Hindawi
Security and Communication Networks
Volume 2021, Article ID 8834616, 17 pages
https://doi.org/10.1155/2021/8834616

Research Article

Efficient Ciphertext-Policy Attribute-Based Encryption Constructions with Outsourced Encryption and Decryption

Hassan El Gafif and Ahmed Toumanari

Laboratory of Applied Mathematics and Intelligent Systems Engineering (MAISI), National School of Applied Sciences (ENSA), Agadir 80999, Morocco

Correspondence should be addressed to Hassan El Gafif; hassan.elgafif@edu.uiz.ac.ma

Received 5 September 2020; Revised 14 November 2020; Accepted 27 April 2021; Published 18 May 2021

Academic Editor: Chalee Vorakulpipat

The invention of the Ciphertext-Policy Attribute-Based Encryption scheme opened a new perspective for realizing attribute-based access control systems without being forced to trust the storage service provider, which is the case in traditional systems where data are sent to the storage service provider in clear and the storage service provider is the party that controls the access to these data. In the Ciphertext-Policy Attribute-Based Encryption model, the data owner encrypts data using an attribute-based access structure before sending them to the storage service, and only users with authorized sets of attributes can successfully decrypt the generated ciphertext. However, Ciphertext-Policy Attribute-Based Encryption schemes employ expensive operations (i.e., bilinear pairings and modular exponentiations) and generate long ciphertexts and secret keys, which makes them hard to implement in real-life applications especially for resource-constrained devices. In this paper, we propose two Ciphertext-Policy Attribute-Based Encryption Key Encapsulation Mechanisms that can be provided as services in the cloud, minimizing the user's encryption and decryption costs without exposing any sensitive information to the public cloud provider. In the first scheme, the ABE Service Provider is considered fully untrusted. On the other hand, the second scheme requires the ABE Service Provider to be semi-trusted (Honest-but-Curious) and does not collude with illegitimate users. Both schemes are proved to be selectively CPA-secure in the random oracle. The theoretical and experimental performance results show that both our first and second schemes are more efficient than the reviewed outsourced CP-ABE schemes in terms of user-side computation, communication, and storage costs.

1. Introduction

In the past, businesses were suffering from the overheads of dealing with their IT infrastructure installation and management. Nowadays, they can easily minimize these costs by externalizing their activities to one of the existing cloud solutions and paying only the amount of resources they consumed. This new paradigm is beneficial for both users and cloud providers, and this is what makes cloud services continue to attract more enterprises and individual users, helping them to start or improve their businesses easily. Cloud Storage is one of the services offered by cloud providers to help companies and individuals store, manage, and share data efficiently. Nevertheless, when outsourcing data, data owners are also outsourcing the control over their data. Therefore, this creates data security and confidentiality challenges against a third party who comprised the cloud server to steal data or even against a curious cloud provider [1]. Hence, data owners should encrypt data before outsourcing them to make sure that only authorized users can decrypt and gain access to the data.

Cryptosystems in traditional public cryptography are one-to-one ciphers, meaning that the data owner should retrieve the public key of all the authorized users and encrypt a copy of his data for each user with the corresponding public key. For example, if a data owner wants to share a document with 100 users, he must create 100 copies of the document, retrieve 100 public keys, and encrypt each copy with the public key of the corresponding user. Thus, this solution is not practical since it produces huge computation, storage, and communication overheads.

In 2005, Sahai and Waters [2] proposed a Fuzzy Identity-Based Encryption (FIBE) scheme with a new model that is not based on users' public keys or identities (as in Identity-Based Encryption schemes), but instead, their model is using attributes to encrypt data and to generate secret keys. In this model, a Trusted Authority (TA) generates users' secret keys based on their sets of attributes, and data owners specify a set of attributes and a threshold (which is the minimum number of attributes in the encryption set of attributes that should exist in the user's set of attributes) and encrypt data using this set of attributes and threshold. Only users with a number of attributes existing in the encryption set of attributes that is greater than the threshold will be able to decrypt the ciphertext using their secret keys.

