Novel Lightweight Ciphertext-Policy Attribute-Based Encryption for IoT Applications

A thesis presented in partial fulfilment of the requirements
for the degree of

Master of Information Science

at Massey University, Auckland, New Zealand

Ping LI

2018

Abstract

As more sensitive data are frequently shared over the Internet of Things (IoT) network, the confidentiality and security of IoT should be given special consideration. In addition, the property of the resources-constraint nodes raises a rigid lightweight requirement for IoT security system. Currently, the Attribute-Based Encryption (ABE) for fine-grained access control is the state-of-the-art technique to enable the secure data transmission and storage in the distributed case such as IoT. However, most existing ABE schemes are based on expensive bilinear pairing with linear size keys and ciphertexts. This results in the increase of the memory and computational requirement on the devices, which is not suitable for the resource-limited IoT applications.

Leveraging on the advantages offered by the Ciphertext-Policy ABE (CP-ABE), this thesis proposes two constructions of lightweight no-paring cryptosystems based on Rivest–Shamir–Adleman (RSA). One realized work is a construction of AND-gate CP-ABE to achieve both constant-size keys and ciphertexts. The result of the evaluation shows that it reduces the storage and computational overhead. The other construction supports an expressive monotone tree access structure to implement the complex access control as a more generic system. Both have respective advantages in different contexts and are provably secure to guarantee the sharing of data, as well as more applicable and efficient than the previous scheme. In this thesis, practical issues are also described about implementations and evaluations of both proposals.

Acknowledgments

I would like to thank my supervisor Associate Professor Julian Jang-Jaccard of Massey University for making the IoT projects possible and supervising my research. Her extensive industry and academic experiences have been extremely valuable in contributing to the successful completion of my projects and thesis. I also want to express my appreciation to Associate CISSP Tim McIntosh for reviewing my thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 IoT Ecosystem

Due to the vigorous development of ubiquitous IoT, IoT application is prevalent in the physical world, from e-health and e-home to smart city, etc. In numerous intelligent applications, IoT is characterized by smart devices and tags connecting sensors to the internet [16]. These sensing nodes work cooperatively to enable the interconnections among the ubiquitous IoT, and between cyber and real world as well. IoT is the innovative technology to facilitate users to access, exchange and store data. Therefore, IoT will be considered as the crucial infrastructure building smart society in the forthcoming era.

Security Challenges of IoT

Since the IoT devices play the key role in transferring the collected data via network, it could bring many challenges to the confidentiality and availability of data. The rise of cloud computing makes the data management more complex in the distributed environments, where servers increasingly provide services across many sites. In this case, IoT devices become vulnerable targets of malicious attacks and information theft [1]. The various attacks including eavesdropping, Denial of Service Attack (DOS) and fabrication could result in serious loss. Additionally, in the event of server device compromise described in [2], possibly, as a result of a software vulnerability exploit, information theft or leak is at a high risk. Hence, with continuously emerging threats, security in the context of IoT requires great attention.

Furthermore, security challenge faced by such devices is escalated by the surge in recent aggregated attacks. Any sensitive data collected, distributed and transmitted in plaintext form will be vulnerable to be attacked. Given the variety, amount, and importance of information stored on these devices, there is concern that personal data could be compromised [2]. From a security standpoint, sensitive data should be stored in encrypted forms so that it will remain confidential even if a server is compromised. Since the purpose of cryptography [17] involves confidentiality, integrity, authentication, access control and so on, data confidentiality and security thus can be accomplished using cryptographic encryption techniques.

Lightweight Security for IoT System

IoT usually consists of resource-starving devices with limited memory, CPU, low power and battery capacity, etc. Examples include mobile devices and digital electronics with Radio Frequency Identification (RFID) tags have been well-known among them. With the popularity and availability of lightweight devices, this has given the increasing demand for designing efficient cryptographic scheme on encipherment and decipherment to offer the lightweight security for IoT systems.

From the survey depicted above, it is imperative to address these issues so that the security of the data in IoT will be significantly enhanced.

1.2 Research Scope and Objectives

Similar to the traditional (wire or wireless) networks, data security in IoT [22] also includes confidentiality, integrity, authenticity and privacy. However, traditional methods [2] on the basis of access control lists need to rely on the storage server for preventing unauthorized data access. This deployment is of limited usage and not particularly suitable for the cloud-based network, where the server is not fully trusted by user. This issue motivates many academic achievements on the technology innovation. A novel cryptographic technique known as ABE approach was proposed to employ data sharing systems at fine-grained level in cloud computing. In ABE systems, secret keys or ciphertexts are associated with a set of attributes composed of descriptive strings. In particular, ABE systems have the property that user can decrypt the ciphertext as long as it satisfies the required attributes, which makes it promising for access control. [11] Instead of trusting and depending on service providers, data owners can enforce selective access control to a particular group of users themselves.

In addition, since most IoT devices are resource-limited, a lightweight cryptographic method is more desirable to implement on a cost-efficient system for encryption and decryption. For instance, in a mobile cloud setting, where internet-enabled devices used to access the cloud services are generally resource-constrained (e.g. limited memory, CPU), it is indispensable to cut down the computation cost and storage overhead (e.g. lengths of ciphertext). Consequently, to alleviate these concerns for plenty of IoT applications, it is obvious that the lightweight ABE cryptosystem is one appealing scheme.

Yang et al. [6] claimed that lightweight security enforcement in cyber-physical systems should achieve three key properties, namely: system lightweightness, outsourcing of expensive computations, and selective protection of critical data. Rather than taking a whole-of-system approach, this thesis focus on developing lightweight cryptographic primitives of the research interest.

As one variant of ABE, CP-ABE allows the data owner to encrypt data with an encoding access policy, whereby only authorized data users with the desired attributes are able to decrypt data. The decoding process is performed after fulfilling the access policy. Clearly, its manifest advantage is more practical for IoT. Unfortunately, it is difficult to implement the existing CP-ABE scheme on the IoT device due to the shortage of the lightweight feature. The lightweight feature [22] implies that it should be more efficient and feasible than traditional solutions in terms of memory capacity and power consumption. But most existing CP-ABE schemes have not been designed taking into consideration energy efficiency. Also, the length of the ciphertext depends on the number of attributes and grows linearly in most previous CP-ABE schemes. Many IoT devices, like sensors and actuators, cannot apply CP-ABE because of their resource limitations.

Although some academic efforts have been made at the expense of incremental computation or resource, they may not be the fit-for-purpose. To the best of my knowledge, a lightweight no-pairing CP-ABE scheme pioneered by Odelu et al. [5] is the only scheme based on RSA algorithm to address these issues on IoT. This thesis discusses the issues of lightweight security techniques to protect data confidentiality, integrity and authentication for IoT local systems. It will consider the lightweight properties of the encryption systems on both communication overhead and computational overhead.

Based on the former research result [5], a novel lightweight CP-ABE encryption system with constant-size keys and ciphertexts is constructed to reduce the usage of storage space. Besides that, a CP-ABE scheme based on access-tree structure is proposed to build up the complex access policies for large systems. Meanwhile, RSA algorithm is used as the underlying cryptographic primitive in the cryptosystems. It improves the efficiency based on no-pairing RSA algorithm instead of pairing-based cryptography. This technology promotes efficient sharing of data and reduce computational overhead.

1.3 Structure of the Thesis

The rest of this paper is structured as follows. In Chapter 2 the author introduces necessary cryptography background and definitions about ABE and CP-ABE. Then, the author discusses the literature review on applied cryptographic techniques in Chapter 3. Next, the author proposes the two cryptosystems and gives their proofs of security in Chapter 4 and Chapter 5 respectively, as well as provides the implementation and the performance measurement of both cryptosystems. Finally, the author generalizes the work of this paper and concludes in Chapter 6.

Chapter 2
Cryptography Background

2.1 ABE Basics

The concept of ABE was presented by Sahai and Waters [3], which is motivated by Identity-Based Encryption (IBE). Unlike an IBE system [3] or Hierarchical Identity-Based Encryption (HIBE) system [18] associated with simple identities, in an ABE system, keys and ciphertexts are labeled with more complex objects – sets of descriptive attributes and access formulas. A secret key can decrypt a particular ciphertext only if there is a match between the attributes of the ciphertext and the secret key. Different users are allowed to decrypt different pieces of ciphertext using their secret keys. This effectively eliminates the requirement that rely on the storage server for the security of data access. It shows that ABE has the advantage for the applicability of IoT. As a result, ABE has become a preferable solution to solving the problem of the data access control.

At present, ABE is mainly used to prevent malicious or unauthorized access to sensitive data in the cloud. Cloud computing is one of main popular settings requiring secure data transmission in the application of IoT. In the ABE system, there are two categories, Key-Policy ABE (KP-ABE) and CP-ABE according to different role of attributes and access structure. KP-ABE (SW[2],GPSW[3]) is a scheme in which the access structure is specified in the private key while the ciphertexts are simply labeled with a set of descriptive attributes. This contrasts with CP-ABE, in which an access structure (i.e. policy) would be associated to each ciphertext while a user's private key would be associated with a set of attributes. Both KP-ABE and CP-ABE cryptosystems are useful in different contexts.

2.2 What is CP-ABE?

In the common set as shown in [4], it can be critical that the person in possession of the secret data be able to determine an access policy based on given knowledge of the underlying data. Moreover, this data owner may not know the exact identities of all other people who should be entitled to access the data, but rather the data owner may only be able to describe them in light of their attributes or credentials.

Traditionally [4], this type of expressive access control is enforced by employing a trusted server to store data locally. Access control relies on software checks to ensure that only the accredited users can access the records or files. Nevertheless, in a large-scale system, it is not easy to identify every potential recipient, acquire and store their public keys. Therefore, it is more desirable to be able to encrypt the data without the complete list of intended recipients in this scenario.

CP-ABE is an emerging approach to deal with this case and ensure user authentication. It can identify whether the user is legitimate when he requests to perform an operation on stored data by checking if the user's attributes fulfill the access policy. Access control policies facilitate granting various access rights to a set of users and provide flexibility in specifying the access privileges for individuals [2]. By using flexible access policies in CP-ABE, a data owner is able to encrypt the data even without the exact information of possible receivers.

Bethencourt et al. [4] introduced the first construction of CP-ABE, as referred to BSW, which present the more expressive models of encrypted access control. In their system, a user's private key is associated with an arbitrary number of attributes expressed as strings. A party encrypting data selectively specifies a policy for who can be granted to decrypt. The eligible user is able to disclose the data contents if and only if his decryption key satisfies the access policy.

An example of CP-ABE is demonstrated below. Suppose in a computer science faculty of a university, the CP-ABE cryptosystem is being used to provide the privacy for the access of confidential documents. The data administrator might want to encrypt some documents to some of executive team members of this faculty under an access policy, such as "CS" AND "executive_team" AND "admin_level > 5". Accordingly, any user who possesses all of these attributes that satisfies the policy could access the documents, whereas any other members are unable to access these files. For example, Bob who has attributes {"CS", "staff_team"}, cannot decrypt the documents.

A ciphertext-policy attribute-based encryption scheme [4] consists of four fundamental algorithms: **Setup**, **Encrypt**, **KeyGeneration**, and **Decrypt**.

Setup (1^{λ})**.** The setup algorithm takes input of the implicit security parameter 1^{λ} and the universe of attributes. It outputs the public parameters PK and a master key MK.

Encrypt (PK, M, $\mathbb{A}$). The encryption algorithm takes as input the public key PK, a message M, and an access structure $\mathbb{A}$ over the universe of attributes. The algorithm will encrypt M under $\mathbb{A}$ and output a ciphertext CT such that only a user that has a set of attributes that satisfies the access structure will be able to decrypt the message. We will assume that the ciphertext implicitly contains $\mathbb{A}$.

KeyGeneration (MK, S). The key generation algorithm takes as input the master key MK and a set S of descriptive attributes. It generates a private key SK that associated with the set S.

Decrypt (PK, CT, SK). The decryption algorithm takes as input the public parameters PK, a ciphertext CT, which contains an access policy $\mathbb{A}$, and a private key SK, which is a private key for a set S of attributes. If the set S of attributes satisfies the access structure $\mathbb{A}$, then the algorithm will decrypt the ciphertext and return a valid message M.

2.3 Summary of Related CP-ABE Schemes

CP-ABE schemes can be further classified into various categories based on access structures. This paper assumes to talk about monotonic access structures (without negated attributes in the policy). They subsume **AND-gates access structure** ZHW[8], CN[12] and **Threshold access structure** for short ciphertexts (see [9], Pirretti et al. [24]). For general access structure, there are some CP-ABE schemes based on **Monotone Tree access structure** which support AND gates, OR gates and threshold ([2], [22]), and based on LSSS (Linear Secret Sharing Scheme [26]).

A threshold access policy was introduced by Sahai and Waters [3], who propose that a user secret key and the ciphertext are both associated with attribute sets. Decryption would only work if the overlap between the two sets is at least as large as the globally defined threshold value. For example, for a set with a total of n attributes, a (t, n)-Threshold gate means that there are any t or more user attributes in a secret key satisfy the required attributes in ciphertext policy. One approach usually uses Secret-Sharing Schemes (SSS) with a variable threshold value t as a (t, n)-Threshold gate. SSS [2] are used to divide a secret among a number of entities. The information assigned to a party is known as the secret share [2] for that party. Every SSS realizes a Threshold-gate access structure that defines the set of parties who should be able to reconstruct the secret by combining their shares.

An access structure of AND-gates is the same as an (*n*, *n*)-Threshold access structure. This essentially means that all shares need to be present, then a single AND-gates with *n* attributes was used. Thus, it is also called (*n*, *n*)-Threshold. Because the attributes in AND-gates access structure can be multi-valued (wildcards), this leads to the realization of Hidden Vector Encryption and non-monotonic access structures. Nevertheless, it means that the required attributes in the access structure and the set of user attributes must be the exact same. This restriction doesn't meet the motivation of CP-ABE for fuzzy decryption.

The Tree access structure is comprehensive approach that consist of Threshold gates and AND-gates, which was first described in Goyal et al [2]. It is a way of sharing a secret element across the attributes of a policy in KP-ABE or CP-ABE, which can then be reconstructed with Lagrange interpolation (see [2], [4]). The construction of Goyal et al. permits not only AND and OR gates, but arbitrary threshold gates with SSS for every node of the tree. Although the Tree access structure may be a generic model for expressing a complicated policy, it is not easy to build up the access tree for a large-scale system.

A Linear Secret Sharing Scheme (LSSS) works on a matrix, where the rows are labelled with attributes of the policy, to produce shares from a secret element *s* hold by a third party called dealer [21]. This party distributes the shares of *s* to other parities such that *s* can be reconstructed by a linear combination of the shares of any authorized set. This could be the same as the outcome for the Tree access structure.

With the rapid growth of IoT application, people also focus on the applicability in IoT, whereby research on the size of ciphertexts and keys is booming. There exist some schemes with constant-size keys property (GSWV [13]) or constant-size ciphertext property (DJ[10]). Considering the practical application scenarios where resource-constrained devices with less sufficient energy and memory are deployed in the IoT network, most of the previous CP-ABE schemes provide linear size keys and ciphertexts , which usually grow with the number of attributes. When there are multiple attributes in the attribute set, this may make severe efficiency shortcomings even fail to work due to the shortage of resources. However, IoT devices using these CP-ABE schemes with constant-size keys and ciphertexts do not need to suffer from the linearly increasing computational and communication overhead. This should be ideal even if the number of attributes is huge, it will only increase slightly in computation, and will be no storage growth. Under this setting, user also do not require to have to set the maximal number of attributes for successful decryption in terms of the capacity of CPU and storage.

Several CP-ABE schemes support both constant size and secret keys, ZZCLL[11], [14] and EMNOS [7]. The EMNOS scheme [7] offers AND-gates CP-ABE with constant-size key on multi-valued attributes structure. However, the access structure is in effect (n, n)-threshold, which requires that the attributes in the access structure need to be completely identical to the set of user attributes.

Nevertheless, with the exception of Odelu et al. [5] and Yao et al. [22], they are all based on bilinear pairing of Elliptic Curve Cryptography (ECC). More precisely, majority of the existing ABE schemes are all based on bilinear maps, which is computational overhead for energy-limited IoT devices. This design drawback for IoT prevents the deployment of CP-ABE schemes on IoT.

Chapter 3
Literature Review

The most existing CP-ABE schemes are built with bilinear pairing of ECC. Only few are based on no-pairing ECC or no-pairing RSA. Their security strengths are determined by the underlying cryptographic algorithms, which in turn is determined by the bit-length of the parameters used.

3.1 ECC-Based CP-ABE

All implementation of pairing-based cryptosystems is constructed with elliptic curves. Apart from ECC based CP-ABE with the pairing cryptosystems, to the best of our knowledge, the solution of Yao et al. [22] is an exclusive no-pairing ECC based CP-ABE. In cryptography, an elliptic curve *E* is defined being over a field of prime order. Compared with other public key cryptography, elliptic curve cryptography uses points on *E* instead of elements of some finite field F^*. The group of points on the elliptic curve is able to be selected for paring, and the group operation is point addition rather than modular multiplication.

3.1.1 Elliptic Curves

An elliptic curve *E* over such a field $K = F_q$ of prime order *q* can be determined by a cubic equation as the form $y^2 = x^3 + ax + b$, where $a, b \in F_q$. Let $\Delta = 4a^3 + 27b^2$ [27], be the discriminant of the cubic in *x*. Then *E* is singular if $\Delta = 0$ and nonsingular otherwise. This thesis always considers nonsingular elliptic curves, whereby supersingular curves is one particular breed that are not singular instead.

The basic operations on the elliptic curves are point operations, where the group operation is built. For elliptic curves *E*, one takes some subgroup *G* of the group of points *E(K)* ($E(F_q)$) with prime order *r* and the operations mainly refer to point addition and point scalar multiplication. An exponentiation is also performed as point multiplication. For any field, $K = F_{q^k}$ define $E(F_{q^k})$ to be the set of all solutions of *E* over F_{q^k} [27], called the finite points along with a special point at infinity denoted O. The number of elements of *E(K)* is denoted as *#E(K)* or |*E(K)*|, where $K = F_q$ or F_{q^k}. To evaluate the amount of *#E(K)*, This thesis quotes a well-known Hasse theorem on an elliptic curve.

Theorem (Hasse [35]). Let $t = q^k + 1 - \#E(F_{q^k})$. Then $|t| \leqslant 2pq^k$. Thus, the number of points on an elliptic curve over a finite field is on the same order as the size of the given field. The quantity t is called the trace of Frobenius.

3.1.2 Elliptic Curve Cryptography

The elliptic curve group operation means that every point on E generates a cyclic group G. Then G can be used for cyclic group cryptography provided that its order is prime. The implementation of pairing-based cryptosystems depends on cyclic groups with particular properties. Roughly speaking, bilinear maps, or pairings, offer cyclic groups additional properties.

A. Bilinear Pairings

Definition (Bilinear Pairing [14]). Let r be a prime. Let G_1, G_2, G_T be cyclic groups of order r. Assume g_1 be a generator of G_1 and g_2 be a generator of G_2. A bilinear map e is a computable function $e : G_1 \times G_2 \rightarrow G_T$ with the next three properties:

- Bilinear: $\forall a, b \in \mathbb{Z}_r$, $e(g_1{}^a, g_2{}^b) = e(g_1, g_2)^{ab}$.
- Non-degenerate: $e(g_1, g_2) \neq 1$.
- Efficiency: $e(g_1, g_2)$ is a generator of G_T and is efficiently computable.

For ECC [27], G_1 and G_2 (or G) are always groups of points on an elliptic E curve over a field F_q, and G_T is always a subgroup of a multiplicative group of a finite field. Both G_1 and G_2 are the group of points $E(F_q)$, which contains a subgroup G of order r by defining $G = E(F_q)\,[r]$.

Depending on the scheme, certain mathematical problems may be assumed to be hard in both G_1 and G_2, or a combination of the two. For example, assume that given g_1, $g_1{}^x \in$ G_1 and $g_2 \in G_2$, there is no efficient algorithm to compute $g_2{}^x$.

B. Weil and Tate Pairings

In ECC cryptography, cryptographic pairings are derived from functions known as the Tate and Weil pairings [27]. Pairing-friendly ECC algorithms are able to be selected to yield cryptographically secure pairings. The Weil and Tate pairings take r-torsion points as input, can be defined using rational functions. They output an element of a finite field that is an *rth* root of unity. Several related definitions need to be introduced here.

1. Torsion Points [27]

Let K be a finite field of characteristic q, so that $K = F_{q^m}$ for some natural number m. Let E be an elliptic curve defined over K, containing n points. Suppose a point $P \in E(K)$ satisfies $rP = O$ such that P has order r or a factor of r. We define P an r-torsion point and denote the set of r-torsion points of $E(K)$ by $E(K)[r]$.

For the curves considered in [27], n and q are always coprime thus r is also coprime to q since $r \mid n$ (r dividing n). It can be proved that for some integer $k \geqslant 1$, $E(F_{q^k})[r]$ contains exactly r^2 points and is in fact the direct product of two cyclic groups of order r, that is, isomorphic to $\mathbb{Z}_r \times \mathbb{Z}_r$. The set of r-torsion points in E is written as $E[r] = E(F_{q^k})[r]$. Thus the above isomorphism may be written as $E[r] \cong \mathbb{Z}_r \times \mathbb{Z}_r$.

