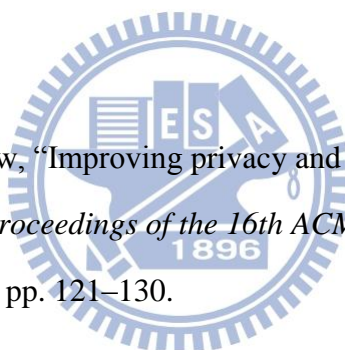
[8] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in *Advances in Cryptology-Eurocrypt 2004*, 2004, pp. 506–522.

[9] D. Boneh, C. Gentry, and B. Waters, “Collusion resistant broadcast encryption with short ciphertexts and private keys,” in *Advances in Cryptology-CRYPTO 2005*, 2005, pp. 258–275.

[10] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” *Theory of Cryptography*, pp. 535–554, 2007.

[11] M. Chase, “Multi-authority attribute based encryption,” *Theory of Cryptography*, pp. 515–534, 2007.

[12] M. Chase and S. S. M. Chow, “Improving privacy and security in multi-authority attribute-based encryption,” in *Proceedings of the 16th ACM conference on Computer and Communications Security*, 2009, pp. 121–130.



[13] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Advances in Cryptology*, 1985, pp. 10–18.

[14] D. Freeman, “Constructing pairing-friendly elliptic curves with embedding degree 10,” *Algorithmic Number Theory*, pp. 452–465, 2006.

[15] V. Goyal, A. Jain, O. Pandey, and A. Sahai, “Bounded ciphertext policy attribute based encryption,” *Automata, Languages and Programming*, pp. 579–591, 2008.

- [16] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proceedings of the 13th ACM conference on Computer and Communications Security*, 2006, pp. 89–98.
- [17] J. Hoffstein, J. Pipher, and J. Silverman, “NTRU: a ring-based public key cryptosystem,” *Algorithmic Number Theory*, pp. 267–288, 1998.
- [18] T. Hupperich, H. Löhr, A. R. Sadeghi, and M. Winandy, “Flexible patient-controlled security for electronic health records,” in *Proceedings of the 2nd ACM SIGHIT Symposium on International Health Informatics*, 2012, pp. 727–732.
- [19] S. Jahid, P. Mittal, and N. Borisov, “EASIER: encryption-based access control in social networks with efficient revocation,” in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011, pp. 411–415.
- [20] M. Karchmer and A. Wigderson, “On span programs,” in *Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual*, 1993, pp. 102–111.
- [21] N. Kobitz and A. Menezes, “Pairing-based cryptography at high security levels,” *Cryptography and Coding*, pp. 13–36, 2005.
- [22] A. Lewko, A. Sahai, and B. Waters, “Revocation systems with very small private keys,” in *Security and Privacy (SP), 2010 IEEE Symposium on*, 2010, pp. 273–285.
- [23] A. Lewko and B. Waters, “Decentralizing attribute-based encryption,” *Advances in Cryptology–EUROCRYPT 2011*, pp. 568–588, 2011.

- [24] A. Lewko and B. Waters, “Decentralizing attribute-based encryption,” <http://eprint.iacr.org/2010/351>, 2010.
- [25] B. Lynn, *The Pairing-Based Cryptography Library*. <http://crypto.stanford.edu/abc/>.
- [26] V. Miller, “Short programs for functions on curves,” unpublished manuscript.
- [27] A. Miyaji, M. Nakabayashi, and S. Takano, “New explicit conditions of elliptic curve traces for FR-reduction,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2001.
- [28] R. Ostrovsky, A. Sahai, and B. Waters, “Attribute-based encryption with non-monotonic access structures,” in *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007, pp. 195–203.
- [29] H. H. Rau, C. Y. Hsu, Y. L. Lee, W. Chen, and W. S. Jian, “Developing electronic health records in Taiwan,” *IT professional*, vol. 12, no. 2, pp. 17–25, 2010.
- [30] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [31] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” *Advances in Cryptology—EUROCRYPT 2005*, pp. 557–557, 2005.
- [32] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

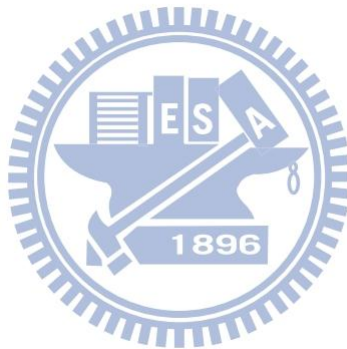
[33] L. C. Washington, *Elliptic curves: number theory and cryptography*. Chapman & Hall, 2008.