Later on, two main variants of FIBE were proposed. Goyal et al. proposed the Key-Policy Attribute-Based Encryption (KP-ABE) scheme [3] where the data owner encrypts data with a set of attributes and users' secret keys are generated based on an access policy that is associated with them. Bethencourt et al. presented the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme [4] where the data owner encrypts data with an access policy and users' secret keys are generated based on their sets of attributes.

The computation overhead in ABE schemes is the most challenging part that makes them hard to be adopted in real-life applications. This is due to the number of expensive modular exponentiations and pairing operations that increases linearly with the size of the access policy.

Many contributions were proposed to optimize this computation overhead. Some of these contributions used different techniques to minimize the number of these operations in the encryption and decryption phases [5, 6] or to split them into two phases: in the first phase, most of the expensive operations are performed offline before knowing the message, and the second phase rapidly assembles the ciphertext [7, 8]. Others replaced the expensive modular exponentiations and pairing operations with the lightweight elliptic curve additions and point-scalar multiplications [9–11]. However, these solutions are still hard to be implemented in the applications where devices are resource-constrained such as the Internet of Things (IoT) and Wireless Sensor Networks (WSN). Computation outsourcing is another direction that achieved better results. In this solution, a big part of the encryption and decryption computation is outsourced to the cloud without revealing any sensitive information to the cloud providers that can help them reveal the plaintexts.

1.1. Our Contribution. Based on [4], we propose two CP-ABE with Outsourced Encryption and Decryption (CP-ABE-OED) Key Encapsulation Mechanisms (KEM) where the public ABE Service Provider performs all the encryption and decryption expensive operations leaving only one modular exponentiation and simple multiplications to be executed by the user when encrypting or decrypting data. The first scheme is suitable for the applications that consider the cloud service provider untrusted. On the other hand, the second scheme requires that the ABE Service Provider is a semi-trusted party that cannot collude with unauthorized users. Both schemes achieve provable CPA-security selectively in the random oracle.

1.2. Organization. The rest of our paper will be organized as follows. In Section 2, we will discuss the related work. Next, we define the preliminaries in Section 3. Later on, we present our 1st and 2nd CP-ABE-OED KEMs and their security analysis in Section 4. Section 5 is dedicated to showing and analysing the performance results. Finally, we conclude our paper in Section 6.

2. Related Work

In 2011, Green et al. [12] proposed the first outsourced CP-ABE scheme, which is selectively CPA-secure. They outsourced a big part of the expensive decryption operations in Waters's large universe construction [13] to a decryption proxy (e.g., Cloud Server), leaving only one modular exponentiation to be executed by the user. In the registration phase, the Trusted Authority (TA) generates a public transformation key TK and a secret decryption key z for each user. To decrypt a ciphertext CT, the user sends his TK to the decryption proxy which transforms CT to a short ElGamal-style [14] ciphertext. Then, using the decryption key z, the user decrypts the transformed ciphertext. To decrypt an ABE ciphertext containing 100 attributes, it takes nearly 30 seconds of sustained computation on a 412 MHz ARM-based iPhone 3G with 128 MB of RAM using the original CP-ABE scheme [13], while it requires only 60 milliseconds using Green et al.'s scheme [12]. Besides, thousands of lines of code, dedicated to determining how a key satisfies the access policy, were removed from the user's side. For instance, in libfenc [15], about 3000 lines are dedicated to access policy handling, excluding dependencies. An improved scheme is also provided in [12] that is selectively secure in the Replayable Chosen-Ciphertext Attack (RCCA) security model using Fujisaki and Okamoto techniques [16].

