12-2013

Group Selection and Key Management Strategies for Ciphertext-Policy Attribute-Based Encryption

Russell F. Martin

Group Selection and Key Management Strategies for Ciphertext-Policy Attribute-Based Encryption

by

Russell F. Martin

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering

Supervised by

Dr. Marcin Lukowiak
Department of Computer Engineering
Kate Gleason College of Engineering

Dr. Stanisław Radziszowski
Department of Computer Science
Golisano College of Computing and Information Sciences

Rochester Institute of Technology
Rochester, New York
December 2013

Approved by:

Dr. Marcin Lukowiak
Thesis Advisor, Department of Computer Engineering

Dr. Stanisław Radziszowski
Thesis Advisor, Department of Computer Science

Dr. Dhireesha Kudithipudi
Committee Member, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Group Selection and Key Management Strategies for Ciphertext-Policy Attribute-Based Encryption

I, Russell F. Martin, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Russell F. Martin

Date

Dedication

To my family.

Acknowledgments

Thank you to everyone who has been in my life during my time at college, pushing me on when I really needed it, especially these past few weeks while writing this thesis. Anyone who has shared a meal with me, given me a place to sleep when traveling, worked on a team project with me, helped me enjoy my time here in Rochester.

My advisors were great to work with, and I need to thank them for letting me do my own research without great amounts of pressure.

Most notably, thank you to my parents, and their constant support of my collegiate choices. I don't know where I would be without them.

And finally, I thank you, the reader interested in my work.

Abstract

Group Selection and Key Management Strategies for Ciphertext-Policy Attribute-Based Encryption

Russell F. Martin

Supervising Professors: Dr. Marcin Lukowiak and Dr. Stanisław Radziszowski

Ciphertext-Policy Attribute-Based Encryption (CPABE) was introduced by Bethencourt, Sahai, and Waters, as an improvement of Identity Based Encryption, allowing fine grained control of access to encrypted files by restricting access to only users whose attributes match that of the monotonic access tree of the encrypted file. Through these modifications, encrypted files can be placed securely on an unsecure server, without fear of malicious users being able to access the files, while allowing each user to have a unique key, reducing the vulnerabilites associated with sharing a key between multiple users.

However, due to the fact that CPABE was designed for the purpose of not using trusted servers, key management strategies such as efficient renewal and immediate key revocation are inherently prevented. In turn, this reduces security of the entire scheme, as a user could maliciously keep a key after having an attribute changed or revoked, using the old key to decrypt files that they should not have access to with their new key. Additionally, the original CPABE implementation provided does not discuss the selection of the underlying bilinear pairing which is used as the cryptographic primitive for the scheme.

This thesis explores different possibilites for improvement to CPABE, in both the choice of bilinear group used, as well as support for key management that does not rely on proxy servers while minimizing the communication overhead. Through this work, it was found that nonsupersingular elliptic curves can be used for CPABE, and Barreto-Naehrig curves allowed the fastest encryption and key generation in CHARM, but were the slowest curves for decryption due to the large size of the output group. Key management was performed by using a key-insulation method, which provided helper keys which allow keys to be transformed over different time periods, with revocation and renewal through key update. Unfortunately, this does not allow immediate revocation, and revoked keys are still valid until the end of the time period during which they are revoked. Discussion of other key management methods is presented to show that immediate key revocation is difficult without using trusted servers to control access.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

While current private key cryptosystems, such as AES, are believed to be mathematically secure, they are only as secure as their keys are. Any user with the encryption key is able to decrypt and subsequently access the data. This proves difficult when needing to authorize a large group of individuals to decrypt the data, as any of the users has a chance to maliciously or accidentally distribute the key to unauthorized individuals, allowing them access to decrypted files they shouldn't normally have. One option in cryptography that allows each user to have a unique key is Ciphertext-Policy Attribute-Based Encryption (CPABE), introduced by Bethencourt, Sahai, and Waters in 2007 [4]. In such a scheme, users have a private key associated only to their identity, with control to various ciphertexts given by a list of attributes they possess. CPABE was not the first cryptosystem satisfying this criteria, with its roots based in Identity Based Encryption (IBE), originally proposed by Shamir in 1984 [31].

Shamir's original IBE scheme was designed to allow encryption and signature verification between two users without any exchange of keys, instead using a trusted server to generate unique keys for each user. Instead of directly having users handle keys, personal information unique to a user, such as address, social security number, name, or a combination of such, were used as the public key. When joining the network, the user would be issued a personal smartcard that contains their private key. With this card, the user can encrypt, decrypt, or sign messages to any other user. To prevent unauthorized access or duplication of a card, the users would be required to enter a personal password to access the private key. The main advantage to this system over other public key infrastructure (PKI) implementations is that subsequent communications did not require any access to such infrastructure. This allows users to be added to the network without having to update any keys or infrastructure such as a user list, nor require it of the individual users. Additionally, there was no restriction that the users must be online or connected to the network in order to verify some of these properties when other users needed them. Shamir designed this scheme to be used for large companies such as banks, and the scheme was very expandable, supporting multiple key distribution centers and millions of users.

This IBE scheme used modular exponentiation to validate signatures and generate private keys for the users. A signature (s, t) for the message m, signed by the user i was verified by the equation

$$s^e = i \cdot t^{f(t,m)}$$

where n is the product of two large primes, similar to RSA, e is another large prime that is

relatively prime to $\phi(n)$, and f is a secure one-way function. The security in this scheme is associated with the factorization of n, as users can know n, but if they know the factorization then they can use it to generate other valid keys. The value i from the above equation is a integer representation of the user's identity, which is equal to $g^e \mod n$, with g being the user's secret key. Shamir did not discuss a structure for encryption and decryption within the scope of that paper, leaving it an open problem.

Sahai and Waters improved on IBE [29], by transforming a user's identity from a single string into a set of attributes, as well as adding an error tolerance, allowing a user to decrypt an encrypted file if their attributes were within a small difference. These errors could not be accounted for in the original IBE scheme, as identities were restricted to an exact string of characters. However, by allowing an error range for attributes, not only were biometric identities like an eye scan allowed, but attribute based encryption (ABE) was created. One of the most important changes made in this scheme is the fact that each attribute was directly linked to the rest of the user's attributes, instead of simply having an independent key for every attribute. With independent keys per attribute, multiple users could collude and use their keys to together decrypt a file that no single user could decrypt on their own. When using fuzzy metrics such as biometrics, this scheme would be used by mapping unique features from the image to attributes and matching based on those attributes.

In order to decrypt files within this ABE scheme, a file would be encrypted with an access policy associated with the identity of the user encrypting the file, and a threshold value d. Users wishing to decrypt the file could only do so if their own private keys contained at least d of the same attributes as that of the access policy. This scheme used bilinear maps to match values of the ciphertext and secret key, as well as polynomial interpolation between the attributes, much of the same way that CPABE does.

One of the largest restrictions to this scheme is the fact that the access policies are restricted only to a threshold of attributes, or a"d out of k" scheme. This means that no other control can be done outside of listing attributes, making it difficult to have fine control of the access to the decrypted file, with the attributes given to users controlled by the key generation center and not the users encrypting the files.

Goyal *et al.* [15] provided a more expressive modification to ABE, called Key-Policy Attribute-Based Encryption (KPABE) which modified the original ABE scheme to allow the user's private key to contain the access structure of multiple levels of properties, while the encrypted texts contain a set of attributes or flags that are used to check if the key's access tree is satisfied. For example, if Alice's key in KPABE contains "X AND Y" as the access policy, the only files she can decrypt are those that have both attributes X and Y. A ciphertext with only attribute X could not be decrypted by Alice, as the access tree would not be satisfied. Multiple levels of access control are supported, with any monotonic access tree able to represent the attributes of the files a user is able to decrypt. An example of this access tree is shown in Figure 1.1.

Similar to that of the original ABE scheme, the keys given to users are also collusion resistant, meaning that two users could not use an overlap of attributes within their keys in order to decrypt files neither would be able to decrypt on their own. This is done through levels of secret sharing over the access tree, with values associated with each attribute in the

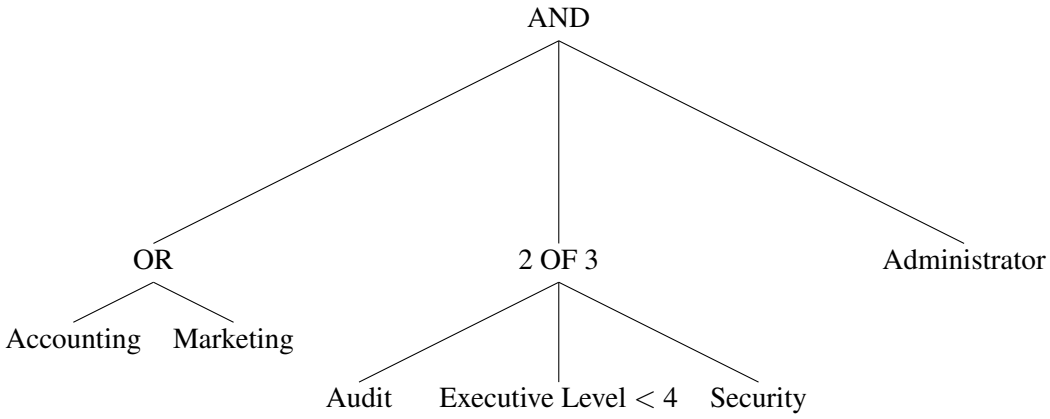

Figure 1.1: Example Access Tree for KPABE and CPABE

key, just like the ABE and CPABE designs. The other details of KPABE are not discussed in depth here, as they are very similar to that of CPABE, but with a reversal of the key and ciphertext values used for decryption.

The main issue with KPABE is the fact that the control of the access to the encrypted data is only controlled by the attributes given to the file, and not a controlled access tree [4]. A user encrypting a text in a KPABE scheme has no control over sets of attributes given to the users, and this makes fine control of access requiring multiple properties challenging. This is one of the reasons that led to the design of CPABE, as CPABE allows the desired attributes to be set by the user encrypting data with fine-grained control, while the key generation consists of issuing properties which cannot be directly connected to a specific ciphertext. An additional restriction of KPABE is the fact that all attributes must be made public, as they are each a component of the public key.

1.1 Thesis Organization

The first several chapters in this thesis provide the neccessary background for the work provided. Readers who are familiar with CPABE, bilinear groups, and elliptic curves may skip these chapters.

CPABE is introduced in Chapter 2, and the mathematical functions and derivations of all of the operations in the scheme are provided. An example is then provided to show an example implementation of CPABE. Additionally, a brief discussion of the limitations of CPABE is presented, to show motivation for the work of the thesis.

Chapter 3 introduces bilinear groups, the underlying crytographic primitive that is used for CPABE and other IBE cryptosystems. The properties of these are explained, and then the desired properties of these groups are shown, and why they are needed in the context of CPABE. This chapter will then cover the new security definitions that are made possible by defining this mapping.

Elliptic curves are then covered as the main topic of Chapter 4. In this chapter, a brief introduction to elliptic curves, point finding, and point counting over the curves is provided.

The rest of this chapter is used to show how a bilinear pairing can be performed over elliptic curves, and an example is provided.

Chapter 5 expands on the previous chapter with the various types and groups of elliptic curves that can be used to provide a bilinear pairing, mostly contained from the curve types supported by Lynn in the Pairing Based Cryptography Library [23]. Additional brief coverage is provided regarding pairings over non-elliptic curve fields, as well as supporting work for using different pairings for Attribute-Based Encryption. Following that, the first section of new work is presented, regarding the possibility of using asymmetric curves for CPABE, and the performance of such curve types compared to the original implementation.

The next chapter, Chapter 6 provides an introduction to key management techniques required by many cryptosystems. It is then shown how CPABE inherently prevents some of these techniques from being adequately performed. The chapter then discusses previous works, including why these techniques are or are not restrictive in the implementation of CPABE. The new work is then provided, with motivation and changes, as well as a short theoretical performance analysis.

Finally, this thesis closes out with Chapter 7, a summary chapter discussing both the work performed through the previous chapters, as well as future work and directions in order to improve and expand upon the work of this thesis.

1.2 Contributions

The original contributions of this thesis are as follows:

1. A detailed proof showing that asymmetric bilinear pairings are suitable for use in a Ciphertext- Policy Attribute-Based Encryption setting.

2. Performance comparison of five curve types of equivalent security provided by the Pairing Based Cryptography library [23] for CPABE functions.

3. Proposal of a Key Insulated CPABE scheme that supports non-immediate revocation and low overhead renewal through temporal keys.

Chapter 2

Ciphertext-Policy Attribute-Based Encryption

CPABE, first defined by Bethencourt, Sahai, and Waters [4], consists of five functions. These functions are Setup (Section 2.2), Key Generation (Section 2.3), Encryption (Section 2.4), Decryption (Section 2.5), and Delegation (Section 2.6). Access to encrypted files is controlled by any monotonic access tree, covered in Section 2.1. Later in this chapter, a detailed example of usage of CPABE is presented in Section 2.8, and is followed up by the inherent limitations of CPABE in Section 2.9.

2.1 Access Trees

The attributes required to decrypt a file within the CPABE scheme are described at the file level through a monotonic access tree τ. τ is represented by a series of nodes in the tree, each representing either an attribute or a boolean/threshold gate. The topmost node within the tree is the root node, denoted by R, and x in the remainder of this section represents some other node. For every non-leaf node in the tree, it has an associated threshold value, k_x, which can range from 1 to the total number of child nodes of that node, denoted by num_x. For a boolean AND gate, this can represent any number of inputs by setting $k_x = num_x$. For a boolean OR gate, this is simply $k_x = 1$. Additionally, the threshold is not restricted to only these two values, and can be any value in between, which allows such nodes to be satisfied when "2 out of 3" children are satisfied, for example, by setting the value $k_x = 2$ and having 3 total leaf nodes for that node.

With the model of the access tree defined, one can then subsequently define functions on the nodes. A unique integer indentifer is also given to each node in the tree, and this is obtained from the result of the function $\mathrm{index}(x)$. $\mathrm{Parent}(x)$ is defined for all nodes in the tree except R, and this function returns the index the parent of the node in the tree. Finally, leaf nodes also have $\mathrm{attr}(x)$, which returns the value of the attribute that is associated with the tree. The final function, performed only when analyzing the tree after being given a set of attributes to test for satisfaction, is denoted by τ_x, returning a value of 1 if the node is satisfied, and either 0 or a blocking value ($\perp$) if it is not satisfied. This evaluation is done recursively on the tree, starting with the root node. Let γ be the set of attributes the user attempting to decrypt the file posseses. For a leaf node x, $\tau_x = 1$ if and only if $\mathrm{attr}(x) \in \gamma$. For a non-leaf node, $\tau_x = 1$ if and only if k_x child nodes return 1.

The aformentioned structure works well for representing attributes that are of string type, but needs to be adapted slightly to work with numerical attributes. CPABE uses a

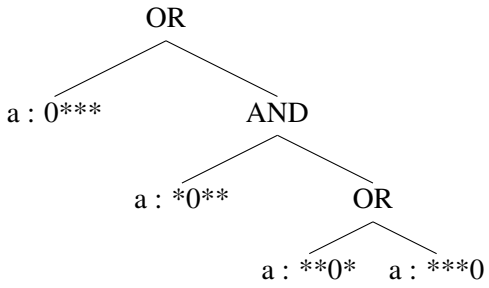

Figure 2.1: Bag of bits representation of $a < 11$

"bag of bits" structure to represent numerical attributes, represented by n child nodes over multiple levels, where n is the number of bits desired in the numerical representation of the attribute. Figure 2.1, originally shown in [4], shows the gate structure that would be used in the case of implementing "$a < 11$" in a four bit representation. In this tree, the attribute is split up and analyzed based on boolean operations on each of the bits. For example, the top level will be satisfied if the most significant bit is 0, as this means the value must be less than 8. If that is not satisfied, then it checks that the other bits are not set to the point where a would be greater than or equal to 11.

One point of note is that in the implementation of CPABE provided by [4], the way these numerical values are defined in these access trees prevent matching with access trees of the same attribute but over a different number of bits. In this case, the access tree shown would not match an access tree implementing "$a < 11$" over five bits.

Definition 2.1 *An access tree τ is monotonic when for all sets of attributes $\mathbb{S}$ that satisfy τ, any superset of $\mathbb{S}$, $\tilde{\mathbb{S}}$ will also satisfy τ.*

In CPABE, a goal is set of being able to realize any monotonic access tree into the access tree restricting access to an encrypted file. A limitation of CPABE is this inability to handle nonmonotonic access trees (See Section 2.9).

2.2 Setup

Prior to any key generation or encryption, CPABE must be initialized to the desired underlying bilinear group, which is used as part of the private key, PK. This is performed within the machine that generates the keys for all users, as this setup step also generates a master key, MK, which is used for key generation. With this master key, one is able to generate specific user keys. This key must remain secure or else malicious users can use it to generate any keys they desire.

The setup process begins with the selection of a bilinear group G_0 of prime order p, with generator g. Details of the choice of this bilinear group is covered in depth in Chapter 5. The next step is to randomly select two elements, α and $\beta \in \mathbb{Z}_p$. With these steps

completed, the setup is complete, and

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

where $e(g,g)$ is a bilinear pairing, $e : G_0 \times G_0 \rightarrow G_1$, and G_1 is a multiplicative cyclic group, also of order p (See Chapter 3) [4].

2.3 Key Generation

Key generation for a user requires two parameters, the master key of the CPABE scheme, as well as $\mathbb{S}$, the set of attributes that the user possesses. This algorithm begins with randomly choosing a value $r \in \mathbb{Z}_p$, as well as random values $r_j \in \mathbb{Z}_p$ for every attribute $j \in \mathbb{S}$. The value of r is chosen as a way of binding the value associated with the attributes to a value which is unique to the user who the key is valid for, preventing collusion between users. The output of this algorithm is a secret key associated with a user, SK,

$$SK = (D = g^{(\alpha+r)/\beta}, \forall j \in \mathbb{S} : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j})$$

where $H(j)$ is a secure hash function which maps any binary string representation of an attribute to an element of G_0.

2.4 Encryption

Encryption of a file or message, M, is performed with the public key generated during the setup step, as well as τ, a monotonic access tree which represents the attributes needed for decryption of the message. In the scope of this thesis, it is assumed that all messages are represented as an element of the output group of the bilinear map, G_1. Details of how a message can be translated to an element are not covered in detail, but Lynn does provide some discussion of hashing values to elements of the bilinear maps in [22].

The encryption begins with the generation of the polynomial q_x for every node x within τ. The degree of each of these polynomials, d_x, is set equal to the threshold value of the node, k_x, minus one. Starting with the polynomial of the root node R of τ, q_R, $q_R(0)$ is set equal to s, where s is a randomly chosen value of $\mathbb{Z}_p$. All other coefficients of the polynomial are set to random values from $\mathbb{Z}_p$. The values for all nodes are randomly generated per user to prevent collusion attacks, as a satisfying attribute from one key will not produce the same value for another user's polynomial. Note that the encrypting users are not required to have the desired attributes as part of their private key, meaning that it is possible that a user could encrypt a file that they would not be able to decrypt with their CPABE key. Additionally, because CPABE is a large universe scheme, with any arbitrary string being used to match an attribute, users may even use attributes that have never been used in the scheme before when specifying their access policy.