[34] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” *Public Key Cryptography–PKC 2011*, pp. 53–70, 2011.

[35] B. Waters, “Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization,” <http://eprint.iacr.org/2008/290>, 2008.

[36] “NIST Recommended Key Sizes.”

http://www.nsa.gov/business/programs/elliptic_curve.shtml.



Appendix A: Source Code

This Appendix includes the core C code for the implementation in this thesis. A.1 is the code of the access tree, A.2 is the code of the linear secret sharing scheme and A.3 is the code of the CP-ABE.

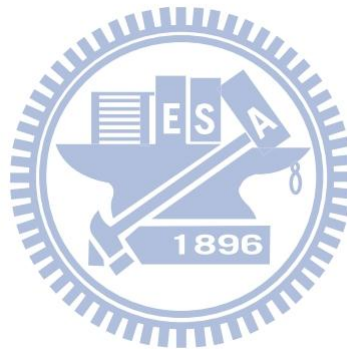
A.1 attr_tree.h

```
#ifndef _ATTR_TREE_H
#define _ATTR_TREE_H

typedef enum _ATTR_NODE_TYPE {
    ATTR_NODE_NULL = 0,
    ATTR_NODE_LEAF,
    ATTR_NODE_AND,
    ATTR_NODE_OR,
    ATTR_NODE_THRESHOLD
} ATTR_NODE_TYPE;

typedef struct _attr_tree_node {
    char * attribute;
    ATTR_NODE_TYPE node_type;
    unsigned int num_subnodes;
    unsigned int threshold_k;
    struct _attr_tree_node ** subnode;
} attr_tree_node;

typedef struct _attr_tree {
```



```

    attr_tree_node * root;

    char * string;
} attr_tree;

typedef struct
{
    int num_components;

    attr_tree_node** components;
}attr_array;

/* Create a policy leaf subnode from an attribute string.

* @param attribute_str      Attribute string.

* @return                   Allocated policy subnode.

*/

attr_tree_node* create_leaf(char *attr_str);

/* Create a policy subnode from an array of subnodes.

* @param node_type         ATTR_NODE_TYPE value.

* @param num_subnodes      Number of subnodes in the array.

* @param threshold_k       Threshold value k.

* @param subnodes          Array of attr_tree_node pointers

* @return                   Allocated policy subnode.

*/

attr_tree_node* create_node(ATTR_NODE_TYPE node_type, int num_subnodes, int threshold_k,
attr_tree_node** subnodes);

/* Recursive the tree in preorder, for each node, run func(node,param)

* @param node              the root of the subtree

* @param func              callback function

* @param param             param for callback function

*/

int traverse_tree_preorder(attr_tree_node* node, int(*func)(attr_tree_node*, void*), void* param);

/* test_satisfy_policy

* @param1: array of attr string

```

```

* @param2: size of the array
* @param3: root the access tree
* @return: 1 if satisfy; 0 if not
*/
int test_satisfy_policy(char** attrs, int num_sk, attr_tree_node* final_policy);
#endif

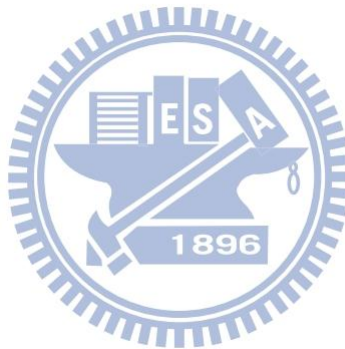
```

A.2 LSSS.h and LSSS.c

```

#ifndef _LSSS_H_
#define _LSSS_H_
#include <stdlib.h>
#include <string.h>
#include <psc/psc.h>
#include "../policy_parse/attr_tree.h"
typedef struct _MSP_{
    int** matrix;
    char** label;
    int rows;
    int cols;
}MSP;
void MSP_init(MSP* msp, int rows, int cols);
void MSP_clear(MSP* msp);
/* use LW's algorithm to make an MSP.
* @param1: msp
* @param2: tree's root
*/
int makeMSP_v2(MSP* msp, attr_tree_node* node);
/* use Our algorithm to make an MSP