Afterward, many contributions added the notion of verifiability (i.e., the ability to verify the correctness of the transformation performed by a proxy) to the mechanism of decryption outsourcing [17–21]. In 2016, Mao et al. [21] proposed a generic construction that transforms any (selectively) CPA-secure ABE scheme with outsourced decryption (e.g., Green et al. [12]) into a (selectively) CPA-secure ABE scheme with verifiable outsourced decryption. In contrast with [17] that separately encrypts an extra random message (which is used to commit to the true message), [21] is encrypting the true message and a random message together. It then commits the random value to the message using a commitment scheme that satisfies the hiding and binding properties (at least computationally). In the decryption phase, the user receives the partially decrypted ciphertext from the decryption proxy and the commitment from the storage server and runs the revealing algorithm of the commitment scheme to verify the correctness of the transformation. The authors showed that the instantiation of this construction in the standard model using Green et al.'s small-universe, backward-compatible, and selectively CPA-secure CP-ABE scheme with outsourced decryption [12] and Pedersen Commitment [22] as the underlying commitment scheme is more efficient than Lai et al.'s scheme [17]. They also proposed a second generic construction to transform any (selectively) CPA-secure ABE scheme with outsourced decryption, that has ciphertext verifiability (i.e., the possibility to verify whether a normal ciphertext will be recovered into the same plaintext under two different decryption keys with two specific attributes) or delegatability (i.e., the capability to use a key to derive another inferior key), into a (selectively) RCCA-secure ABE scheme with verifiable outsourced decryption. They claimed that this is the first RCCA-secure construction that does not rely on a random oracle. In this construction, they combined a secure encapsulation scheme, a strong one-time message authentication code, and a secure commitment scheme.

Obviously, the previous schemes are not suitable for IoT applications where lightweight devices encrypt data and not only decrypt them (e.g., Wireless Sensor Networks) because the encryption cost produced in these schemes is still high. Accordingly, outsourcing the encryption operations in addition to the decryption operations became a new direction [23–27].

Based on [4], Zhou et al. proposed a CP-ABE scheme with outsourced encryption and decryption [23]. They outsourced a big part of the encryption operations by subdividing the access policy T into two parts: T_{DO} (data owner's access tree) and T_{ESP} (Encryption Service Provider's access tree) such that $T = T_{\mathrm{ESP}}$ AND T_{DO}. The data owner generates a random number $s \in Z_p$ and a random 1-degree polynomial $q_R(x)$, where $q_R(0) = s$, $q_R(1) = s_1$ and $q_R(2) = s_2$ and computes $\mathbb{C} = M.e(g,g)^{\alpha.s}$ and $C = g^{\beta.s}$. Then, he generates the ciphertext components C_y and C'_y for his subtree T_{DO} in the same way as CP-ABE [4] using s_2 as the shared key and sends $\left\{\mathbb{C}, C, \left\{C_y, C'_y\right\}_{y \in Y_{\mathrm{DO}}}, T_{\mathrm{DO}}, T_{\mathrm{ESP}}, s_1\right\}$ to the Encryption Service Provider (ESP). Similarly, ESP computes the ciphertext components C_y and C'_y for T_{ESP} using s_1 as the shared key. The final ciphertext is $\mathrm{CT} = \left\{T = T_{\mathrm{ESP}} \wedge T_{\mathrm{DO}}, \mathbb{C}, C, \left\{C_y, C'_y\right\}_{y \in Y_{\mathrm{ESP}} \cup Y_{\mathrm{DO}}}\right\}$. The decryption outsourcing is achieved using almost the same key-blinding technique of Green et al. [12]. However, an untrusted ESP can reveal the encrypted data by colluding with unauthorized users with sets of attributes that satisfy T_{DO}. Therefore, this solution is suitable only for applications where the ESP is at least semi-trusted.