For any other node x in τ, the polynomial q_x is defined by $q_x(0) = q_{parent(x)}(index(x))$, while the other d_x coefficents are again set to d_x random values of $\mathbb{Z}_p$. Following completion of the polynomial generation, the encryption and performed and the ciphertext is

$$CT = (\tau, \tilde{C} = M \cdot e(g,g)^{\alpha s}, C = h^s,$$
$$\forall\, y \in Y : C_y = g^{q_y(0)}, C'_y = H(attr(y))^{q_y(0)}),$$

where Y is the set of all leaf nodes in τ.

In the above listing of the encryption, τ is listed as part of the ciphertext output. This is debatable and depends on the restrictions of the application. In some cases, by having this value public, users are able to discern some details of the plaintext, such as which group this file is being communicated within. There is no strict need that τ must be public, and it can be a hidden value if needed. All this means is that users will not know if they are able to decrypt the file until they attempt decryption.

2.5 Decryption

Decryption occurs when a user with personal key SK wishes to decrypt the ciphertext represented by CT. Decryption is a two step process, with the first decrypting the nodes to determine if the ciphertext's access tree τ is satisfied by the user's properties within SK, and the second performing the decryption of the message itself. For the first step, the DecryptNode function is called recursively, starting with the root node R of τ. Let F_z be the value returned by the recursive call to the child node z.

If z is a leaf node, let $i = attr(z)$. If $i \in \mathbb{S}$, then

$$\begin{aligned} DecryptNode(CT, SK, z) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} = \frac{e(g^r \cdot H(i)^{r_i}, g^{q_z(0)})}{e(g^{r_i}, H(i)^{q_z(0)})} \\ &= \frac{e(g^r, g^{q_z(0)}) \cdot e(H(i)^{r_i}, g^{q_z(0)})}{e(g^{r_i}, H(i)^{q_z(0)})} \\ &= \frac{e(g^r, g^{q_z(0)}) \cdot e(H(i), g)^{r_i q_z(0)}}{e(g, H(i))^{r_i q_z(0)}} \\ &= e(g^r, g^{q_z(0)}) \\ &= e(g,g)^{r \cdot q_z(0)}. \end{aligned}$$

However, if the attribute $i \notin \mathbb{S}$, then DecryptNode returns a blocking error value, denoted by $\perp$.

On a non-leaf node, DecryptNode first collects the results from all of its child nodes. It then chooses a random subset of k_x child nodes of z, where $F_x \neq \perp$ for each chosen node. If there are not k_x child nodes satisfying this criteria, then F_z returns $\perp$. If there is such a set S_x of these child nodes, a polynomial interpolation is performed on the nodes,

by letting $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$, and then computing

$$\begin{aligned} i &= index(x), \\ S'_m &= \{index(x) : x \in S_x\}, \\ F_z &= \prod_{x \in S_x} F_x^{\Delta_{i,S'_x}(0)} \\ &= \prod_{x \in S_x} \left(e(g,g)^{r \cdot q_x(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{x \in S_x} \left(e(g,g)^{r \cdot q_{parent(x)}(index(x))}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{x \in S_x} e(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\ &= e(g,g)^{r \cdot q_z(0)} \end{aligned}$$

Once this returns a value for the root node, if it is not a blocking value, decryption on the ciphertext can be performed. If it is not blocking then the result from the root node is $e(g,g)^{r \cdot q_R(0)} = e(g,g)^{r \cdot s} = A$. Decryption on the entire ciphertext is defined by

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

2.6 Delegation

Delegation is a method which allows a user to generate another secret key, $\tilde{SK}$, based on a subset $\tilde{\mathbb{S}}$ of the user's attribute set $\mathbb{S}$. This key is equal in privilege to that of a key with the attribute set $\tilde{\mathbb{S}}$ generated through an official key generation of section 2.3, but is able to be generated by a user, without requiring the master key.

Similar to the key generation algorithm, the delegate key is generated by first randomly choosing a value $\tilde{r}$ along with a random value $\tilde{r_k}$ for each attribute k within $\tilde{\mathbb{S}}$. This creates the delegate key

$$\tilde{SK} = (\tilde{D} = D \cdot f^{\tilde{r}}, \forall k \in \tilde{\mathbb{S}} : \tilde{D_k} = D_k \cdot g^{\tilde{r}} \cdot H(k)^{\tilde{r_k}}, \tilde{D'_k} = D'_k \cdot g^{\tilde{r_k}}).$$

One point of note is that the toolkit implementation of CPABE, provided by [4], does not contain support for this delegation method. Additional inspection shows that this could present issues with collusion when this is used, due to the fact that a user can choose $\tilde{r}$ during this operation, and in turn be able to bind attributes maliciously. However, this is not a direct issue due to the fact that the user generating these delegate keys already possesses these attributes, and will achieve no additional benefit from knowing these values. There are still issues though when the user creates delegate keys for multiple users and decides to reuse the value of $\tilde{r}$. In this case, other users would not know that these delegate keys are

malicious, and two delegate keys generated by the same user could be combined to gain access to ciphertexts the delegate users should not have access to. At the same time, this is a bit roundabout, as the user generating the delegate keys will always already have access to decrypt any ciphertexts, and could instead share the plaintexts. Another detriment of the delegation is with regards to key management strategies, as users could maliciously generate keys for themselves unknown to anyone else including the key management servers, and subsequently keep these keys to use after they have been revoked.

2.7 Security

For CPABE, the security of it is proven by the authors under the generic group model [4]. Additionally, due to the random values used during the key generation and encryption, care must be taken to ensure that all values are properly generated randomly, without the worry of reuse, as that may cause collusion attacks to occur. With the hashing function used, note that this needs to be public due to the fact that the encrypting users will need to compute these values in order to properly encrypt their messages. This means one can not put any security in the fact that these values will essentially be public. However, due to the multiplications within the encryption and key generation functions, these values are blinded and users are not able to use them to find out the other values they were multiplied by. Additional properties for security related to the bilinear groups used are discussed in section 3.4.

2.8 Example

Let an organization which is using CPABE to be split into four departments. These departments are engineering, marketing, accounting, and customer service. An example will be shown of the attributes which users contain in their keys, as well as various ciphertext access trees that can be satisfied by each of the keys. Each of the departments will have separate subgroups and teams which need to communicate between each other (or with another team). In the engineering department, these teams are design, programming, and testing. Within accounting, the two teams are payroll and investment. Teams could also use specific attributes associated with projects they are working on, which could stretch within departments. For example, the secret project could require communications between programming, testing, and marketing. The default CPABE scheme allows attributes to be produced as needed, without all of them required to be defined during the setup step.

Additional numeric properties are assigned to each user, such as executive level, and starting date. Executive level starts at 0, and increases by one for each lower level employee. This means that the CEO/President would have a value of 0, vice presidents a level of 1, and lower employees higher numbers. To adhere to the convention required by bag of bits, a four bit value would be used to represent this value, meaning there is also a limitation of 15 being the highest possible level. If more levels were to be added at a later date, each user's key would need to be regenerated in order to maintain consistency with the bits used

	Departments				**Teams**					**Projects**		**Numeric**
	Eng	Market	Account	CS	Des	Prog	Test	Pay	Invest	Secret	Web	EL
Alice	+					+					+	5
Bob	+					+					+	4
Charles		+								+		2
Dan	+	+	+	+	+	+	+	+	+	+	+	0
Erin			+					+				4
Francis			+						+	+	+	3
Greg				+						+		4
Heather	+				+					+		5

Table 2.1: Example of Users and Attributes

to represent the property within other keys as well as access trees. Hire dates would be represented as 32 bit values consistent with modern operating systems, at midnight UTC of the date the employee began employment with the company. Note that this is not desirable in this example, as a 32 bit value requires 32 attributes for the bag of bits representation, so performance would be hindered because all keys would have many more nodes in the access tree than actual properties.

We will now look at several users within the scheme, and three ciphertexts with varied access trees, that they will or will not be able to decrypt. Alice is a member within the programming team, hired on June 4, 2012. She is focusing on the web development project, and has an executive level of 5. On the other hand, her direct manager Bob, leader of the project, would have the same attributes as Alice, but his executive level would be 4. Charles is a manager of the secret project, in the marketing department, with an executive level of 2. Additionally, Dan is the CEO of the company, and this allows him to have an executive level of 0, as well as membership in all departments. Note that the CPABE scheme does not restrict attributes based on the possession of other attributes, or that membership in one department does not prevent a user from having membership in other departments in his or her key. The attributes that each user possesses, as well as several other users, is summarized in table 2.1. For the boolean attributes, + represents that the user in that row has that attribute, and a blank box represents that the user does not possess that attribute. For numeric attributes, the corresponding decimal value is shown. The hire date is not shown, as it is usually not a good metric for measuring access to files, as a company could have a recently hired CEO who would not be able to access some files, which is counterintuitive.

These users are attempting to communicate with each other using CPABE. Usually, most of the files being encrypted are restricted between the employees of each project, while others are restricted by the department (such as accounting). Let Alice encrypt a file discussing technical details of the project she is working on, meaning that it is only relevant to employees of the programming team, and is not needed by any other teams or departments. The second restriction would be that no employees with a lower executive level would be able to decrypt the file, such as new employees who did not have access

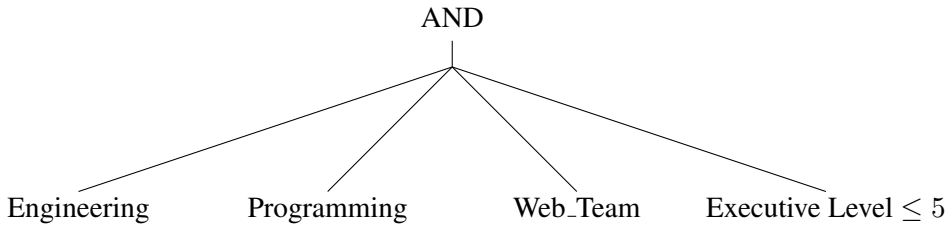

Figure 2.2: Access Tree of First Example

to project details, for example. Because of this, to ensure that only authorized employees could access the file, she would set the access tree of this file to "ENGINEERING and PROGRAMMING and WEB_TEAM and EXECUTIVE_LEVEL$\leq$5". With this access structure, the users from table 2.1 who contain the proper attributes to decrypt the file would be Alice, Bob, and Dan. The access tree associated with this file is shown in figure 2.2. Note that there would be a bag of bits used to represent the numeric value, which would in total take 5 leaf nodes and 3 non-leaf nodes to fully represent that attribute.

A second example file could be between the high level members of the secret project, regardless of team. Charles wants to send a message to the members of the project, except only for the executives who have an executive level of 3 or less. The access tree for this would simply be "SECRETPROJECT and EXECUTIVE_LEVEL$\leq$3". With this tree, Charles, Dan and Francis would be the only users from Table 2.1 with properties able to decrypt this message.

The final example message that could be used in this scheme would be composed by Francis, discussing her findings about possible investments for the secret project. This message has a much more complex access tree, requiring either a vice-president (executive level of 1), other members of the same project, also working in investment, or a somewhat lower executive within the accounting department, regardless of team. This access tree would be "EXECUTIVE_LEVEL ≤ 1 or (SECRETPROJECT and INVESTMENT) or (ACCOUNTING and EXECUTIVE_LEVEL ≤ 3)". The access tree associated with this structure is shown in Figure 2.3. Again, the bag of bits details covering the value of the executive level are not shown. The check for "EXECUTIVE_LEVEL ≤ 1 would take 3 leafs and 1 non-leaf node to represent, while the check for "EXECUTIVE_LEVEL ≤ 3" would take 2 leafs and 1 non-leaf node to represent. This file would then be able to be decrypted by Dan or Francis only.

2.9 Limitations

2.9.1 Negative Attributes and Nonmonotonic Access Trees

One of the major restrictions, outside of the lack of key revocation (see Chapter 6), is the structure of the access trees, and how it is unable to handle negative attributes. A negative attribute is when one wishes to restrict access based on a certain property. For example, if someone wants to restrict access to a file for all users of the engineering department, the

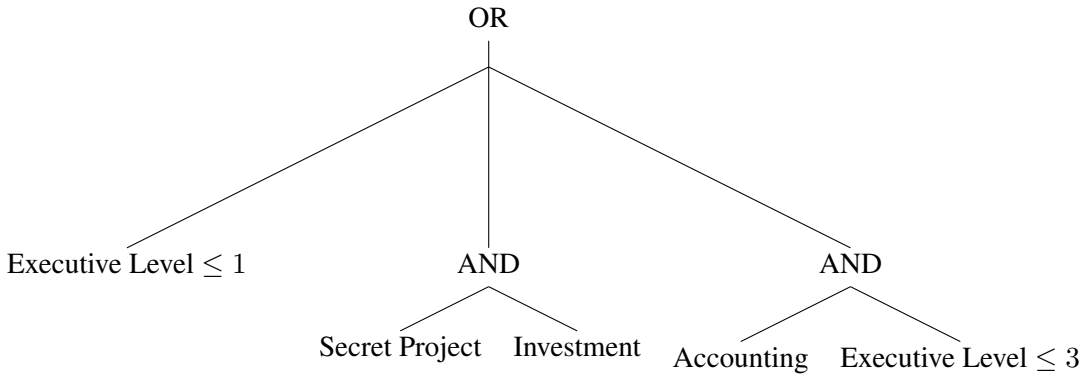

Figure 2.3: Access Tree of Final Example

access tree "not ENGINEERING" would not be valid within the CPABE scheme. In order to allow this scheme to be valid, they would have to explicitly label all other departments with an or gate, such as "ACCOUNTING or MARKETING or CUSTOMER_SERVICE", in the case of the previous example scheme. While this doesn't seem bad from this example, in applications in which there are many attributes at that same level, it would create extremely large access trees.

This restriction is due to the definition of monotonic access trees which are used to represent access trees within CPABE. By the definition of monotonic access trees from section 2.1, if a set of properties $\mathbb{S}$ satisfies the access tree, any superset $\tilde{\mathbb{S}}$ of those properties will also satisfy the access tree. When NOT gates are used within access trees, such as in the example above, and $\mathbb{S}$ does not contain the property "ENGINEERING" but satisfies the access tree, adding "ENGINEERING" to $\tilde{\mathbb{S}}$ will make it such that the monotonic property is violated.

2.9.2 Ciphertext Size

A second limitation is the size of the ciphertext. Regardless of message size, the ciphertext begins at 630 bytes due to the overhead needed to handle it. Additionally, every attribute produces 250-300 [36] more bytes, which can make expressive policies require a large overhead. Bethencourt claims that for an access formula of n attributes, the complexity of encryption and decryption becomes $O(n^{3.42})$ [19]. When trying to transfer messages over a limited medium, this overhead makes CPABE not a good candidate to be used for that application.

However, some variations of CPABE, such as the work of Goyal [14], present the ability to create an upper bound on the size of the access tree, and subsequently, the ciphertext. The requirement for a bound like this is by limiting the number of nodes in the access tree during the Setup step, as well as the maximum tree depth and maximum number of child nodes. A transformation is then used to ensure that all trees meet this desired format. Depending on the application using CPABE, these limits may not be feasible.

Chapter 3

Bilinear Groups

Bilinear groups were first used in 2001 by Boneh and Franklin [5] in order to provide an efficient implementation of the Identity Based Encyryption scheme proposed by Shamir in 1984 [31]. Soon after, the usage of these groups expanded to digital signature schemes, such as the BLS signature scheme [6]. The key reason for this usage is the fact that not only do these groups support Identity Based Encryption, they support it in a hierarchical manner such as that of an access tree, shown in section 2.1.

Definition 3.1 *A bilinear mapping, or bilinear pairing on two input groups G_1 and G_2 with generators g_1 and g_2, respectively, and output group G_T as the efficiently computable*

$$e : G_1 \times G_2 \rightarrow G_T$$

The function $e(x, y)$ will compute a value of G_T from a value $x \in G_1$ and $y \in G_2$.

For the scope of this thesis, only bilinear groups over finite elliptic curves are considered, except for a brief discussion of other pairings, covered in section 5.2. Therefore, the groups G_1 and G_2 will each be a set of points on an elliptic curve over the finite field $\mathbb{F}_p$, and the output group G_T is elements of the extension field $\mathbb{F}_{p^k}$, where k is a small integer. The value of k, also known as the embedding the degree of the curve, is defined based on the type of curve that is chosen (see Chapter 5).

Based on the curves chosen, a pairing is referred to as symmetric if $G_1 = G_2$, and nonsymmetric otherwise. The definition of CPABE in the previous chapter assumes that the pairing is symmetric, due to the flipping of elements in the pairing as seen during the DecryptNode function in section 2.5. However, this does not prevent asymmetric curves from being used for CPABE, and by careful rearrangement of the pairings during the DecryptNode stage of decryption, shown in section 5.3, these pairings can also be used for CPABE.

Ideally, one would like to find a bilinear map that is self-bilinear, that is $G_1 = G_2 = G_T$. However, there are no known examples of such a mapping, and Cheon and Lee [10] have stated that while some mappingss are possible that meet these properties, they are not cryptographically useful, and can easily be rewritten as a mapping that is not self-bilinear. They then proved that such a mapping cannot exist for the Computational Diffie-Hellman assumption in the group to be hard (see Section 3.4).

3.1 Properties

Two main properties are defined for a bilinear pairing, known as bilinearity and nondegeneracy. Assuming that the pairing is symmetric, meaning that both inputs are elements of the group G with generator g, these properties are as such.

Definition 3.2 *A bilinear mapping is bilinear if*

$$e(g^a, g^b) = e(g, g)^{ab} \, \forall a, b \in \mathbb{Z}$$

Definition 3.3 *The bilinear mapping is nongenerate if and only if*

$$e(g, g) \neq 1$$

Additionally, the pairings are also distributive, meaning that $e(xy, z) = e(x, z) \cdot e(y, z)$, which is also shown during the DecryptNode step in section 2.5.

Gagne [12] makes note of one property that is also defined by the non-degeneracy, which is the fact that assuming that g is a generator of G, then $e(g, g)$ is a generator of G_T.

When a pairing is asymmetric, a distortion map is defined as the relationship between G_1 and G_2. Assume that g_1 and g_2 are generators of G_1 and G_2, respectively. A distortion map is a mapping $\phi : G_2 \rightarrow G_1$ such that $g_1 = \phi(g_2)$ [22]. By the application of such a distortion map, it is guaranteed that the two points used for the pairing are linearly independent. If the points are linearly dependent, then pairing output will be degenerate, or 1.

3.2 Generic Bilinear Groups

Lynn states that the restrictions from the previous section are too constraining, and pairings can be performed on generic elliptic curve groups that instead satisfy the properties below. For a generic bilinear map $e : G_1 \times G_2 \rightarrow G_T$, the properties can be rewritten as follows.