```



```

* @param1: msp
* @param2: tree's root
*/

int makeMSP_v3(MSP* msp, attr_tree_node* node);

/* Computes Shares
* @param1: msp
* @param2: input vector[s, r1, r2, ...]
* @param3: output vector
*/

void computeShares(MSP* msp, element_t* input_array, element_t* output_array);

/* use Gauss-Jordan to reduce msp's matrix
* @return: 1 if success 0 if failed
*/

int reduceMSP(MSP* msp);
void swap_vector(int** matrix, int i, int k);
void swap_label(char** label, int i, int k);

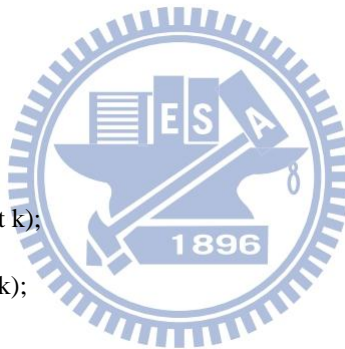
/* divide the row by n
* @param1: matrix
* @param2: index of row
* @param3: number of columns
* @param4: n
*/

void divide_row_by_n(int** matrix, int i_row, int cols, int n);
void eliminate(int** matrix, int i_row, int i_col, int rows, int cols, int **new_matrix);

#endif

int makeMSP_v3(MSP* msp, attr_tree_node* node){
    int** matrix = msp -> matrix;
    char** label = msp -> label;

```



```

int cols = msp -> cols;

int rows = msp -> rows;

attr_tree_node* queue[100] = {0};

int* vectors [100] = {0};

int write_i = -1;

int read_i = -1;

int* v = calloc(cols, sizeof(int));

v[0] = 1;

queue[++write_i] = node;

vectors[write_i] = v;

int count = 1;

int index = 0;

while( read_i < write_i){

    attr_tree_node* node = queue[++read_i];

    int* v = vectors[read_i];

    if( node -> node_type == ATTR_NODE_OR ){

        int i = 0;

        for( i; i < 2; i++){

            queue[++write_i] = (node -> subnode)[i];

            vectors[write_i] = v;

        }

    }

    else if( node -> node_type == ATTR_NODE_AND ){

        int* l_v = calloc(cols, sizeof(int));

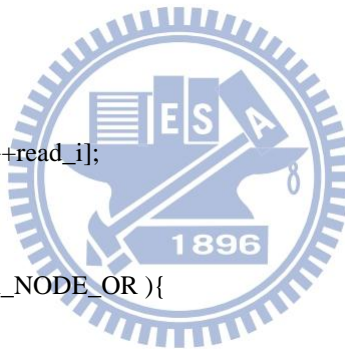
        int* r_v = calloc(cols, sizeof(int));

        memcpy(l_v, v, cols*sizeof(int));

        l_v[count] = 1;

        {

```



```

int i = 0;

for(i; i < cols; i++)

    r_v[i] = v[i] - l_v[i];

}

count++;

queue[++write_i] = (node -> subnode)[0];

vectors[write_i] = l_v;

queue[++write_i] = (node -> subnode)[1];

vectors[write_i] = r_v;

}

else if( node -> node_type == ATTR_NODE_THRESHOLD ){

int i = 0;

for( i; i < node->num_subnodes; i++){

int* l_v = calloc(cols, sizeof(int));

memcpy(l_v, v, cols*sizeof(int));

int j = 0;

for( j; j < node->threshold_k - 1; j++){

l_v[count+j] = power(i+1, j+1);

//printf("power:%d\n", power(i+1, j+1));

}

queue[++write_i] = (node -> subnode)[i];

vectors[write_i] = l_v;

//free(l_v);

}

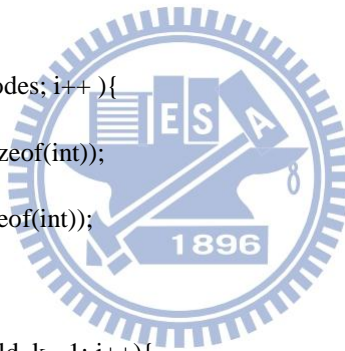
count = count + (node->threshold_k) - 1;

}

else if( node -> node_type == ATTR_NODE_LEAF){

memcpy( matrix[index], v, cols*sizeof(int) );