2. Rational Functions [27]

Let E be an elliptic curve $y^2 = x^3 + ax + b$ over F_{q^k}. $E(X, Y)$ takes two variables X, Y is a polynomial for such cubic equation. The polynomial $E(X, Y)$ with coefficients in F_{q^k} is denoted by $F_{q^k}[E] = F_{q^k}[X, Y]$. If there exists a polynomial $f(X, Y)$ is a multiple of $E(X, Y)$, it satisfies $f(X, Y) = 0$ whenever $E(X, Y) = 0$. Define $F_{q^k}(E)$ to be the field of fractions of $F_{q^k}[E]$. Elements in $F_{q^k}(E)$ can be shown to be of the form $f(X, Y)/g(X, Y)$ where f and g are polynomials in two variables X, Y.

When considering the quotient of two polynomials $f(X)/g(X)$, the **zeroes** are called the roots of f and the **poles** are the roots of g, and f/g is completely determined up to a constant by its zeroes and poles and their multiplicities. Extending this idea for $f(X, Y)/g(X, Y)$, the zeroes are the zeroes of f and the poles of g, and the poles are the poles of f and the zeroes of g.

Divisors are defined in [27] to notate zeroes and poles and their multiplicities, or orders. Let D be the divisor with zeroes and poles $P_1, ..., P_n$ of multiplicities $a_1, ..., a_n$. D shall be represented as $D = (p_1)^{a_1}...(p_n)^{a_n}$.

Thus, by using *(f)* to denote the divisor of *f*, in the notation for any rational functions *f*, *g* we have $(f)(g) = (fg)$.

3. Weil Pairings

In the case of the Weil pairing, both inputs are *r*-torsion points. It produces an *rth* root of unity.

Definition (Weil pairing [42]). Let *E* be an elliptic curve over a field F_q and $n = \#E(F_q)$. Let *G* be a cyclic subgroup of $E(F_q)$ of order *r* with *r* and *q* coprime. Let *k* be the smallest positive integer such that $E(F_{q^k})$ contains all of $E[r]$. The definition of the Weil pairing $f: E[r] \times E[r] \rightarrow F_{q^k}$ is as follows.

For a pair of points $P, Q \in E[r]$, select any $R, S \in E[F_{q^k}]$ such that $S \neq R, P+R, P+R-Q, R-Q$. Let f_P be a rational function with divisor $(f_P) = (\mathrm{P} + \mathrm{R})^r/(\mathrm{R})^{\mathrm{r}}$. In standard notation this is $r(P+R) - r(R)$, where *r* zeroes are at $P + R$ and *r* poles are at *R*. and similarly let f_Q be a rational function with divisor $(f_Q) = (\mathrm{Q} + \mathrm{S})^r/(\mathrm{S})^{\mathrm{r}}$. Define $f(P, Q)$ is quotients as [42].

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

From this definition, it can be shown that choosing *R*, *S* and finding explicit expressions for these functions are not easy for producing cryptographically useful pairings.

4. Tate Pairing

Let *E* be an elliptic curve containing n points over a field F_q. Let *G* be a cyclic subgroup of $E(F_q)$ of order *r* with *r*, *q* coprime. Let *k* be the smallest positive integer such that $r \mid q^k - 1$. An equivalent characterization is that $K = F_{q^k}$ is the smallest extension of F_q containing the *rth* roots of unity. The Tate pairing is defined as follows [41].

Let f_P be a rational function with divisor $(f_P) = (P)^r$. Choose an $R \in E(K)$ such that $R \neq P, P - Q, \mathrm{O}, -Q$. Define $f(P, Q)$ as

$f(P, Q) = f_P\ (\mathrm{Q} + \mathrm{R})/f_P(\mathrm{R})$

"The Weil pairing is a bilinear map takes pairs of elements from $E[r]$ and outputs an *rth* root of unity in F_{q^k}. The Tate pairing is similar but only the first input is from $E(F_q)[r]$ (see [27])." By defining $G_1 = G$, $G_2 = E[r]$, and G_T to be the *rth* roots of unity in F_{q^k}, the Weil pairing and the Tate pairing are in accordance with the abstract definition of a bilinear pairing given above.

C. Embedding Degree of the Curve

Recall that the Tate pairing and the Weil pairing described above must be defined and computed in some field extension L of K. In order for the Tate pairing and the Weil pairing to be efficiently computed, operations must be efficient in L. Thus, the fields L needs to be small enough so that field operations are fast. It turns out that such a field L will always contain the nth roots of unity. Then, the embedding degree on the elliptic curve is defined. It can be viewed as measuring the size of L compared to the field K.

Definition (Embedding degree of the curve [27]). Let E be an elliptic curve defined over $K = F_q$. Let $G \subseteq E(F_q)$ be a cyclic group of order r. Let k be the smallest positive integer such that $r \mid q^k - 1$ holds. Then we have that the embedding degree of G is k.

From such definition, the aim seems to seek the minimized embedding degree, so that L is as close as possible to K. In practice, the embedding degree of G shall often be determined by finding the smallest integer k such that $n \mid q^k - 1$, where $n = \#E(F_q)$. In this way [27], for a cyclic subgroup $G \subset \#E(F_q)$ the embedding degree k of $E(F_q)$ may not be the smallest positive integer such that $r \mid q^k - 1$. However, in general [27], when we use such a group G and r is a large prime factor of $\#E(F_q)$ it is almost always the case that F_{q^k} is the smallest field extension that allows the computation of the pairing. In other words, when r is a large prime factor of $\#E(F_q)$, the embedding degree k is always preferred. At this point, an embedding degree of a curve $E(F_q)$ implies the embedding degree of the subgroup of $E(F_q)$ of order r, where r is the largest prime dividing $\#E(F_q)$.

Because a desired embedding degree is $k > 1$, both the Tate pairing and the Weil pairing may be computed by performing field arithmetic in F_{q^k}. By carefully tailoring the complex multiplication method of constructing elliptic curves, the curves of a certain embedding degree can be produced. From an efficiency standpoint, choosing the input groups to be certain subgroups is desirable for all pairings. As stated in [27], supersingular curves are guaranteed to have a small embedding degree and are easy to construct. They have been completely classified. Operations on some of them can be efficiently processed by optimization.

D. ECC Key Generation

To generate a public and private key pair for ECC communications, an entity would perform the following steps [35]:

1) Find an elliptic curve $E(K)$, where K is a finite field such as F_q, and choose point Q of order n on $E(K)$.
2) Choose a pseudo random number x such that $1 \leqslant x \leqslant (n - 1)$.
3) Compute point $P = xQ$.
4) ECC key pair is (P, x), where P is the public key, and x is the private key.

Elliptic curve cryptography depends on the ability to compute a point multiplication. Their key strength derives from a computational hard problem called the Elliptic Curve Discrete Logarithms (ECDL).

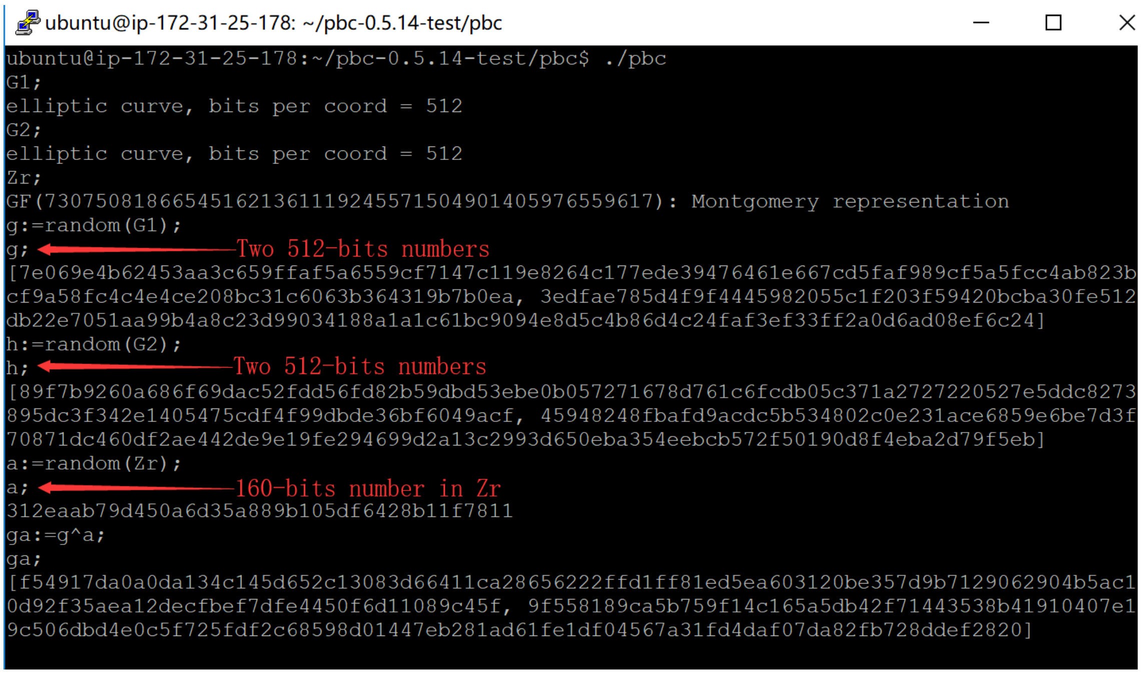
E. ECDL

Definition (ECDL [22]). For elliptic-curve-based protocols, it is assumed that no the discrete logarithm of a random elliptic curve element with respect to a publicly known base point has ever been discovered; this is the infeasibility to compute the x such that $P = xQ$ if given points Q and P on the elliptic curve. The security of ECC is based on the intractability of ECDLP. The size of the elliptic curve determines the difficulty of such problem.

3.1.3 Key Size in the Implementations of ECC-Based CP-ABE

To the best of author's knowledge, all full developed instances of CP-ABE with the pairing (e.g. Bethencourt et al. [39]) are built on Pairing-Based Cryptography (PBC [40]) encryption scheme integrated ECC.

It is stated that PBC with some appealing features can offer a desired security level using smaller key size as the general ECC. In practice, this conclusion made directly from one-sided theoretical analysis is not accordance with the physical world. More precisely, the security of ECC is working on the computational complexity of ECDL. Since the intractable level of ECDL defines the security level of ECC, it requires that the order n of the base point P satisfying $n > 2^{160}$. This means the underlying field need larger enough so that the DL (Discrete Logarithms) in $F_q{}^*$ is considered intractable. On the other hand, it requires to ensure the efficiency of pairing computations in PBC. In a word, in a running case [25] PBC has to work with the key size of 1024-bits, instead of that of 160-bits input as supposed, in order to offer 80 bits security level. An example of this is shown in Figure 1. In this example, PBC works on a fast curve over 512-bits cyclic group. It will produce the element of 1024-bits size taken as representation of the key as shown because a point (x, y) in the curve is composed of x and y of 512-bits each.



```
ubuntu@ip-172-31-25-178:~/pbc-0.5.14-test/pbc$ ./pbc
G1;
elliptic curve, bits per coord = 512
G2;
elliptic curve, bits per coord = 512
Zr;
GF(730750818665451621361119245571504901405976559617): Montgomery representation
g:=random(G1);
g;
[7e069e4b62453aa3c659ffaf5a6559cf7147c119e8264c177ede39476461e667cd5faf989cf5a5fcc4ab823b
cf9a58fc4c4e4ce208bc31c6063b364319b7b0ea, 3edfae785d4f9f4445982055c1f203f59420bcba30fe512
db22e7051aa99b4a8c23d99034188a1a1c61bc9094e8d5c4b86d4c24faf3ef33ff2a0d6ad08ef6c24]
h:=random(G2);
h;
[89f7b9260a686f69dac52fdd56fd82b59dbd53ebe0b057271678d761c6fcdb05c371a2727220527e5ddc8273
895dc3f342e1405475cdf4f99dbde36bf6049acf, 45948248fbafd9acdc5b534802c0e231ace6859e6be7d3f
70871dc460df2ae442de9e19fe294699d2a13c2993d650eba354eebcb572f50190d8f4eba2d79f5eb]
a:=random(Zr);
a;
312eaab79d450a6d35a889b105df6428b11f7811
ga:=g^a;
ga;
[f54917da0a0da134c145d652c13083d66411ca28656222ffd1ff81ed5ea603120be357d9b7129062904b5ac1
0d92f35aea12decfbef7dfe4450f6d11089c45f, 9f558189ca5b759f14c165a5db42f71443538b41910407e1
9c506dbd4e0c5f725fdf2c68598d01447eb281ad61fe1df04567a31fd4daf07da82fb728ddef2820]
```

Figure 1. ECC Key Size in a PBC Running Case

In addition, a core operation in PBC is to compute a bilinear pairing (e.g. the Weil or the Tate pairing), which is computationally expensive but often used. From the above two points of view, PBC is losing its significant advantage over ECC although it looks ideal.

3.2 RSA-Based CP-ABE

The RSA algorithms, in addition to be the first publicly known examples of high-quality public-key algorithms, have been among the most widely used. In such a cryptosystem, the encryption key is public while the decryption key is kept secret (private). RSA encryption and decryption relies on modular arithmetic and not bilinear maps.

In an RSA cryptosystem, a composite number N is picked from the product of two large prime p and q. Let $\mathbb{Z}_N$ ($\mathbb{Z}/n\mathbb{Z}$) be the set of congruence classes of the integers modulo N. An RSA key pair $((N, e), d)$ can be generated using the multiplicative inverse of an integer e modulo N if given e being co-prime to $\varphi(N)$ in $\mathbb{Z}_N$. $\varphi(N)$ is Euler's totient function which gives the order of the multiplicative group of integers modulo N. Euler's totient function is also called Euler's phi function. For a prime n, we have the value of $\varphi(n) = n$-1, since all positive integers less than n are coprime to it.

In number theory, even if knowing e and N it can be extremely difficult to find the private key d, which is also defined as the modular inverse of e modulo N. The inverse exists if and only if $gcd(N, e) = 1$, which means that value 1 is the largest number to divide both N and e. The *gcd* is called the greatest common divisor. If $gcd(m, n) = 1$ then we say that m and n are coprime or relatively prime. An efficient method of computing the *gcd* of two integers is called Euclidean algorithm. The algorithm is shown below in the Table 1.

Euclidean Algorithm Computing the Great Common Divisor of Two integers([37])

Input: two non-negative integers m and n with $m \geq n$
Output: the greatest common divisor of m and n

```
Procedure Euclidean_gcd($m$, $n$)
    While $n > 0$, do
        Set $temp\_r = m \bmod n$
        $m = n$
        $n = temp\_r$
    return $m$
```

Table 1. Euclidean Algorithm

RSA uses groups whose order must be unknown to the attacker. An RSA-based scheme uses residues of integer modulus rather than bilinear pairing. Furthermore, an RSA cryptosystem arises for finite fields, where one picks a prime e and uses a subgroup G of

$\mathbb{Z}_N^*$ of prime order φ(N). Thus, the group operation in such RSA cryptosystem is field multiplication.

3.2.1 Textbook RSA Cryptosystem

A classic RSA cryptosystem is called the "textbook" RSA cryptosystem. The term "textbook" indicates that, although the RSA cryptosystem as presented below appears in many papers and books, there definitely exits some differences with the RSA used in the deployed systems. In particular, public key encryption is most often used to securely transmit symmetric keys (the functionality is often called key encapsulation), rather than to encrypt data.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and N such that with modular exponentiation for all integer m (with $1 \leqslant m < N$), $(m^e)^d \equiv m$ (mod N) holds. In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies $(m^d)^e \equiv m$ (mod N). This works because multiplication is commutative.

The RSA algorithm includes four procedures: key generation, key distribution, encryption and decryption.

a. Key Generation Algorithm

An RSA public and private key pair can be generated using the practical algorithm below.

1) Choose two random prime numbers p and q, of approximately equal size such that their product $N = pq$ is of the required bit length, e.g. 1024 bits.
2) Compute $N = pq$ and $\varphi(N) = (p - 1)(q - 1)$.
3) Choose a random integer e such that $e < \varphi(N)$ and $gcd(e, \varphi(N)) = 1$.
4) Compute the secret integer d such that $ed \equiv 1 \bmod \varphi(N)$.
5) (N, e) is the public key, and d is the private key, where N is known as the modulus, e is known as encryption exponent or public exponent and d is known as the decryption exponent or secret exponent.

This step may require trusted third parties and public key infrastructures to perform.

b. Key Distribution [36]

Suppose that Alice's RSA public key is the pair of integers (*N*, *e*) and her private key is the integer *d*. Bob wants to send message to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (*N*, *e*) to Bob via a reliable, but not necessarily secret channel. Alice's private key (*d*) is never distributed.

c. Encryption

To encrypt a message to Alice, sender Bob does the following.

1) Obtains the recipient Alice's public key (*N*, *e*).
2) Encodes the plaintext message as a positive integer *m* with $1< m < n$. Note that in practice one may assume that $m \in (\mathbb{Z}/n\mathbb{Z})^*$.
3) Computes the ciphertext as $c = m^e \pmod N$.
4) Transmits the ciphertext *c* to Alice.

d. Decryption

Recipient Alice does the following.

1) Uses her private key (*d*) to compute $m = c^d \pmod N$.
2) Decodes to obtain the plaintext from the message representative *m*.

This thesis will explore the above RSA algorithm in depth. The value *m* is not usually an actual message or document (which might be huge) but a short integer that is the output of some (non-injective) compression function (such as a hash function). Sometimes *m* is called a message digest as defined in [36]. The idea is that exponentiation to the power *e* modulo *N* is a one-way function: a function that is easy to compute but such that it is hard to compute pre-images. Indeed, exponentiation modulo *N* is a one-way permutation on $(\mathbb{Z}/n\mathbb{Z})^*$ when *e* is co-prime to φ(*N*). The private key *d* allows the permutation to be efficiently inverted and is called a trapdoor [36]. Therefore, RSA is often described as a trapdoor one-way permutation. The RSA system can also be used as a digital signature algorithm.

e. Signing [36]

When sending a message to Bob, Sender Alice does the following.

1) Creates a message digest for the message to be sent.
2) Represents this digest as an integer m between 1 and $n-1$.
3) Uses her private key (N, d) to compute the signature as $s = m^d (\text{mod } N)$
4) Sends this signature s to the recipient, Bob.

f. Verification [36]

When Bob receives (m, s) he obtains an authentic copy of Alice's public key and then verifies as follows.

1) Uses Alice's public key (N, e) to compute integer $v = s^e$ (mod N).
2) Independently computes the message digest H' of the information that has been signed.
3) Computes the expected representative integer v' by encoding the expected message digest H'.
4) If the verification equation $v = v'$ holds, the signature is valid, then Bob believes that the message m does come from Alice.

From a mathematical point of view, decryption and signing are identical as both use the private key. Similarly, encryption and verification both use the same numerical operation with the public key, where $m = (m^e)^d = (m^d)^e$ (mod N).

3.2.2 Notes on RSA Practical Applications

To indicate some of the applications of the "textbook" RSA cryptosystem as described above, this thesis presents several simple practical issues below.

1. Random Number Generator

To generate the primes p, select a random number of bit length $k/2$, where k is the required bit length of the modulus N; set the low bit (this ensures the number is odd) and set the two highest bits (this ensures that the high bit of N is also set); check if prime using the algorithm of prime test; if not, increment the number by two and check again until you find a prime. Repeat for q starting with a random integer of length $k-k/2$. Alternatively, instead of incrementing by 2, just generate another random number each time.

2. Choices for *e*

In practice [36], to speed up encryption common, choices for e are 3, 5, 17, 257 and 65537 (2^{16}+1). These particular values are chosen because they are primes and make the modular exponentiation operation faster, having only two bits of value 1. Now suppose it is tempting to use small encryption exponents, such as $e = 3$. The sender A is only sending a very small message $0 < m < N^{1/3}$ to recipient B; Then $c = m^3$ in $\mathbb{Z}_N$, i.e., no modular reduction has taken place. An adversary can therefore compute the message m from the ciphertext c by taking cube roots in N using numerical analysis techniques. The usual choice for e is as large as 65537 = 0x10001. Also, having chosen e, it is simpler to determinate whether $gcd(e, p-1)=1$, and $gcd(e, q-1)=1$ while generating and testing the primes p and q. Values of p or q that fail this test can be rejected there.

3. Security of RSA Algorithm

A good encryption scheme should make it possible that an adversary to learn absolutely nothing about a message from the ciphertext. Nevertheless, some sorts of attacks may be serious for certain applications. Hence, the security of RSA cryptosystems is based on integer factorization as well as cryptosystems based on the discrete logarithm problem. In addition, to secure RSA against attacks of various form, one effective work as far as the mathematics concerned is making the encryption process randomized. The other is feasible by using padding schemes that encode short messages as sufficiently large integers.

4. Extended Euclidean Algorithm

The modular inverse d of an integer e is defined as the integer value such that $ed \equiv 1 \bmod \varphi(N)$. It only exists if e and φ have no common factors. To obtain the value for a large number d, the Extended Euclidean Algorithm is used to calculate a modular inversion written as $d = e^{-1} \bmod \varphi(N)$. As present in [37], the idea of this algorithm is that if we need to find $d = e^{-1} \bmod N$ and we can find integers x and y such that $ex + Ny = 1$, then the inverse d is the value of x. The Extended Euclidean Algorithm is a novel way of doing what the Euclidean algorithm did above. It comprises using three extra values to compute *mx + ny = gcd(m, n)* described in the Table 2.