Definition 3.4 *A generic bilinear map is bilinear if*

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} \, \forall a, b \in \mathbb{Z}, g_1 \in G_1, g_2 \in G_2$$

Definition 3.5 *A generic bilinear map is nondegenerate if*

$$e(g_1, g_2) = 1_{G_T} \forall g_2 \in G_2 \textit{ if and only if } g_1 = 1_{G_1} \textit{ and}$$
$$e(g_1, g_2) = 1_{G_T} \forall g_1 \in G_1 \textit{ if and only if } g_2 = 1_{G_2}$$

By using these properties instead to define the mapping, the usage of asymmetric curves is allowed. If the original equations for the map properties are used (3.2 and 3.3), then it is impossible to hash to G_2 securely. Depending on the application of the cryptosystem, this may be ideal, but sometimes it is not ideal. If hashing to all groups is required of the

scheme (as is the case for CPABE), then the generic properties for the pairing must be used. This only applies to hashing to elliptic curves, and may not be the case for possible other pairing groups, but that is not covered within this thesis.

Additional benefits are allowed by this generic definition. We are no longer restricted to only groups of prime order. These properties are defined for any groups where G_1 and G_T are of order r, and G_2 is of an order dividing r. Additionally, this allows G_2 to not be a cyclic group. One point of note, however, is that given the generators of the input groups, the output of their pairing is not guaranteed to be a generator of G_T. The pairing result $e(g_1, g_2)$ may then only generate an order r subgroup of G_T [22].

3.3 Ideal Properties

Selection of the pairing curve is dependent on the requirements of the cryptosystem it is being used in. Lynn states that ideally, three properties would be satisfied for cryptographic usage of bilinear pairings.

1. G_1 and G_2 are cyclic
2. One can securely hash to G_1 and G_2
3. There exists a one-way isomorphism $\phi : G_2 \rightarrow G_1$

However, no known pairing satisfies all three of these properties. On the other hand, one can easily find a pairing that satisfies any two of these [22]. Selection of a curve is dependent on both which of these properties are met by that curve, as well as which security problems one needs to be hard in the scheme.

3.4 Security

Three basic security problems are at the core of the usage of elliptic curves and subsequently bilinear pairings in cryptography. The first of these is the discrete logarithm problem, which is a problem used for many cryptosystems.

Definition 3.6 *(Discrete Logarithm Problem) Given g and g^x, find x.*

Ideally, this problem is only solvable by brute force, or multiplying successive powers of g until g^x is obtained. While there are attacks which make the problem computable in less than brute force, the assumption is that this problem is still hard, as no polynomial time algorithms are known. In the case of bilinear pairings, this problem is hard in both the elliptic curve elements used as the inputs, as well as on the pairing itself.

Two other closely related security problems are also defined at the core of the security of many schemes, the Computational Diffie-Hellman (CDH) problem and the Decisional Diffie-Hellman (DDH) problem.

Definition 3.7 *(Computational Diffie-Hellman Problem) "Given g, g^x, and g^y, find g^{xy}".*

Definition 3.8 *(Decisional Diffie-Hellman Problem) "Given g, g^x, g^y and g^z, determine if $z = xy$"*

Note that these problems are closely related, and any break to the discrete log problem would subsequently break both CDH and DDH [22].

With the nature of the bilinear pairings, there are also modifications to CDH and DDH that take advantage of these pairings, providing additional security problems outside of the three major ones. The Bilinear Diffie-Hellman is the equivalent of the CDH problem, defined below.

Definition 3.9 *(Bilinear Diffie-Hellman Problem) "Given g, g^x, g^y, and g^z, find $e(g,g)^{xyz}$".*

Once again, there is another duality of the DDH for bilinear maps, known as the Bilinear Decisional Diffie-Hellman (BDDH) problem.

Definition 3.10 *(Bilinear Decisional Diffie-Hellman Problem) "Given g, g^x, g^y, g^z and $e(g,g)^w$, determine if $w = xyz$".*

Just as is the case with the original CDH and DDH problems, any attack which makes the discrete log problem easy would make both of the bilinear problems easy as well. The bilinear equivalents incorporate a third input element, to challenge the properties of the bilinear pairings. Both of these bilinear problems are assumed to be hard within the bilinear map setting [3].

Within the scope of CPABE, some of the aforementioned problems may be easy to solve. Just because a problem is made easy within the scope of a scheme does not mean the scheme is inherently insecure. For example, Boneh and Franklin [5] state that in a symmetric bilinear pairing, DDH is easy. This is due to the application of pairings to solve this problem. Take the values given (g, g^x, g^y and g^z) and apply the pairing to g^x and g^y. By the bilinearity property, $e(g^x, g^y) = e(g,g)^{xy}$. With the other values given, apply a pairing again, to find $e(g, g^z) = e(g,g)^z$. If the output of both of these pairings are equal, then it is true that $z = xy$, thus DDH is easy. However, on the other hand, CDH is still hard to compute. The ability to solve DDH is only easy due to the fact that g^z is known. The internals of the pairing does not allow the values to be easily reversed, and even though $e(g^x, g^y) = e(g,g)^{xy} = e(g, g^{xy})$, that does not allow g^{xy} to be known.

Definition 3.11 *A group G where DDH is easy but CDH is hard is referred to as a Gap Diffie-Hellman (GDH) group [3].*

As for the BDDH problem, Boneh and Franklin yet again state that the hardness of this problem is equivalent to the hardness of the CDH within pairings. Additionally, any possible algorithms that make CDH easy would also make BDDH easy. Finally, the fact that DDH is easy in the input group does not mean it is easy in the output group of the pairing, as if it was, then it would be easy to invert the bilinear pairing [12].

For asymmetric pairings, Lynn mentions that another security problem can be defined, the Computational Co-Diffie-Hellman problem.

Definition 3.12 *(Computational Co-Diffie-Hellman Problem) "Given* $g_1, g_1^x \in G_1$ *and* $g_2 \in G_2$*, find* g_2^x*".*

As mentioned previously, the fact that the internals of the pairings can not be computed this way make it such that this problem is hard for the aymmetric pairing. Similarly, the Decisional Co-Diffie-Hellman problem can also be defined.

Definition 3.13 *(Decisional Co-Diffie-Hellman Problem) "Given* $g_1, g_1^x \in G_1$ *and* $g_2, g_2^y, g_2^{xy} \in G2$*, distinguish* g_1 *and* g_1^x *from* g_2*,* g_2^y *and* g_2^z*".*

Additionally, the hardness of DDH in an asymmetric mapping is dependent on whether the third property from section 3.3 regarding an isomorphism is met. If such an isomorphism exists, then DDH is easy in the asymmetric mapping. If there is no such isomorphism, DDH is hard [3].

Boyen [7] states that for generic bilinear groups, if any of these assumptions are not hard for those generic groups, there is no reason to believe that the scheme will be secure for lesser models. Additionally, proof of security restricted to only the generic group model is not strong enough to guarantee security in other models, as the generic model assumes that a malicious user needs access to an oracle in order to perform any of the operations such as bilinear pairings, which is not usually the case in attacks. This is an issue with the inital implementation of CPABE, as its security is only proven for this generic group model.

Waters realizes this in his own modification to CPABE [33], and presents the parallel bilinear Diffie-Hellman assumption, which takes $q+2$ random elements of $\mathbb{Z}_p$, $a, s, b_1, ..., b_q$, and the generator of the group G, where p is the prime order of the group. With this, security can be proven over a more secure model than the generic group.

Definition 3.14 *(Parallel Bilinear Diffie-Hellman Assumption) Given* $g, g^s,$ $g^a, ..., g^{a^q}, g^{a^{q+2}}, ..., g^{a^{2q}}$*, a challenger is unable to tell the difference between* $e(g,g)^{a^{q+1}s}$ *and a random element in the output group of the bilinear mapping with any polynomial time algorithm.*

Finally, these security problems are in no way restrictive, and only the ones closest to CPABE have been covered within this thesis. Hohenburger presents many more complexity assumptions in [16], as well as a discussion of the relationship between these problems.

Chapter 4

Elliptic Curves

This chapter covers the basics of elliptic curves, and how pairings are performed on elements of the elliptic curve. It shows an example of both point counting as well as pairing on a trivial curve.

Definition 4.1 *An elliptic curve over a finite field $\mathbb{F}_q$, where q is a prime greater than 3, is the set of points (X, Y) that satisfy*

$$E : Y^2 = X^3 + aX + b \tag{4.1}$$

as well as a point of infinity, O, also used as the identity element, for a given $a, b \in \mathbb{F}_q$.

Definition 4.2 *The discriminant of the elliptic curve is $\Delta = 4a^3 + 27b^2$.*

When $\Delta = 0$, then the curve has a repeated root, and is referred to as a singular curve. Only nonsingular curves are cryptographically useful, as singular curves provide no extra benefit over finite fields, and have slower operations. One may notice that the definiton of E does not account for variations such as a rotation, or shearing of the curve. However, there exists an affine linear transformation between such a curve and a curve in the above format, which allows one to easily find an equivalence of the points. Because of this, the assumption is made that all curves referred to in this thesis have been transformed into this format [22].

Two examples of elliptic curves are shown in Figures 4.1 and 4.2.

4.1 Point Addition

The points of an elliptic curve form an additive group. Because the group operation is addition, multiplication of a point by a constant is equivalent to exponentiation within integers. This multiplication is composed by a series of point additions on the curve. This is performed by doubling a point, P, to get $2P$, and doubling again to get $4P, 8P$, and so on, adding the results together to get points in between. Addition between two points is defined by the Chord-Tangent Law.

Definition 4.3 *(Chord-Tangent Law) The addition between any two points P and Q, $P \neq Q$, on the curve can be found geometrically by drawing a line on the elliptic curve between*

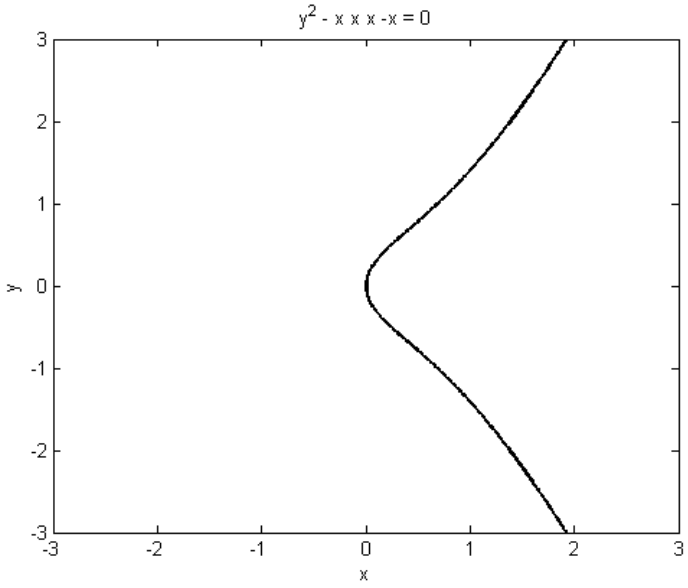

Figure 4.1: Elliptic Curve $y^2 = x^3 + x$

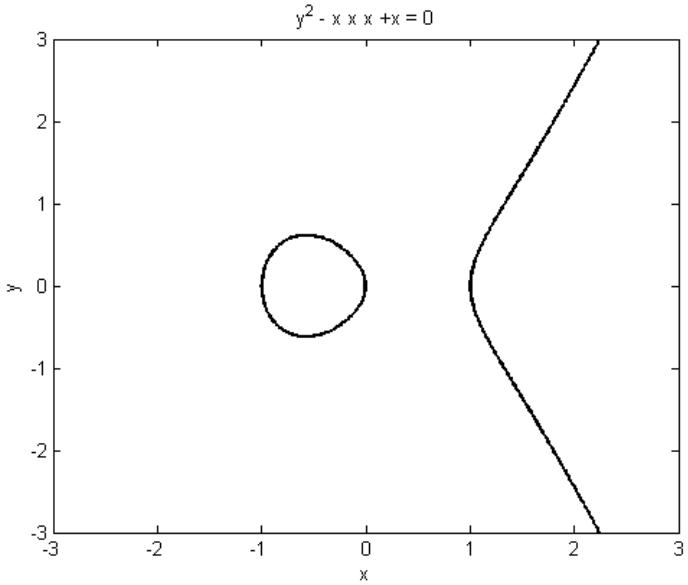

Figure 4.2: Elliptic Curve $y^2 = x^3 - x$

the two points, and finding a third point, $R = (x, y)$ on the curve that the line intersects. The sum $P + Q$ is equal to the inverse $(x, -y)$.

The proof that a third point exists on the line is not shown within the scope of this thesis. There are a few cases, however, that change this definition of addition. The first occurs when $P = (x, y)$ and $Q = (x, -y)$, as the line between the two points is vertical, and does not intersect any third point on the curve. In this case, the result of the addition is defined as O. A second variation is when adding $P = Q$, or finding $2P$. One first takes the line tangent to the curve at P, and there is a second point of intersection between the line and the curve, which when the y value is inverted is the result of the addition. The final difference occurs when one of the points is O, in which case basic axioms apply, and $P + O = P$.

One can numerically compute the point additions directly through the following equations. When one of the points is equal to O, identity operations do not require these steps. When adding $P = (x_1, y_1)$ to $Q = (x_2, y_2)$, one can compute the slope of the line between the two points. If $P = Q$, let the slope of the tangent line be

$$\lambda = \frac{3x_1^2 + a}{2y_1}. \tag{4.2}$$

Otherwise, the slope of the line between the two distinct points is

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}. \tag{4.3}$$

Regardless of which equation was needed to find the slope of the line, the coordinates of the sum, (x_3, y_3) can be found by computing

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

4.2 Point Finding and Counting

Due to the fact that the elliptic curves are defined over a finite integer field, and the fact that not all integers are valid coordinates for points on these curves, an algorithm to map any values of the finite field, K, to a point on the curve is needed. Let x be the value of the finite field that we are trying to map to the elliptic curve. The first step is to simply solve the curve equation (4.1) for Y by substituting the value of x into the equation, and taking the square root of Y^2.

Given a value of Y^2, the square roots are found using the Tonelli-Shanks algorithm [22], which finds $b = \sqrt{a}$, where $b \in \mathbb{F}_q$ and q is prime. The algorithm first finds a value $g \in \mathbb{F}_q$ that does not have a square root, which also means $g^{(q-1)/2} = -1$. Additionally, there is no requirement that this g be unique for each square root being found, and can easily be precomputed. Additionally, since q is odd (as the only even prime is trivial), write

$q - 1 = 2^s t$. For all values of i from 2 to s, if $(ag^{-e})^{(q-1)/2^i} \neq 1$, set an intermediate value $e \leftarrow 2^{i-1} + e$. Once this is looped for all values of i, set $h \leftarrow ag^{-e}$, and the square root $b = g^{e/2}h^{(t+1)/2}$.

Algorithm 1 Tonelli-Shanks Algorithm

Randomly choose an element $g \in F_q$ that is not a square.
Because of this, find an odd t such that $q - 1 = 2^s t$
$e \leftarrow 0$
for $i \leftarrow 2$ **to** s **do**
  if $(ag^{-e})^{(q-1)/2^i} \neq 1$ **then**
    $e \leftarrow 2^{i-1} + e$
  end if
end for
$h \leftarrow ag^{-e}$
$b \leftarrow g^{e/2}h^{(t+1)/2}$
Return b

If a valid square root is found, we have found two points on the curve, (x, Y) and $(x, -Y)$. Occasionally this will only produce a single point, where $Y = 0$, and the point lies on the x-axis. However, this can only occur for a maximum of three values of x for each curve. If a square root is not found, then there are no points for that x value, and a new x must be chosen. From this, hashing to an elliptic curve is possible. One possible algorithm first uses a hashing function to generate a value of x from the message that one is trying to hash to a point. Using this value of x, deterministically generate a value of $x \in \mathbb{F}_q$ until a valid value of y is found. From that, one of the values of y is again deterministically chosen from the two, and the hash output is the point (x, y).

Another method of finding the square roots is by determining the quadratic residues of the values obtained by substituting x into the elliptic curve equation.

Definition 4.4 *A value z is a quadratic residue mod p if and only if* $z^{(p-1)/2} = 1 \mod p$.

Additionally, an extension of this allows efficient computation of the square roots if the condition $p = 3 \mod 4$ is also met. Let z equal a value found by substituting x into an elliptic curve over $\mathbb{F}_p$, where $p = 3 \mod 4$ and is prime. If z is a quadratic residue of the curve over this field, the square roots of z are $\pm z^{(p+1)/4} \mod p = \pm z^3 \mod p$ [32]. If z is a valid quadratic residue, the curve has two points for the value of x used. If it is not, then there are no points for the value of x. An exception to this occurs when $z = 0$, as there is only a single point on the elliptic curve for this value, at (x,0).

Lynn [22] notes that these hashing and point finding algorithms allow an efficient manner for selecting a random point on the curve. One first chooses a random value of $x \in K$, and attempts to solve the curve equation for Y. Repeat these steps until an x that yields a valid value of Y is found. One this occurs, randomly select one of the two solutions of Y, and return the random point as (x, y). This is not truely random, as it prevents the point at infinity from being chosen via this algorithm, as well as the fact that points where $Y = 0$ have a higher probability of being picked. This can be ignored because rarely in cryptography is the point at infinity desired for a random point, and the definition of the elliptic

curves will have at most three points which are of a higher probability, almost negligible for the number of points on the curve. By having this algorithm for finding a random point instead being deterministic, hashing to points on the curve is now allowed. This algorithm would then take the input, map it to x, and if no points are found for x, deterministically modify the value and repeat from the beginning until a point is found. While point counting algorithms are not explicitly covered in this thesis, the number of points on the curve is roughly the size of the finite field, or q^k.

Definition 4.5 *The order n of an elliptic curve E, is equal to the number of points on the curve, including the point at infinity.*

Definition 4.6 *The order r of a point P on an elliptic curve is the minimum positive integer k such that $kP = O$.*

Because of the fact that every point on E generates a cyclic group G, this cyclic group is what is used for cryptographic operations. The difficulty with these operations is ensuring that a randomly chosen or hashed points lie within G and not elsewhere on the curve. To do so, find a prime r such that $r \mid n$, but $r^2 \nmid n$. Operations will all be of points on E and G of order r. For the above mentioned random elements, proceed normally, except multipling by n/r after a random or hashed point on E is found [22].

4.3 Example

Let E be the elliptic curve defined by $y^2 = x^3 + x$ over the finite field $\mathbb{F}_{11}$. Note that this curve is the same that is used by the initial implementation of CPABE [4], albeit over a much smaller finite field for ease of this example. Since $p = 11 = 3 \mod 4$, the quadratic residue method can be used to find the points on this curve. In this case, a value z is a quadratic residue if $z^{(p-1)/2} = z^5 = 1 \mod 11$. Table 4.1 shows the values of z that correspond to quadratic residues, and Table 4.2 shows how these residues are used to find the points of the elliptic curve.

With all of the points listed in Table 4.2, the number of points on this elliptic curve is 12, the 11 points shown in the table and the point at infinity. The order of each point is then determined by successive additions. Additions are performed due to the fact that the group operation is addition for elliptic curves.

To find the order of an element on the elliptic curve, successive point additions are performed until the point at infinity is obtained. Begin with the point (5,3), denoted by P for this example. Using equation 4.2 due to the fact that the points added are the same, the λ value obtained for $2P$ is $(3 \cdot 5^2 + 1)/(2 \cdot 3) \mod 11 = 10 \cdot 6^{-1}$. Because this calculation is over a finite field, the modulus must be applied, and the inverse is used to represent division in such fields. Solving for the inverse, it is found that $\lambda = 9$. With this value, the coordinates of the new point can be found, with $x_3 = 9^2 - 5 - 5 = 71 = 5 \mod 11$ and $y_3 = 9(5 - 5) - 3 = 8$, and $2P = (5, 8)$.