In 2014, Asim et al. [25] proposed a new CP-ABE scheme where they outsourced a part of the encryption operations to a semi-trusted proxy A and they outsourced the decryption phase to a semi-trusted proxy B following the same technique employed in [12]. Using an encryption secret key generated by the Trusted Authority, the proxy A computes $\widehat{g^{s}}$ and uses it as the access policy root's secret to generate the access policy's leaf nodes' components $\widehat{g^{s_j}}$. Afterward, it multiplies each leaf node's component with the corresponding attribute component $\widetilde{C}_j = H_1(a_j)^{-s}$ in the partially encrypted ciphertext received from the host. The authors claim that their construction is secure in the generic group model under the assumption that proxy A and proxy B will not collude with unauthorized users and will not collude with each other. However, unauthorized users with at least one attribute (a_x) that exists in the access policy can reveal the plaintext using the partially encrypted ciphertext $\widetilde{\mathrm{CT}}$ and the ciphertext generated by proxy A (CT). For each leaf node j of the access policy, the attacker retrieves $\widetilde{C}_j = H_1(a_j)^{-s}$ from $\widetilde{\mathrm{CT}}$ and $C_j = \widehat{g^{s_j}} \cdot H_1(a_j)^{-s}$ from CT and computes $\widehat{g^{s_j}} = C_j/\widetilde{C}_j$. Then, he executes PolicyGeneration function backward to retrieve $\widehat{g^{s}}$ and computes $A^s = (e(\breve{C}, D^1)/e(\widehat{g^{s}} \cdot H_1(a_x)^{-s}, D^2) \cdot e(\breve{C}, D_x^3))^z$. Finally, the attacker reveals the plaintext $M = C \oplus H_2(A^s)$. In addition, their scheme is not correct (i.e., given an SK of a set of attributes S that satisfies the access policy τ, $\mathrm{Dec}(\mathrm{Enc}(M, \tau), \mathrm{SK}) \neq M$). In the PolicyGeneration phase, they used $\widehat{g^{s}}$ (instead of $\widehat{s}$) as the shared key to get the shares $\widehat{g^{s_j}}$. However, in the decryption phase they used the polynomial interpolation on $\widehat{s}_j$, which will result in a value that is different than $\widehat{g^{s}}$ and, as a result, the decryption output will be different than M.

Subsequently, Zhang et al. [26] presented a fully outsourced CP-ABE scheme that, for the first time, achieves outsourced key generation, encryption, and decryption simultaneously. In their system, two Key Generation Service Providers (KGSP1, KGSP2) help TA to generate Intermediate Secret Keys (ISKs), and two Encryption Service Providers (ESP1, ESP2) help users to generate Intermediate Ciphertexts (ITs). Decryption outsourcing is achieved using the same key blinding used in Green et al.'s scheme [12]. The extra communication costs that had arisen from outsourced key generation and encryption are offline, meaning that TA and users can communicate with the cloud servers in their spare time. The system is proved to be secure under the assumption that two KGSPs (ESPs) do not collude with each other, so the final combined ISK (IT) should be information-theoretically hidden from two servers. It is selectively CPA-secure against corrupt users colluding with KGSP1, ESP1, and SSP and corrupt users colluding with KGSP2, ESP2, and SSP who can obtain the conversion key at Decryption Service Provider.

Other contributions proposed outsourced CP-ABE schemes using trusted parties such as fog nodes [28] or a trusted private cloud provider [29].

3. Preliminaries

3.1. Bilinear Maps [4]. Let G and G_T be two multiplicative cyclic groups of prime order p. Let g be a generator of G and e be a bilinear map, $e : G \times G \longrightarrow G_T$, that has the following properties:

(i) Bilinearity: for all $u, v \in G$ and $a, b \in Z_p$, we have $e(u^a, v^b) = e(u, v)^{a.b}$

(ii) Non-degeneracy: $e(g, g) \neq 1$

We say G is a bilinear group if the group operation in G and the bilinear map $e: G \times G \longrightarrow G_T$ are both efficiently computable.

3.2. Access Structure [30]. Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ is monotone if $\forall B, C:$ if $B \in A$ and $B \subseteq C$ then $C \in A$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$; i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\} \setminus \{\varnothing\}}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.

In our context, we will use a monotone access structure where the attributes play the role of the parties, which means that the access structure $\mathbb{A}$ will contain the authorized sets of attributes.

3.3. Linear Secret Sharing Scheme (LSSS) [13]. A secret-sharing scheme Π over a set of parties P is called linear (over Z_p) if the following is satisfied:

(i) The shares for each party form a vector over Z_p.