```



```

        label[index] = node->attribute;

        index++;

    }

}

}

int reduceMSP(MSP* msp){

    int** matrix = msp->matrix;

    char** label = msp->label;

    int cols = msp->cols;

    int rows = msp->rows;

    int** new_matrix = calloc(rows, sizeof(int*));

    {

        int i = 0;

        for(i; i < rows; i++){

            new_matrix[i] = calloc(rows, sizeof(int));

            new_matrix[i][i] = 1;

        }

    }

    // Gauss-Jordan

    int i_row = 0;

    int i_col = 0;

    while( i_row < rows && i_col < cols ){

        //look for a non-zero entry in col i_col at or below row i_row

        int k = i_row;

        while( k < rows && matrix[k][i_col] == 0 ) k++;

        if( k <= rows ){

            if( k != i_row ){

                //swap

```



```

        swap_vector(matrix, i_row, k);

        swap_vector(new_matrix, i_row, k);

        swap_label(label, i_row, k);
    }

    if( matrix[i_row][i_col] != 1){

        //divide

        int div = matrix[i_row][i_col] / 1;

        divide_row_by_n( matrix, i_row, cols,  div);

        divide_row_by_n( new_matrix, i_row, rows, div);

    }

    //eliminate

    eliminate(matrix, i_row, i_col, rows, cols, new_matrix);

    i_row++;

}

i_col++;

}

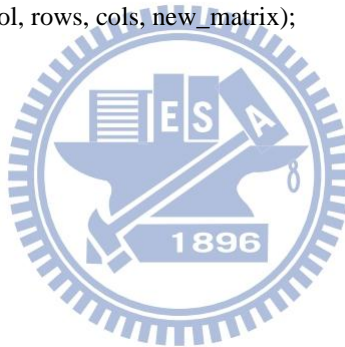
msp->matrix = new_matrix;

msp -> cols = msp -> rows;

return 1;

}

```



A.3 CPABE.h

```

#ifndef _LW11_H_

#define _LW11_H_

#define TYPE_A_GROUP_ORDER_LENGTH 512

#define TYPE_A_BASE_FIELD_LENGTH 1536

```



```

#define PUBLIC_DATABASE "abe.db"

#include <pbk/pbc.h>

#include <openssl/sha.h>

#include <sqlite3.h>

#include <stdio.h>

#include <string.h>

#include "pairingio.h"

#include "LSSS.h"

typedef struct _AUTHORITY_SK_{

    char attribute[256];

    element_t alpha;

    element_t y;

}authority_sk;

typedef struct _CIPHER_INFO_{

    element_t C1;

    element_t C2;

    element_t C3;

    int owner_id;

    int pk_id;

    char matrix_row[128];

}cipher_info;

typedef struct _PK_{

    char PK1[4096];

    char PK2[4096];

    int owner_id;

    int pk_id;

}PK;

typedef struct _DECRYPTION_KEY_ {

```



```

char* key_text;

char* attribute;

int gid;

}DK;

/* return a malloc DK*/

DK* new_DK(char* key_text, char* attr, int gid);

/* free a DK*/

void delete_DK(DK* _dk);

/* read all authority's secret keys from file into an array
* return: an array of authority_sk*, the array is ended with NULL. If failed, return NULL.
*/
authority_sk** get_authority_keys_from_file(pairing_t pairing, const char* filename);

/* Produce system parameters: the pairing parameter file and the g parameter file
* param1: the file where pairing parameter will be written
* param2: the file where g parameter will be written
* return: if success return 0. else return 1
*/

int global_setup_LW11(const char* pairing_param_filename, const char* g_param_filename);

/* Produce public keys and secret keys. The public keys will be stored into sqlite3 database.
* The private keys will be stored into a file(param4.secretkey)
* param1: the pairing
* param2: the generator g
* param3: owner's name
* param4: the file contains the owners' all controlled attributes
* return: if success return 0. else return 1
*/

int authority_setup_LW11(pairing_t pairing, element_t g, const char * owner_name, const char * attributes_file);

```

```
/* Connect the sqlite database
```

```
* param1: sqlite3 database file name
```

```
* param2: the address of sqlite3 handle
```

```
* param3: error function, when connection failed, this function will be called
```

```
* return: 0 if succeed, else return whatever error_func returns
```

```
*/
```

```
int connect_database(const char* databasename, sqlite3** handle, int (*error_func)(const char* ));
```

```
/* Check if value exists in table's column
```

```
* return: the id of the row if exist, -1 if not
```

```
*/
```

```
sqlite3_int64 is_value_in_column_of_table(sqlite3* handle, const char* table_name, const char* column_name, const char* value);
```

```
/* Check insert a column's value in table
```

```
* return: 0 if success, 1 if failed
```

```
*/
```

```
int insert_value_in_column_of_table(sqlite3* handle, const char* table_name, const char* column_name, const char* value);
```

```
/* Produce a decryption key of owner:attribute to user(gid)
```

```
* param1: (output)the produced decryption key
```

```
* param2: generator g
```

```
* param2: pairing
```

```
* param3: array generated by `get_authority_keys_from_file`
```