Now extending on this, one can find $3P = P + 2P$ using the same method. When this is applied, however, one finds that λ can not be found due to the fact that the denominator

z	$z^5 \mod 11$	Quadratic Residue?
0	0	No
1	1	Yes
2	10	No
3	1	Yes
4	1	Yes
5	1	Yes
6	10	No
7	10	No
8	10	No
9	1	Yes
10	10	No

Table 4.1: Quadratic Residues of Elliptic Curves over $\mathbb{F}_{11}$

x	$z = (x^3 + x) \mod 11$	Quadratic Residue?	z^3	Points
0	0	Exception	0	(0,0)
1	2	No	-	-
2	10	No	-	-
3	8	No	-	-
4	2	No	-	-
5	9	Yes	± 3	(5,3), (5,8)
6	2	No	-	-
7	9	Yes	± 3	(7,3), (7,8)
8	3	Yes	± 5	(8,5), (8,6)
9	1	Yes	± 1	(9,1), (9,10)
10	9	Yes	± 3	(10,3), (10,8)

Table 4.2: Points on Elliptic Curve $y^2 = x^3 + x$ over $\mathbb{F}_{11}$

requires the computation of 0^{-1}. In this case, note that $(5, 8) = (5, -3)$, and that $y_1 = -y_2$, meaning that by definition of elliptic curve addition, the result is the point at infinity. Because the point at infinity was obtained at $3P$, the order of P is 3.

4.4 Pairing

Two pairing algorithms, Weil and Tate, are used for calculation of pairings over elliptic curves. Instead of directly using points from the curve, they use r-torsion points of the curve as the input fields, and the output is an rth root of unity of the finite field.

4.4.1 Torsion Points

Definition 4.7 *An r-torsion point is the set of points P such that $rP = O$.*

The set of r-torsion points can also be denoted by $E(K)[r]$ or $E[r]$, where $K = \mathbb{F}_{q^m}$ is the finite field the elliptic curve is over. In other terms, this means that the order of P is r or a factor of r. For all curves, the only r-torsion points which are cryptographically useful are those where r and q are coprime.

Definition 4.8 *An elliptic curve E is supersingular if and only if the only p^{th} point of torsion for any prime p is the point at infinity, or the number of points on the curve $\#E(\mathbb{F}_{p^m}) = p^m + 1 - t$, for $p \mid t$ [13].*

Contrary to its name, a supersingular curve is not a singular curve. Any nonsupersingular curve is referred to as an ordinary curve. Additionally, for any curve E, there exists a value k such that $E(\mathbb{F}_{q^k})[r]$ contains r^2 points, and no more r-torsion points will exist for any greater value of k.

Care must be taken to ensure that r is coprime to q, or the set of points is vulnerable to a anomalous attack, which will solve the discrete log within $E[r]$ in linear time. This can be avoided by ensuring that the order of the curve is coprime to q [22].

4.4.2 Embedding Degree

With this, we can define the concept of the embedding degree of E.

Definition 4.9 *The embedding degree of E is the smallest value of k such that all r^2 r-torsion points of the subgroup $G \subset E(\mathbb{F}_q)$ of order r and $r \nmid q - 1$ exist on the curve.*

This value is also the minimum value of k where $r \mid q^k - 1$, allowing this to be easily computed. In terms of an elliptic curve pairings, this degree is approximately the ratio of the input elements to output element sizes, in bits. For example, if the embedding degree of E is 2, then typical pairings will consist 512-bit elements as inputs, and 1024-bit elements for the output. One method of finding this embedding degree is just solving $n \mid q^k - 1$, which also ensures that the coprime requirement from section 4.4.1 is also met.

In the curves used in the PBC Library, embedding degrees up to 12 are supported, depending on the type of curve. One point of note is the fact that there are no known supersingular curves with an embedding degree higher than 6 [22]. If an application requires curves of a higher degree, or if attacks are found such that the security of lower embedding degree is compromised, then only ordinary curves would be able to meet the security properties needed for the bilinear maps.

4.4.3 Distortion Maps

Distortion maps are used for symmetric pairings, as a means of creating an isomorphism between the two inputs. This is denoted as $\Psi(x, y) = (-x, iy)$, and is used in the bilinear mapping as $e(P, Q) = f(P, \Psi(Q))$, where f is the Weil or Tate pairing function [22]. The usage of these distortion maps is required due to the fact that when in a symmetric mapping, the pairing $e(P, P) = 1$, which violates the degeneracy property for the bilinear pairing. Therefore, distortion maps are needed in order to guarantee the two input points for a symmetric pairing are linearly independent. Maas states that this map is only valid when the curves are over an integer field of prime order p, and $p \equiv 3 \mod (4)$ [24].

4.4.4 Weil Pairing

Definition 4.10 *The Weil pairing is a mapping of two r-torsion points to an element of the finite field as such:*

$$f : E[r] \times E[r] \rightarrow \mathbb{F}_{q^k}$$

The algorithm computing the Weil Pairing is based on the evaluation of two rational functions f_P and f_Q. Details behind how such rational functions are defined, and details such as the Poles and Zeroes of such functions, is left to Lynn's work [22].

To compute this pairing of two input points P and Q, two additional points R and $S \in E[\mathbb{F}_{q^k}]$ must be chosen, where $S \neq R, P + R, P + R - Q$, or $R - Q$. Following that, define the rational function f_P with divisor $(f_P) = (P + R)^r/(R)^r$, and f_Q with divisor $(f_Q) = (Q + S)^r/(S)^r$. With these two functions, the pairing output is the value $f(P, Q)$ obtained from equation 4.4.

$$f(P, Q) = \frac{f_P(Q + S)/f_P(S)}{f_Q(P + R)/f_Q(R)} \tag{4.4}$$

4.4.5 Tate Pairing

Definition 4.11 *The Tate pairing is:*

$$f : E[r] \cap E(K) \times E(K)/rE(K) \rightarrow K^*/K^{*r}$$

Note that in this case, only a single point of torsion is used for an input, as opposed to two of which are used as the inputs of the Weil pairing. The second point can be any point on $E(K)$, regardless of order.

Similar to the Weil pairing, computation of the Tate pairing begins by defining a rational function f_P with divisor $(f_P) = (P)^r$. Note that this again only requires a single function as such, compared to the two required for the Weil pairing. Once such a function is found, the pairing chooses R, a point of $E(K)$, where $R \neq P, P - Q, O$, or $-Q$. The result from the pairing, $f(P, Q)$, is shown by equation 4.5

$$f(P, Q) = \frac{f_P(Q + R)}{f_P(R)} \tag{4.5}$$

4.4.6 Miller's Algorithm

Miller's algorithm is designed for computing the pairing of two points, using linear functions based on the input points. The linear functions are defined on points as such:

1. L: The line between the points U and V on the curve as defined by the Chord-Tangent law (see Section 4.1), in the form $L_{U,V} = 0$.
2. T: The line tangent to the curve, with only point of intersection at $U = V$. Since there is only one point of intersection, this can be defined easily as $T_U = 0$. Additionally, $T_U = L_{U,U}$.
3. V: The vertical line between the points U and $V = -U$, in the form $V_U = 0$. Similar to the case with T, $V_U = L_{U,-U}$. If $U = -U$ then $T_U = V_U$ [22].

Evaluation of these functions is performed on a point $S = (X, Y)$. These evaluations are shown clearly in the example shown in section 4.4.7.

This leads us to Miller's algorithm for finding the Tate pairing on points P and Q. Note that a modification of this algorithm can be used to find the Weil pairing as well, but the Weil pairing is not used for the pairings of curves used in this thesis. The value r_i referred to in this algorithm is the binary bit i in the order r of the point P, in the form $r = r_t...r_0$.

Algorithm 2 Miller's Algorithm

$x \leftarrow 1$
$Z \leftarrow P$
for $i \leftarrow t - 1$ **to** 0 **do**
  $x \leftarrow x^2 \cdot T_Z(Q)/V_{2Z}(Q)$
  $Z \leftarrow 2Z$
  if $r_i = 1$ **then**
    $x \leftarrow x \cdot L_{Z,P}(Q)/V_{Z+P}(Q)$
    $Z \leftarrow Z + P$
  end if
end for

Following the completion of this algorithm, one final exponentiation is performed in order to generate the output of the pairing. The output of the pairing is $x^{(q^k-1)/r}$.

4.4.7 Example

A trivial example of the Tate pairing using Miller's algorithm is now shown. The same curve used from section 4.3 is used, $E : y^2 = x^3 + x$ over the field $\mathbb{F}_q$, where $q = 11$.

As shown in section 4.3, the order of the curve is $n = 12$. The first step in computing this pairing is to find a large prime r that divides n. Due to the fact that this is such a trivial example, the largest prime which is a factor of n is $r = 3$, which is the value that will be used for this pairing.

Once r is found, it must be tested to find a k value such that r divides $q^k - 1$ but r^2 does not divide that value. Starting with $k = 1$, $q - 1 = 10$. Since 3 does not divide into 10, $k = 1$ is not valid. Incrementing for $k = 2$, $q^2 - 1 = 120$, and because $r \mid 120$ while $r^2 = 9 \nmid 120$, $k = 2$ is the smallest value which satisfies these properties. That means that the embedding degree for pairings on this curve is 2.

The next step is to find all of the r-torsion points on E. Using the list of points found from the previous example in section 4.3, the only points of order 3 are $(5, 3)$ and $(5, 8)$. Due to the fact that r is prime, there are no other factors of r, so one does not have to check for points with order equal to a factor. What this means, however, is that Tate pairings are only defined for $P \in E[r]$, but Q is not restricted, and can be any other point on this curve. Note that if the Weil pairing was to be computed, the selection of points would be restricted to $P, Q \in E[r]$.

Now we will compute the Tate pairing of $P = (5, 3)$ and $Q = \phi(P) = (6, 3i)$ using Miller's Algorithm. The algorithm starts by setting $x = 1$ and $Z = P$. It then loops through values of t. Since $r = 3 = 11_2$, $t = 1$, and $r_1 = 1$, $r_0 = 1$. The algorithm will only loop a single time, for $i = 0$.

With the loop at $i = 0$, $x = x^2 \cdot T_Z(Q)/V_{2Z}(Q)$. Since $x = 1$ to start, we can ignore the x values and just compute the two linear functions. $T_Z = T_P =$ the line tangent to $P = 2x + y + 4$. When evaluated at Q, this output is $2 \cdot 6 + 3i + 4 = 16 + 3i = 5 + 3i$ when simplified for the finite field used. The next function needed is V_{2Z}. Since $Z = P$, $2Z = 2P = (5, 8)$. This works out to be the same vertical line through P, and the function for this line is $x + 7$, which when evaluated at Q gives the value 2. Miller's algorithm now computes $(5 + 3i)/2 = (5 + 3i)(2)^{-1} = 8 + 7i$. The algorithm then sets $Z = 2Z = (5, 8)$, and checks if $r_0 = 1$. Since this value is true, the if statement is then executed as well. The value $x \cdot L_{Z,P}(Q)/V_{Z+P}(Q)$ needs to be computed. Because $Z = (5, 8) = 2P$ for this statement, $L_{Z,P} = V_P = x + 7$. Additionally, since $Z = 2P$, $Z + P = 3P = O$ because P is a 3-torsion point. In this case, this statement would be the equivalent of dividing by zero. In order to perform this calculation, this zero value is instead set to 1, equivalent to the division just not occuring. Still, $L(Q)$ needs to be evaluated, and this value is found to be 2. This means that $x = x \cdot 2 = (5 + 3i)$. Since this is the final iteration of the loop, the final x value, $5 + 3i$, is the result of the algorithm.

The final step of the pairing is to then find $x^{q^k-1/r} = x^{40} = 5 + 3i$. To ensure that this pairing is correct, one important step is to ensure that $(5 + 3i)^r = (5 + 3i)^3 = 1$, which is true. If this value was not equal to 1, then the output of the pairing would definitely be incorrect due to the fact that the output would not reside in the output field defined by the Tate pairing.

Chapter 5

Group Selection

While many different types of curves have been proved to be cryptographically useful in the scope of bilinear pairings, there is limited research published regarding the performance of these different curves (or a mapping outside of elliptic curves) within CPABE.

This chapter starts off by introducing the types of bilinear pairings supported in Lynn's PBC library [23], and a brief summary of how these curves are found or generated. Following that, a small discussion regarding the usage of maps other than elliptic curves is provided. This section is followed by a proof to show that asymmetric pairings are possible for usage in CPABE.

The rest of the chapter regards performance analysis performed for different types of curves supported in the PBC library, both symmetric and asymmetric curves. Previous work is discussed, and a detailed analysis of speed and size is given for these curves within the scope of CPABE. The chapter concludes with recommendations for group selection based on the application of CPABE.

5.1 Elliptic Curves

The Pairing Based Cryptography Library [23] was designed to support seven types of elliptic curves as the underlying fields for the pairings, denoted type A through type G. Of these curves, types A,B, and C are symmetric pairings over supersingular curves, and types D through G are asymmetric pairings on ordinary curves.

5.1.1 Supersingular Curves

Type A Curves

A Type A curve is an elliptic curve of the form

$$y^2 = x^3 + ax, a \in Z_p.$$

over a finite field $\mathbb{F}_{q^2}$, where q is prime and $q = 3 \bmod 4$. The values of a that provide the best performance are -3, 1, and -1 [22]. However, in the library that is used for the CPABE implementation, this is hardcoded to only support the value $a = 1$, as it enables certain optimizations. These curves have an embedding degree of 2, and also allow easy

computation of the number of points on the curve, as $\#E(\mathbb{F}_q) = q + 1$, and $\#E(\mathbb{F}_{q^2}) = (q + 1)^2$.

Type B Curves

Type B curves are curves satisfying

$$y^2 = x^3 + b, b \in Z_p$$

over the finite field $\mathbb{F}_q$, where $q = 2 \mod 3$. Similar to the Type A curves, pairings using these curves are also of embedding degree 2. However, curves of this type are not implemented in Lynn's PBC library due to the fact that they provide little or no advantage over the usage of Type A curves.

Type C Curves

A type C curve is a set of points which satisfies the equation

$$y^2 = x^3 + 2x \pm 1$$

over the finite field $\mathbb{F}_{3^l}$. For these curves, they have an embedding degree of 6. However, there are very few curves which are cryptographically useful in this type, and there are only seven known curves which have a field size of less than or equal to 265 bits.

Similar to type B curves, type C curves are again not implemented in the main library due to the fact that they are vulnerable to a Coppersmith attack on the low characteristic field [22].

5.1.2 Ordinary Curves

Complex Multiplication

For many of the ordinary curves, a method known as complex multiplication is used in order to find curves of a certain embedding degree with a specified order. The complex multiplication equation is

$$DV^2 = 4q - t^2 \tag{5.1}$$

where D, V, q, and t are all integers, with q prime and $D > 0$ as well as no odd prime squared divides D. From here, let j be any root of the Hilbert polynomial for D, except for $j = 0$ or 1728, and $k = j/(1728 - j)$. The elliptic curves formed by this complex multiplication method are

$$E : y^2 = x^3 + 3kc^2x + 2kc^3 \tag{5.2}$$

for any nonzero value of c. Curves formed by this method will have order $q + t - 1$ or $q - t - 1$.

Type D Curves

Type D, or MNT, curves have an embedding degree of $k = 6$, and are constructed by substituting $U = 3x \pm 1$, $q = x^2 + 1$ and $t = \pm x + 1$ into the complex multiplication equation. From here, select a value of D and solve $U^2 - 3DV^2 = -8$. By the definition of U, $U = \pm 1$ mod 3, so we can set $x = (-1 \pm U)/3$. The next step is to check if q is prime, and that $q - t + 1 = q \mp x$ has a large prime factor r. Another check needs to be made to ensure that $r \nmid q^j = 1$ for any positive integer $j < k$. However, [22] notes that this is unlikely to ever occur. Regardless, if these checks pass, then the complex multiplication method will produce a curve of order $q \mp x$.

Type E Curves

Curves denoted as Type E have an embedding degree of 1, and are designed by setting $t = 2$, $D = 7$, and $q = 28r^2h^2 + 1$ for a positive r and some value of h. The complex multiplication equation becomes $7V^2 = 4(28(rh)^2 + 1) - 4$ after these substitutions, and has a solution $V = 4rh$. Take H as the hilbert polynomial for $D = 7$, which is $H(x) = x + 3375$, which will produce a curve $y^2 = x^3 + 3kx + 2k$ of order either $q - 1$ or $q + 3$. If this curve has order $q + 3$, using the same k value found from the complex multiplication method, instead use the curve $y^2 = x^3 + 3kc^2x + 2kc^3$, where c is any quadratic nonresidue in $\mathbb{F}_q$.

Type F Curves

Type F curves, referred to as Barreto-Naehrig [2] curves, have a large embedding degree of 12, the largest possible for elliptic curves. These curves are once again designed through the complex multiplication method, setting $q = 36x^4 + 36x^3 + 24x^2 + 6x + 1$, $t = 6x^2 + 1$, and $D = 3$, which has solution $V = 6x^2 + 4x + 1$. Additionally, this has the property that $q + 1 - t \mid q^1 2 - 1$. To produce a curve of Type F, take any value of x , and check that q is prime for that value of x and that $n = q - t + 1$ has a large factor r, ideally prime. Then try values of k until any point of $E : y^2 = x^3 + k$ has order n. This curve that satisfies these properties is the curve to be used for the pairing.

Type G Curves

The final type of curves supported in [23] are Type G, which are of embedding degree 10. Similar to Type F curves, start with $q = 25x^4 + 25x^3 + 25x^2 + 10x + 3$ and $t = 10x^2 + 5x + 3$. Then find any value of D that is both squarefree and either 43 or 67 mod 120. Once that is completed, find a solution (u, v) of $u^2 - 15Dv^2 = -20$. u will always be equal to ± 5 mod 15, meaning that $x = (-5 \pm u)/15$. Just as in Type D curves, check that both q is prime and that $n = q - t + 1$ has a large prime factor r. If it does, use the complex multiplication method to produce this curve.

5.2 Outside of Elliptic Curves

Lynn [22] states that RSA and integer exponentiation also fufill the conditions of a bilinear map. This allows the mapping

$$e(g, a) = g^a, g \in \mathbb{Z}_r, a \in \mathbb{Z}_{r-1}, r \in \mathbb{Z}, r > 1$$

to be used. However, for security reasons, the recommended sizes are still that of the normal RSA algorithm, which are much larger than the 160-512 bits needed for elliptic curve elements. On average, the operations required for 1024-bit RSA computations take three times the time as that of the equivalent 160-bit EC operation [16]. This is only for standard operations, however, and one bilinear pairing over a 521-bit elliptic curve is equivalent to eight modular exponentiations of corresponding finite fields of the same security level.

Hyperelliptic curves are also a possibility to be used for a bilinear pairing. [22] notes that while this is possible, no cryptographic usages are shown that produce a higher security level. While a higher embedding degree is allowed for these curves, its only a minor increase in this value, while the complexity of the elements and pairings is much more difficult.