(ii) There exists a matrix M with l rows and n columns called the share-generating matrix for Π. For all $i = 1, \ldots, l$, the i'th row of M, we let the function ρ define the party labeling row i as $\rho(i)$. When we consider the column vector $v = (s, r_2, \ldots, r_n)$, where $s \in Z_p$ is the secret to be shared, and $r_2, \ldots, r_n \in Z_p$ are randomly chosen; then $M.v$ is the vector of l shares of the secret s according to Π. The share $(M.v)_i$ belongs to party $\rho(i)$.

It is shown in [30] that every linear secret sharing-scheme according to the above definition also enjoys the linear reconstruction property, defined as follows: suppose that Π is an LSSS for the access structure A. Let $S \in A$ be any authorized set, and let $I \subset \{1, \ldots, l\}$ be defined as $I = \{i: \rho(i) \in S\}$. Then, there exist constants $\{w_i \in Z_p\}_{i \in I}$ such that, if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $\sum_{i \in I} w_i.\lambda_i = s$.

Furthermore, it is shown in [30] that these constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix M.

We note that we use the convention that vector $(1, 0, 0, \ldots, 0)$ is the "target" vector for any linear secret sharing scheme. For any satisfying set of rows I in M, we will have that the target vector is in the span of I. For any unauthorized set of rows I, the target vector is not in the span of the rows of the set I. Moreover, there will exist a column vector w such that $(1, 0, 0 \ldots, 0) \cdot w = -1$ and $M_i \cdot w = 0$ for all $i \in I$.

Using standard techniques [30], one can convert any monotonic Boolean formula into an LSSS representation. An access tree of l nodes will result in an LSSS matrix of l rows. We refer the reader to the appendix of [31] for a discussion of how to perform this conversion.

3.4. CPA-Security Game

(i) *Setup*. The challenger runs the Setup algorithm and gives the public parameters PK to the adversary.

(ii) *Phase 1*. When the adversary $\mathscr{A}$ queries the decryption key and the transformation key on S, the challenger passes S on to the key generation oracle to get the corresponding decryption key and transformation key and then returns the result to $\mathscr{A}$.

(iii) *Challenge*. The adversary $\mathscr{A}$ submits the access structure (M^*, ρ^*) (which is not satisfied by any of the sets of attributes S passed in phase 1) to be challenged on and requests the challenge Key*. The challenger flips a random coin $b \in \{0, 1\}$. If $b = 0$, it returns CT* to $\mathscr{A}$, where the first element in CT* is a random Key*. If $b = 1$, it returns CT* to $\mathscr{A}$, where the first element in CT* is a well-constructed Key*.

(iv) *Phase 2*. Phase 1 is repeated with the restriction that the adversary cannot obtain a decryption key for a set of attributes that satisfies (M^*, ρ^*).

(v) *Guess*. The adversary outputs 0 if Key* is random and 1 if Key* is a well-constructed key.

4. Our Proposed Constructions

4.1. The 1st Proposed CP-ABE-OED Key Encapsulation Mechanism. In this scheme, the ABE Service Provider is considered to be an untrusted party.

4.1.1. The Construction. (1) *Setup Phase*. In this phase, we execute the function setup(λ) that takes as input a security parameter λ, which determines the size of the groups. setup(λ) chooses a bilinear group G of prime order p with a generator g and a bilinear map $e: G \times G \longrightarrow G_T$.

It also defines a hash function $H_1: \{0,1\}^* \longrightarrow G$ mapping each attribute (described as a binary string) to a random group element, and a hash function $H_2: \{0,1\}^* \longrightarrow Z_p$.

Afterward, it generates two random numbers $\alpha, \beta \in Z_p$. Then, it secretly stores the master key $\text{MK} = \{g^\alpha, \beta\}$ and publishes the public parameters:

$$\text{PK} = \left\{G,\ g, e(\cdot,\cdot), H_1(\cdot), H_2(\cdot),\ h = g^\beta, e(g,g)^\alpha\right\}. \quad (1)$$

(2) *Registration and Key Generation Phase*. In Figure 1, Alice and Bob represent two users. Bob plays the role of the data owner and Alice plays the role of the data receiver. In the registration phase, both Alice and Bob behave in the same way.