Extended Euclidean Algorithm Computing the Polynomial Great Common Divisor ([37])

Input: two non-negative integers m and n with $m \geq n$
Output: d = gcd(m, n), and the coefficients x, y satisfying $mx + ny = d$

```
Procedure Extended_Euclidean_gcd(m, n)
    If n = 0, then
        Set d = m
        Set x = 1 and y = 0
        return (d, x, y)
    Set coeff_x2 = 1 and coeff_x1 = 0
    Set coeff_y2 = 0 and coeff_y1 = 1
    While n > 0, do
        q = floor(m/n)
        tmp_r = m – qn
        x = coeff_x2 – q.coeff_x1
        y = coeff_y2 – q.coeff_y1
        Set m = n and n = tmp_r
        coeff_x2 = coeff_x1
        coeff_x1 = x
        coeff_y2 = coeff_y1
        coeff_y1 = y
    Set d = m
    Set x = coeff_x2  and y = coeff_y2
    return (d, x, y)
```

Table 2. Extended Euclidean algorithm

5. Using the Chinese Remainder Theorem (CRT) with RSA

When using RSA algorithm, a practical issue needs to be considered: Is the cryptosystem efficient in the sense of computation time and ciphertext size? Key generation is only carried out occasionally and so computational efficiency is less of an issue. In the encryption and decryption, the basic calculation is known as modular exponentiation. The time to carry this out increases with the number of bits in the exponent. For encryption, an appropriate choice of *e* can reduce the computational effort required for the modular exponentiation using the binary left-to-right method. However, as stated in [37], it is not secure to contrive a specific value in bits set to one for the decryption exponent *d*. So, decryption using the standard method of modular will take longer than encryption.

With the RSA cryptosystem one can decrypt the ciphertext c with RSA private key (d) by computing c^d (mod N). This comes to calculate the modular exponentiation that is computationally expensive in the resource-limited devices. Assume a modulus N of k bits, the private exponent d will also be of similar length, with approximately half being '1'. The cost to compute the exponent is proportional to k^3, which have a lot computational overhead. This can be computed efficiently using the Chinese Remainder Theorem (CRT).

The Chinese Remainder Theorem [37]. Let n_1, n_2 ..., n_r be positive integers such that $gcd(n_i, n_j)$=1 for $i \neq j$. Then the system of linear congruence

$x \equiv c_1$ (mod n_1); $x \equiv c_2$ (mod n_2); ... ; $x \equiv c_r$ (mod n_r)

has a simultaneous solution which is unique modulo $n_1 n_2 \ldots n_r$.

CRT states that there is a unique solution yet actually no telling about how to solve it. Another Gauss's algorithm will be needed to solve simultaneous linear congruence from number theory.

Gauss's algorithm [37]. Let N=$n_1 n_2 \ldots n_r$, then $x \equiv c_1 N_1 d_1 + c_2 N_2 d_2 + \ldots + c_r N_r d_r$ (mod N), where N_i= N/n_i and $d_i \equiv N_i$-1 (mod n_i).

The latter modular inverse d_i is easily calculated by the extended Euclidean algorithm. The details of CRT algorithm from [38] are as below.

a) Given p, q with $p > q$, precompute the following values.

dP = (1/e) mod (p-1)

dQ = (1/e) mod (q-1)

$qInv$ = (1/q) mod p

, where the expression (1/e) notation means the modular inverse, also written as e^{-1}, and x is any integer that satisfies $ex \equiv 1$ (mod N). In this case, where $N = n = pq$, we use the unique value x in $\mathbb{Z}_N$.

b) Given c, compute the message m to do

$m_1 = c^{dp}$ mod p

$m_2 = c^{dQ}$ mod q

$h = qInv\ .\ (m_1 - m_2) \bmod p$

$m = m_2 + h.q$

The private key is stored as the quintuple (p, q, dP, dQ, $qInv$).

The CRT representation of numbers in $\mathbb{Z}_N$ can be used to perform modular exponentiation about four times more efficiently using extra variables precomputed from the prime factors of *N*.

3.2.3 Key Length

Generally, the key length of an RSA key refers to the length of the modulus *N* in bits. The minimum recommended key length for a secure RSA transmission is currently at least 1024 bits. A key length of 512 bits is no longer considered secure, although cracking it is still not a trivial task for the non-cryptanalyst. Typical bit lengths are k=1024, 2048, 3072, 4096, ..., with increasing computational expense for larger values.

The longer the key is used, the stronger the encrypted information would be kept secure. Note RSA can't encrypt anything larger than its modulus, which is generally less than or equal 4096 bits. When defining the key length, one convention, sometimes used, is that the key length is the number of bytes needed to store *N* multiplied by eight, i.e. $(\log_{256}(n+1))\times 8$.

3.3 Hardness Assumptions Relative to RSA

Cryptographic algorithms are designed around certain computational hardness assumptions, rendering such algorithms hard to break in practice by any opponent. A computational problem is specified by the input in a form and an output of certain properties related to the input.

Integer Factorization (IF) Assumption

Public-key algorithms are based on the computational hardness of various problems. The most famous of cryptographic hardness assumptions (see [37]) is integer factorization (e.g., the foundation of RSA algorithm). The presumed difficulty is at the heart of RSA algorithm based on a problem relative to large integer factoring. The integer factorization problem states, given a composite number n, find two primes p and q such that their product is n, more generally, find primes $p_1,\ldots,p_k$ such that $n = \prod_i p_i$. It is a major challenge to find an effective algorithm for solving large integer factorization that runs in polynomial time.

Diffie–Hellman Problem (DHP)

Diffie–Hellman problem is a computational assumption about a certain problem involving discrete logarithms. In cryptography, the discrete log problem is the most important, for if it could be solved, all other relevant problems could also be solved. Much public-key cryptosystem concerns numerical algorithms is built on top of this computational problems. Given two group elements g and h, the discrete logarithm problem ([27]) finds an integer x such that $h = g^x$. Compared the discrete log problem with integer factorization, their computational complexities are closely related.

Most cryptographic protocols related to the discrete log problem actually rely on the stronger Diffie–Hellman problem [27]: Given group elements g, g^x and g^y, where x and y are random integers, the DHP is to compute g^{xy}. For certain groups, the DHP is difficult to be solved efficiently.

Chapter 4
RSA-based CP-ABE Scheme with Constant-size Keys and Ciphertexts on AND-gate Access Structure

At present, in pairing-based CP-ABE schemes, the length of the keys and ciphertexts depend on the number of attributes. Building on an earlier work of [5], this paper generalizes its technology and yields a new variant of RSA-based CP-ABE scheme with constant-size keys and ciphertexts. Moreover, to improve the efficiency, the complexity of the original primitive is reduced to make it feasible in the infrastructure of IoT. Nevertheless, there will always exist a tradeoff between efficiency and reliability. Under the factoring assumption and computationally hard problem, this CP-ABE scheme was provably secure against adaptive Chosen Ciphertext Attack (CCA) and collusion attack. This paper introduces the new algorithm with efficient property to implement both constant-size keys and ciphertexts.

4.1 Preliminary

The existing conventional cryptography based solutions are employed by combining public key cryptography and symmetric cryptography, which are called hybrid cryptosystems. This CP-ABE scheme follows the implementation of hybrid cryptosystem [20], in which the less efficient public-key cryptosystem is most often used to securely deliver symmetric key, while the bulk of the work in encryption/decryption is accomplished by the more efficient symmetric algorithm. To be precise, the generated ciphertext is composed of two parts: One is the encrypted file (data) and the other is the access policy in cipher concatenated as the header. Because Advanced Encryption Standard (AES) was developed as a secure and efficient alternative to the resource intensive Data Encryption Standard (DES) [19], for efficient encryption, the plaintext file is encrypted by AES using a random AES key, which is encoded by the CP-ABE scheme. Only if the access policy for the ciphertext is satisfied, the random AES key is extracted and the cipher file encrypted with AES is able to be decrypted.

AND-gate Access Structure

Formally, this section starts with the definitions of the attribute. Let $U = \{A_1, A_2, \cdots, A_n\}$ be the universe of n attributes, where A_i is one attribute with subscript i, $i = 1, 2, \cdots, n$.

An attribute set of a user is denoted by A ⊆ *U*, and A is represented with an *n*-bit string $a_1a_2\cdots a_n$ defined as follows [5]:

$$\begin{cases} a_i = 1, & \text{if } A_i \in \text{A} \\ a_i = 0, & \text{if } A_i \notin \text{A} \end{cases}$$

For example, assume $n = 4$ and A = $\{A_1, A_3, A_4\}$, the 4-bit string corresponding to A would be 1011. In this scheme, the AND gate access structure is utilized to specify an access policy *P* composed of attributes in *U*, and *P* is represented with an *n*-bit string $b_1b_2\cdots b_n$ defined as follows:

$$\begin{cases} b_i = 1, & \text{if } A_i \in \text{P} \\ b_i = 0, & \text{if } A_i \notin \text{P} \end{cases}$$

For example, for $n = 4$, *P* with the set of the attributes $\{A_1, A_3\}$ is denoted by the string 1010.

Definition 1 [5]. Suppose that attribute set A is labeled with an n-bit string $a_1a_2\cdots a_n$, and the access policy *P* is associated with an n-bit string $b_1b_2\cdots b_n$. Then, *P* ⊆ A if and only if $a_i \geq b_i$, for all *i* from 1 to *n*. We call that the attribute set A fulfills the access policy *P* if and only if *P* ⊆ A.

Complexity Assumptions

This scheme is based on the following two computational problems: Integer Factorization Problem (IFP) and Computational Diffie-Hellman Problem (CDHP). The complexity assumptions are stated below.

Definition 2 **(IFP [5])**. Suppose that RSA modulus $N = pq$, where *p* and *q* are unknown ρ-bit primes. Let *B* be a polynomial-time algorithm which takes an input *N* with 1^ρ and outputs (*p*, *q*). Integer factorization problem related to *B* states that given *N*, it is computationally infeasible problem to derive *p* and *q*, except with negligible probability.

Definition 3 **(CDHP [5]).** The problem of breaking the Diffie-Hellman scheme dependent on an RSA modulus $N = pq$ and base *g* is equivalent to the problem of computing a value of the following function: CDHP(N, g, g^x, g^y)= g^{xy} (mod *N*). As claimed in [29], any algorithm that will break the CDHP for a non-negligible proportion of the possible inputs can be used to factor the modulus *N*. This implies any algorithm that will break the CDHP for a given modulus N can also be used to break the original Diffie-Hellman scheme for the desired element relatively prime to N in the group $\mathbb{Z}_N$.

Key Construction on Defined Access Structure

In this section, the key construction in the defined access structure will be discussed. It is inspired by the scheme [28], which is proven secure against key recovery attacks. The basic idea is that consider $\mathbb{Z}_N$ be the set of congruence classes of the integers modulo $N = pq$, where p and q are RSA secure primes with $p \neq q$. For any non-zero integer $a \in \mathbb{Z}_N$ chosen relatively prime to N, that is $gcd(a, N) = 1$, there exists a multiplicative inverse b for a (mod φ(N)) if and only if $ab \equiv 1 \pmod{\varphi(N)}$ or $b \equiv a^{-1} \pmod{\varphi(N)}$ holds. In modular arithmetic, multiplicative inverse can be computed efficiently using the extended Euclidean *gcd* algorithm, where $1 \in \mathbb{Z}_N$ is the multiplicative identity. The elements with a multiplicative inverse form a finite group that can be viewed as a subgroup of $\mathbb{Z}_N$ under multiplication modulo N, denoted $\mathbb{Z}_N^*$. The key construction in this CP-ABE scheme is to choose RSA key pairs corresponding to the attributes A_i that are (e_i, d_i). The key construction is described below.

Pick a secure prime number e_i with $gcd(e_i, \varphi(N)) = 1, \forall i = 1, 2, \cdots, n$, to each attribute $A_i \in U$. Then, compute the inverse d_i of e_i such that $e_i d_i \equiv 1 \pmod{\varphi(N)}$, where $e_i \neq e_j$ if and only if $i \neq j$. Let $\{\varphi(N), d_1, \cdots, d_n\}$ be secret parameters, and make $\{N, e_1, \cdots, e_n\}$ public parameters. Since factoring the product $N = pq$ is computationally hard problem described by [5], computing $\varphi(N) = (p-1)(q-1)$ without the knowledge of secure primes p and q is also computationally infeasible. This implies that computing the secret primes d_i using the corresponding public prime e_i is based on the integer factorization problem, and as a result, computing the prime d_i such that $e_i d_i \equiv 1 \pmod{\varphi(N)}$ is also computationally hard problem. This joint RSA-pair generation based on a subset product construction also makes it possible to implement the system of encryption and decryption.

Proposition and Proof

Choose a random number g satisfying $2 < g < N - 1$ and $gcd(g, N) = 1$, and compute the secret keys K_A and K_P associated with the attribute set A and access policy P, respectively, as follows:

$$K_A = g^{d_A} \pmod{N},$$

$$K_P = g^{d_P} (\text{mod N}),$$

where $d_A = \prod_{i=1}^{n} {d_i}^{a_i}$, $a_i \in$ A, and $d_p = \prod_{i=1}^{n} {d_i}^{b_i}$, $b_i \in P$.

◆ **Proposition 1 [5]**

The attribute set A fulfills access policy P (that is, $P \subseteq$ A) if and only if $\frac{e_A}{e_P}$ is an integer, where $e_P = \prod_{i=1}^{n} {e_i}^{b_i}$, $b_i \in P$ corresponding to the access policy P, $e_A = \prod_{i=1}^{n} {e_i}^{a_i}$, $a_i \in P$ corresponding to the user attribute set A. In this case, $\frac{e_A}{e_P}$ written as $\frac{e_A}{e_P} = \prod_{i=1}^{n} {e_i}^{a_i - b_i}$ is an integer, and $K_P = k_A^{\frac{e_A}{e_P}}$ (mod N).

Proof [5]. Suppose that A does not fulfill P, that is, $P \nsubseteq$ A. Then, $a_i - b_i \in \{-1,0,1\}$ as a_i, $b_i \in \{0,1\}$.This implies that in the fraction $\frac{e_A}{e_P}$ at least one inverse term exists, such as e_j^{-1}, and thus, computing e_j^{-1} without factoring $N = pq$ is a hard problem. As a result, $\frac{e_A}{e_P}$ cannot be an integer when $P \nsubseteq$ A.

◆ **Proposition 2**

Let an attribute set A of a user is represented with an n-bit string a_A, written as $a_1a_2 \cdots a_n$, where $a_i = 1$, if $A_i \in$ A, and $a_i = 0$, if $A_i \notin$ A. Let an access policy P is represented with an n-bit string b_p, written as $b_1b_2 \cdots b_n$, where $b_i = 1$, if $A_i \in$ P, $b_i = 0$, if $A_i \notin$ P. The attribute set A fulfills access policy P (that is, $P \subseteq A$) if and only if a_A & $b_p = b_p$ and $a_i \geqslant b_i$, for all $i = 1, 2, \cdots, n$.

Proof. Suppose that A fulfill P, that is, $P \subseteq$ A. Then, a_i & $b_i = b_i$ as $a_i \geqslant b_i$, and a_i, $b_i \in \{0,1\}$. Thus a_A & $b_p = b_p$ holds.

◆ **Deduction 1**

On the other hand, if $P \subseteq$ A, the secret key K_p is generated as follows.

$$K_P = k_A^{\frac{e_A}{e_P}} \quad (\text{mod } N)$$

$$= (g^{d_A} \ (mod\ N))^{\frac{\prod_{i=1}^{n} {e_i}^{a_i}}{\prod_{i=1}^{n} {e_i}^{b_i}}} \ (\text{mod } N)$$

$$= g^{(\prod_{i=1}^{n} {d_i}^{a_i})(\prod_{i=1}^{n} {e_i}^{a_i - b_i})} \ (\text{mod } N)$$

$= g^{(\prod_{i=1}^{n} {d_i}^{a_i-b_i+b_i})(\prod_{i=1}^{n} {e_i}^{a_i-b_i})} \pmod N$

$= g^{(\prod_{i=1}^{n} {d_i}^{b_i})(\prod_{i=1}^{n} (e_i d_i)^{a_i-b_i})} \pmod N$

$= g^{(\prod_{i=1}^{n} {d_i}^{b_i})} \pmod N$

$= g^{d_P} \pmod N$

And return the result.

Notations List:

$N = pq$	RSA modulus with large primes p and q, $p \neq q$
$\mathbb{Z}_N$	Set of congruence classes of integers modulo N
$\mathbb{Z}_N^*$	Multiplicative group of integers modulo N
φ(.)	Euler's phi (totient function) $\varphi(N) = (p\text{-}1)(q\text{-}1)$
n	Number of attributes in attribute set U $\lvert U\rvert$
U	Attribute Universe $U = \{A_1, A_2, \cdots, A_n\}$
A	Set of user attributes $A \subseteq U$
P	Access policy $P \subseteq A$
M	Random chosen AES key

4.2 Description of Construction

This scheme provides constant-size secret keys and ciphertexts without using bilinear maps. That is no prime order pairing and instead the group $\mathbb{Z}_N$ is a group of congruence classes of integers modulo N where $N = pq$. The RSA-based CP-ABE cryptosystem works on $\mathbb{Z}_N$ and its subgroup written in multiplication notation as $\mathbb{Z}_N^*$ or $(\mathbb{Z}/n\mathbb{Z})^*$, which is called multiplicative group of integers modulo N. The key pair generation is constructed on the integer factoring. This encryption scheme is also designed to be based on discrete log problem. The idea is that exponentiation to the power e modulo N is easy to compute but such that it is hard to compute pre-images. In fact, exponentiation modulo N is a one-way permutation on $\mathbb{Z}_N^*$ when e is co-prime to φ(N). Exponentiation and Modular multiplication are most frequent operations in the discrete log-based scheme.

A. SETUP PHASE

Let $U = \{A_1, A_2, \cdots, A_n\}$ be the universe of attributes. Additionally, the security parameter ρ as input will determine the bit size of the groups. This setup algorithm consists of the following steps.

A1. Choose two RSA primes p and q with $p \neq q$, and compute $N = pq$. Then, randomly select the RSA public exponent e_i with $\gcd(e_i, \varphi(N)) = 1$, and compute d_i such that $e_i d_i \equiv 1 \pmod{\varphi(N)}$ corresponding to each attribute $A_i \in U$, $\forall i = 1,2,\cdots,n$. Further, pick two system private keys k and x that satisfy $gcd(k, \varphi(N)) = 1$ and $gcd((k+x), \varphi(N)) = 1$. Next, select a random number g such that $gcd(g, N)=1$ and $2 < g < N-1$.

A2. Compute the public parameters $D_U = g^{d_U}$, $Y = g^x$ and $R = g^k$, where $d_U = \prod_{A_i \in U} d_i$.

A3. Finally, publish the master secret key MSK and master public key MPK, where

MSK = $\{\varphi(N), k, x, d_1, \cdots, d_n\}$, MPK = $\{N, n, e_1, \cdots, e_n, Y, R, D_U\}$.

B. ENCRYPT PHASE

The new algorithm takes an access policy $P \subseteq U$, where $|P| \neq 0$, the master public key MPK and a hidden message M (e.g., random chosen AES key) as inputs. The value M is not usually an actual plaintext message or document (which might be huge) but a short integer that is the output of some element or function. The ciphertext $C = \{E_p, E, Y_m, R_m\}$ is output using the following computations.

B1. Compute $E_p = \{e_p, b_p\}$, $e_p = \prod_{i=1}^{n} {e_i}^{b_i}$, $b_i \in P$ corresponding to the access policy P, b_p is the n-bit string $b_1 b_2 \cdots b_n$.

B2. Calculate K_m as $K_m = K_p^h = D_u^{h\frac{e_U}{e_P}}$

$$= (g^{d_U})^{h\frac{e_U}{e_P}} = g^{d_P h},$$

From Deduction 1, K_P is computed as $K_P = D_u^{\frac{e_U}{e_P}} = (g^{d_U})^{\frac{e_U}{e_P}} = g^{d_P}$

B3. Choose a number h at random from $\mathbb{Z}_N$, compute:

$E = MK_m = MK_p^h = Mg^{d_p h}$ $d_p = \prod_{i=1}^{n} {d_i}^{b_i}$, $b_i \in P$

$Y_m = g^{xh}$, $R_m = g^{kh}$

In the end, output the ciphertext C as $C = \{E_p, E, Y_m, R_m\}$

C. KEYGEN PHASE

In this phase, the key generation algorithm takes a user attribute set A, master public key MPK and master secret key MSK as inputs, and then generates a user secret key K_U as follows.

C1. Compute $d_A = \prod_{i=1}^{n} {d_i}^{a_i}$, $a_i \in$ A, where a_i= 1 if $A_i \in$ A and a_i=0 if $A_i \notin$ A;

$E_A = \{e_A, a_A\}$, $e_A = \prod_{i=1}^{n} {e_i}^{a_i}$, $a_i \in$ A corresponding to the user attribute set A, a_A is an n-bit $a_1a_2 \cdots a_n$.

C2. Use k and x in master public key MPK to compute the inverse of (k+x) as inv=$\frac{1}{k+x}$ (mod φ(N)), and $K_1 = inv * d_A$ (mod φ(N)). K_U is then published as $\{E_A, K_1\}$.

D. DECRYPT PHASE

The decryption algorithm takes the secret key $K_U = \{E_A, K_1\}$ corresponding to the user attribute set A and ciphertext $C = \{E_p , E, Y_m , R_m\}$ corresponding to the access policy P, and outputs the hidden message M doing the following:

D1. From Proposition 2, a_A & $b_p = b_p$ holds, if and only, if $P \subseteq$ A. In this case, compute

$$\begin{aligned} K_m &= (Y_m R_m)^{k_1 \frac{e_A}{e_P}} \\ &= (g^{xh} g^{kh})^{\frac{d_A}{(k+x)} \frac{e_A}{e_P}} \\ &= (g^{hd_A})^{\frac{e_A}{e_P}} \\ &= g^{hd_P} = {K_P}^{h} \end{aligned}$$

Otherwise, a_A & b_p is not equal to b_p; $\frac{e_A}{e_P}$ is not an integer, thus, computation of K_m is infeasible.