A vector of integers with inner product defined also constitutes a bilinear pairing, mapping two vectors onto a value within an integer field. One example of this is shown in [25], but there are little cryptographic implementations shown for these vectors, as each is simply a set of integer values, and would likely need to adhere to those security lenghts (such as 1024 bits), which impose the same difficulties as using exponentiation as the pairing function. Besides that, many of the pairings on elliptic curves could easily be translated to integer finite field representations.

5.3 Asymmetric Pairings in CPABE

Due to the fact that only one specific supersingular curve can be implemented using the PBC library, in order to perform a performance comparison of various curves, it must be ensured that asymmetric bilinear pairings are possible in CPABE.

To use asymmetric pairings, two functions within CPABE need to be modified, the DecryptNode and final decryption steps, due to the fact that those functions are the locations where the the order of the input to the pairings is needed. Note that during the key generation and encryption steps, two values are generated for each attribute, D_i and D_i' for the key, and C_x and C_x' for the ciphertext. Asymmetric usage can be obtained by using one of the input groups for one of each of these, and the other input group for the other two.

One proposed modification for this usage is obtained by having D_i and C_x' to be elements of G_1, and D_i' and C_x within G_2. In order to properly decrypt at each node, the order of one of the inputs must be switched, as the original decrypt node does manipulate the order of attributes to properly decrypt. Of course, this also assumes that the setup step is also modified to support two input groups, with generators g_1 and g_2, respectively. These modifications can be properly applied by:

$$DecryptNode(CT, SK, z) = \frac{e(D_i, C_x)}{e(C'_x, D'_i)} = \frac{e(g_1^r \cdot H(i)^{r_i}, g_2^{q_z(0)})}{e(H(i)^{q_z(0)}, g_2^{r_i})}$$
$$= \frac{e(g_1^r, g_2^{q_z(0)}) \cdot e(H(i)^{r(i)}, g_2^{q_z(0)})}{e(H(i)^{q_z(0)}, g_2^{r_i})}$$
$$= e(g_1, g_2)^{r \cdot q_z(0)}.$$

For the interpolation during decryption, no changes are needed due to the fact that the results are mainly multiplied from the node decryption step. When it comes to the final decryption, however, the interpolation will provide the value $e(g_1, g_2)^{r \cdot s}$. Additional modifications are needed during the setup and key generation steps:

1. During the setup step, set $h = g_1^{\beta}$.
2. The public key contains both input groups, $\mathbb{G}_1$ and $\mathbb{G}_2$, both generators g_1 and g_2, and the pairing $e(g_1, g_2)^{\alpha}$.
3. For key generation, set $D = g_2^{\alpha + r/\beta}$.
4. In encryption, $\tilde{C} = M \cdot e(g_1, g_2)^{\alpha s}$.

With all of these modifications, the final decryption can be performed as such:

$$\frac{\tilde{C}}{\frac{e(C,D)}{A}} = \frac{\tilde{C}}{\frac{e(h^s, g_2^{(\alpha+r)/\beta})}{e(g_1,g_2)^{r \cdot s}}} = \frac{M \cdot e(g_1, g_2)^{\alpha \cdot s}}{\frac{e(g_1,g_2)^{\beta \cdot s \cdot \frac{\alpha+r}{\beta}}}{e(g_1,g_2)^{r \cdot s}}} = \frac{M \cdot e(g_1, g_2)^{\alpha \cdot s}}{\frac{e(g_1,g_2)^{\alpha \cdot s} \cdot e(g_1,g_2)^{r \cdot s}}{e(g_1,g_2)^{r \cdot s}}} = M$$

5.4 Supporting Work

Two known implementations of CPABE that incorporate asymmetric bilinear pairings are the CHARM library, developed by Akinyele, Green, and Rubin [1], in addition to PIRATTE by Jahid and Borisov [20]. In the first implementation, several pairing based cryptosystems are demonstrated, and the version of CPABE supported in the CHARM library does allow for asymmetric pairings to be used, by specifying that D, D_j, and C'_x are elements of G_2, with D'_j and C_x as elements of G_1, the opposite of the group assignments previously proposed. However, this framework, in addition to the published papers relating to it, does not contain any details about the performance or justification of using ordinary curves. The author does state that there is some overhead obtained during the encryption, but the key generation and decryption is comparable to the C implementation. They believe that this overhead is due to the parsing the attributes into an access policy.

On the other hand, the PIRATTE scheme is required to use asymmetric pairings due to the fact that if the same group is used for both inputs, users can maliciously combine these to skip the key revocation policies that the scheme was designed for, similar to collusion attacks. More details regarding this scheme and its usage for key management is provided

in section 6.3. They did provide a comparison of the usage of MNT curves to the type A curves in the original CPABE implementation, and found that there was a performance decrease in using the type D MNT curves, with the MNT curves taking 2-3 times as long as the original implementation for encryption, decryption, and key generation.

Lynn [22] states that the size of the elements in the curve is related to whether a symmetric or asymmetric pairing is chosen. For a symmetric pairing over a supersingular curve, security is achieved for 512 bit elements. With an asymmetric pairing, security is retained for smaller element sizes, 160 bits for G_1 and 320 bits for G_2. A tradeoff for the smaller elements is the fact that the pairings are slower. Overall, Type F and G curves required the smallest input size, and types A, B, D, and E produced the smallest output sizes. The fastest pairings were for Type A and B curves. For the usage of asymmetric curves within CPABE, a cursory glance makes type D curves appear most plausible for usage, as they are faster than type F or G curves, with only a slight size increase. For these group sizes, type F and G curves have the smallest input sizes, and type A,B,D, and E have the smallest output sizes.

On the contrary, Page *et al.*[27] found that MNT curves had better performance for bilinear pairings in an IBE scheme than the supersingular curves. This performance increase was roughly 50-60 % over supersingular curves of the same size. However, the cost of this higher performance was that keys resulting from the MNT curves were three times the size of the type A curves.

Another article, by Chen *et al.*[9], backed up this research by showing another modification to a pairing based scheme, this time the Direct Anonymous Attestation algorithm. The original implementation of this modification consisted of symmetric pairings, and was converted to an asymmetric pairing. Upon doing so, there was no additional overhead in terms of performance when using these pairings, and the security remained the same if not higher.

5.5 Experiment Design

For the tests performed as part of this thesis, timing analysis was performed for the various CPABE functions, over a variety of different curves supported. In order to provide easy usage of different curves, the CHARM framework [1] was used, which provides a quick python prototype of CPABE that is generic and can be used with any curve supported by the PBC library. All underlying computations are still using Lynn's library, with the major changes provided by this framework allowing easy generation of access trees and serialization. The first modifications performed were to allow support for type E, F, and G curves, which were easily added by using the same curves provided by the PBC Library.

Two main types of access trees were designed for these tests, to show the extremes of different access trees supported. The first type of access tree is referred to as a "horizontal" tree, which will always consist of a two level access tree. The first level of this access tree is the root node, which is an n-input and gate, where n is the number of properties in the access tree. The second level is all of these properties which are required in order to decrypt the file. An example of such a tree can be seen in figure 5.1

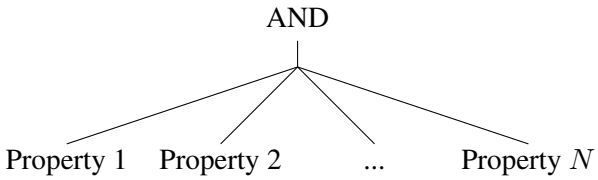

Figure 5.1: Horizontal Access Tree

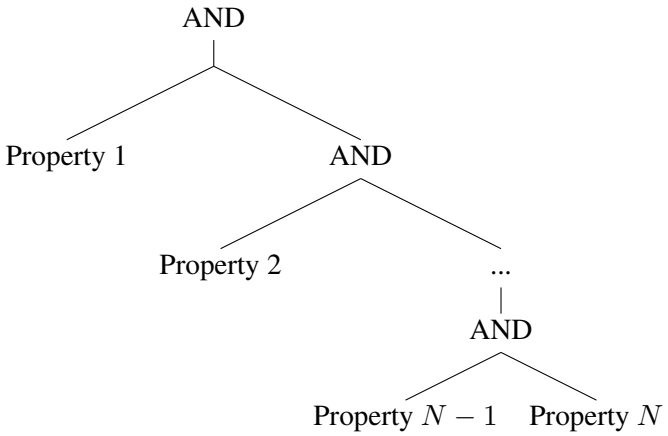

Figure 5.2: Vertical Access Tree

On the other hand, a "vertical" access tree consists of many more levels, with each node having either zero or two children. The root node is a 2-input and gate, with the first child being one property, and the second child being another 2-input and gate, in which one child of that gate is the second property, and the other child is again another 2-input and gate. This continues until all of the properties are within the tree, the last two properties having the same parent node, the one non-leaf node which has two leaf nodes as children. An easier way of visualizing this tree is provided by figure 5.2.

For both of these access tree designs, tests were run for both encryption and decryption, using access policies of 1 to 100 attributes. The implementation was a CHARM example implementation of the original CPABE scheme [4], which supported both symmetric and asymmetric pairings. Using the CHARM benchmarking tools timing analysis, as well as operation count, including a breakdown of operations in each group, was performed. There was a single curve used for types A, D, E, F, and G, all of comparible security, 512 bit inputs for type A, with roughly 160 bits for G_1 and 320 bits for G_2 in the asymmetric pairings. Other curves supported by CHARM, such as a 1024 bit input type A curve, were also measured, but these results are not shown here since they are larger curves and thus their resulting operations were much more costly and any comparison would reflect as such.

5.6 Results

All tests performed in this section were run on a modern workstation, with Fedora 18 as the operating system, 16GB of RAM, and an AMD A10-5800K quad-core processor.

5.6.1 Operation Breakdown

With the different input groups being used for the operations, knowledge of the timing for internal operations within the individual groups would help find dominant mathematical operations, allowing for optimizations to be applied. Using the CHARM benchmarking suite, the time needed for elliptic curve multiplications, divisions, and exponentiations in the input groups was measured, as well as the same operations in the output groups and the pairing function. Operation counts for integer groups, or addition within the elliptic curve groups were not available, but a total operation count was provided, with the gap assumed to be integer operations. These results are summarized in table 5.1. An N/A output is used to denote that in the symmetric type A curve, the two input groups are the same, so the operations are also exactly the same. All times shown in this table are in seconds.

Additionally, there was found to be an error of around 4-5% between the minimum and maximum times for a single command, which can be attributed to the random numbers that were used for the different operations.

From this table, a few interesting observations are obtained. The first of which is that group multiplications are fastest in G_1 for types D, F, and G curves, but G_2 multiplication is much slower by a magnitude of 10 for D and G, and only double in type F curves. This is expected, as these curves have much smaller input groups than type A or type E curves, and the fact that G_2 is a larger group than G_1, so it would be expected that the operation time would reflect this.

For division within these curves, the same curve types were optimal for division operations within the two input groups, with the ratio of performance between G_1 and G_2 being almost the same. The main difference between multiplication and division occured when comparing the time for the operation within the output group G_T. For type A, D, and F curves, the division in G_T is approximately 10 times slower than multiplication in the same group. In pairings of type E or type G curves, the difference is much smaller, almost negligible difference in the type E case.

Exponentiation had much of the same trends between the two input groups, except in the case of type G curves, which were 100 times slower in G_2 than the same operation in G_1. The pairing operation found that pairings for the type A curves were fastest, and slowest for type F curves.

5.6.2 Key Generation

As seen in figure 5.3, the fastest key generation occured for the type F curves. Following that, type A curves were the fastest, followed by D, E, and G curves. Additionally, there were no changes in the number of operations when using different curve groups. If optimizations were made, these counts could change. Since they are the same, the data is easily

Curve	G_1 **Mult**	G_2 **Mult**	G_T **Mult**	G_1 **Div**	G_2 **Div**	G_T **Div**
A	6.290×10^{-6}	N/A	1.474×10^{-6}	6.146×10^{-6}	N/A	7.501×10^{-5}
D	2.558×10^{-6}	2.360×10^{-5}	6.992×10^{-6}	2.516×10^{-6}	2.360×10^{-5}	2.108×10^{-5}
E	1.505×10^{-5}	1.513×10^{-5}	1.154×10^{-6}	1.495×10^{-5}	1.521×10^{-6}	1.2336×10^{-6}
F	2.557×10^{-6}	5.358×10^{-6}	2.887×10^{-5}	2.510×10^{-6}	5.299×10^{-6}	1.331×10^{-4}
G	2.491×10^{-6}	4.747×10^{-5}	1.937×10^{-5}	2.570×10^{-6}	4.759×10^{-5}	4.529×10^{-5}
Curve	G_1 **Exp**	G_2 **Exp**	G_T **Exp**	**Pairing**		
A	1.700×10^{-3}	N/A	2.563×10^{-4}	1.173×10^{-3}		
D	5.895×10^{-4}	5.314×10^{-3}	1.100×10^{-3}	3.798×10^{-3}		
E	3.795×10^{-3}	3.794×10^{-3}	1.102×10^{-4}	4.448×10^{-3}		
F	5.780×10^{-4}	1.148×10^{-3}	5.198×10^{-3}	2.200×10^{-2}		
G	5.363×10^{-4}	1.028×10^{-2}	3.003×10^{-3}	1.148×10^{-2}		

Table 5.1: Time Needed for Individual Operations in Pairing Groups

comparable, and it was found that the type of operations that are used for key generation are best in type F.

An analysis of the operation count showed that for key generation of an n attribute key in CPABE, CHARM required $n + 1$ multiplications, 1 division, and $2(n + 1)$ exponentiations. All of the multiplications computed during this function were in G_2, while the exponentiation was almost evenly distributed between the two input groups. The divison occured outside of the elliptic curve groups, assumed to be in the integer field, for the computation of the exponent $(\alpha + r)/\beta$. Because of the fact that the multiplications are found to be in G_2 here, it would be likely advantageous to all curve types to switch the groups in which D_i and D_i' are a part of, to reflect the work proposed in section 5.3 as opposed to the input groups of the secret key elements from CHARM. This is due to the results from table 5.1, which shows that G_1 multiplications are no slower than multiplications in G_2. Note that this would also require to switch C_x and C_x' as well, due to the pairings within decrypt node needing to be elements of different input groups

Focusing on the three curve types which performed the best for this function, A, D, and F, using the operation breakdown an estimation of how much time is spent during exponentiation could be calculated. For type A curves, 46% of the time spent during key generation was for exponentiation, while type D only spent roughly 30 % of the time computing exponentiations during this same function. On the other hand, type F curves were dominated by this exponentation, and 86% of the key generation time could be attributed to these computations. This helps suggest locations for possible modifications and optimizations in order to minimize the times needed for this key generation.

5.6.3 Encryption

Using all of the supported curve types, the times needed to generate different length ciphertexts with a varying number of attributes was calculated for both horizontal and vertical encryptions. The horizontal encryption results can be seen in figure 5.4, while the vertical encryption results are shown in figure 5.5. The first point of note is that while the horizontal

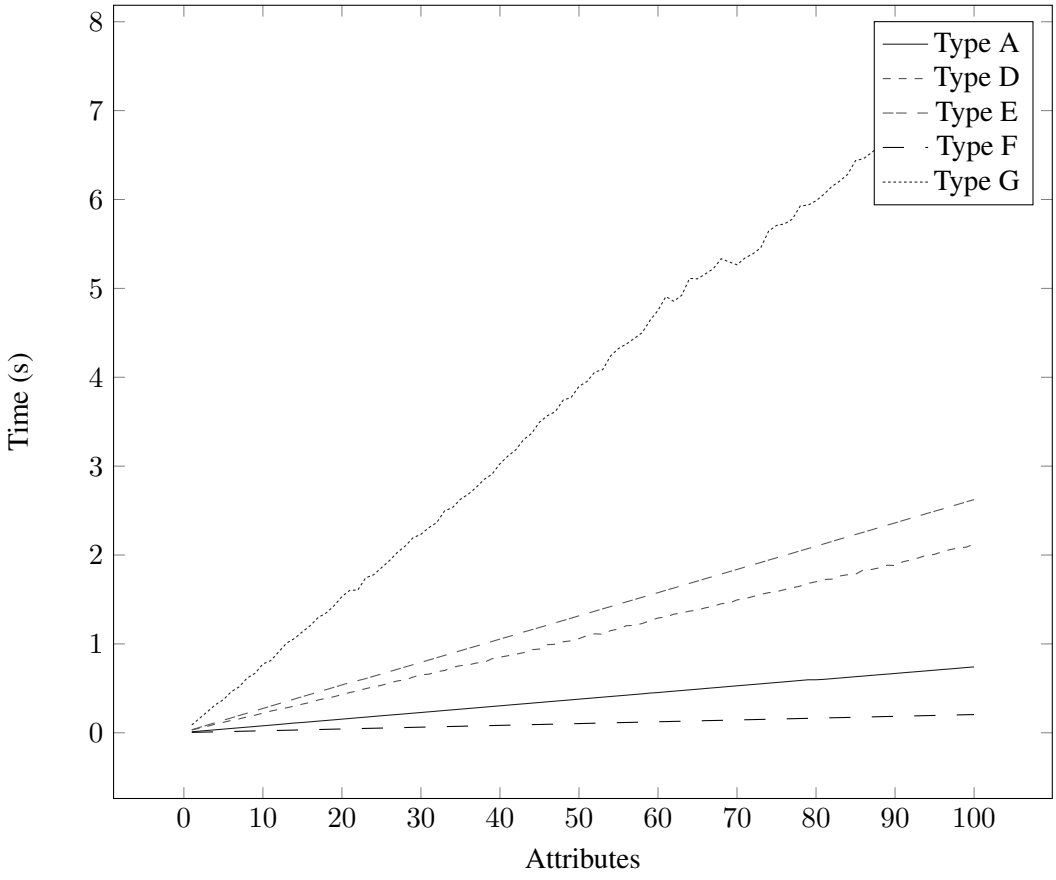

Figure 5.3: Time for Key Generation

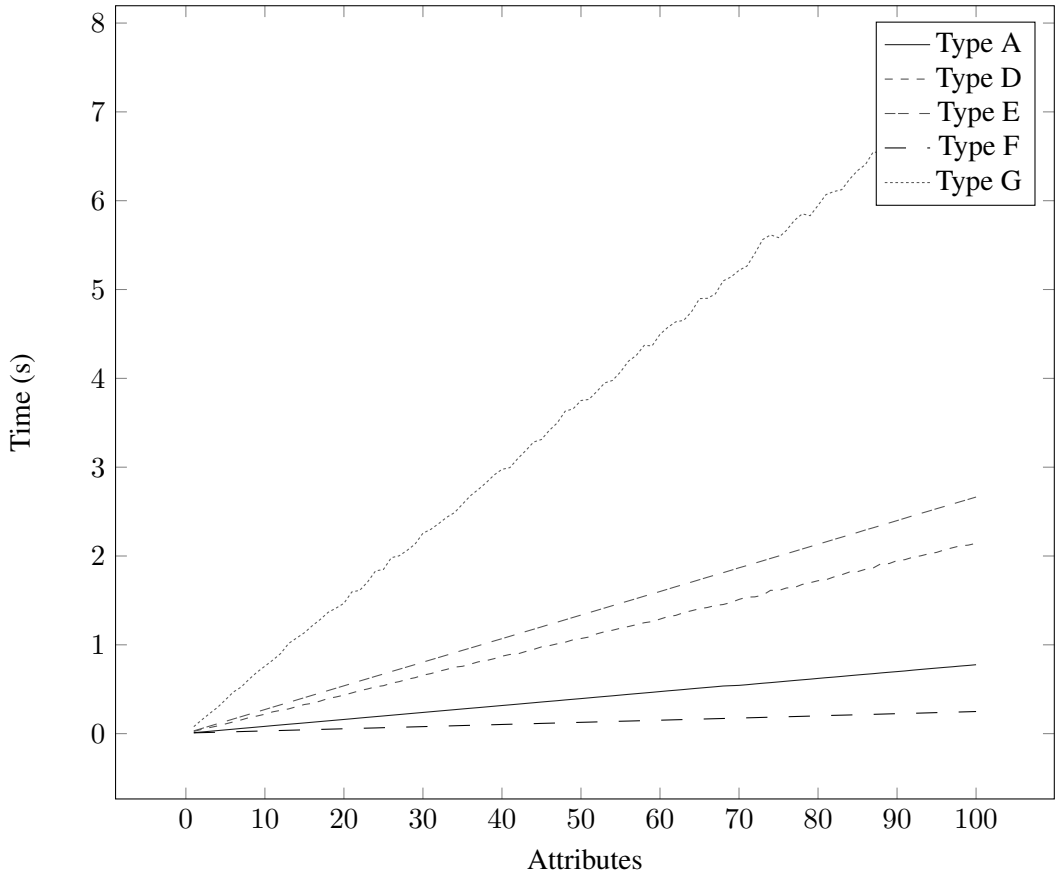

Figure 5.4: Time for Horizontal Encryption

encryption results were computed for up to 100 attributes, the vertical encryption results only go to 81 attributes. This is due to the fact that CHARM does not support access trees with a depth higher than 80 levels.