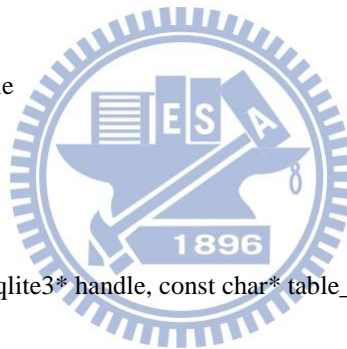
```
* param4: attribute
```

```
* param5; user's gid
```

```
* return: 0 if success, 1 if failed
```

```
*/
```

```
int keygen_LW11(element_t key, pairing_t pairing, element_t g, authority_sk** array_of_asks, const char * attribute, int gid);
```



```

/* Encrypt file under aes128 and M is an element of GT, will be used as a key of AES-128

* @param1: an element of GT

* @param2: filename

* @param3: cipher id in database

*/

int LW11_encrypt_file(element_t M, char* file, int cipher_id);

/* Compute  $Me(g,g)^s$ , stored it into first parameter

* @param1: output

* @param2: M(session key)

* @param3: e(g,g)

* @param4: s

*/

int LW11_encrypt_sessionkey(element_t Megg_s, element_t M, element_t egg, element_t s);

/* insert policy and encrypted session key into database

* @param1: policy string

* @param2: encrypted session key (type is GT)

* @return: the insert row id, if failed -1

*/

int insert_cipher_into_database(char* policy_input, element_t Megg_s);

/* get a cipher_info

* @param1: formate owner:attribute

* @return: NULL if failed

*/

cipher_info* get_cipherinfo(char* raw_attr, int* matrix_row, int cols, element_t omega, element_t lambda,
element_t r, element_t egg, element_t g);

/* store the cipher_info into the database

* @param1: cipherinfo

* @parma2: cipher_id

```

```

* @return 0 if failed

*/

int store_cipher_info(cipher_info* cipherinfo, int cipher_id);

/* get cipher infos from public db, return a array of cipher_info

* @param1: id of the cipher

* @param2: output, size of array

* @return: a pointer or NULL

*/

cipher_info** get_cipherinfo_from_db(int cipher_id, int* size_of_array, pairing_t pairing);

/* print ci's member*/

void print_cipher_info(cipher_info* ci);

/* get_attr_string_array

* @param1: array of dk

* @param2: size of the array

* @return: the array or null

*/

char** get_attr_string_array(DK** dks, int num_sk); /* get an msp by array of cipher_info

* @return: an MSP or null

*/

MSP* getMSP_by_cipher_info(cipher_info** cis, int size_of_array, char** attrs, int size_of_attrs);

/* count a character appears in a string

* @param1: the string

* @param2: the char

* @return: the count

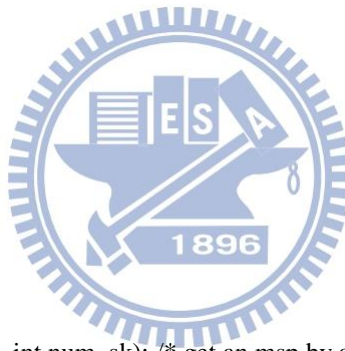
*/

int count_chars(const char*, char);

/* find the attribute's pk id from database

* @param1: string and the form is owner:attribute

```



```

* @return: the pk id or -1
*/

int find_pk_id(char* str);

/* find a DK* from array of DK

* @param1: array of DK

* @param2: array size

* @param3: attribute string

* @return: NULL pointer or DK*

*/

DK* find_DK_by_attribute(DK** dks, int size, char* str);

cipher_info* find_cipher_info_by_attribute(cipher_info** cis, int size_of_array, char* attribute);

/* get encrypted sk from database by cipher id

* @param1: element_t, must be GT

* @param2: the cipher id

* @return: -1 if failed

*/

int get_encrypted_sk(element_t M, int cipher_id);

#endif

```