D2. Compute $M = E/k_m$ to obtain M (the AES key), otherwise, output null ($\perp$).

4.3 Security Analysis

In this section the author proceeds to analyze the model of security for proposed RSA-Based CP-ABE with constant-size keys and ciphertexts (CPABE-CSKCT) encryption. Selective security for CP-ABE aims at the indistinguishability of messages and the collusion-resistance of secret keys, which implies that attackers cannot generate a new secret key by combining their secret keys see ([12]).

Security Model for CP-ABE

A selective-set model in [4] was defined for proving the security of CP-ABE schemes.

Init The adversary declares the set of attributes γ that he wishes to be challenged upon.

Setup The challenger runs the setup algorithm of CP-ABE and publishes the public parameters to the adversary.

Phase 1 The adversary is allowed to issue queries for repeated private keys corresponding to sets of attributes $\gamma_1 \ldots, \gamma_q, \gamma_j \in \gamma$ for all j.

Challenge The adversary submits two equal length messages M_0 and M_1. Additionally, he gives a challenge access structure $\mathbb{A}$ such that none of the sets γ_j from Phase 1 satisfy $\mathbb{A}$ .The challenger flips a random coin v and encrypts M_v with access structure $\mathbb{A}$. The ciphertext is passed to the adversary.

Phase 2 Phase 1 is repeated with the condition on the event that none of sets of attributes $\gamma_j \in \gamma$ satisfy the access structure $\mathbb{A}$ for all j.

Guess The adversary outputs a guess v' of v.

The advantage of an adversary A in this game is defined as $Pr[v' = v] - \frac{1}{2}$ and this security model can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2.

Definition 4 An attribute-based encryption scheme is secure in the Selective-Set model of security if all polynomial-time adversaries have at most a negligible advantage ϵ in such game.

From the definition 4, the author notes that the advantage of an adversary A in this game is $\epsilon := |Pr[v' = v] - \frac{1}{2}|$. Thus, we have $Pr[v' = v] = \frac{1}{2} + \epsilon$. Now the author defines the model of security for the proposed RSA-Based CPABE-CSKCT.

Security Definitions and Proofs of the Model

In [29], They declare a variation of the Diffie-Hellman key distribution scheme working on the multiplicative group $\mathbb{Z}_N^*$ with composite n that in effect combines the security of the original scheme with the difficulty of factoring large integers. It proves a precise theorem concerning the system security, from which it follows that any algorithm that will break the Computational Diffie-Hellman (CDH) scheme for a non-negligible proportion of the possible inputs can be used to factor the modulus N. With this mind, this paper proposes the model of n-IF-CDH of security as follows.

A. Security Definitions of IF and CDH

For a probabilistic polynomial-time (PPT) algorithm $\mathcal{B}$, the factoring advantage is defined by the following factoring hardness assumption. Let Gen be an algorithm which takes an input 1^ρ and outputs (N, p, q). For an algorithm $\mathcal{B}$, its factoring advantage follows that of [30] as

$$Adv_{1Gen.\mathcal{B}}^{IFP}(\rho) := Pr[(N, p, q) \leftarrow IGen(1^\rho) : \mathcal{B}(N) = \{p, q\}].$$

Define that for $\mathcal{B}$, $(t_{IFP}, \epsilon_{IFP})$-IF holds if $\mathcal{B}$ runs in time t_{IFP} and $Adv_{1Gen.\mathcal{B}}^{IFP}(\rho) \leq \epsilon_{IFP}(\rho)$. The factoring assumption (with respect to $\mathcal{B}$) states that $Adv_{1Gen.\mathcal{B}}^{IFP}(\rho)$ is negligible in ρ for every PPT $\mathcal{B}$.

Assume the advantage of adversary $\mathcal{A}$ in solving the CDH is: $Adv_{Z_N.\mathcal{A}}^{CDH}(\rho) = Pr[\mathcal{A}(N, g, g^a, g^b) = g^{ab}]$. We say that $(t_{CDH}, \epsilon_{CDH})$-CDH assumption holds if $Adv_{Z_N.\mathcal{A}}^{CDH}(\rho) \leq \epsilon_{CDH}$, for any sufficiently small $\epsilon_{CDH} > 0$, with its running time at most t_{CDH}.

The (t,ϵ)-Hard n-IF-CDH problem defines as follows. A t-polynomial time algorithm $\mathcal{B}$, which outputs a bit $\gamma \in \{0,1\}$, has an advantage $Adv_{Z_N.\mathcal{B}}^{IF-CDH}(\rho) = \epsilon$ in solving the n-IF-CDH problem in $\mathbb{Z}_N$, where

$$Adv_{Z_N.\mathcal{B}}^{IF-CDH}(\rho) = Pr[\mathcal{B}(N, e_1, \ldots, e_n, g, g^k, g^x, g^{kh}, g^{xh}, g^{d_P}, g^{d_P h}) = T],\ T = g^{d_P h} \in \mathbb{Z}_N.$$

Then, n-IF-CDH problem reduces to decide whether T is equal to $g^{d_P h}$ or a random element in $\mathbb{Z}_N$.

B. Security Assumption on the Specific Construction of Secret keys

In this scheme, the valid secret user key K_U corresponding to the attribute set A consists of $d_A = \prod_{i=1}^{n} d_i{}^{a_i}$, where $a_i \in A$, which could be derived from the inverse d_i of e_i modulo φ(N). Assume $\prod_{i=1}^{n} d_i{}^{a_i} \neq \prod_{j=1}^{n} d_j{}^{a_j}$, $a_i \in A$, $a_j \in A'$ for $A \neq A'$. If there exist A and A' ($A \neq A'$) such that $\prod_{i=1}^{n} d_i{}^{a_i} = \prod_{j=1}^{n} d_j{}^{a_j}$, $a_i \in A$, $a_j \in A'$, a user with the attribute list A' can decrypt a ciphertext associated with P, where $P \nsubseteq A'$ and $P \subseteq A$.

Remark1 that the assumption holds with overwhelming probability $\frac{N(N-1)\ldots(N-(a-1))}{N^a} > \frac{(N-(a-1))^a}{N^a} = (1 - \frac{a-1}{N})^a > 1 - \frac{a(a-1)}{N} > 1 - \frac{a^2}{N}$, where a := $\sum_{i=1}^{n} a_i$, $a_i \in A$. Therefore, if each secret key d_i is chosen at random from $\mathbb{Z}_N$, then the assumption is naturally hold.

C. Proof of Security

Theorem 1 This scheme satisfies the indistinguishability of messages under the n-IF-CDH assumption.

Proof. Suppose that the adversary $\mathcal{A}$ wins the selective game for CP-ABE with the advantage ϵ. We can then construct an algorithm $\mathcal{B}$ that breaks the IF-CDH assumption with the advantage $\frac{\epsilon}{2}(1 - \frac{a^2}{N})$, where $a := \sum_{i=1}^{n} a_i$, $a_i \in A$,a_i= 1 if $A_i \in A$ and a_i=0 if $A_i \notin A$ is the number of expressed attributes for a user's attribute set.

The adversary $\mathcal{A}$ is allowed to issue queries on the set of attributes for private keys. $\mathcal{B}$ responds them with the valid decryption keys that don't satisfy access policy P'. In the challenge phase of the selective game, the adversary $\mathcal{A}$ will submit two challenge messages m_0 and m_1 to the algorithm $\mathcal{B}$. $\mathcal{B}$ flips a fair binary coin v and computes a ciphertext of m_v. The ciphertext is output as: $C' = \{ E_P', E' = m_v K_m', Y_m', R_m' \}$, where $K_m' = g^{d_{p'} h'}$, $d_{p'} = \prod_{i=1}^{n} d_i{}^{b_i}$, $b_i \in P'$, corresponding to the challenged access policy P'. Since h' is a random number in Z_N, and all d_i will be chosen independently at random as in the construction. Therefore, $K_m' = g^{d_{p'} h'}$ is a random group element and the ciphertext is a valid random encryption of message m$_v$. Also, the challenged ciphertext C' is indistinguishable with a real ciphertext. The adversary outputs a guess v' of v and wins the game if $v' = v$. In this case, there exists A' such that $\prod_{i=1}^{n} {d_i}^{a_i} = \prod_{j=1}^{n} {d_j}^{a_j}$, $a_i \in A'$, $a_j \in P'$ holds. According to the above result of Remark 1, this probability is at most $\frac{a^2}{N}$. Otherwise, $K'_m = g^{{d_p}'h'}$ is a random element in $\mathbb{Z}_N$.

Furthermore, the advantage of algorithm $\mathcal{B}$ in solving the CDH problem in the RSA group $\mathbb{Z}_N$ is $Adv_{Z_N.\mathcal{B}}^{CDH}(\rho)$. Let S be an event that the adversary $\mathcal{A}$ queries an element $g^{{d_p}'h'} \in \mathbb{Z}_N$. Then, $Pr[S]$ follows $\mathcal{B}$'s advantage in the CDH game [3]is

$$Pr[S] = \frac{1}{2}(Pr[\mathcal{B} \rightarrow 1|v' = v] + Pr[\mathcal{B} \rightarrow 0|v' \neq v]) - \frac{1}{2} = \frac{1}{2}\frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}*\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$$

From the above analysis, $\mathcal{B}$ knows the private keys, which do not satisfy the challenge ciphertext C'. Then, $g^{{d_p}'h'}$ can be computed only if the CDH problem can be solved in the RSA group $\mathbb{Z}_N$.When $\mathcal{B}$ chooses randomly a set of random element corresponding to the queried attributes in the query list, the probability that a t-polynomial time algorithm $\mathcal{B}$ gains the information about v is equal to $g^{{d_p}'h'}$ given by solving the n-IF-CDH problem. Consequently, the overall advantage of $\mathcal{B}$ is given by $(1-\frac{a^2}{N})Pr[s]$, that is, $Adv_{Z_N.\mathcal{B}}^{IF-CDH}(\rho) = \frac{\epsilon}{2}(1-\frac{a^2}{N})$. Hence, the theorem 1 is proved.

Theorem 2. Under the hardness of solving the integer factorization problem, this scheme is secure against an adversary (also a legitimate user u) for deriving the valid user secret key K_U corresponding to the attribute set A with $P \not\subseteq A$.

Proof: Assume that a group of users u_i with t numbers of attributes, $i = 1,\cdots, t$, corresponding to the attribute set A_i collaborate among each other and try to derive the system private key pair (k, x) using their valid secret keys $k_{U_i} = (K_1^i)$, where

$$k_1^i = inv(\mathrm{k+x}) .d_{A^i} \pmod{\varphi(N)} \qquad (1)$$

From the KEYGEN algorithm of this scheme, we have

$$d_{A_i} = \prod_{i=1}^{n} {d_i}^{a_i} \pmod{\varphi(N)} \qquad (2)$$

Proof. It is clear that if d_{A^i} is unknown, it is impossible to solve k and x. To compute d_{A^i}, the adversary $\mathcal{A}$ can randomly choose d_i in $\mathbb{Z}_N$, and module $\varphi(N)$ in their product. From the security assumption on the secret key, Equation (2) is computationally infeasible and requires to randomly guess $t + 1$ unknowns under the hardness of solving the IF problem. Moreover, to compute the value k_{U_i}, $\mathcal{A}$ requires the system private key pair (k, x) and RSA secret d_{A_i} from Equation (1). Even if d_{A_i} is known, it will have infinitely many solutions to produce the original value of k and x. Furthermore, because Equation (1) and (2) depend on the Euler's totient function $\varphi(N) = (p-1)(q-1)$. Thus, generating the valid user secret key K_U is computationally infeasible problem for $\mathcal{A}$ due to the intractability of the IFP. Hence, it is computationally infeasible problem for the user u to derive the valid secret key K_U corresponding to the attribute set A.

Theorem 3. Under the hardness of solving n-IF-CDH problem, this scheme is secure against deriving the secret K_m from a ciphertext $C = \{E_p, E, Y_m , R_m\}$ by a group of collaborative unauthorized users u_i, who use their corresponding attribute sets. Hence, the scheme has the property of collusion resistance. That is, attackers are not able to generate a new user secret key by combining their secret keys (see [7]).

Proof. This paper proves this theorem for any pair of users and the same argument is then extended for a group of users. Suppose u_1 and u_2 be two users corresponding to the attribute sets A and B, respectively, and try to decrypt the cipher $C = \{E_p, E, Y_m, R_m\}$, where $P \nsubseteq A$ and $P \nsubseteq B$, and $P \subseteq (A \text{ OR } B) = D$. From Theorem 2, It follows that it is computationally infeasible for both u_1 and u_2 to derive the valid secret key k_{U_i} corresponding to the attribute policy D such that $P \subseteq D$. However, they can derive $g^{d_A h}$ and $g^{d_B h}$ using their own secret keys k_{U_1} and k_{U_2} respectively. Let $g_1 = g^h$, and then we have $g^{d_A h} = g_1^{d_A}$ and $g^{d_B h} = g_1^{d_B}$. Since solving the CDH problem in $\mathbb{Z}_N$ is as hard as solving factorization of RSA modulus $N = pq$, then no collaborative users can compute the secret K_m as follows:

$$g_1^{d_A d_B} \leftarrow \text{CDHP}(g_1, g_1^{d_A}, g_1^{d_B}),\ k_m = g_1^{d_P} = ((g_1^{d_A d_B})^{e_C})^{\frac{e_D}{e_P}},$$

where $C = A$ AND B.

It turns out that computing the valid K_m is computationally infeasible for the adversary $\mathcal{A}$. Therefore, the theorem 3 is obtained under n-IF-CDH assumption, and the security of this proposed scheme is proved.

4.4 Implementation

This section will detail the implementation of the CPABE-CSKCT scheme proposed in section 4.2, including the process of programming and a description of the developed toolkit. This cryptosystem is built with C programming language under Ubuntu platform. Only command line is available for the user interface in this system at present. The implementation mainly uses the GNU Multiple Precision arithmetic library (GMP) that is a high-performance arbitrary precision arithmetic toolkit appropriate for cryptography. Additionally, few functions in PBC library are also invocated for the usage of element operations of finite group.

The AES 128-bit (AES-128) in Cipher Block Chaining (CBC) mode is used for the payload encryption. As described in [22], the key size of 128-bit is securely sufficient for most of the objects in the IoT paradigm. The implementation uses OpenSSL library for AES encryption.

The cpabe Toolkit

For the sake of convenient usage, the construction of section 4.2 has been implemented as a set of tools that provides four shell commands as below.

A. cpabe-setup [OPTION ...]

Takes as input file path of an attribute universe, publishes a public key and a master key with respect to the attribute universe.

Options are not mandatory arguments.

-h, --help	print this message.
-p, --output-public-key FILE	write public key to FILE.
-m, --output-master-key FILE	write master secret key to FILE.
-u, --universe FILE	read attribute universe from FILE.

B. cpabe-keygen [OPTION ...] PUB_KEY MASTER_KEY ATTR [ATTR ...]

Given a master secret key, a public key and the listed attributes, generates a private key for the certain set of user attributes declared.

- Command-line parameters description

PUB_KEY:	file for a public key
MASTER_KEY:	file for a master secret key
ATTR	arbitrary number of attributes string.

Attributes come in the form of non-numerical. Non-numerical attributes are simply any string of letters, digits, and underscores beginning with a letter.

- Options are listed as follows.

-h, --help	print this message.
-o, --output FILE	write user private key to FILE.

C. cpabe-enc [OPTION ...] PUB_KEY FILE [POLICY]

Given a public key and a file, encrypts the file under a specified access policy on AND-gate structure. If the access policy is not specified, the policy will be read from stdin.

Command-line parameter is described as below.

PUB_KEY:	file for a public key
FILE:	the original file for encryption
POLICY:	attribute strings with 'and' delimiter.

The keywords 'and' is reserved for the policy language and may not be used for either type of attribute. The whitespace around the attribute is optional.

- Options are not mandatory arguments.

-h, --help	print this message.
-o, --output FILE	write the ciphertexts to FILE.

D. cpabe-dec [OPTION ...] PUB_KEY PRIV_KEY FILE

Given a public key and a private key, decrypts an encrypted file.

- Command-line parameters

PUB_KEY:	file for a public key
PRIV_KEY:	file for a user private key

- Options

-h, --help	print this message.
-o, --output FILE	write output to FILE.

The GMP Library

The GMP library that performs the mathematical operations has a rich set of functions with regular interface. GMP defines the *mpz_t* type for multiple precision integer, *mpq_t* for rational and so on. Some important features in using GMP [43] are:

- Output arguments generally precede input arguments.
- The same variable can be used as input and output in one call.
- Before a variable may be used it must be initialized exactly once. When no longer needed it must be cleared.
- In general, in GMP, if a function modifies an input variable, that variable remains modified when control result is returned to the caller.
- GMP variables automatically allocate memory when needed. By default, malloc() is called but this can be changed.
- GMP functions are mostly reentrant. A reentrant function can be safely called recursively or from multiple tasks.

Serialization and Deserialization

In the cryptosystem, all generated keys and ciphertexts are of file form stored under specific directory. Accordingly, process instances are dependent on these files to conveniently pass the information. However, how to organize the file operation seems a problem as no such built-in function is available. To conduct this operation, the functions of serialization and deserialization have to be designed and written manually. Serialization is the process of writing a data structure to a format, a sequence of bytes, which can be stored in a file, a memory buffer or transmitted over a network. On the other hand, deserialization is instead the reconstruction from this format back into the data structure.

GByteArray structure type is used to generate arrays of arbitrary bytes which grow automatically as elements are added. A sequence of bytes in C is represented by a byte array, i.e. an array of *uint32_t*. Note that serialization is hardware dependent as the endianness of the architecture might differ. Functions for serializing element can be seen in Figure 2 and are described as follows.

```
/* Serialization of an element.
* vector b: byte array where the serialized element will be
*           stored;
*        e: element data
*/
void serialize_element ( GByteArray* b, element_t e )
{
      uint32_t len;
      unsigned char* buf;
       /* length in bytes the element will take to represent
       */
      len = element_length_in_bytes(e);
      /* serialize the length of an element as the head*/
      serialize_uint32(b, len);
      buf = (unsigned char*) malloc(len);
      /* Converts element to byte, writing the result into
      * the buffer */
      element_to_bytes(buf, e);
     /* Adds the given bytes to the end of the GByteArray */
      g_byte_array_append(b, buf, len);
      free(buf);
}
```

Figure 2. Function for Serializing an Element

A serialization function frequently used is serialize_element designed for element data type. The number of bytes it will write can be determined from calling the function element_length_in_bytes. It also makes the length of an element serialized as the head byte array to indicate the following amount of allocation. Then a result array of the necessary size is allocated. Next, it converts the element to byte, writes the result into the buffer. In the end, a byte array for the sequence is produced and appended to a block of storage.

The process of deserialization shown in Figure 3 needs to initialize the element group and $\mathbb{Z}_N$ group before reconstructing the element in them.

```
/* Deserialization of an element
*  Vector b, byte array containing the serialized element;
*         e: element data
*/
void unserialize_element ( GByteArray* b, int* offset,
                        element_t e )
{
      uint32_t len;
      unsigned char* buf;
      /* Deserialization of the length of element in bytes */
      len = unserialize_uint32(b, offset);
      buf = (unsigned char*) malloc(len);
      memcpy(buf, b->data + *offset, len);
      *offset += len;
      /* Reads e from the buffer data */
      element_from_bytes(e, buf);
      free(buf);
}
```

Figure 3. Deserialization Function of an Element

As an example of deserialization, unserialize_element is to read the number of data of bytes from the given byte array and then copy the data in this number of bytes to a allocated buffer. Eventually, it generates the element in the form of binary data using element_from_bytes, which needs to set the element of group with limb space. A limb means the part of a multi-precision number that fits in a single machine word.

Algorithm Description

A. Setup Algorithm

In the setup algorithm, argument vector for input is null or document path of attribute universe. Output arguments are a public key and a master key. To use the results, put the output in the files indicated the keys. The algorithm is present in Algorithm 1.The detailed process proceeds in the following steps.

Algorithm 1: Setup Algorithm in CSKCT

```
Procedure CSKCT_setup(k, universe)
1    begin
2        Select a prime value of k/2 bits for p and q
3        N = pq
4        phi_n = (p-1)(q-1)
5        Initialize(pubkey, mskkey)
6        repeat do
7            g ← random()
8        until gcd(g, N)==1
9        Read universe into attr_buf
10       Parse attr_buf into attr_shm
11       repeat do
12           k ← random()
13       until gcd(k, phi_n) == 1
14       repeat do
15           x ← random()
16       until gcd(x, phi_n) == 1
17       foreach A_i ∈ {A_1, A_i, ⋯,A_n} do
18           repeat do
19               pub_exponent[i] ← random()
20           until gcd(pub_exponent[i], phi_n) == 1
21           prv_inverse[i] ← inverse(pub_exponent[i])
22       end
23       foreach A_i ∈ {A_1, A_2, ⋯, A_n} do
24           du ← du * prv_inverse [i] mod phi_n
25       end
26       Write serialize(pubkey) to pub_file
27       Write serialize(mskkey) to msk_file
28       Output pub_file msk_file
29   end
```

Table 3. Algorithm 1 Setup Algorithm in CSKCT

A1. (see 1-8 of Algorithm 1). If no input, the program will take a default path of document for attribute universe as input file. The main code for this step is in Figure 4.

```
/* process 1). select prime p and q*/
pbc_mpz_randomb(p, k/2); /* random number with k/2 bits length*/
pbc_mpz_randomb(q, k/2);
mpz_nextprime(p, p);     /*identify prime and set*/
mpz_nextprime(q, q);
mpz_mul(N, p, q);    /* N=pq */
/* start 2). find g*/
do {
      element_random(g);          /*randomly chosen g*/
      element_to_mpz(mpz_g, g);
      mpz_gcd(divisor, mpz_g, N);
} while (mpz_cmp_d(divisor, 1)); /*decide the gcd == 1?*/
```

Figure 4. Step 1 of Setup Algorithm in CSKCT

1) For the 1024 group size, randomly select RSA prime *p* and *q* be 512 size using inner random number generator, which read them from the Linux kernel's /dev/urandom in given size.