Similar to the key generation results, the fastest three curve types were type F, type A, and type D, in order. Types E and G were not as high performing as these three types, with type G taking up to 7 times as long as type A or F curves.

There were no major changes in performance between the horizontal and vertical encryption schemes. Almost all of the results were within 2-3% of the same curve for the same number of attributes, more than the difference that could occur just from different random numbers used as values to the multiplications and exponentations. However, what was unusual is that these differences were consistent, hinting that there may be some very minor differences in the underlying operations performed. For type A and D curves, the vertical encryption took slightly longer. For type F curves, instead the horizontal encryption took longer.

During the encryption of a message with n attributes in its access policy, it was found that $(6(n-1))+1)$ multiplications, and $2(n+1)$ exponentations were computed. Of

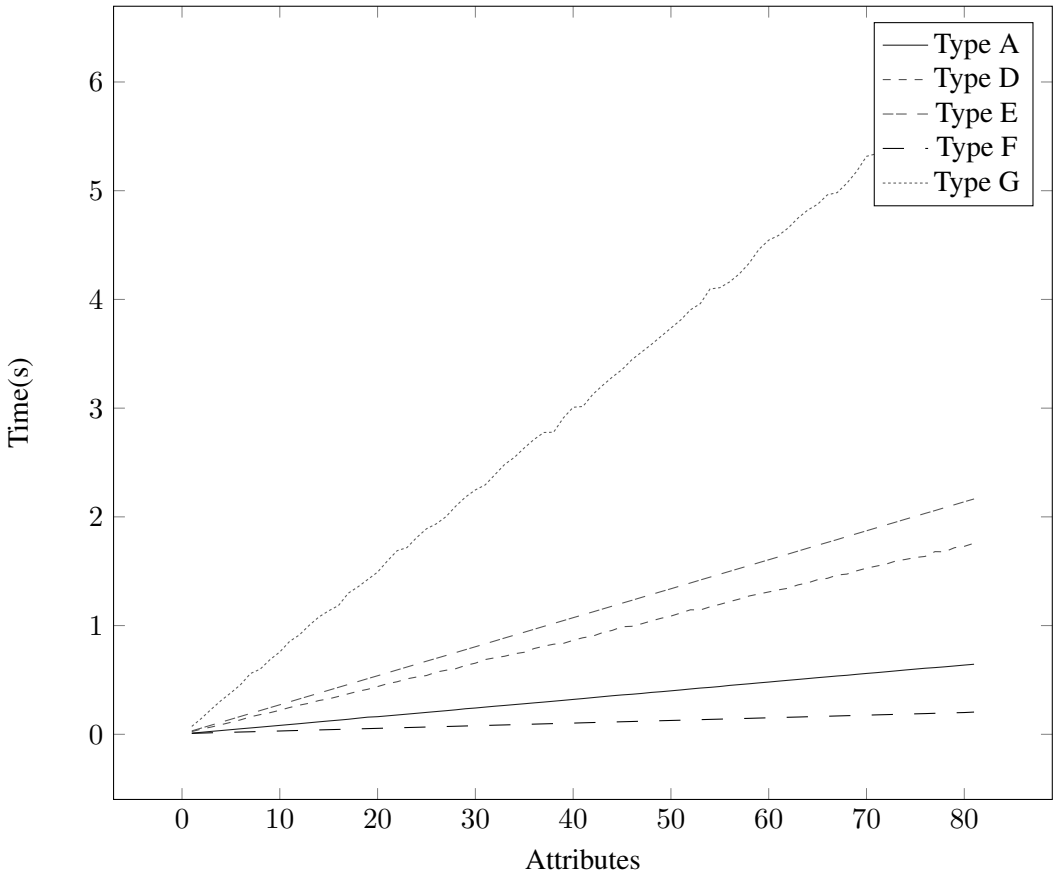

Figure 5.5: Time for Vertical Encryption

these multiplications, all except one were done within the integer fields. The one exception occured in the output group. On the other hand, the exponentations were mostly even between the two input groups, with again a single computation done in G_T. Again, using these operation counts and the timing for individual operations, it provided an estimate of 43% of the encryption time was computing exponentations for type A, and 27% for type D. Type F encryptions were also dominated by the exponentations, with 70% of the time needed to compute these operations for that curve type.

For these encryptions, a single block message was encrypted under different access trees, either horizontal or vertical. A single block consists of any random element of the output group G_T. If an actual file was to be encrypted, there would need to be some additional translation and mapping of binary files to an element of the output group. This would add some additional time for the encryption, but is assumed to be constant, as this has no dependency on the length of the access policy. There could be some differences between groups that have larger output groups, such as type F curves. Due to the fact that this is an efficient mapping, such as a simple binary translation and padding, it would be surprising if this makes type F curves require more time and perform worse than type A or D curves.

5.6.4 Decryption

The most notable observation from the decryption results is that while type F curves had the best performance for key generation and encryption, it had the worst performance for decryption, worse than even type E or G curves. The type A pairing performed best, followed by almost identical performance for type D and E curves. However, these curve types still took approximately three times as long as the type A curves. Type G curves did perform better than the type F curves, but still was five to six times slower than that of the fastest type A curves. The performance decrease for type F curves was tremendous and took over ten times the time needed to decrypt the same file in a type A curve.

Similar to the difference between horizontal and vertical encryptions, the decryption time for the different access trees was also very close to the other, with only minor differences between the two. However, these differences were a bit more pronouced, allowing up to 5% of a difference in some cases, but still minor compared to the differences between individual runs. As was the case with encryption, type A curves took longer for the vertical access trees, while types D and F curves took longer for the horizontal encryption.

An exploration into the operations needed for decryption helps explain why type F curves performed the worst in this function. For the decryption of a file with n attributes, the operations consisted of $3n$ divisions, $(5(n-1))+1$ multiplications, $n-1$ exponentiations, and $2n+1$ pairings. All of these operations occured in either the output group G_T, or the integer field. The fact that these operations were focused on the output groups is why type F took so long. Due to the fact that type F curves have an embedding degree of 12, the output group is much larger than the other curve types. In the curve used, the type F curve has an input group size of 160 bits, meaning that the output group size is $12 \cdot 160 = 1920$ bits. On the other hand, all other curve types have an output group size of approximately 1024 bits. Due to this large difference, it would be expected that all of the operations in

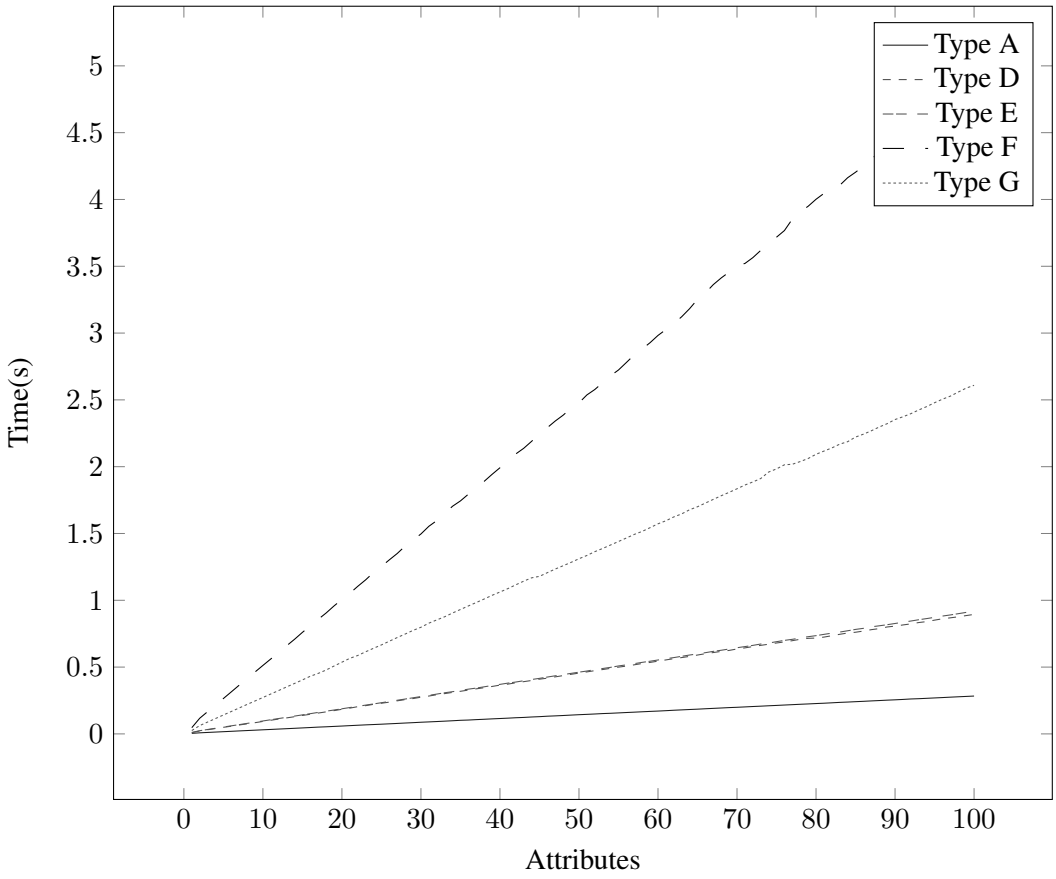

Figure 5.6: Time for Horizontal Decryption

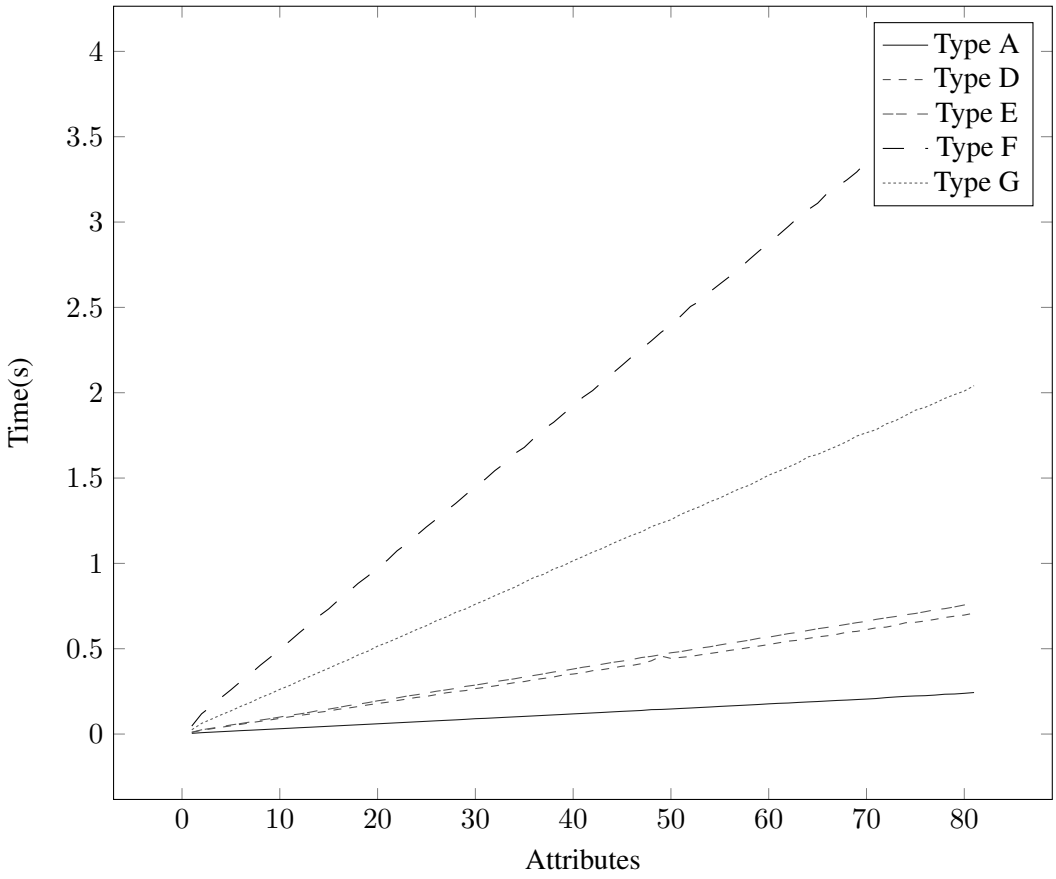

Figure 5.7: Time for Vertical Decryption

Attributes	Type A	Type D	Type F
1	1536	795	790
5	5632	2703	2686
10	10752	5088	5056
25	26112	12243	12166
50	51712	24168	24016
75	77312	36093	35866
100	102912	48018	47716

Table 5.2: Sample Key Sizes based on Number of Attributes

the output group would take much longer, and this likely explains the slow performance of type F curves for decryption. Note that for key generation and encryption, operations in the output group were extremely limited, so type F performed better. Additional research of operation breakdown showed that for all of the curve types, the decryption time was 80% spent on pairings, meaning that this would be the place to optimize. This is consistent with the results from 5.6.1, which had type F curves taking about ten times as long to perform a pairing operation compared to types A and D, and also had some of the slowest operations in G_T.

5.6.5 Size

Due to the fact that the CHARM library had consistent containers to hold all of the input and output elements, no detailed information regarding the size of keys and ciphertexts could be calculated through the performance tests. However, by using a theoretical approach based on the design of the asymmetric scheme and using the size of the input groups for the different curves, an estimation and comparison of the different types could be calculated. Due to the lackluster performance of type E and G curves overall, only the size of type A, D, and F curves were considered for this section.

The public key was the first item looked at for size. Unlike the secret keys and ciphertexts, there is no relation between attributes and the public key, so the public key would remain constant size. Using the group assignments from CHARM explained in 5.3, the public key had three elements of G_1, g_1, h, and f. The second generator g_2 is the only part of the public key from G_2, and there is a single element of G_T, $e(g_1, g_2)^\alpha$. The total size of the public key is then $3|G_1| + |G_2| + |G_T|$. With type A curves, this adds up to 3072 bits. Type D curves have a total public key size of 1749 bits, while type F has a total of 2686 bits.

On the other hand, secret keys in CPABE consisted of D, an element of G_2, as well of one element from each input group for every attribute in the key . Overall, the size of secret keys for a user possessing n attributes would be $|G_2| + n(|G_1| + |G_2|)$. Sample sizes based on a number of attributes are shown in table 5.2. As seen by these numbers, type A has very large keys, while type D and type F are roughly half the size, with type F being narrowly smaller than type D keys.

For the ciphertexts created in CPABE, each consists of one element of G_T, which is the

Attributes	Type A	Type D	Type F
1	2560	1590	2528
5	6656	3498	4424
10	11776	5883	6794
25	27136	13038	13904
50	52736	24963	25754
75	78336	36888	37604
100	103936	48813	49454

Table 5.3: Sample Ciphertext Sizes based on Number of Attributes

encrypted message, $\tilde{C}$. On top of that, there is a single value C, which is an element of G_1. Similar to the secret keys, there is one element from each input group used for every attribute in the access policy of the ciphertext. This totals $|G_T|+|G_1|+n(|G_1|+|G_2|)$ for the entire ciphertext. There is also additional overhead needed, such as how to store the access policy and trees, but that is assumed to be constant and will not vary at all for the different curve types. A sample size for ciphertexts with a varying number of leaf nodes/attributes is shown in table 5.3. For ciphertexts with a very small number of attributes, types A and F have comparible sizes, but type F curves do grow at a much slower rate, and are comparable to type D curves for a large number of attributes. Type D does have the smallest ciphertexts, again less than half that of the type A equivalents. These size differences are due to the smaller input groups that the asymmetric curves have compared to the 512 bit type A input groups.

5.7 Summary

Overall, it was found that the performance of type A, D, and F curves were acceptable for the usage of CPABE, while type E and G curves were slower and unlikely to be recommended. For key generation and encryption, type F curves performed the best, but these same curves also had the worst performance for decryption due to the larger output group size. Based on the needs of the application, the usage of type F curves is plausible for CPABE, as the key generation and encryption is fast, but decryption is much slower. Due to this slow decryption, type F curves would not be recommended for many applications.

For the application of CPABE, a breakdown of the various operations showed that for encryption and key generation, improvements could be made by focusing on the dominant operations, which for both was exponentiations in types A and F, and multiplications of type D. Further improvements for key generation could be achieved by switching the input groups of some of the elements, which was shown in section 5.3, but not used in the CHARM library in which the tests were performed. For decryption, the dominant function was pairings, and all curve types would be improved by optimizations to the pairing operation. Additionally, due to this fact that exponentiations were not the limiting factor for type D curves in key generation and encryptions, it is plausible that an extremely efficient multiplication operation for this curve type would provide some of the best performance.

The size of both keys and ciphertexts were greatly minimized by the usage of type D or F curves over that of type A curves. The smallest private keys were obtained by the usage of a type F curve, and the smallest ciphertexts overall were from type D curves.

Chapter 6

Key Management

The initial CPABE scheme has issues with the incorporation of key management. Because of this, it is inherently difficult to revoke a key and prevent a user from subsequently using this key. Additionally, this also requires some method of allowing key renewal, for either the entire key, or individual attributes.