2) The algorithm needs the test to find a random *g* who satisfy the greatest common divisor of *g* and *N* is only 1. This is accomplished by calling mpz_gcd function through a while loop.

A2. (see 9-10 of Algorithm 1). First, to parse the attributes from the input file, the algorithm needs to read the file into a string buffer allocated the actual size of attribute file. Then, the parse_attribute_list function depicted in Figure 5 is used to collect all of attributes from the given file of attribute universe. It takes the buffer of attribute universe stored in string form as input parameter.

```
/* Parse every attribute stored in the string buffer */
* Vector attrs_str pointing to the sting buffer of attributes
*/
int parse_attribute_list(char * attrs_str)
{     ...
      /* formalize all strings by squeezing out the '\n' and
         blank*/
      squeeze(attrs_str, '\n');
      if ((substr = strtok(attrs_str, BLANK))== NULL)
            parse_attribute(&attrlist, substr);
      while(substr){
            parse_attribute(&attrlist, substr);
            substr = strtok(NULL, BLANK);
      }
      /* strings in the list will be sorted by string
         comparisons*/
      attrlist = g_slist_sort(attrlist, comp_string);
      n = g_slist_length(attrlist);
      attrs = malloc((n + 1) * sizeof(char*));
      /* a fixed-length array of string where the attribute
         strings will be placed in order */
      for( ap = attrlist; ap; ap = ap->next )
            attrs[i++] = ap->data;
      attrs[i] = 0;
}
```

Figure 5. Function for Parsing Attribute Set of Setup Algorithm in CSKCT

The function processes all strings by squeezing out the '\n' and blank such that every string is formalized and extracted into a list by using parse_attribute function. Next, formalized strings in the list will be sorted by size. In the end, a fixed-length array of string objects is allocated to place the attribute strings in order.

Shared memory is the fastest form of IPC (Interprocess Communication) because the data does not need to be copied between the processes. Consequently, the setup algorithm places the array of attribute strings into a shared memory region until the shared memory segment will be detached in the next setup running.

A3. (see 11-22 of Algorithm 1).The procedure of this step can be seen in Figure 6.

```
for (i=0; i< num_attrs; i++) {
      do {
            pbc_mpz_random(pub->ei[i], N); /*random chosen e_i*/
            mpz_gcd(divisor, pub->ei[i], phi_n);
        }while (mpz_cmp_d(divisor, 1)); /*decide the gcd==1?*/
      /*compute the multiplication inverse*/
      mpz_invert(msk->di[i],  pub->ei[i], msk->phi_n);
}
/*select random k with gcd(k, phi_n) = 1 */
do {
      pbc_mpz_random(msk->k, N);
      /*compute the gcd(k, phi_n)*/
      mpz_gcd(divisor, msk->k, phi_n);
}while (mpz_cmp_d(divisor, 1));
```

Figure 6. Step3 of Setup Algorithm in CSKCT

A loop for each attribute is used to randomly select the RSA public exponent e_i and compute the multiplicative inverse d_i of e_i by calling mpz_invert function. Further, pick two system private keys *k* and *x* that satisfy the greatest common divisor between them and φ(*N*) is only 1. This is implemented by calling pbc_mpz_random and mpz_gcd functions, respectively through a while loop.

A4. (see 23-29 of Algorithm 1).The procedure of this step is displayed in Figure 7.

```
/*compute the component d_U */
for (i=0; i< num_attrs; i++) {
      mpz_mul(du, du, msk->di[i]);
      mpz_mod(du, du, msk->phi_n);
}
element_pow_mpz(pub->gk, g, msk->k); /*compute Y = g^x */
element_pow_mpz(pub->gx, g, msk->x); /*compute R = g^k */
element_pow_mpz(pub->g_du, g, du);   /*compute D_U */
/*output public key file and master key file*/
spit_file(pub_file, cskct_pub_serialize(pub),1);
spit_file(msk_file, cskct_msk_serialize(msk),1);
```

Figure 7. Step4 of Setup Algorithm in CSKCT

Multiply all d_i together to compute d_U so that calculate the component D_U of the public parameters with the base g and the element_pow_mpz function. Other public parameters $Y = g^x$ and $R = g^k$ are computed in the same way. Eventually, the algorithm serializes the public key and master secret key to the associated file and outputs this file.

B. Key generation algorithm

In the key generation algorithm, public key, master private key associated files and attributes strings are all necessary input variables. Output is user private key file indicated by -o option. If -o option for the output argument is not input, the default file name will be used by the name of input file with the extension ending in .cpabe. The process is described in Algorithm 2 below.

Algorithm 2: Key Generation Algorithm in CSKCT

```
Procedure CSKCT_keygeneration(pub_file, msk_file, user_attrs)
1    begin
2    attrstr array, num_attrs ← shmget()
3    userattr array ← parse_arguments()
4    foreach userattr[index] ∈ {$A_{index}$ | 0 < index < num_attrs} do
5        foreach attrstr ∈ {$A_1 \cdots A_i$} do
6            If userattr[index] == attrstr[i] then
7                Set ea = attribute bitmap ← 1
8                ai[i] ← 1
9            end
10       end
11   end
12   pubkey ← deserialize(read_file(pub_file) )
13   mskkey← deserialize(read_file(msk_file) )
14   prvkey ← InitPrvParams(pubkey)
15   foreach attrstr ∈ {$A_1 \cdots A_i$} do
16       If ai[i] == 1 then
17           mul_diai = mul_diai * prv_inverse[i] mod phi_n
18           ea = ea * pub_exponent[i] mod phi_n
19       end
20   end
21   add (k, x)
22   Compute invert(k+x) mod phi_n
23   prv_key1 = invert(k+x) * mul_diai mod phi_n
24   Output(serialize(prvkey))
25   end
```

Table 4. Algorithm 2 Key Generation Algorithm in CSKCT

K1. (1-14 of Algorithm 2). The function first obtains the data of attribute universe from the shared memory. In this way, multiple processes are able to communicate one another. The process needs to map the attribute universe into its address spaces so as to access shared attributes memory.

1) After parsing the array of attributes string, this can be compared with that of user attributes input to compute the bitmap of user attributes. This detail can be seen in Figure 8.

```
/* process 1). compare the attributes universe with user
    attributes to obtain the bitmap of user attributes. */
for(i=0; i<num_userattr; i++)
{
      for(index=0; index<num_attrs; index++) {
           if (!strcasecmp(user_attrs[i], attrs[index])){
                attrmap |= 1<<index; /*set attributes bitmap*/
                abit[index] = 1;
                 break;
            }
      }
}
/* start 2). read the input file and deserialize the public key
   and  master key*/
pub = cskct_pub_unserialize(suck_file(pub_file),1);
msk = cskct_msk_unserialize(suck_file(msk_file),1);
/*initialize mpz_t variable in the private key parameters */
prv = init_prv_params(pub);
prv->ea = attrmap;
```

Figure 8. Step1 of Key Generation algorithm in CSKCT

2) Then it executes the deserialization of public key file and master secret key file to obtain the public key and master key; initialize *mpz_t* variables in the private key parameters by calling the mpz_init function for an integer initialization.

K2. (15-20 of Algorithm 2). The procedure is shown in Figure 9. For each bit with a value of 1 in the string bitmap, multiply the corresponding d_i with a loop together using the functions for multiplication and modulus.

```
/* use the multiplication and modulus for each bit == 1 in
   the string bitmap. multiply the corresponding the inverse
   d_i to compute d_A, multiply the corresponding the inverse e_i
   to compute e_A,*/
for( i =0; i< num_attrs; i++ )
{
      if (abit[i] == 1) {
        mpz_mul(prv->mul_diai, prv->mul_diai, msk->di[i]);
        mpz_mod(prv->mul_diai, prv->mul_diai, msk->phi_n);
        mpz_mul(prv->ea, prv->ea, pub->ei[i]);
        mpz_mod(prv->ea, prv->ea, msk->phi_n);
      }
```

Figure 9. Step2 of Key Generation Algorithm in CSKCT

Hereafter, we obtain the result of $d_A = \prod_{i=1}^{n} {d_i}^{a_i}$, if $a_i \in A$ and a_i= 1. Again, we could compute the E_A = e_A = $\prod_{i=1}^{n} {e_i}^{a_i}$, $a_i \in A$ corresponding to the user attribute set A in the same way.

K3. (21-25 of Algorithm 2). The process can be seen in Figure 10.

```
/*compute inv = 1/k+x mod φ(N)*/
mpz_add(tmp, msk->k, msk->x);
if (! mpz_invert(prv->inv_kx, tmp, msk->phi_n))
      return -1;
/*compute user private key K1 */
mpz_mul(prv->k1, prv->inv_kx, prv->mul_diai);
/* serialize user private key, write into the output file*/
spit_file(out_file, cskct_prv_serialize(prv),1);
```

Figure 10. Step3 of Key Generation Algorithm in CSKCT

To compute the inverse of (k+x) as inv=$\frac{1}{k+x}$ (mod φ(N)), k and x in master public key MPK are used to perform the addition and inverse operations. If no inverse exists the return value is zero, this algorithm exits with prompt of that. The component K_1 of the user's secrete key is computed with multiplication result of inv and d_A modulo φ(N). The user's private key is then derived as {E_A, K_1} and written into the corresponding file by serializing.

C. Encryption Algorithm

The necessary argument vector consists of a public key and an input file that will be encrypted. If the access policy is not specified, the policy will be read from standard input. The encrypted file will be output, and the original file will be removed. The following flowchart in Figure 11 describes the entire process of encryption algorithm.

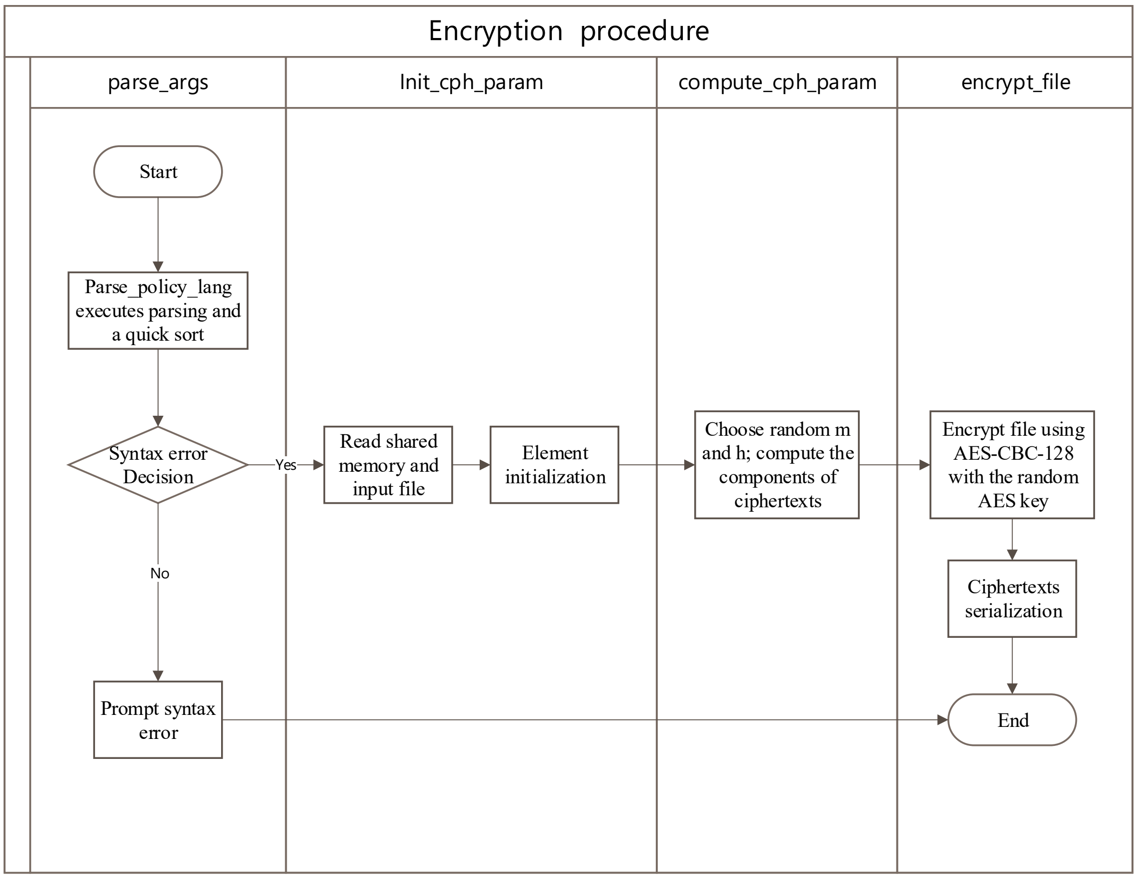

Encryption procedure
parse_args
Init_cph_param
compute_cph_param
encrypt_file
Start
Parse_policy_lang executes parsing and a quick sort
Syntax error Decision
Yes
No
Read shared memory and input file
Element initialization
Choose random m and h; compute the components of ciphertexts
Encrypt file using AES-CBC-128 with the random AES key
Ciphertexts serialization
Prompt syntax error
End

Figure 11. Encryption Procedure in CSKCT

E1. The function parse_polocy_lang calls yyparse to conduct parsing. This function reads tokens, executes actions, and ultimately returns when it encounters end-of-input or an unrecoverable syntax error.

```
/* execute policy language parsing, return the formalized policy
* Vector s,   current policy buffer
*/
char* parse_policy_lang( char* s )
{
      cur_string = s;
      /* reads tokens, parse syntaxes and executes actions */
      yyparse();
      /* execute a quick sort for arrays of string */
      tidy(parse_policy);
      parsed_policy = format_policy_postfix(parse_policy);
      ...
}
```

Figure 12. Function for Parsing Policy Language

This function returns a value of 0 if the input it parses is valid according to the given grammar rules. It returns 1 if the input is incorrect and error recovery is impossible. A quick sort for arrays of string objects is implemented according to string comparisons.

E2. This program of main steps is shown in Figure 13. Each element of the ciphertext structure needs to be initialized before using.

```
cph->ep = attrmap;
/* randomly choose the element m as AES key and h */
element_random(m);
element_random(h);
for( i =0; i< num_attrs; i++ )
      if (bbit[i] == 0)
             mpz_mul(eu_div_ep,  eu_div_ep, pub->pi[i]);
/* compute KP */
element_pow_mpz(cph->m_kph, pub->g_du, eu_div_ep);
element_pow_zn(cph->m_kph, cph->m_kph, h);
element_mul(cph->m_kph, cph->m_kph, m); /* compute E */
```

Figure 13. Step2 of Encryption Algorithm in CSKCT

After initializing, compute K_P and E_p corresponding to the access policy *P*. It assigns uniformly random elements to *m* and *h*. The element m is thought as the concealed AES key. Subsequently, the component *E*, Y_m, R_m in the ciphertext are computed with exponentiation operation, where the base *g* will exponentially power elements or *mpz_t* data.

E3. The file encryption processing reads the input file into arrays of bytes and passes it as one input parameter to Aes-cbc-128-encrypt function for encryption. The key for the AES-128 in CBC mode is taken from random element m. The AES-128 function returns the buffer stored encrypted file, whose size is identical to that of the original file. The algorithm runs the serialization for the ciphertexts, which writes the access control in cipher and the encrypted file into the output file. The implementation of this step can be seen in Figure 14.

```
/* serialize the cyphertext policy*/
cph_buf = cskctabe_cph_serialize(cph);
cskct_cph_free(cph);
/*read the input file into a buffer that will be encrypted*/
plt = suck_file(in_file);
file_len = plt->len;
/* use AES to encrypt file */
aes_buf = aes_128_cbc_encrypt(plt, m);
write_cpabe_outfile(out_file, cph_buf, file_len, aes_buf);
```

Figure 14. Step3 of Encryption Algorithm in CSKCT

D. Decryption Algorithm

Decryption algorithm takes a public key, a user private key and an encrypted file as input. It output the decrypted file. If the name of this encrypted file is X.cpabe, the decrypted file will be written as X and the encrypted file will be removed. Otherwise the file will be decrypted in place. The usage of the -o option overrides this behavior. The entire process of encryption algorithm is shown in the flowchart of Figure 15.

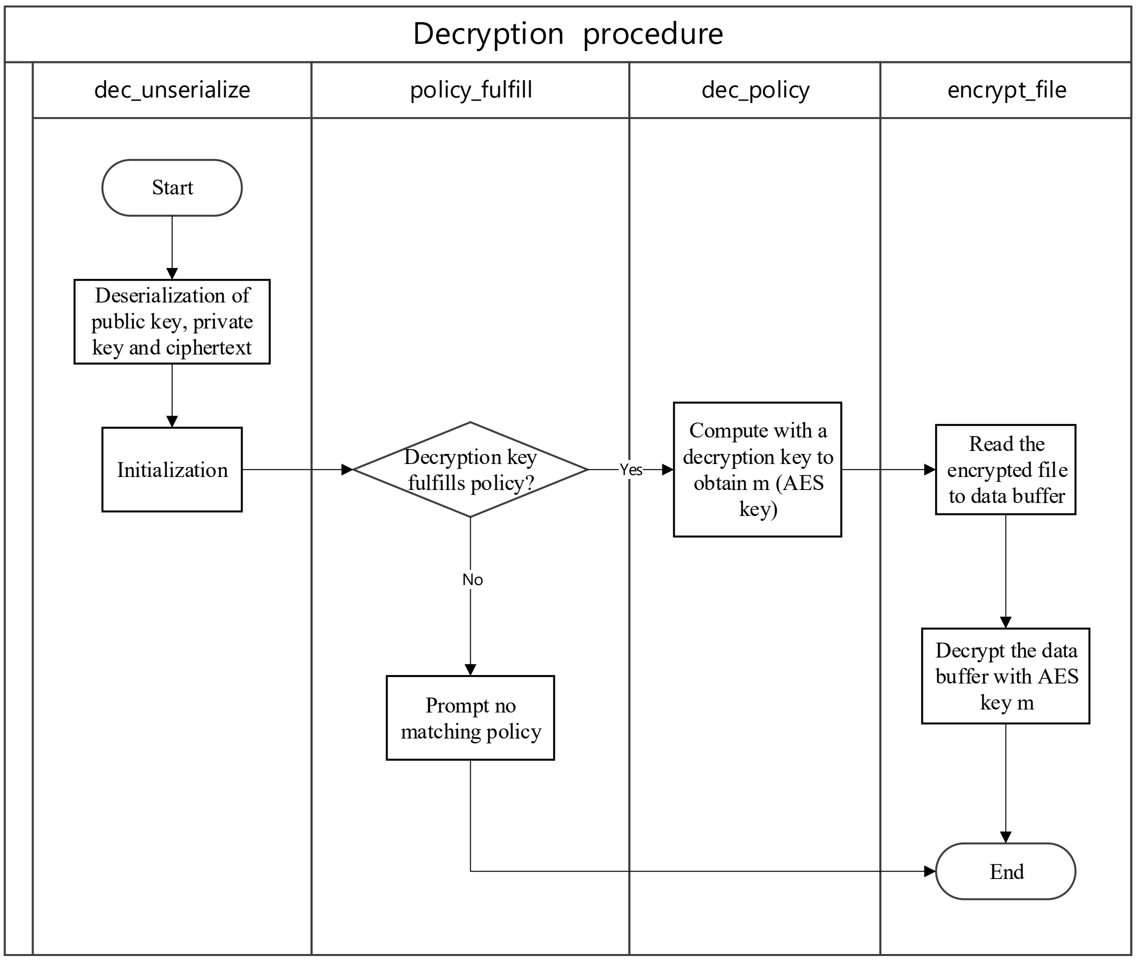

Decryption procedure
dec_unserialize
policy_fulfill
dec_policy
encrypt_file
Start
Deserialization of public key, private key and ciphertext
Initialization
Decryption key fulfills policy?
Yes
No
Compute with a decryption key to obtain m (AES key)
Read the encrypted file to data buffer
Decrypt the data buffer with AES key m
Prompt no matching policy
End

Figure 15. Decryption Procedure in CSKCT

D1. The algorithm executes a series of deserialization for public key, user secret key and ciphertext, whereby are assigned and restored to their structure. Next, it initializes element and *mpz_t* variables that will be used in the decryption by calling initialization function for element and integer. The main steps can be seen in Figure 16.

```
/* deserialization of public key, user secret key and ciphertext*/
pub = cskct_pub_unserialize(suck_file(pub_file),  1);
prv = cskct_prv_unserialize(suck_file(prv_file), 1);
cph = cskct_cph_unserialize(cph_buf, 1);
/* initialization of element and mpz_t data in the decryption*/
element_init(m, Zn);
mpz_init_set_ui(ea_div_ep, 1);
mpz_init(dp);
element_init(kph, Zn);
```

Figure 16. Step1 of Decryption Algorithm in CSKCT

D2. In this step, we will decide whether the decryption key fulfills access policy through bitwise operation. The execution procedure refers to Figure 17.

```
/* perform AND operation on the access policy and the user
   attribute set */
ai_bi = prv->ea & cph->ep;
/*decide whether decryption key satisfies the access policy */
if (ai_bi != cph->ep) {
      printf("No matching policy, decryption fail.\n");
      return -1;
}
ai_bi =  prv->ea ^ cph->ep;
...
mpz_mul(dp, prv->mul_diai, ea_div_ep);
mpz_mod(dp, dp, pub->phi_n); /* compute dp */
element_mul(kph, cph->gkh, cph->gxh);
element_pow_mpz(kph, kph, prv->inv_kx);
element_pow_mpz(kph, kph, dp);     /* compute Km */
```

Figure 17. Step2 of Decryption Algorithm in CSKCT

To be specific, the AND operation is performed on b_P and a_A, corresponding to the access policy P and the user attribute set A, respectively. If the operation result is identical to the b_P, this means decryption key satisfies the access policy. Otherwise the access policy is not satisfied, and the decryption process will terminate with indication of that.

Then, the algorithm performs exponentiation and multiplication operations to decrypt the random AES key m. It computes K_m with three times exponentiation operation. Subsequently, it uses element_div function to obtain the random m (the AES key) by computing m as E/k_m.

```
/*read the encrypted file into array of bytes will be stored
  in aes_buf */
read_cpabe_file(in_file, &file_len, &aes_buf);
cskct_cph_free(cph);
/*use AES to decrypt the encrypted file store in aes_buf*/
plt = aes_128_cbc_decrypt(aes_buf, m);
g_byte_array_set_size(plt, file_len);
g_byte_array_free(aes_buf, 1);
/*output the decrypted file*/
spit_file(out_file, plt, 1);
unlink(in_file);
```

Figure 18. Step3 of Decryption Algorithm in CSKCT

D3. The decryption processing in Figure 18 reads the encrypted file into array of bytes. AES-cbc-128-encrypt function takes as input the data buffer in array of bytes for decryption. The decryption key for the AES-128 in CBC mode is taken from random element m. The AES-128 decryption function decrypts the blocks one by one and returns the buffer stored the decrypted data, which will finally be output as file.

4.5 Evaluation

This section will evaluate this proposed CP-ABE scheme on two aspects of communication overhead and computational overhead. In order to illustrate the distinguished feature of the proposed scheme, this scheme is compared with the previous CP-ABE schemes in these two aspects.

This paper first proceeds the evaluation on the size of private keys and ciphertexts in the several relevant CP-ABE systems with constant-size keys or ciphertexts.

Storage overhead

In table 5, storage cost comparison is made in terms of the size of the decryption key and the ciphertext within the following schemes.

Scheme	Length of decryption key	Length of ciphertext
EMNOS [7]	2G	$2G + G_t$
GSWV [31]	2G	$(n - \|p\| + 2)G + G_t$
ZH [33]	$(\|A\| + 1)G$	$2G + G_t$
ZHW [8]	$(2n + 1)G$	$2G + G_t$
ZZCLL [11]	$(n + 1)G$	$2G + G_t$
This scheme	G	3G

Table 5. Comparison of Communication Cost

In Table 5, the notation *G* is an element in a group. For public key bilinear pairings, *G* represents the element of G_1 and G_2, which are elliptic curve groups. Let G_t be multiplicative pairing group with prime order. For the no-pairing scheme, *G* stands for an element in $\mathbb{Z}_N^*$. The table demonstrates the comparative schemes that provide either constant size secret keys or ciphertexts. The ZZCLL scheme [11], the ZH scheme [33] and the ZHW scheme [8] fail to provide constant size keys for the users. Although the GSWV scheme [31] is lightweight with shorter constant secret keys, but it cannot offer constant size ciphertexts. From the table 5, it is clear that the EMNOS scheme [7] and this proposed scheme are uniquely designed to have constant size for both secret keys and ciphertexts. However, the EMNOS scheme provides only (*n*, *n*)-threshold access structure, which does not meet the requirement of CP-ABE for flexible access control.

In this scheme, the private key contains exactly an element and an integer for describing user attributes and one group member of $\mathbb{Z}_N^*$. Practically, the size of each element on $\mathbb{Z}_N^*$ is around 128 bytes for the 1024-bits group. Thus, the ciphertext head is bounded within 352 bytes. The proposed CP-ABE CSKCT scheme provides lightweight property on the length of ciphertexts and keys, which are constant and do not depend on the number of attributes. Accordingly, decryption is computationally constant too. Consequently, the CPABE-CSKCT scheme is easily deployed on IoT devices.

Computational overhead

This paper will evaluate the computational overhead based on the benchmark experiments for the CPABE-CSKCT operations. The benchmark was performed on a workstation which has Intel Core Xeon(R) CPU E5 and 1GB memory running Ubuntu operating system. The running environment taken the measurements displays in Table 6.

CPU	Intel Core Xeon(R) CPU E5 @ 2.40GHz
RAM	1GB DDR3
OS	Ubuntu 16.04.09
Compiler	gcc version 5.4.0
Language	C
Library	GMP 6.1.2 OpenSSL 1.0.2g

Table 6. Running Environment for Measurement

To the best of author's knowledge, the prior ABE schemes are almost all based on bilinear paring of ECC. In the performance evaluation, PBC is used for pairing, and the implementation selects a 160-bit elliptic curve group based on the fast supersingular curve $y^2=x^3+x$ over a 512-bit finite field *G*. On the test machine, the PBC library can perform a pairing in approximately 1.2ms, exponentiations in *G* and G_t take about 1.6ms and 0.2ms respectively. Each of operation is run 10 times and the average value is rated as performance result listed in Table 7

Parameter	Time-Consuming
T_e	1.20 ms
T_G	1.62 ms
T_{G_t}	0.19 ms
T_{Z_N}	0.48 ms

Table 7. Execution Time for Various Parameters

Let T_G be time to execute a point exponentiation in the group G and T_{G_t} be time to execute an exponentiation in the pairing group G_t. Also, T_e is time to execute a bilinear map. Additionally, the computational exponentiation time over multiplicative group $\mathbb{Z}_N^*$ ($|N| = 1024$) denoted by T_{Z_N} requires 0.48ms in the experiment. According to this, the author summarizes the computational overhead among this scheme and other oriented schemes. The comparison results are shown in Table 8.

Scheme	Encryption	Decryption
EMNOS [7]	$(n + 1)T_G + 2T_{G_t}$	$2T_{G_t} + 2T_e$
GSWV [31]	$(2(n - \|P\|) + 2)\ T_G$	$2(\|A\| - \|P\|)T_G + T_{G_t} + 3T_e$
ZH [33]	$2T_G$	$(2\|P\| + 1)\ T_e$
ZZCLL [11]	$3T_G$	$2T_e$
This scheme	$3T_{Z_N}$	$3T_{Z_N}$

Table 8. Comparison of Computational Cost

As Demonstrated in table 8, this scheme is the only one to provide efficient encryption and decryption with $\mathcal{O}(1)$ time-complexity without pairing. For the measurements, assume the total number of attributes in the system be $n = 50$. Also, $|P| = 20$ and $|A| = 25$ are the number of attributes associated with access policy P and user attribute set A, respectively. Table 9 displays performance measurements of encryption time and decryption time produced on the related schemes.

Scheme	Encryption (ms)	Decryption (ms)
EMNOS [7]	16.08	2.86
GSWV [31]	100.44	19.99
ZH [33]	3.24	49.20
ZZCLL [11]	4.86	2.4
This scheme	1.46	1.10

Table 9. A Comparative Summary on Computational Cost from the Experiment

From the table 9, it is observed that this scheme takes only 1.46ms and 1.10ms for encryption and decryption. The execution of the CPABE-CSKCT encryption algorithm takes minimal time for both encryption and decryption, requires lowest computational cost in comparison with other related CP-ABE scheme.

Overall, the experiment results are accordance with the author's expectation and analysis. This lightweight feature makes it feasible for practical deployment in IoT setting.

4.6 Discussion

Although the proposed scheme has remarkable advantages in efficiency, it still has some limitations, which would be discussed as follows.

1. It only provides AND-gate access policy without multiple values. A CP-ABE scheme supporting multi-valued AND-gate can reduce the number of attributes. Designing a CP-ABE scheme based on AND-gate with multiple values is the future research.

2. Similar to most of the CP-ABE schemes, this scheme is not flexible enough in revoking and adding attributes, which means that it does not provide to change the attributes set after Setup phase. If a user wants to decrypt a ciphertext on an access structure including newly changed attributes, then the user has to obtain a new secret key corresponding to the new attributes set.

Further work will include implementing this CPABE-CSKCT scheme in the IoT devices (e.g. Raspberry Pi) with the aims of further fine-tuning the scheme for real application.

Chapter 5
RSA-based Access-Tree CP-ABE Scheme

5.1 Background

Most existing public key encryption schemes allow for data encryption by a restrictive policy. But their access policies are unable to efficiently enforce more expressive types of access control. In Bethencourt's work [4], they provided the first construction of a CP-ABE with a monotonic "access tree", where the interior nodes of the access structure were composed of threshold gates and the leaves were associated with descriptive attributes. Bethencourt et al. [4] noted that AND-gates could be constructed as n-of-n threshold gates and OR gates as 1-of-n threshold gates; more complex access controls such as numeric ranges can be handled by converting them to small access trees. At a higher level, the work of this proposed scheme is similar to their design. Therefore, the access structure of this scheme supports AND, OR gates and the comparison of numerical attributes.

In the design pattern, private keys will be identified with an arbitrary number of user attributes. A party that wishes to encrypt a message will specify a policy through an access tree structure, which private keys must satisfy for decryption. A user will be able to decrypt a ciphertext with a private key if and only if the given key contains all of attributes assigned to the leaves of the tree such that the tree is satisfied. This paper uses the same notation as [2] to describe the access-tree structure.

Tree Access Structures Definition

Access Structure [2]. Let $\{P_1, P_2, ..., P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{p_1 p_2,\dots,p_n)}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, ..., P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{p_1 p_2,\dots,p_n)}\backslash\{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.

In this scheme, the attributes will describe the role of the participants. The access structure $\mathbb{A}$ will be labelled different encrypted data with the authorized sets of attributes, denoted n attributes as $\mathbb{A} = \{A_1, A_2, \cdots, A_n\}$, where A_i is an attribute with the ordering of i =1,2,$\cdots$,n. The author restricts the attention to monotone access structures in this paper. However, as noted in [4], it is also possible to (inefficiently) realize general access structures using the extension of this scheme by having the ‘not’ of an attribute as a separate attribute altogether. If doing in this way, the number of attributes in the system will be doubled.

Construction for Access Tree

Access tree T [4]. Let *T* be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. Suppose that num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND-gate. Each leaf node x of the tree is associated with a descriptive attribute and its threshold value $k_x = 1$. To facilitate working with the access trees, this paper defines a few functions below.

parent(x) denotes the parent of the node x in the tree

att(x) is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree.

order(x) is defined only if x is a leaf node and denotes the index of the attribute associated with the leaf node x in $\mathbb{A}$.

index(x) defines an ordering between the children of every node, that is, each children of a node x is associated with a number from 1 to *num* called an index value. The index values are uniquely assigned to nodes in the access structure for a given key in an arbitrary manner.

Satisfying an access tree [4]. Let *T* be an access tree with root *r* written as T_r. The subtree of *T* rooted at the node x is denote by T_x. If a set of attributes γ satisfies the access tree T_x, it is denoted as $T_x(\gamma) = 1$. $T_x(\gamma)$ is computed recursively as follows. If x is a non-leaf node, evaluate $T_{x'}(\gamma)$ for all children x' of node x. $T_x(\gamma) = 1$ if and only if at least k_x children return 1. If x is a leaf node, then $Tx(\gamma) = 1$ if and only if att(x) $\in \gamma$.

Lagrange polynomials [34] are used for polynomial interpolation. Given a set of k data points $(x_0, y_0), \cdots, (x_j, y_j), \cdots, (x_k, y_k)$, where no two x_j are the same, the Lagrange polynomial is the polynomial of lowest degree that assumes at each value x_j the corresponding value y_j. The interpolating polynomial of the least degree is unique. An n degree polynomial in the Lagrange form is a linear combination.

$$P(x) = \sum_{i=0}^{n}(\prod_{0\le j\le n, j\ne i} \frac{x-x_j}{x_i-x_j})y_j$$

According to this definition, we also define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_n$ and a set, S, of elements in $\mathbb{Z}_n$: $\Delta_{i,S}(x) = \prod_{j\in S, j\ne i} \frac{x-j}{i-j}$. The method to construct the Lagrange polynomial is known as polynomial interpolation, the polynomial $P(x)$ is called Lagrange interpolation polynomial. Suppose that the polynomial $P(x)$ is in the form:

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

Polynomial interpolation [34] is to find a polynomial p of lowest possible degree with the property $P(x_i) = y_i$ for all $i \in \{0,1,\cdots,n\}$.

5.2 Proposed Construction

Comparable to the prior CP-ABE scheme based on less efficient bilinear map of ECC, this proposal is constructed with no-pairing RSA on a group $\mathbb{Z}_N$, which is of congruence classes of integers modulo N where $N = pq$. This RSA-based CP-ABE cryptosystem works on $\mathbb{Z}_N$ and its subgroup written in multiplication notation as $\mathbb{Z}_N^*$ or $(\mathbb{Z}/n\mathbb{Z})^*$, which is called multiplicative group of integers modulo N. The generation of key pair is built on the integer factoring. This encryption scheme is also designed to be based on discrete logarithm problem.

A. SETUP PHASE

The authorized set of attributes in the system is $\mathbb{A} = \{A_1, A_2, \cdots, A_n\}$ and the mapped ordering of attributes is defined as $\{1, 2, \ldots, n\}$. The setup algorithm will take the security parameter ρ and $\mathbb{A}$ as inputs. This algorithm consists of the following steps:

1) Choose two RSA primes p and q with $p \ne q$, and compute $N = pq$. Next, pick system private key k that satisfy $gcd(k, \varphi(N)) = 1$. Then, randomly select the RSA public exponent e_i with $\gcd(e_i,\varphi(N)) = 1$, and compute d_i such that $e_i d_i \equiv 1 \pmod{\varphi(N)}$ corresponding to each attribute $A_i \in A$, $\forall i = 1,2,\cdots,n$. Further, select a random integer g with $gcd(g, N) = 1$ and $2 < g < N-1$.

2) Compute the public parameter $R = g^k$.

3) Finally, generate the following master secret key MSK and master public key MPK, where MSK = $\{\varphi(N), k, d_1, \cdots, d_n\}$, MPK = $\{N, R, e_1, \cdots, e_n\}$.

B. ENCRYPT PHASE

The encryption algorithm encrypts a message M (here is random chosen AES key) under the tree access structure T. The algorithm first chooses a Lagrange polynomial q_x for each node x (including the leaves) in the tree T. These polynomials are chosen in the following way in a top down manner [2], starting from the root node R. For each node x in the tree, the degree d_X of the polynomial q_x is set to be one less than the threshold value k_X of that node, that is, $d_X = k_X - 1$. Now, for the root node R, the algorithm chooses a random $s \in \mathbb{Z}_N$ and sets $q_R(0) = s$. Then, it chooses d_R other points of the polynomial q_R randomly to define it completely. For any other node x, it sets $q_x(0) = q_{\text{parent}(x)}(\text{index}(x))$ and chooses d_X other points randomly to completely fix q_x. The ciphertext CT is then computed by giving the tree access structure T as follows.

1) The algorithm first chooses a random $s \in \mathbb{Z}_N$, and sets $q_R(0) = s$. Then, constructs a polynomial q_x for each non-leaf node x in the tree T. Let, Y be the set of leaves in T.

2) Next, random $h \in \mathbb{Z}_N$ and compute $Y_m = g^{kh}$, $CT: \ E = M\text{R}^{\text{hs}} = MY_m^s$

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

Ultimately, output the ciphertext as $CT = \{T, E, Y_m, \{E_X\}_{\forall X \in Y}\}$.

C. KEYGEN PHASE

The key generation algorithm will take as input a set of attributes S and output a key that identifies with that set. it computes $H(\text{att}(x))$ for each attribute $x \in S$. assign the key as $SK = \{k_X = H(\text{attr}(x)) / d_i, x \in S, i = \{\text{order}(x)\}\}$

D. DECRYPT PHASE

The algorithm begins by simply calling the function on the root node R of the tree T If the tree is satisfied by a set S of attributes. This paper specifies the decryption procedure as a recursive algorithm. For ease of explanation, the author presents the simplest form of the decryption algorithm. First define a recursive algorithm DecryptNode(CT, SK ,x) that takes as input a ciphertext $CT = \{T, E, Y_m, \{E_X\}_{\forall X \in Y}\}$, a private key SK, which is associated with the set *S*, and a node *x* in *T* . Let the node *x* be a leaf node and define as follows: If $i \in S$, *i*={order(*x*)}, then

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

If $i \notin S$, then define DecryptNode(*CT*, *SK*, *x*) = $\perp$.

Now consider the recursive case when *x* is a non-leaf node. The algorithm DecryptNode(*CT*, *SK*, *x*) then proceeds as follows [2]: For all nodes *z* that are children of *x*, it calls DecryptNode(*CT*, *SK*, *z*) and stores the output as F_z. Let s_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$. Otherwise, compute

$F_Z = \sum_{z \in s_x} q_z(0) \,.\, \Delta_{i,s_z}(0)$, where i=index(z), s_z= {index(z):z∈ s_x}

$= \sum_{z \in s_x} q_{parent(z)}(index(z)) \,.\, \Delta_{i,s_z}(0)$ (by construction)

$= \sum_{z \in s_x} q_z(i) \,.\, \Delta_{i,s_z}(0)$

$= q_x(0)$ (using polynomial interpolation)

$= \text{s}$

And return the result.

The last equality F_Z is derived from using polynomial interpolation in the sum. Thus, instead of exponentiating at each level, for each node $x \in S$, Lagrange polynomial interpolation can be computed by doing multiplication in $\mathbb{Z}_N$.

Now that the function DecryptNode have been defined, the decryption algorithm can be implemented by calling the function on the root node R of *T*. If the tree is satisfied by *S*, then set *A*= DecryptNode(*CT*, *SK*, *R*) = *s*. The algorithm now decrypts by computing.

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

5.3 Security analysis

The proposed scheme is proven secure under two assumptions, Integer Factorization Problem (IFP) and Computational Diffie-Hellman Problem (CDHP). If the published keys are chosen large enough as to render both problems intractable using current algorithms, then the system with two secure properties will remain secure so long as at least one of the two problems remains intractable.

The following theorem gives a lower bound on the advantage of a generic adversary in breaking the proposed CP-ABE scheme. Let α be a union bound on the total number of group elements it receives from queries it makes and from its interaction with the CP-ABE security game. A random oracle model is also given to represent the hash function H.

Theorem 4 For any adversary $\mathcal{A}$ who can attack the scheme during a polynomial time, the negligible advantage of $\mathcal{A}$ in the CP-ABE security game is at most $\mathcal{O}(\alpha^2/\text{N})$.

if an adversary that acts generically on the construction underlying this scheme can break the security of this scheme with a reasonable probability, then a stimulator $\mathcal{B}$ can be constructed to break the DH problem with a non-negligible advantage.

Proof. The author follows a selective-set model in [2] here.

Init The stimulator $\mathcal{B}$ choose the attributes set S, which is the adversary $\mathcal{A}$ wish to challenge. Assume that the access tree associated with the S is Γ.

Setup The stimulator sets k co-prime to φ(N) at random from $\mathbb{Z}_N^*$ (range from 1 to N-1) and sets public parameter e_i associate with each attribute in S by choosing random integers co-prime to φ(N). It calculates the value d_i of multiplicative inverse of e_i. The public key parameters $R = g^k$, e_i, $i \in \{1, \ldots, |S|\}$ are assigned and sent to the challenger.

Phase1 The adversary $\mathcal{A}$ makes many requests for private keys where the attributes may be set overlap between the attributes for each requested key. When the adversary makes its j'th key generation query for the set S_j of attributes, any value $d_i^{(j)}$ for every $i \in S_j$ is chosen independently at random from $\mathbb{Z}_N$. In addition, when the adversary asks for the evaluation of H hash function on any string i, a new random number is chosen from $\mathbb{Z}_N$, and the stimulation assigns a value t_i as the response to $H(i)$. Random oracle [32] are typically used as an output of cryptographic hash functions in schemes where strong randomness assumptions are needed.

The simulator then computes $K_i = H(i)/\ d_i = t_i\ /\ d_i$ and passes the assignments to the adversary $\mathcal{A}$.

In the key generation phase, the solution randomizes the key value d_i such that they cannot be combined. Therefore, collusion attacks will not succeed since the blinding values have the property of strong randomness for the private key of a particular user.