By default, CPABE assumes that all files encrypted will be able to be placed anywhere, such as a non-secure server. By the properties of the scheme itself, these files are claimed to be secure and only users with keys that satisfy the access tree are able to decrypt these files. Unlike other cryptographic schemes, CPABE is similar to IBE in that each user has a unique key, so issues with key sharing are not as important. But what happens when a user is no longer a member of the organization using CPABE? By default, the user would still have a valid key, and could subsequently decrypt and files, even ones that they are no longer authorized to do so. This applies indefinitely, and future encrypted files are still valid as no part of CPABE requires a user to delete or validate their keys when decrypting, as would happen if a trusted server was used. This is why such key management strategies are needed.

This chapter begins with a short introduction to key revocation and renewal, and then discusses what properties are desired in CPABE in order to incorporate these features. Then several previous works are presented, with the advantages and disadvantages of each. Using these works as a guideline, a new set of assumptions are designed. The new work, a key insulation scheme for CPABE, is shown, as well as a theoretical performance analysis of this new scheme. This chapter then finishes with another brief discussion of how these ideas can be used in order to revoke or renew individual attributes, but for the most part the focus is on key management for full keys of users.

6.1 Definitions

6.1.1 Revocation

Key Revocation is the act of preventing any future usage of a key towards decrypting ciphertexts within an encryption scheme. For CPABE, this could either be in the form of revoking an entire user's key, or a specific property of a user's key. The initial CPABE implementation [4] explains that one can use an expiration date for each property, but it still presents the challenge of both time synchronization, and the fact that a malicious user with

a revoked key would be able to circumvent this by manipulating the time on the system used for decryption. In order to prevent this malicious access from occuring, one would need to coordinate with a trusted server, which goes against the wishes of CPABE. Additionally, this would require approximately three times the number of attribute gates in the access tree, which also increases key and ciphertext sizes, which are already quite large when using a large number of attributes.

Other examples of Key Management are used for the public key infrastructure (PKI) that allows many cryptographic communications over unsecured channels, such as the Internet. The standard for PKI key management is X.509 [17], which uses a hierarchy of trusted authories to generate and present certificates to users. X.509 defines revocation as a list of certificates which are revoked and no longer valid. An issue with this type of policy in CPABE is the fact that users are the one who make the choice whether to listen to the revocation list, and not servers. In X.509, a server (should) always reject revoked certificates and assume they are coming from malicious users. However, since in CPABE the malicious user still controls their own key, and no access is required to get the ciphertext, they are the only ones preventing themselves from being able to use their key to decrypt the file.

6.1.2 Renewal

Renewal of keys will need to be performed when a user either has an attribute changed, or after a certain time period, in which case the entire key will need to be renewed. In some systems, just the renewal of a specific attribute may be performed, but this is not the case for CPABE. In order to prevent collusion between users with different attributes, the attributes of the same key must all be linked to not work for decryption with attributes of another key. A possible workaround for this is the fact that the key generation keeps tracks of the current attributes every user possesses, and generating portions of the key compatible with the user's current variation of their key.

A straightforward usage of CPABE would have the key renewal performed by simply running the original key generation algorithm, performed exactly the same way as in Section 2.3, after a specified expiration period. The difficulty with this is there is much overhead required, as it is unlikely that every user will have an attribute changed every time period. This can also be done on demand, however, and having a user voluntarily update their private key every time one of their attributes is updated. This is based on trust, and in the case that the user loses an attribute they possessed, they could still keep their old key and use it to decrypt messages that require their revoked attribute. In this case, a method of revocation must also be presented in order to prevent the user from decrypting messages with their previous key(s).

6.2 Desired Properties

In a CPABE scheme with key management, the desired properties must first be discussed in order to measure the ability of the key management to meet these properties. Different

key management ideas will only work for some kinds of server and cryptographic schemes.

Ideally, a CPABE scheme supporting key management allows

1. No requirement for trusted servers for the decryption of files. This includes proxy servers that provide users with parts of a key to be used at decryption time. If any communication is required during decryption, it would ideally be decentralized. Additionally, this restriction also implies that the encrypted files will be placed anywhere, and not a trusted server.

2. Minimal overhead during key revocation/renewal. This means that there should not be a large list of different values sent to mark revoked keys, and that upon renewal smaller values are sent rather than a brand new key. This is due to the fact that keys are very large at times, and requiring renewal from many users may cause an issue, especially if there is only a single key generation server.

3. Immediate revocation of a user's key(s).

6.3 Supporting Work

6.3.1 PIRATTE

One previous implementation for key management in CPABE was provided by the PIRATTE scheme designed by Jahid and Borisov [20]. This work was important because it allowed revocation at the cost of an additional pairing, while working with asymmetric elliptic curve pairings. The revocation provided by this scheme was allowed by the usage of a proxy which gives a third part of the user's key for every attribute, or C''_x. When the user attempts to decrypt a file, the user is given this value by the proxy, and this allows the decrypt node function to provide the correct value.

The proxy key in this scheme is a controlled secret sharing scheme. Upon initial setup, the proxy key is just t random points that blind a secret value forming a $t-1$ degree polynomial , where t is the maximum number of users that can be revoked at a single time. Each user is given a value that blinds only one of these random shares, the coefficient $P(0)$. The main change occurs during the key generation step, where the user's private key is blinded by an exponentiation by $P(0)$. Revocation occurs by giving the proxy a list of users which need to be revoked. Based on this, the values of the t points used for secret sharing are modified for these revoked users, preventing them from being able to subsequently compute the value of $P(0)$.

The issue with the usage of this scheme is the fact that performance suffers greatly, from both the usage of asymmetric pairings as well as the introduction of an additional pairing needed for every leaf node in the access tree. A performance analysis done by the authors of this paper found that in addition to the overhead of using a type D curve for CPABE, the additional pairing required adds a much longer length for the decryption operation, which for a larger amount of attributes was up to a 400% overhead in total decryption time. Additionally, the extra exponentiation needed for key generation added 20-30 % more time

for the function over the same key generation of type D pairings. Because of this, and the need for proxy server communication, it was not further explored in this thesis.

6.3.2 Broadcast Revocation

Two different schemes supporting revocation of keys for recipients of broadcast methods have been provided by Du *et al.* [11] and Zhou [35]. While the fact that they are for broadcast messages makes it such that they don't match the goals for complete revocation of files, they still provide revocation techniques for an ABE scheme, and some may be applicable to the goals of this thesis. The work of Du *et al.* supports revocation for KPABE by associating users with various user groups, and encrypting files aimed towards specific user groups using a linear secret sharing scheme. By encrypting based on these user groups, revocation occurs by updating the groups to remove a user. When a user is removed, they no longer belong to the new user group, and because of this, they will not match up with the attributes needed to decrypt the key. An advantage of this scheme is that everything is updated solely with the third parties, transparent to users. The disadvantage is that this is a broadcast scheme which assumes there is a single central figure who sends all messages that are being broadcast, effectively making it a one-way scheme. Another issue is that this scheme assumes every attribute will be valid for multiple users, which may not always be the case.

Zhou provided a similar method for broadcast revocation which supports revocation of ABE keys while retaining minimal overhead for both storage and communications. Unlike the work from Du above, this work does support many-to-many communications where any user can broadcast to multiple recipients. This is provided in the scheme by having a group controller for each group or attribute within the scheme. When a member joins a group, they are linked a unique ID number, each bit associated with a specific secret. By having these unique secrets and random to each group controller, collusion between members of different groups is prohibited. By having the contoller know every unique ID, they can combine these IDs through boolean expressions when a user is revoked, and give updates to the remaining users. Valid users must use these secrets to in turn combine find the group key, which is used to decrypt broadcast messages intended for that group. Through this scheme, immediate broadcast key revocation is obtained with low complexity communication and key storage.

6.3.3 Dynamic Revocation

One method of providing key revocation was supplied by Xu and Martin [34], which allows the revocation of keys without requiring any modifications to ciphertexts or other keys. Similar to PIRATTE, this scheme uses a proxy which is supplied with an additional part of the keys, and uses this part of the key at decryption through an additional pairing. Like other schemes, secret sharing is used to convert the user's share with the values from the proxy in order to get the proper value to compute the pairing. Overall, this method is not compatible with the goals of this thesis due to the fact that it is based around storage-centric environments, which already handle some issues with key management by having files on

a trusted server. The keys revoked through this are revoking the access from the trusted servers, not local keys from users.

6.3.4 User Blacklist

Maas's thesis [24] discusses the equivalent of PKI blacklists to be used in an IBE scheme. With this, revocation of users' keys can be done by publicly noting the revoked keys. Due to the fact that an IBE scheme has two keys, verification of public keys can be done for revoked users, and revoked users will be unable to change their keys. However, while this initially may seem promising, ABE schemes do not have multiple keys per users, and this means there is not as much of a direct translation into a CPABE context. An additional restriction is the fact that these revoked keys are still valid for previously sent messages, only blocking messages that would be sent after revocation occurs. A final difficulty with such a scheme is the fact that there is a requirement to have some infrastructure for such communication of the blacklist and ensuring that the users receive a valid blacklist from a trusted server.

6.3.5 Hierarchical Access Structure

One option of a key management scheme that could be adapted for CPABE is the work of Li [21], which discusses a hierachical access structure for power systems such as utilities. This original work was not designed for any ABE scheme, but the relationships provided in it between keys and attributes is close to the design of KPABE. In this work, one or more users is given a specified role. The users' access privileges to ciphertexts or servers are controlled by the possession of these roles. These roles additionally form a hierarchy of various security classes that allow an inheritance relationship. The inheritance occurs through trusted servers, using one-way functions to pass on keys granting access from the initial role owners to other users. The usage of this in CPABE is discussed in section 6.4.1.

6.3.6 Distributed Multiple Authority CPABE

As part of Servos's work discussing the application of CPABE to health services [30], the possibility of using a distributed multiple authority ciphertext-policy shared attribute-based encryption (DMACPSABE) scheme is presented. Through this scheme, CPABE is adapted to use multiple key generation authorities, each granting a subset of attributes to the users. Additionally, this work allows for NOT boolean operators to be incorporated into access structures. In this situation, revocation is provided by these NOT operations during encryption time. While this does work, it is not ideal as the requirement of preventing access is to be done by the entity encrypting the file, and it adds more attributes which in turn decreases performance of the encryption and decryption. This overhead is also added to the keys, and some keys are incredibly large in this situation, taking up to 50MB total. This may work in the application of health systems, but for generic usage of CPABE based off the properties originally desired, wanting no centralized server outside of key generation, this may prove difficult.

6.3.7 Key-Insulated ABE

One method that allows revocation for an ABE scheme is provided by Chen *et al.* [8], which provided a key-insulated approach. Unlike previous works discussed, this method does not provide immediate revocation, but instead uses temporal keys that are valid for a specific value. Once the period ends, if a user's key is revoked, their key will not work for messages encrypted in future time periods. The goal of this scheme was to minimize the work done by the authority in renewing keys, by using a helper key to generate a single value, and giving this value to the users. Because of the minimal overhead needed for this renewal process, and the fact that it is not required during the actual decryption step, these helpers which produce the renewed values can be redundant and distributed.

Key revocation in this scheme is provided by the same mechanism that allows key renewal. If a user's key is revoked, they will not get a renewed value for the next time period. This value is generated by a pseudorandom function which is seeded for every user. During each time period, the next pseudorandom value will be generated for each user, and provided to them. The security of this method is dependent upon the inability for a malicious user to compute the next pseudorandom value, and ensuring that all renewal authorities have a synchronized revocation list. A detailed derivation of the functions for KIABE is shown in section 6.4.2.

6.3.8 Attribute Revocation

Ibraimi *et al.* [18] provides an option for attribute revocation within a CPABE scheme. However, this mediated CPABE scheme does not provide much functionality with regards to revocation, as it is not immediate, requiring users to wait until a time period ends. The revocation is performed by requiring a mediator to hold each half of the user's key for each attribute. These two shares are combined after decryption, with both the user and the mediator decrypting using their shares, giving a single decrypted message. An advantage of this scheme is that it does allow the mediators to be distributed, not requiring a single proxy server. However, each proxy still has to contain the same shares of every user's key, and a single compromised server can still give a malicious user the half of every key. Due to the fact that this only provides revocation of single attributes, and the fact that it is not immediate, it is not a route worth pursuing for the key management wanted for this thesis.

Another scheme providing attribute revocation is provided by Qiuxin and Miao [28]. In this scheme, linear secret sharing matrices are used as a means of creating semi-functional keys and ciphertexts. One tradeoff of using such matrices are that it requires a small universe scheme, in which all attributes used must be known during the setup step, and if additional attributes need to be subsequently added, all keys are no longer valid. The revocation works by simply managing a list of attributes which need to be revoked. If an attribute is revoked, then during decryption, ciphertexts requiring any of the revoked attributes are ignored, and the threshold for the associated access tree, and allowing decryption without that. However, this revocation is unrelated to the goals aimed for earlier in this chapter, and this makes it difficult for full key revocation, as there is no restriction that prevents any user key from being used still.

6.4 Proposed Changes

6.4.1 Hierarchical Access Structure

A possible translation of the hierarchical access structure, presented in Li's work [21], to CPABE would be to allow different users to form a hierarchy for various attributes, and using these as ways to verify or guarantee user possession of an attribute, while handling key revocation for that specific attribute. By using this, any attacks or collusion would only provide a single attribute, and collision would still be difficult due to the hierarchy and different "gatekeepers" of the attributes, preventing the malicious user from knowing which user possesses the role they are trying to collude. Each gatekeeper would be in charge of communicating with the servers that provide revocation, and allow a method to verify a user's identity, and then giving them a token that can be used for decryption. While this method does require that a user communicates with multiple entities during decryption, these are decentralized and unpredictable, reducing the risk of having a single central authority being compromised. The hierarchy is retained by having the delegation function used to produce these values. The difference is this delegation is orchestrated by the central authority, and not initiated by the user producing the delegate key. This scheme works by splitting one of the values for each atrribute in the key, D_j and D'_j, and one of these is the value that is given to the user upon validation to the gatekeeper.

While this looks promising, there are issues with both communication with other general users in the system, as well as collusion resistance. The first issue is regarding anonymity and the fact that the gatekeepers would know various users who posses the same attributes. In some applications, this is a security issue itself. A workaround for this would be to have generic anonymous references to the users communicating, but would still be possbile to track the communications. The other main issue is the collusion resistance and how that impacts the storage required for these values. Because of the possibility of collusion, the values users are given from the gatekeepers must be unique for each user. If not, users could collude to reuse the values between them. However, these values must also be unique in order to allow revocation, as the same value could be kept by the users and reused after their key is no longer supposed to be valid. This in turn requires that there would be a unique value given for each user and each ciphertext, which gives the gatekeeper a large amount of information such as the number of ciphertexts requiring each attribute, and requires a large storage space for each of these values, one for each user/ciphertext pair.

Another possible solution involves using temporal keys, simular to the key insulation approach. This method uses a series of temporal key values, one given to the gatekeeper of each attribute. Upon decryption, the shares given from each are combined to get a specific value, which is also used for decryption. This does not work, however, for OR gates in the access tree, as secret sharing is designed for AND or threshold gates. Additionally, this method is much more inefficient to the proposal from section 6.4.2, due to the communication overhead.

6.4.2 Key Insulated CPABE

KIABE

Originally provided by Chen *et al.*; the original Key-Insulated ABE (KIABE) scheme defines the following methods as part of the scheme.

Setup This function generates the parameters needed for the rest of the functions. In addition to the selection of the initial pairing group, a pseudorandom function $F_\omega(x)$ must be defined. This function's seeds will be unique to each user, with x representing the current time period in which an output is requested. ω is used within the rest of this scheme to represent a user's identity and attribute set as a single variable.

$$
\begin{gathered}
g = \text{ Generator of } G_1 \\
y \in Z_p \\
g_2, h_1, v_1, ...v_{n+1} \in G_1 \\
g_1 = g^y \\
V(x) = g_2^{x_n} g^{h(x)} \\
H_w(x) = g_1^x h_1 \\
PK = (g_1, g_2, h_1, v_1, ...v_n) \\
MK = y
\end{gathered}
$$

Key Generation Key generation is defined by the following equations. After this function is completed, the value HK is given to the helper, and TK is given to the user.

$$
\begin{gathered}
HK_\omega = \{0,1\}^\kappa \\
k_{\omega,0} = F_{HK_\omega}(0) \\
q(x) = d-1 \text{ degree polynomial such that } q(0) = y \\
TK_{\omega,0} = (g^{k_{\omega,0}}, \forall i \in \omega : \{g_2^{q(i)} V(i)^{r_i} H_\omega(0)^{k_{\omega,0}}, g^{r_i}\})
\end{gathered}
$$

κ is the length of the seed required by the pseudorandom function, so the initial HK value is set to a κ bit string unique to the the user. One possibility for creating this would to be a hashed value of the user's idenity.

Helper Update At the expiration of the time period $t-1$, the helper needs to generate a new value UI which is valid for the next time period t for each user. These values are only generated and given to the user if their keys are still valid. Revoked users will not get an updated UI value for time period t, and subsequently their keys will no longer provide the

ability to decrypt messages.

$$k_{\omega,t} = F_{HK_\omega}(t), k_{\omega,t-1} = F_{HK_\omega}(t-1)$$
$$UI_{\omega,t-1,t} = (H_\omega(t)^{k_{\omega,t}}/H_\omega(t-1)^{k_{\omega,t-1}}, g^{k_{\omega,t}})$$

User Update Once given the updated value UI for the time period t, the user can update their temporary key TK to be valid for this same time period. Note that the values used for the attribute objects will cancel out, leaving the key in a format similar to that created during key generation, but valid for time period t instead of time period 0.

$$TK_{\omega,t} = (g^{k_{\omega,t}}, \forall i \in \omega :$$
$$\{g_2^{q(i)} V(i)^{r_i} H_\omega(t-1)^{k_{\omega,t-1}} \cdot H_\omega(t)^{k_{\omega,t}}/H_\omega(t-1)^{k_{\omega,t-1}}, g^{r_i}\})$$

Encryption Encryption is performed on a message M valid for time period t with attribute set ω' as such. Note that ω' is defined that the message can be decrypted if and only if the user with attribute set ω posseses at least d of the same attributes required by ω'.

$$s \in Z_p$$
$$E = (t, \omega', E' = M \cdot e(g_1, g_2)^s, E'' = g^s, E''' = H_\omega(t)^s, \forall i \in \omega' : E_i = V(i)^s)$$

Decryption Decryption of the ciphertext E by the user with attribute set / identity ω is done as such.

$$S = \omega \cap \omega' \text{ such that } |\omega \cap \omega'| \geq d$$
$$M = E' \prod_{i \in S} \left(\frac{e(g^{r_i}, E_i) e(g^{k_{\omega,t}}, E''')}{e(g_2^{q(i)} V(i)^{r_i} H_\omega(t)^{k_{\omega,t}}, E'')} \right)$$
$$= M \cdot e(g_1, g_2)^s \prod_{i \in S} \left(\frac{e(g^{r_i}, V(i)^s) e(g^{k_{\omega,t}}, H_\omega(t)^s)}{e(g_2^{q(i)} V(i)^{r_i} H_\omega(t)^{k_{\omega,t}}, g^s)} \right)$$
$$= M \cdot e(g_1, g_2)^s \prod_{i \in S} \left(\frac{e(g^{r_i}, V(i)^s) e(g^{k_{\omega,t}}, H_\omega(t)^s)}{e(g_2^{q(i)}, g^s) e(V(i)^{r_i}, g^s) e(H_\omega(t)^{k_{\omega,t}}, g^s)} \right)$$
$$= M \cdot e(g_1, g_2)^s \prod_{i \in S} \left(\frac{1}{e(g, g_2)^{q(i)s}} \right)$$
$$= M \frac{e(g_1, g_2)^s}{e(g, g_2)^{ys}} = M \frac{e(g^y, g_2)^s}{e(g, g_2)^{ys}} = M$$

KICPABE

A KICPABE scheme is now presented. The KIABE scheme was modified in order to use the temporal keys in both the decrypt node and final encryption functions, with no change needed during the interpolation at non-leaf nodes. Note that in CPABE, each user has a random value r that is used to blind the values until these interpolation functions. This is where the temporal keys for KIABE are used, and the r value is replaced by the k_ω value.