Challenge The adversary will submit two messages M_0, $M_1 \in \mathbb{Z}_N$, and the access tree $\mathcal{T}$ to the stimulator $\mathcal{B}$. $\mathcal{B}$ flips a fair binary coin v and does the following in order to returns an encryption of M_v. First, it chooses a random value s corresponding to the root node r of $\mathcal{T} \in \Gamma$ from $\mathbb{Z}_N$. Thereafter, it needs to define a polynomial Q_x of degree d_x for each node x in the access tree $\mathcal{T}$ so as to construct the access structure for all relevant attributes. It sets $q_x(0)$ and rest of the nodes uniformly and independently at random from $\mathbb{Z}_N$. In this way, the process is performed recursively until q_x are completely set up for each node of $\mathcal{T}$. Finally, the simulation chooses a random $h \in \mathbb{Z}_N$, and built the encryption message as $E = MR^{hs}$, $Y_m = g^{kh}$ and $E_i = H(\text{i}) \cdot q_i(0) \cdot e_i$ for each relevant attribute i. The ciphertext is passed to the adversary. Obviously, in order to decrypt, an attacker need recover R^{hs} blinded by some values, g^{kh} and $q_x(0)$. The value of $q_x(0)$ need be extracted out using polynomial interpolation if and only if the correct key component K_i of the user satisfy the component E_i embedded in the ciphertext.

Phase 2 The simulator and the adversary acts exactly as they did in Phase 1.

Guess The adversary outputs a guess v' of v to the stimulator. When $v \neq v'$, the adversary cannot gain any information about v.

It will show that with probability of $1-\mathcal{O}(\alpha^2/N)$, the simulation can be considered secure. The adversary in turn has the advantage of at most $\mathcal{O}(\alpha^2/N)$ in the game.

In the encryption, the simulator takes over the random oracle model and the random choice of the variable values. It employs a hash function $H\colon \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ modeled as a random oracle. The function will map any attribute described by a binary string to a random unique group element, whose size equals the size of the RSA modulus. Random oracles have long been considered in computational complexity theory, and many schemes have been proven secure in the random oracle model. In general, to break the random oracle assumption, attacker must discover some unknown and undesirable property of the applied hash function, where such properties are yet believed unlikely.

Recall that the setup of access structure $\mathcal{T}$, when a polynomial q_r of degree d_r is defined for the root node r, the procedure chooses a random integer θ and set $q_r(0) = \theta$. Then it recursively defines polynomial q_x for each node x of $\mathcal{T}$. Lagrange coefficients from various levels of access tree are chosen uniformly at random from $\mathbb{Z}_N$ in the secret way, where q_x is decided for each node x. For the adversary, final polynomial $Q_x(.) = yq_x(.)$ is built for each node x in the tree. Notice that this sets $Q_r(0) = y\theta$. The ciphertext component E_i corresponding to each leaf node is given using its polynomial, $E_i = t_i \cdot q_x(0) \cdot e_i$, where $i = \text{attr}(x)$. E_i will be a random element of $\mathbb{Z}_N$ and contains no information about $q_x(0)$. Under the factoring hardness assumption and computationally hard problem, it is hard to break E_i using known approach.

From the adversary view, an oracle query R^{hs} can be viewed as an algebraic expression $\nu = \lambda\mu$. Let ν_0, ν_1 be two random encodings of the group $\mathbb{Z}_N$. Here this paper will keep track of two queries to illustrate an unexpected collision. When an adversary makes a query to the group oracles, there are N distinct values in the range of both ν_0 and ν_1 corresponding to two distinct rational functions $\lambda\mu$, $\lambda'\mu'$. Due to the random choices of these variable' values, a collision occurs only if $\lambda\mu = \lambda'\mu'$. The probability of this event is $\mathcal{O}(1/N)$. Accordingly, the same argument is then extended for a union bound. The author can conclude that the probability that any such collision happens in this scheme is at most $\mathcal{O}(\alpha^2/N)$, as stated. The theorem 4 is proved.

5.4 Implementation

This section now discusses the implementation for the RSA-based access-tree CP-ABE scheme. This cryptosystem is programmed with C language under Ubuntu platform. The development mainly uses the GMP arithmetic library that is suitable for cryptography. Additionally, few functions in PBC library are also invocated for the usage of element operations of finite group.

Policy Tree

Each non-leaf node of the access tree is a threshold gate and the leaves are described by a set of attributes. In this scheme, a (t, n)-Threshold gate can be simply expressed as t of *n*, instead using complex expressions. A tree with "AND" and "OR" gates can be represented by using respectively n of n and 1 of n threshold gates.

To handle and compile a numerical attribute "*a* = *k*", for some *n*-bit integer *k* it can be converted into a "bag of bits" representation described in [4], producing n (non-numerical) attributes which specify the value of each bit in *k*. For instance, to describe a secret key with the 4-bit attribute "*a* = 11", this paper would instead involve such "*a*: 1***", "*a*: *0***", "*a*: **1*", and "*a*: ***1" in the key.

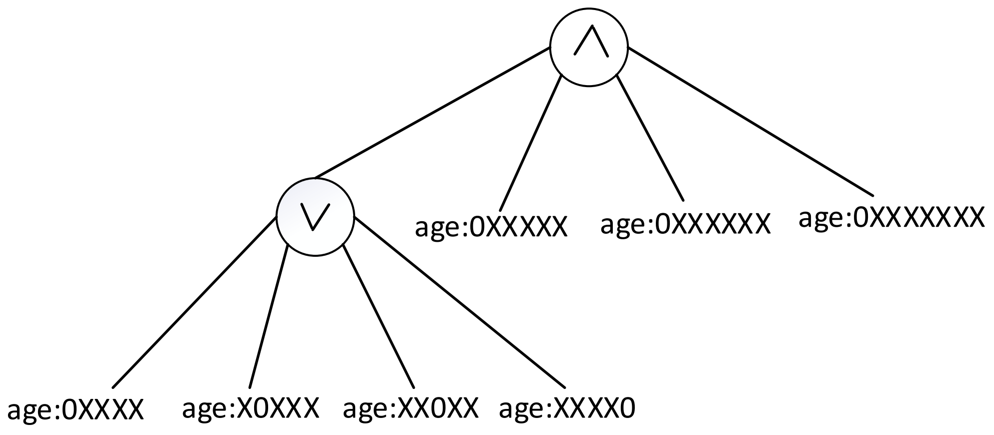


Figure 19. Policy Tree for the Integer Comparison "age < 30"

The policy languages of AND and OR gates can then be used to implement integer comparisons over such numerical attributes, as illustrated for "*age* < 30" in Figure 19. The numerical comparisons are handled into their gate-level implementation. There is a direct correspondence between the bits of the constant 30 and the choice of gates. Policies for $\leqslant$, >, $\geqslant$, and = can be implemented in a similar way with at most *n* gates, or possibly fewer depending on the value of constant.

cp-abe Toolkit

The author has developed cp-abe toolkit for the user interface that provides command line tools. The toolkit supports the numerical values and integer range queries described as below.

A. cp-abe-setup [OPTION ...]

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

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

options are set.

-h, --help	print this message.
-p, --output-public-key FILE	write public key to FILE.
-m, --output-master-key FILE	write master secret key to FILE.
-u, --universe FILE	read attribute universe from FILE.

B. cp-abe-keygen [OPTION ...] PUB_KEY MASTER_KEY ATTR [ATTR ...]

Generate a key with the listed attributes using public key PUB_KEY and master secret key MASTER_KEY. Output will be written to the file related to private key unless the -o option is specified.

- Command-line parameters are listed below.

PUB_KEY:	file for a public key
MASTER_KEY:	file for a master secret key
ATTR	arbitrary number of attributes string.

- Options are set.

-h, --help	print this message.
-o, --output FILE	write resulting key to FILE.

C. cp-abe-enc [OPTION ...] PUB_KEY FILE [POLICY]

Encrypt FILE under the decryption policy POLICY using public key. The encrypted file will be written to FILE.cpabe unless the -o option is used.

- Command-line parameters are shown as below.

PUB_KEY:	file for a public key
FILE:	the original file for encryption
POLICY:	attribute strings with 'and', `or', and `of' delimiter.

The keywords 'and', 'or', and 'of', are reserved for the policy language and may not be used for either type of attribute.

- Options are set as.

-h, --help	print this message.
-o, --output FILE	write resulting key to FILE.

D. cp-abe-dec [OPTION ...] PUB_KEY PRIV_KEY FILE

Decrypt an encrypted file using a given public key PUB_KEY and private key PRIV_KEY. The -o option specifies the name of the decrypted file. Otherwise If the name of FILE is X.cpabe, FILE will be decrypted and removed. The decrypted file will be written as X.

- Command-line parameters

PUB_KEY:	file for a public key
PRIV_KEY:	file for a user private key

- Options

-h, --help	print this message.
-o, --output FILE	write output to FILE.

Also, the cp-abe toolkit provides a flexible policy language to specify access policies. These features are illustrated in the sample usage session of the following Figure 20.

```
$ cp-abe-keygen  -o  sara_priv_key  pub_key  master_key \
        female  it_department  ‘age = 32’  ‘admin_level = 1’

$ cp-abe-keygen  -o  carol_priv_key  pub_key  master_key \
        male  executive_team  ‘age = 35’  ‘admin_level = 7’

$ cp-enc  -o  pub_key  enc_report.pdf  \
        ( (executive_team or it_department) and age < 30 )  or
             ‘admin_level > 5’
```

Figure 20. Example Usage of the cp-abe Toolkit

In this example, two private keys are issued for both employees, “Sara” and “Carol”, with various sets of attributes (normal and numerical) using cp-abe-keygen tool. A document is encrypted under a complex policy using cp-abe-enc. As demonstrated by the above example, the police language allows the general threshold gate of the underlying scheme, but also provides AND and OR gates for convenience. Members of the executive team or IT department may decrypt the encrypted report if their age is less than 30. In addition to that, anyone whose administration level is larger than 5 also can decrypt. In this case, Carol would be able to use his key stored as carol_priv_key to decipher the enciphered report, whereas Sara would not be able to use hers to decrypt the document.

Algorithm Description

A. Setup Algorithm

The procedure is similar to the setup algorithm in CSKCT scheme refers to Section 4.4

B. Key Generation Algorithm

Argument vector is composed of public key, master private key associated files and attributes strings. A user secret key will be written to the file of the default name as output unless the -o option is specified. Numerical attributes are specified as `attr = *N*', where *N* is a non-negative integer less than $2^{64}$ and `attr' is another string. The process of the key generation is displayed in Algorithm 3 and Figure 21.

Algorithm 3: Setup Algorithm in Access-Tree CPABE

```
Procedure Tree_CPABE_keygeneration(pub_file, msk_file, user_attrs)
1   begin
2   * user_attrs ← parse_arguments()
3   pubkey ← deserialize(read_file(pub_file) )
4   mskkey← deserialize(read_file(msk_file) )
5   prvkey ← init_prv_params(pubkey)
6   foreach user_attrs[i] ∈ {$A_1 \cdots A_i$} do
7       initialize(dp)
8       initialize(hash_attr)
9       initialize(invert_attr)
10      Map user_attrs[i] to element hash_attr
11      Invert hash_attr to invert_attr
12      dp ← multipy(invert_attr, prv_inverse[i])
13      Add dp to prvkey comparions structure
14  end
15  prvkeybuffer ← serialize(prvkey)
16  Write to prvkeybuffer prv_file
17  Output prv_file
18  end
```

Table 10. Algorithm 3 Key Generation Algorithm in Access-Tree CPABE

```
/* process 1).deserialization of public key and master secret
   key file */
parse_args(argc,argv);
pub = treecpabe_pub_unserialize(suck_file(pub_file),1);
msk = treecpabe_msk_unserialize(suck_file(msk_file),1);
/* user private key structure is allocated and initialized.*/
prv = init_prv_params(pub);
/*start 2). compute private key*/
while (*user_attrs) {
      treecpabe_prv_comp_t c;
      element_t h_attr, inv;

      c.attr = *(user_attrs ++);
      element_init(c.dp, Zn);
      element_init(h_attr, Zn);
      element_init(inv, Zn);
      /* map each attribute string to one element of the finite
        group */
      element_from_string(h_attr, c.attr);
      element_invert(inv, h_attr);
      element_mul_zn(c.dp, inv, msk->di[i]); /*compute k1 */
      element_clear(h_attr);
      element_clear(inv);
      /*append the generated elements to bytes of array in the
      key structure */
      g_array_append_val(prv->comps, c);
}
/* serialize private key, writing into the output file */
spit_file(out_file, treecpabe_prv_serialize(prv),1);
```

Figure 21. Step1 of Key Generation Algorithm in Access-Tree CPABE

1) The function first calls the deserialization of public key file and master secret key file to obtain the public key and master key. Then user private key structure is allocated and every parameter in the structure is initialized.

2) Next, each declared attribute string need be mapped to one element of the finite group. For each attribute in the loop, Secure Hash Algorithm 1 (SHA1) hash function is used to hash an arbitrary string to a value. The function element_from_hash is used to generate an element e deterministically from the bytes number stored in the data buffer. To add elements to a byte array, g_array_apend_val is used to append the generated elements to the private key structure. After completing this process, the user decryption key is then derived and serialized into the output file.

C. Encryption Algorithm

For the encryption, this paper only focuses on the encryption procedure of access policy. The procedure of policy encryption is depicted in the following Figure 22.

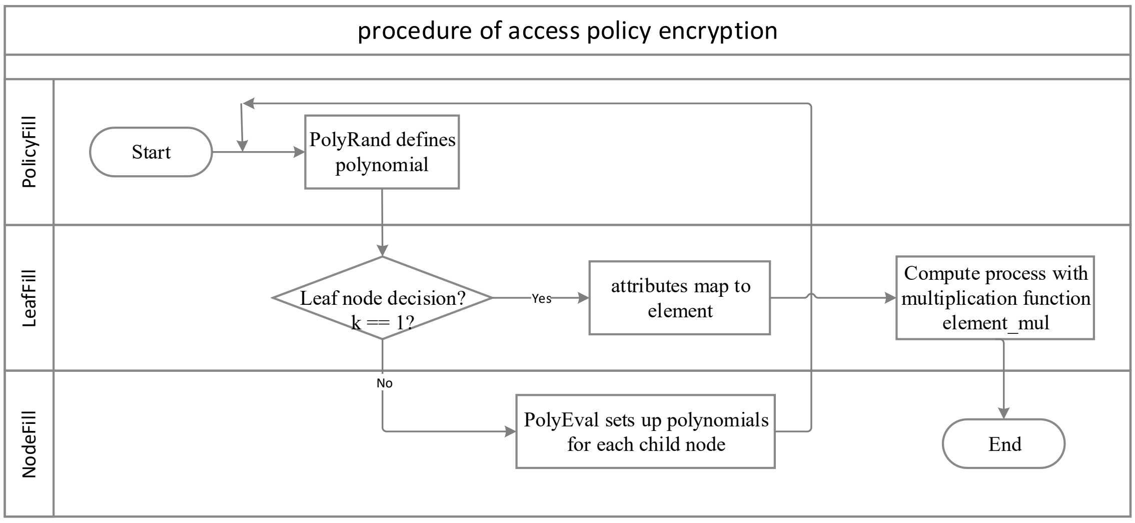

procedure of access policy encryption
PolicyFill
LeafFill
NodeFill
Start
PolyRand defines polynomial
Leaf node decision? k == 1?
Yes
attributes map to element
Compute process with multiplication function element_mul
No
PolyEval sets up polynomials for each child node
End

Figure 22. Encryption Procedure of Policy in Access-Tree CPABE

The encryption algorithm defines the following function, namely, PolicyFill ($\mathbb{T}_x$, γ, pub, *s*) described in Figure 23.

```
/* set up the polynomials for the nodes of an access subtree
*  vector p: access policy represents access tree
*         pub: public key
*         e:  random element
*/
void PolicyFill( treecpabe_policy_t* p, treecpabe_pub_t* pub,
                    element_t e )
{
      /* set up a polynomial of k-1 degree for the node. set q_x(0) =
       s and Lagrange coefficient of rest of the points at random to
       completely define q */
      p->q = PolyRand(p->k - 1, e);
      if( p->children->len == 0 )  /* process 2 for leaf node */
      {
            element_init(p->cp, Zn);
            element_from_string(h, p->attr);
            element_mul(p->cp, h, p->q->coef[0]);
            element_mul(p->cp, p->cp, ch);
      }
      else /* process 1) for internal node */
            for( i = 0; i < p->children->len; i++ )
            {
                  element_set_si(r, i + 1);
                  /* set polynomials for each child node
                     recursively */
                  PolyEval(t, p->q, r);
                  PolicyFill(g_ptr_array_index(p->children, i),
                                    pub, t);
            }
}
```

Figure 23. Function for Building up an Access Tree in Access-Tree CPABE

1) This procedure sets up the polynomials for the nodes of an access subtree with satisfied root node. The function takes an access tree $\mathbb{T}_x$ (with root node x) as input along with a set of attributes γ and a random integer s. It first calls the function PolyRand(deg, zero_val) to set up a polynomial of k_x-1 degree for the node x, where k_x is its threshold value. The function PolyRand sets $q_x(0) = s$ and Lagrange coefficient of rest of the points at random to completely define q_x. Then for the non-leaf nodes, the function PolicyFill calls PolyEval($\mathbb{T}_x$, q_x(index(x)) to set polynomials for each child node x' of x by recursively invocate the procedure PolyEval($\mathbb{T}'_x$,

$q_x(\text{index}(x'))$. Notice that in this manner, for each child $q_{x'}(0) = q_x(\text{index}(x')$ for each child node x' of x and the polynomial for the access tree $\mathbb{T}_x$ is built.

2) As for the leaf node, the corresponding attribute needs to be mapped to an element of the group using the function element_from_string. It uses SHA1 hash algorithm to produce the hash values corresponding to the attribute strings. Every attribute string is mapped to a 160-bit message digest stored in the data buffer. Then it invocates the function of element_from_hash to generate an element in the group structure deterministically from the buffer data. The addition and multiplication functions perform addition and multiplication operations in the field group.

D. Decryption Algorithm

The decryption processing is shown in Figure 24.

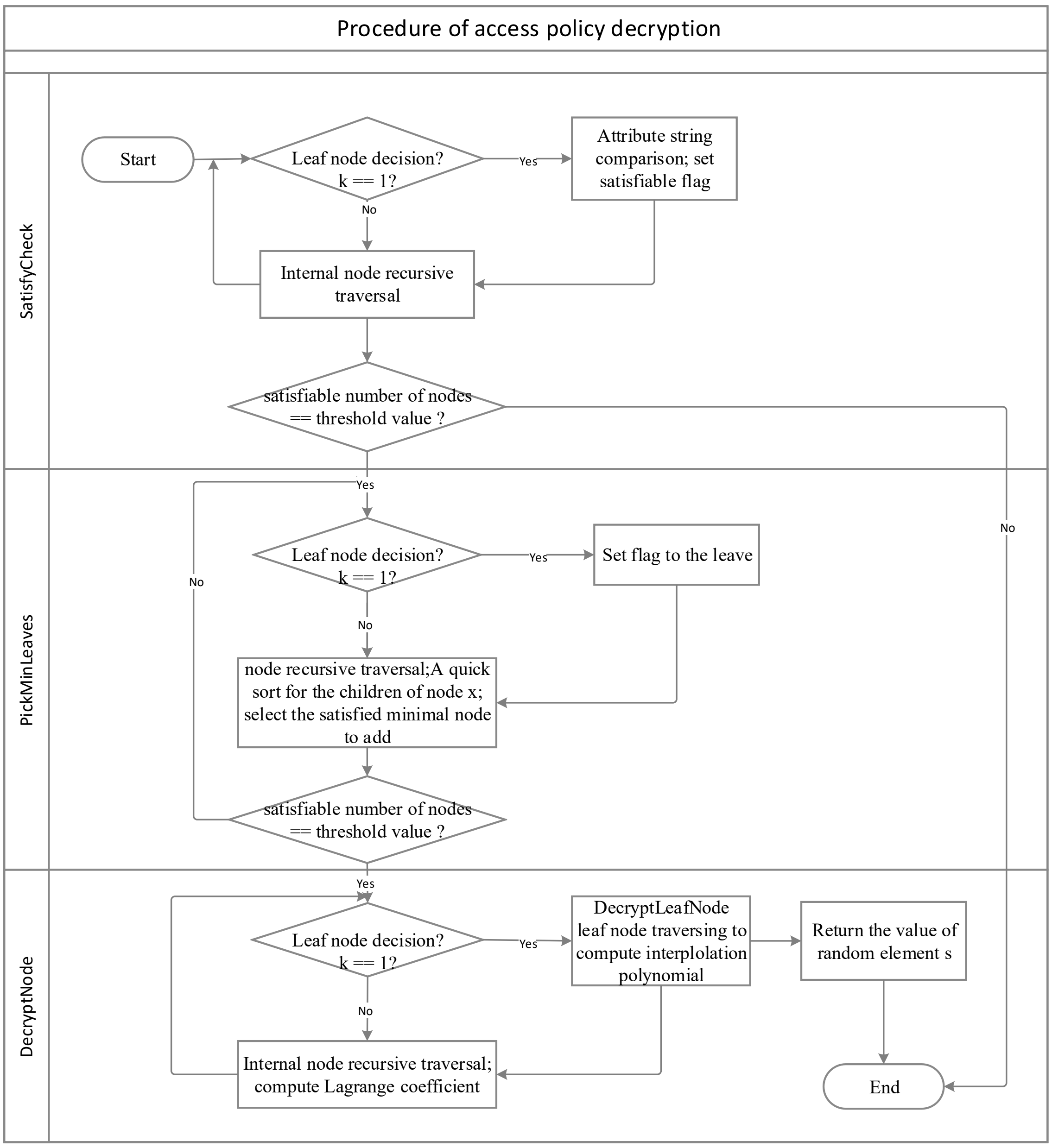

Procedure of access policy decryption
SatisfyCheck
Start
Leaf node decision? k == 1?
Yes
Attribute string comparison; set satisfiable flag
No
Internal node recursive traversal
satisfiable number of nodes == threshold value ?
No
Yes
PickMinLeaves
Leaf node decision? k == 1?
Yes
Set flag to the leave
No
No
node recursive traversal;A quick sort for the children of node x; select the satisfied minimal node to add
satisfiable number of nodes == threshold value ?
Yes
DecryptNode
Leaf node decision? k == 1?
Yes
DecryptLeafNode leaf node traversing to compute interplolation polynomial
Return the value of random element s
No
Internal node recursive traversal; compute Lagrange coefficient
End

Figure 24. Decryption Procedure of Policy in Access-Tree CPABE

The decryption algorithm begins with a series of deserialization and initialization. In this phase, the author only concentrates on the decryption procedure of access policy. First it checks whether the decryption key SK satisfies the access tree T_x by the SatisfyCheck(P, prv) function. This function is recursively run as follows, displayed in Figure 25.

```
/*check whether the key SK satisfies the access tree Tx
* vector p: access policy represents access tree
*      prv: decryption key
*/
void SatisfyCheck ( treecpabe_policy_t* p, treecpabe_prv_t* prv )
{
      if( p->children->len == 0 )   /* leaf node */
      {
            for( i = 0; i < prv->comps->len; i++ )
                if( !strcmp(g_array_index(prv->comps,
                      treecpabe_prv_comp_t, i).attr,  p->attr) )
               {
                    p->satisfiable = 1;
                    p->attri = i;
                    break;
                }
      } else {    /* internal node */
            for( i = 0; i < p->children->len; i++ )
                 SatisfyCheck(g_ptr_array_index(p->children,
                              i),prv);
           /* statistics the satisfiable number of the nodes */
            for( i = 0, l=0; i < p->children->len; i++ )
                if( ((treecpabe_policy_t*)g_ptr_array_index(p->chi
                                 ldren, i))->satisfiable )
                    l++;
            if( l >= p->k )
                p->satisfiable = 1; /* set satisfiability */
}
```

Figure 25. Function for Checking the Satisfiability in Access-Tree CPABE

If x is a leaf node, perform string comparison to determine if it is satisfied. The function sets the satisfiable flag for the leaf node if and only if the attribute string in the decryption key is equal to that attached in this leaf node. If x is an internal node, then evaluate $T_{x'}$ for all children x' of node x. The function statistics the satisfiable number of the nodes and returns 1 if and only if at least k_x (threshold value) children are satisfied. Otherwise it returns 0 and the decryption algorithm exits with prompt of that.

For the decryption algorithm, one important idea stated in [2] is to selectively tailor the access tree relative to the ciphertext attributes before it makes cryptographic computations. At the very least the algorithm should first discover which nodes are not satisfied without performing cryptographic operations on them. Apart from this, the algorithm may find out which leaf nodes should be used in order to minimize the number of computations as follows [2].

For each node x, define a set S_x. if x is a leaf node, then $S_x = \{x\}$. For non-leaf node x, let k be the threshold value of x. From among the child nodes of x, choose k nodes $x_1, x_2, \ldots, x_k$ such that S_{x_i} (for $i = 1, 2, \ldots, k$) are first k sets of the smallest size. If x is a non-leaf node, then $S_x = \{x' : x' \in S_{x_i}, i=1, 2, \ldots, k\}$. The set S_r corresponding to the root node r denotes the set of leaf nodes that should be used in order to minimize the number of computations. PickMinLeaves(P_Υ, prv) function is designed for this process shown in Figure 26.

```
/* find out k sets of the smallest size satisfies the access tree Tx
*  vector p: access policy represents access tree
*         prv: decryption key
*/
void PickMinLeaves ( treecpabe_policy_t* p, treecpabe_prv_t* prv )
{
      if( p->children->len == 0 )   /* leaf node */
      {
            p->min_leaves = 1;
      } else {    /* recursively make a traversal of internal node */
            for( i = 0; i < p->children->len; i++ )
              if( ((treecpabe_policy_t*)g
                     _ptr_array_index(p->children, i))->satisfiable )
                   PickMinLeaves (g_ptr_array_index(p->children, i),
                                    prv);
            qsort(c, p->children->len, sizeof(int), leaves_cmp_int);
            p->satl = g_array_new(0, 0, sizeof(int));
            for( i = 0,l = 0; i < p->children->len && l < p->k; i++ )
             if( ((treecpabe_policy_t*) g_ptr_array_index(p->children,
                                c[i]))->satisfiable )
             {
                   l++;
                   p->min_leaves += ((treecpabe_policy_t*)
                   g_ptr_array_index(p->children, c[i]))->min_leaves;
                   k = c[i] + 1;
                   g_array_append_val(p->satl, k);
             }
     }
}
```

Figure 26. Function for Picking the Minimized Node in Access-Tree CPABE

In this function, A tree policy is taken as inputs, which contains an access tree $\mathbb{T}$ with root *r* and a set of attributes γ satisfies it. Let *S* be a subset of the nodes in an access tree $\mathbb{T}$. This function is to pick a set *S* such that the number of leaves in s is minimized. In other words, no internal node has more children than its threshold *k*. This is easily accomplished with a recursive algorithm that makes a single traversal of the tree.

As shown in Figure 27, the decryption algorithm proceeds by simply calling a recursive function DecryptNode defined in section 5.2.

```
void DecryptNode ( element_t r, element_t exp,
                  treecpabe_policy_t* p, treecpabe_prv_t* prv)
{
      if( p->children->len == 0 )
            DecryptLeafNode (r, exp, p, prv);
      else
            DecryptInternalNode (r, exp, p, prv);
}

void DecryptInternalNode ( element_t r, element_t exp,
                 treecpabe_policy_t* p, treecpabe_prv_t* prv )
{
      for( i = 0; i < p->satl->len; i++ )
      {
           /* computes the Lagrange coefficient Δ_{i,S} */
            lagrange_coef(t, p->satl, g_array_index(p->satl,
                          int, i));
            element_mul(expnew, exp, t); /* num_muls++; */
            /* recursively call to pass through all children*/
            DecryptNode (r, expnew, g_ptr_array_index
           (p->children, g_array_index(p->satl, int, i)-1), prv);
      }
}
```

Figure 27. Function for decryption of the nodes in Access-Tree CPABE

If x is a non-leaf node, the algorithm DecryptNode then proceeds as follows: For all nodes z that are children of x, it computes the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_n$ and a set s of elements in $\mathbb{Z}_n$ as $\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$. Then, we have $\Delta_{i,s_z}(0) = \prod_{j\in S, j\neq i} \frac{-j}{i-j}$ according to definition, and recursively call DecryptNode to make a traversal of children node.

Now consider the recursive case when the node x is a leaf node, it executes the function DecryptLeafNode to compute interpolation polynomial and stores the result in the sum. In the end, it returns the sum as output, which is the value of random element s by using polynomial interpolation. Since $E = MY_m^s$, the decryption algorithm simply divides out Y_m^s and recovers the message M.

5.5 Evaluation

This paper now considers the efficiency of this scheme in terms of ciphertext size, private key size, and computation time for decryption and encryption. Both storage cost and computational cost are taken into account.

Storage cost

The public parameter and master secret parameter in the system will be of size linear with the number of attributes defined by the universe. The storage cost on these parameters grows linearly depending on the number of attributes in the universe.

User's private key will consist of a group element associated with every attribute. Accordingly, user's private key in size grows linearly with the size of the attributes set used to identify this user. The total number of group elements that compose a user's private key may be more than the number of attributes in the ciphertext.

The ciphertext overhead will be approximately one group element for every leaf in the access tree. Finally, the size of a ciphertext grows linearly with the number of leaf nodes in the access tree.

Computational Cost

This evaluation experiment was performed by using a workstation with Ubuntu 16.04.09 on Intel Core Xeon(R) CPU E5 and 1GB memory. The details of execution setting refer to Table 6 in section 4.5. In performance measurement, BSW scheme that is an ECC based public key cryptosystem using PBC with a fast curve over 512-bits cyclic group is compared with the RSA-based access-tree CP-ABE encryption system. Both systems are based on comparable monotone tree access structure. A series operation of encryption conducted on both schemes to produce the running time displayed in TABLE 11, and the increasing tendency of encryption time with respect to number of attributes is shown in Figure 28.

Scheme \ Number of Attributes	3	6	9	12	15
ECC(512-bits group)	82.56	137.51	202.83	269.11	337.17
Our(1024-bits group)	4.75	5.75	6.78	8.27	9.95

Table 11. Observation of Encryption Time Depending on the Number of Attributes

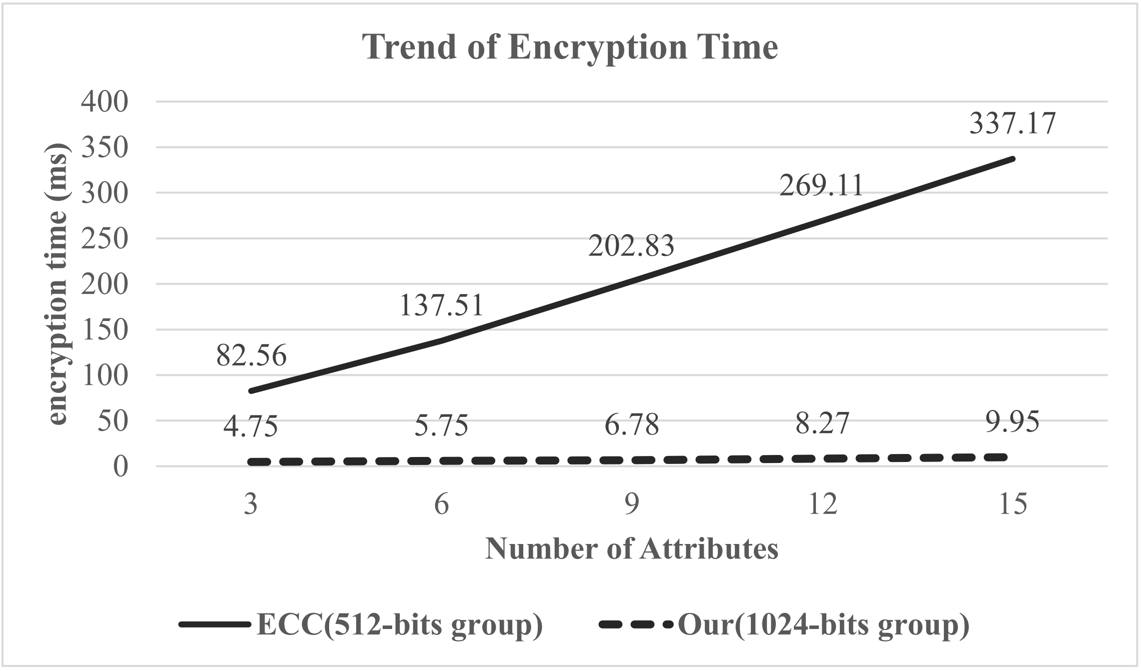


Figure 28. the Trend of Encryption Time Depending on the Attributes

As expected, the encryption execution of this scheme takes a particularly small amount of time and grows slightly with the number of attributes. Encryption in this system is significantly efficient than that in the CP-ABE cryptosystem based on bilinear pairing of ECC.

The time to encrypt is also precisely linear with respect to the number of leaf nodes in the access policy but remains much slower increase than that of BSW scheme. This is because polynomial operations on the tree in this scheme amount to a modest number of multiplications and do not significantly contribute to the running time. Through the evaluation, this proposed scheme is quite feasible for even the larger system instances.

5.6 Discussion

To the best of our knowledge, this scheme is the first attempt to design such a new lightweight CP-ABE system on access-tree structure using RSA. In the future, it would be appealing to consider that access-tree CP-ABE systems will provide more familiar language of expression with which to specify different forms of policy such that more complex and flexible applications could be created.

Chapter 6
Conclusion

IoT is a recent widespread paradigm which comprises of various smart devices interconnected over the internet. As they require the frequent communication of sensitive information, the privacy and security of IoT are of great importance. Presently, IoT technology draws many research attentions on its security. Lightweight security technique is of the best choice to the lightweight encryption system which aims to achieve the practicality and feasibility for IoT.

This thesis presents two novel lightweight RSA-based CP-ABE encryption schemes. The first proposed construction can be viewed as an extension of the previous techniques, further provide an efficient CP-ABE cryptosystem with constant-sizes keys and ciphertexts, regardless of the number of attributes. To gain more efficient, the encryption algorithm is simplified so as to reduce the complexity of computation. The thesis also demonstrates its implementation, as well as proves the model of security against attacks. As regards the evaluation results, as expected, the decryption key is mainly composed of one group elements only and the size is constant with 1024 bits each under 80-bit security requirement. The scheme has lightweight property appropriate for IoT.

Apparently, the AND-gate access structure of this primitive is lack of flexibility on the expression, seems to limit its applicability to larger systems. For the enforcement of comprehensive access control in some particular cases, again the author developed an RSA-based Access-Tree CP-ABE Scheme on generic access structure. The second result has a certain value as a heuristic work on the complex access model. The performance evaluation shows that the CP-ABE system based on no-pairing RSA is significantly efficient than other CP-ABE cryptosystem based on bilinear pairing of ECC.

These results suggest that the systems using public key encryption at sensors are feasible. The first scheme has the lightweight property on storage and computational overhead; the second one can achieve greater efficiency in the IoT scenarios where flexible access policies need be set.

References or Bibliography

[1] Bhardwaj, I., Kumar, A., & Bansal, M. (2017, September). A review on lightweight cryptography algorithms for data security and authentication in IoTs. In *Signal Processing, Computing and Control (ISPCC), 2017 4th International Conference on*(pp. 504-509). IEEE.

[2] Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2006, October). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security* (pp. 89-98). Acm.

[3] Sahai, A., & Waters, B. (2005, May). Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 457-473). Springer, Berlin, Heidelberg.

[4] Bethencourt, J., Sahai, A., & Waters, B. (2007, May). Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on* (pp. 321-334). IEEE.

[5] Odelu, V., Das, A. K., Khan, M. K., Choo, K. K. R., & Jo, M. (2017). Expressive CP-ABE scheme for mobile devices in IoT satisfying constant-size keys and ciphertexts. *IEEE Access*, *5*, 3273-3283.

[6] Yang, Y., Lu, J., Choo, K. K. R., & Liu, J. K. (2015, September). On lightweight security enforcement in cyber-physical systems. In *International Workshop on Lightweight Cryptography for Security and Privacy* (pp. 97-112). Springer, Cham.

[7] Emura, K., Miyaji, A., Nomura, A., Omote, K., & Soshi, M. (2009, April). A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In *International Conference on Information Security Practice and Experience*(pp. 13-23). Springer, Berlin, Heidelberg.

[8] Zhou, Z., Huang, D., & Wang, Z. (2015). Efficient privacy-preserving ciphertext-policy attribute based-encryption and broadcast encryption. *IEEE Transactions on Computers*, *64*(1), 126-138.

[9] Ge, A., Zhang, R., Chen, C., Ma, C., & Zhang, Z. (2012, July). Threshold ciphertext policy attribute-based encryption with constant size ciphertexts. In *Australasian*

Conference on Information Security and Privacy (pp. 336-349). Springer, Berlin, Heidelberg.

[10] Doshi, N., & Jinwala, D. C. (2014). Fully secure ciphertext policy attribute-based encryption with constant length ciphertext and faster decryption. *Security and Communication Networks*, *7*(11), 1988-2002.

[11] Zhang, Y., Zheng, D., Chen, X., Li, J., & Li, H. (2014, October). Computationally efficient ciphertext-policy attribute-based encryption with constant-size ciphertexts. In *International Conference on Provable Security* (pp. 259-273). Springer, Cham.

[12] Cheung, L., & Newport, C. (2007, October). Provably secure ciphertext policy ABE. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 456-465). ACM.

[13] Guo, F., Mu, Y., Susilo, W., Wong, D. S., & Varadharajan, V. (2014). CP-ABE with constant-size keys for lightweight devices. *IEEE transactions on information forensics and security*, *9*(5), 763-771.

[14] Odelu, V., Das, A. K., Rao, Y. S., Kumari, S., Khan, M. K., & Choo, K. K. R. (2017). Pairing-based CP-ABE with constant-size ciphertexts and secret keys for cloud environment. *Computer Standards & Interfaces*, *54*, 3-9.

[15] Salomaa, A. (2013). *Public-key cryptography*. Springer Science & Business Media.

[16] Ning, H., Liu, H., & Yang, L. (2013). Cyber-entity security in the Internet of things. *Computer*, 1.

[17] Agrawal, M., & Mishra, P. (2012). A comparative survey on symmetric key encryption techniques. *International Journal on Computer Science and Engineering*, *4*(5), 877.

[18] Lewko, A., Okamoto, T., Sahai, A., Takashima, K., & Waters, B. (2010, May). Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 62-91). Springer, Berlin, Heidelberg.

[19] Simmons, G. J. (1979). Symmetric and asymmetric encryption. *ACM Computing Surveys (CSUR)*, *11*(4), 305-330.

[20] Hofheinz, D., & Kiltz, E. (2007, August). Secure hybrid encryption from weakened key encapsulation. In *Annual International Cryptology Conference* (pp. 553-571). Springer, Berlin, Heidelberg.

[21] Beimel, A. (1996). *Secure schemes for secret sharing and key distribution*. Technion-Israel Institute of technology, Faculty of computer science.

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

[23] Oualha, N., & Nguyen, K. T. (2016, August). Lightweight attribute-based encryption for the internet of things. In *Computer Communication and Networks (ICCCN), 2016 25th International Conference on* (pp. 1-6). IEEE.

[24] Pirretti, M., Traynor, P., McDaniel, P., & Waters, B. (2010). Secure attribute-based systems. *Journal of Computer Security*, *18*(5), 799-837.

[25] Cao, Zhengjun, and Lihua Liu. "On the Disadvantages of Pairing-based Cryptography." *IACR Cryptology ePrint Archive*2015 (2015): 84.

[26] Waters, B. (2011, March). Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography* (pp. 53-70). Springer, Berlin, Heidelberg.

[27] Lynn, B. (2007). *On the implementation of pairing-based cryptosystems* (Doctoral dissertation, Stanford University).

[28] Akl, S. G., & Taylor, P. D. (1983). Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS)*, *1*(3), 239-248.

[29] McCurley, K. S. (1988). A key distribution system equivalent to factoring. *Journal of cryptology*, *1*(2), 95-105.

[30] Hofheinz, D., & Kiltz, E. (2009, April). Practical chosen ciphertext secure encryption from factoring. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 313-332). Springer, Berlin, Heidelberg.

[31] Guo, F., Mu, Y., Susilo, W., Wong, D. S., & Varadharajan, V. (2014). CP-ABE with constant-size keys for lightweight devices. *IEEE transactions on information forensics and security*, *9*(5), 763-771.

[32] Koblitz, N., & Menezes, A. J. (2015). The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography*, *77*(2-3), 587-610.

[33] Zhou, Z., & Huang, D. (2010, October). On efficient ciphertext-policy attribute based encryption and broadcast encryption. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 753-755). ACM.

[34] Berrut, J. P., & Trefethen, L. N. (2004). Barycentric lagrange interpolation. *SIAM review*, *46*(3), 501-517.

[35] Koblitz, N., Menezes, A., & Vanstone, S. (2000). The state of elliptic curve cryptography. *Designs, codes and cryptography*, *19*(2-3), 173-193.

[36] Galbraith, S. D. (2012). *Mathematics of public key cryptography*. Cambridge University Press.

[37] Katz, J., Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press.

[38] Kaliski, B., & Staddon, J. (1998). *PKCS# 1: RSA cryptography specifications version 2.0* (No. RFC 2437).

[39] Bethencourt, J., Sahai, A., & Waters, B. (2013). Advanced crypto software collection. *The cpabe toolkit, http://acsc. cs. utexas. edu/cpabe*.

[40] Lynn, B. (2010). The pairing-based cryptography (PBC) library, *http://crypto.stanford.edu/pbc/*

[41] Miller, V. S. (1985, August). Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques* (pp. 417-426). Springer, Berlin, Heidelberg.

[42] Boneh, D., Lynn, B., & Shacham, H. (2001, December). Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 514-532). Springer, Berlin, Heidelberg.

[43] Granlund, T. (1991). GMP, the GNU multiple precision arithmetic library.

Appendix 1. List of Abbreviations

ABE	Attribute-Based Encryption
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CCA	Chosen Ciphertext Attack
CDH	Computational Diffie-Hellman
CDHP	Computational Diffie-Hellman Problem
CP-ABE	Ciphertext-Policy Attribute-Based Encryption
CPABE-CSKCT	CP-ABE with Constant-size Keys and Ciphertexts
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DHP	Diffie–Hellman Problem
DOS	Denial of Service Attack
ECC	Elliptic Curve Cryptography
ECDL	Elliptic Curve Discrete Logarithms
GMP	GNU Multiple Precision arithmetic library
HIBE	Hierarchical Identity-Based Encryption
IBE	Identity-Based Encryption
IF	Integer Factorization
IFP	Integer Factorization Problem
IoT	Internet of Things
KP-ABE	Key-Policy Attribute-Based Encryption
LSSS	Linear Secret-Sharing Schemes
PBC	Pairing-Based Cryptography

RFID	Radio Frequency Identification
RSA	Rivest–Shamir–Adleman
SHA1	Secure Hash Algorithm 1
SSS	Secret-Sharing Schemes