Similar to CPABE and KIABE, the following functions are defined as part of this scheme. These definitions are based upon a symmetric CPABE scheme, but using the modifications from chapter 5 asymmetric pairing groups can also be used.

Setup The setup in this case is exactly the same as the original CPABE scheme. The only difference is that the pseudorandom functions as well as the one-way functions needed for the keys and ciphertext values must be defined as well.

$$PK = \mathbb{G}_0, g, h = g^\beta, f = g^{1/\beta}, e(g,g)^\alpha$$
$$MK = (\beta, g^\alpha)$$

Key Generation As mentioned, F_{seed} is a pseudorandom function, the same that was used in the original KIABE scheme, with the seed being a unique κ bit representation of the user. H is the same hash function from CPABE that maps attribute names to elements of G_1. The function T is also a one-way hashing function, and may be either the same or a different function, mapping instead integers representing the time period to elements of G_1.

$$HK_\omega = \{0,1\}^\kappa$$
$$k_{\omega,0} = F_{HK_\omega}(0)$$
$$TK_0 = (D = g^{(\alpha + k_{\omega,0})/\beta},$$
$$\forall j \in \mathbb{S} : D_j = g^{k_{\omega,0}} \cdot H(i)^{r_i} \cdot T(0), D'_j = g^{r_j})$$

Helper Update Just as in the KIABE scheme, the goal of this function is to have the helper generate a value which will be given to user ω to update their key from time period $t-1$ to time period t.

$$UI_{\omega,t-1,t} = (g^{k_{\omega,t}}, g^{(\alpha + k_{\omega,t})/\beta}, T(t)/T(t-1))$$

One point of note is the contrast between this update function and that of the KIABE scheme. In the KIABE, the update function exponentiates by the $k_{\omega,t}$ value for each time period. This is not done here due to how the decryption is setup. The reasoning behind this is explained in the summary of that function. One point is that this may allow malicious users to discover the values output from T, but that is not an issue due to the fact that these are only part of the mask used for the time period, and the r_i value is still needed to produce an advantage.

An additional difference is the fact that there is a third value given to the user opposed to the KIABE version. This is due to the fact that with the replacement of the value r with thw k value makes it such that the user's key requires both to properly update - the first for the D_j value, the second for the D value. Given only one of the k value or g^k, users are not able to properly update their D value for the new time period, as β is a hidden value part of the secret key.

User Update There are no changes to the purpose or design of this function compared to the KIABE version.

$$TK_t = (D = g^{(\alpha + k_{\omega,t})/\beta},$$
$$\forall j \in \mathbb{S} : D_j = g^{k_{\omega,t}} \cdot H(j)^{r_j} \cdot T(t-1) \cdot T(t)/T(t-1), D'_j = g^{r_j})$$

Encryption Again there are no functional changes to the usage of this outside of the adaptation from the KIABE version.

$$s \in Z_p$$
$$q_R(0) = s$$
$$CT = (t, \tau, \tilde{C} = M \cdot e(g,g)^{\alpha s}, C = h^s,$$
$$\forall\, y \in Y : C_y = g^{q_y(0)}, C'_y = H(y)^{q_y(0)} \cdot T(t))$$

Decryption The first step is the decrypt node function. Since the r value per user was replaced with k, that is desired in the output from this function. This is also where the H or T function outputs in the helper update can not be exponentiated by k, due to the fact that the setup of decrypt node requires that these values cancel out, which would leave another value exponentiated by k. In order to cancel out properly, the D'_i value would be also needed to be exponentiated, which in turn would add many more values to the update function. Exponentiation in the C_z values can not occur as these are unique to the ciphertext while the k values are unique to the user.

$$DecryptNode(CT, SK, z) = \frac{e(D_i, C_z)}{e(D'_i, C'_z)} = \frac{e(g^{k_{\omega,t}} \cdot H(i)^{r_i} \cdot T(t), g^{q_z(0)})}{e(g^{r_i}, H(i)^{q_z(0)} \cdot T(t))}$$
$$= \frac{e(g^{k_{\omega,t}}, g^{q_z(0)}) \cdot e(H(i)^{r_i} \cdot T(t), g^{q_z(0)})}{e(g^{r_i}, H(i)^{q_z(0)} \cdot T(t))}$$
$$= e(g,g)^{q_z(0) \cdot k_{\omega,t}}$$

For interpolation during the decryption, the same as CPABE is performed here, but instead producing the value $A = e(g,g)^{k_{\omega,t} \cdot q_R(0)} = e(g,g)^{k_{\omega,t} \cdot s}$ due to the usage of k instead of r.

Given this value A from the interpolation of the root node, the final decryption occurs as follows.

$$\frac{\tilde{C}}{\frac{e(C,D)}{A}} = \frac{\tilde{C}}{\frac{e(h^s, g^{(\alpha+k_{\omega,t})/\beta})}{e(g,g)^{s\cdot k_{\omega,t}}}} = \frac{M\cdot e(g,g)^{\alpha s}}{\frac{e(g,g)^{\beta\cdot s\cdot\frac{\alpha+k_{\omega,t}}{\beta}}}{e(g,g)^{s\cdot k_{\omega,t}}}} = \frac{M\cdot e(g,g)^{\alpha s}}{\frac{e(g,g)^{\alpha\cdot s}\cdot e(g,g)^{s\cdot k_{\omega,t}}}{e(g,g)^{s\cdot k_{\omega,t}}}} = M$$

6.5 Performance

6.5.1 Speed

The design of the key insulation function was made as such to minimize any extra pairings and exponentiations during decryption time. Because of the fact that keys are updated externally to the decryption, there are no additional operations required for decryption over that of the original CPABE scheme. It would be expected that the time required for this decryption would match that obtained from the previous chapter for decryptions of the same group used.

For encryption and key generation, there are a couple extra multiplications that must be done in order to account for the T values. The operation breakdown from the previous chapter showed that the group multiplication is on the order of 10^{-6} seconds, which also was equivalent to an additional attribute in key generation, and less due to the number of multiplications already used for encryption. On top of this, there are the one-way and pseudorandom functions that need to be computed, but these are again expected to take much less time than the other operations needed.

During the helper and user update functions, one must look into the time required for them. The helper update takes two exponentiations in an input group, as well as two integer divisions. The exponentiations are a limiting factor here, espeically when taking into account that this function must be computed for every user at the end of every time period. Adjusting the time period to longer values would reduce this load, but at the cost of security as invalid keys would take longer to be revoked. One note is that the key insulation does allow these helpers to be distributed to minimize the work each helper has to do. Additionally, there is no work for the helper to be doing in between the expiration of these time periods, meaning it could precompute the values as needed, and only distribute them at the beginning of the new time period. Upon receipt of the update, the user needs to perform n G_1 multiplications, one for every D_j value, where n is the number of attributes in the user's key.

6.5.2 Size

Again, because of the design of the scheme, there are no modifications to the size of keys or ciphertexts produced by KICPABE. This is because every multiplication of values from T still reduces to elements within the same input group as previous.

For the update value sent, this consists only of 3 elements of G_1. This is again on a per user basis, and is smaller here if nonsymmetric pairings are chosen. This is as most 1536

bits needed, assuming that a type A curve is used here. This should not create much of a network bottleneck unless hundreds of thousands of users are in the scheme.

The size restriction comes into play when one looks at how users handle decrypting files from multiple time periods. One assumption here can be made that users will toss out all keys once that time period expires. This then means that users are unable to decrypt previous files, as they were encrypted during previous time periods. The other option here is to require that users keep all previous keys, at which point multiple time periods would require a large burden of storage for every user.

6.5.3 Disadvantages

As mentioned above, there is an issue when determining how to handle storage of keys for multiple time periods. Either the users can keep previous keys, which require a large storage overhead, or they are tossed, preventing the users from decrypting any ciphertexts from previous time periods.

One work around for the storage would be the assumption that users will automatically decrypt any file they can as soon as it is encrypted. This would reduce the storage needed, but at the same time also require that users instead store plaintext files, which may cause an issue because if they are compromised then the attackers would get access to each of the plaintexts as well. User compromise is not covered here because it would be assumed that the attacker could also get the keys and subsequently decrypt the files. This could be avoided by using some guarantee that there is a two-factor authentication needed to access the keys to decrypt, but this same issue is plausible with even the original CPABE scheme.

On the other hand, even if a user keeps all previous keys, there needs to be some method of handling users who enter the system after the initial setup. By the design of this, either their initial key becomes huge, as they need to get all previous keys as well, or they are only starting with a key valid for the current time period, preventing them from accessing previously encrypted files.

One option is to require that after a specified number of time periods, the user tosses out all keys except the n most recent, and all ciphertexts encrypted before the last n time periods are reencrypted. This adds additional issues, such as how to handle files that no users are able to decrypt, or dealing with the large amount of files that all much be encrypted again at once.

This issue is nonexistent depending on the application. For example, if there was to be a broadcast application, which the recipients either immediately decrypt, or ignore and don't care for ever decrypting, there is no need to store previous key values.

6.5.4 Security

For security of the key insulated CPABE scheme, it is important to look at the values that must be determined for a malicious user to use a revoked key. The main difference between CPABE and KICPABE is the modification of the k value in place of the r value, and that is important to look at for security.

Assume that a user does not have a valid k value for the time period $t + 1$, but they do for time period t. This would imply that a user's key has been revoked and not renewed for the next time period. Because of the fact that the k values are generated by a pseudorandom function, the security of the k values are related to the security of this function as well as the seed. If a malicious user is able to determine the seed, they are then subsequently able to derive any k values and generate keys for any time period. Therefore the security of the entire scheme is defined by the inability of the users to compute seeds or determine the pseudorandom function. This is external to the CPABE scheme, and is a future avenue of exploration.

The setup of the T function and the fact that the outputs are not exponentiated means that it may be possible for a user to discern $T(t)$ and $T(t-1)$. If the hash function is known, then the user can subsequently generate $T(t+1)$ and so on. This may not be a flaw, as if files are to be reencrypted after a certain number of time periods, this allows users to perform this encryption prior to the actual time period. This is fine because no users will have keys valid for the future time periods, and would be unable to decrypt them until they access their key from that period. Additionally, if a user is given only the T values, they are unable to generate new keys or any advantage due to the fact that even with malicious T values, they do not have the r_i values needed to be used in the decrypt node function. Even if their values are modified, it is still difficult to discern the other part of their key, the H value for each attribute. These values are only used to cancel out based on the attribute, and even if the user is able to forge having a valid key for the time period for that decrypt node, their result will still have the incorrect k value during interpolation and final decryption. However, it is recommended that the T values are made public, because the advantages outweigh the possible security flaws for this value, with it being required on the k value still. The other option is to simply just only give the current T value for the time period, as that is still needed by the users in order to properly encrypt based on that time period.

6.6 Summary

Several options for key management are possible based on the desired restrictions of the scheme and the application in which it is being used. Due to the fact that CPABE desires to not use trusted servers, key management is difficult, especially for immediate key revocation. Previous works have shown that this tradeoff is difficult, and many schemes make immediate revocation difficult without trusted servers.

A key insulated approach to CPABE was presented, designed by substituting the r value in the CPABE user keys and replacing with a pseudorandom value k, as done in the key insulated ABE scheme [8]. With this, keys are valid for specific time periods, and renewal and revocation is done by small updates to keys from helpers. Outside of this modification, there are no additional operations required in order to decrypt files compared to the original CPABE scheme. The security of this scheme is directly related to the pseudorandom function used to generate values used as exponents for the helper keys.

Chapter 7

Conclusion & Future Directions

Two avenues related to the performance of Ciphertext-Policy Attribute-Based Encryption were explored as part of this thesis. The first was looking into the different types of elliptic curve pairings supported by the Pairing Based Cryptography library [23], and performing a size and performance analysis of these types through CHARM, a python cryptography prototyping framework [1]. It was found that type F, or Barreto-Naehrig, curves, had the fastest key generation and decryption, but the slowest decryption of all curve types. It was found that this slow decryption is due to the fact that the output group is almost twice the size of other groups, and operations take much longer in this group.

The second half of new work was with regards to key management for a CPABE scheme. While looking at immediate revocation proved difficult for a CPABE scheme, the usage of key insulation allowed both revocation during specific time periods, as well as providing an efficient mechanism for key renewal.

7.1 Future Work

Other elliptic curve libraries may end up producing different results outside of the PBC library which was the focus of this work. An additional option is the MIRACL library [26], which is additionally supported by CHARM, and may have different underlying operations which change the results obtained from this thesis.

For group selection, a key direction would be into the underlying operations required for the asymmetric curve types. As is, due to the fact that decryption, the most vital function, takes the longest for the type F curves, it makes them not recommended for many applications. Any optimizations which improve decryption on type F curves would be promising to see. Additionally, due to the high embedding degree of these curves, some attacks on other curves of lower embedding degrees may require that these curves are used, and the optimizations will be needed for comparable performance.

The key management scheme provided as part of this thesis works best when used in a broadcast application. In the works covered in this thesis, some provided efficient key revocation for broadcast applications of attribute based encryption schemes. A comparison in both efficiency, key size, and security of KICPABE to these schemes would allow different views of which is most efficient of these schemes.

For security of the KICPABE scheme, analysis of different pseudorandom and one-way functions would be a place to start. Additionally, critics of CPABE have noted that

the initial scheme [4] is only secure under a weak generic group model. Waters does improve this in his work [19] in order to prove the security under the Bilinear Diffie-Hellman model. Therefore, with the modifications proposed from KICPABE, it would be promising to modify it under this scheme and prove its security under this model.

Finally, more research into various key management methods would prove useful, allowing various different choices that may not have been covered in the scope of this thesis. Ideally, immediate key revocation would be allowed with no requirement on trusted servers at all.

Bibliography

[1] J.A. Akinyele, C. Garman, I. Miers, M.W. Pagano, M. Rushanan, M. Green, and A.D. Rubin. CHARM: A Framework for Rapidly Prototyping Cryptosystems. *Journal of Cryptographic Engineering*, pages 1–18, 2013.

[2] P. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected areas in cryptography*, pages 319–331. Springer, 2006.

[3] J. Bethencourt. Introduction to bilinear maps. Computer Sciences Department, Carnegie Mellon University. Available at `http://www.cs.berkeley.edu/bethenco/bilinear_maps.pdf"`, 2006.

[4] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-Policy Attribute-Based Encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[5] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology-CRYPTO 2001*, pages 213–229. Springer, 2001.

[6] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Advances in Cryptology-ASIACRYPT 2001*, pages 514–532. Springer, 2001.

[7] X. Boyen. The Uber-Assumption Family. *Pairing-Based Cryptography–Pairing 2008*, pages 39–56, 2008.

[8] J.H. Chen, Y.T. Wang, and K. Chen. Attribute-Based Key-Insulated Encryption. *Journal of Information Science and Engineering*, 27:437–449, 2011.

[9] L. Chen, P. Morrissey, and N. Smart. Pairings in Trusted Computing. *Pairing-Based Cryptography–Pairing 2008*, pages 1–17, 2008.

[10] J.H. Cheon and D.H. Lee. A Note on Self-Bilinear Maps. Available at `http://basilo.kaist.ac.kr/mathnet/thesis_file/11_B08-219.pdf`, 2009.

[11] Q. Du, G. Wang, and Q. Liu. A Scalable Encryption Scheme for Multi-Privileged Group Communications. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEEIFIP 8th International Conference on*, pages 597–602. IEEE, 2010.

[12] M. Gagne. *Applications of Bilinear Maps in Cryptography*. Master's thesis, University of Waterloo, 2002.

[13] S.D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[14] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded Ciphertext-Policy Attribute-Based Encryption. *Automata, Languages and Programming*, pages 579–591, 2008.

[15] 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*, pages 89–98. ACM, 2006.

[16] S. Hohenberger. *Advances in Signatures, Encryption, and E-cash from Bilinear Groups*. PhD thesis, Massachusetts Institute of Technology, 2006.

[17] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459 (Proposed Standard), 1999.

[18] L. Ibraimi, M. Petkovic, S. Nikova, P. Hartel, and W. Jonker. Mediated Ciphertext-Policy Attribute-based Encryption and its Application. In *Information Security Applications*, pages 309–323. Springer, 2009.

[19] L. Ibraimi, Q. Tang, P. Hartel, and W. Jonker. Efficient and Provably Secure Ciphertext-Policy Attribute-Based Encryption Schemes. *Information Security Practice and Experience*, pages 1–12, 2009.

[20] S. Jahid and N. Borisov. PIRATTE: Proxy-based Immediate Revocation of ATTribute-based Encryption. 2012.

[21] C.J. Li. *Cryptographic Key Management for Role-Based Access Control Model in Power System Computer Networks*. Master's thesis, Ryerson University, 2005.

[22] B. Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2007.

[23] B. Lynn. The Pairing-Based Cryptography Library. Available at `http://crypto.stanford.edu/pbc/`, 2010.

[24] M. Maas. *Pairing-Based Cryptography*. PhD thesis, Technische Universiteit Eindhoven, 2004.

[25] J. Milnor and D. Husemoller. *Symmetric Bilinear Forms*. Springer-Verlag, 1973.

[26] MIRACL Cryptographic SDK. Available at `http://www.certivox.com/miracl`, 2013.

[27] D. Page, N.P. Smart, and F. Vercauteren. A Comparison of MNT Curves and Supersingular Curves. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):379–392, 2006.

[28] W. Qiuxin and Z. Miao. Adaptively Secure Attribute-Based Encryption Supporting Attribute Revocation. *China Communications*, 9(9):22–40, 2012.

[29] A. Sahai and B. Waters. Fuzzy Identity-Based Encryption. *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473, 2005.

[30] D. Servos. *A Role and Attribute Based Encryption Approach to Privacy and Security in Cloud Based Health Services*. PhD thesis, Lakehead University, 2012.

[31] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology*, pages 47–53. Springer, 1985.

[32] D.R. Stinson. *Cryptography: Theory and Practice*, volume 36. CRC Press, 2006.

[33] B. Waters. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. *Public Key Cryptography–PKC 2011*, pages 53–70, 2011.

[34] Z. Xu and K.M. Martin. Dynamic User Revocation and Key Refreshing for Attribute-Based Encryption in Cloud Storage. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 844–849. IEEE, 2012.

[35] Z. Zhou. *On Efficient and Scalable Attribute Based Security Systems*. PhD thesis, Arizona State University, 2011.

[36] Z. Zhou and D. Huang. On Efficient Ciphertext-Policy Attribute Based Encryption and Broadcast Encryption. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 753–755. ACM, 2010.