First, the Trusted Authority (TA) registers Alice and Bob and associates a set of attributes to each of them (S_A for Alice and S_B for Bob) and executes KeyGen(U, MK, S_i).

KeyGen(U, MK, S_i) is defined as follows:

(i) First, it generates the encryption key $\text{EK}_i = s_i$ where s_i is picked randomly in Z_p, and the decryption key $\text{DK}_i = z_i$ where z_i is a random number in Z_p.

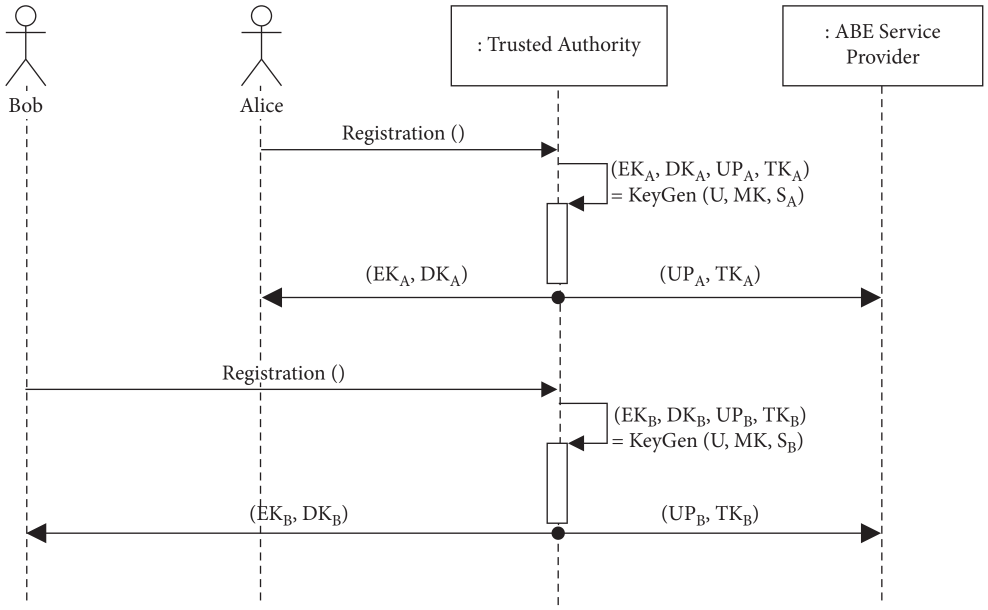

Figure 1: Registration and key generation's sequence diagram.

(ii) Afterward, it computes the user's parameters as follows: $\mathrm{UP}_i = \left\{ \begin{array}{c} \mathrm{UP}_{i,u,1} = g^{1/H_2(u \| s_i)} \\ \mathrm{UP}_{i,u,2} = H_1(u)^{1/H_2(u \| s_i)} \end{array} \right\}_{\forall u \in U}$.

(iii) It also computes the Transformation Key (TK_i). First, it chooses a random number $r_i \in Z_p$ and for each $j \in S_i$ it picks $r_{i,j} \in Z_p$ randomly. Then, it computes

$$\mathrm{TK}_i = \left\{ \begin{array}{c} S_i \\ D_i = g^{(\alpha + r_i)/\beta \cdot z_i} \\ \forall j \in S_i: \left\{ \begin{array}{c} D_{i,j} = g^{r_i/z_i} \cdot H_1(j)^{r_{i,j}/z_i} \\ D'_{i,j} = g^{r_{i,j}/z_i} \end{array} \right\} \end{array} \right\}.$$

(iv) Finally, it outputs $(\mathrm{EK}_i, \mathrm{DK}_i, \mathrm{UP}_i, \mathrm{TK}_i)$.

TA sends $(\mathrm{EK}_i, \mathrm{DK}_i)$ securely to the user i and sends $(\mathrm{UP}_i, \mathrm{TK}_i)$ publically to the ABE Service Provider.

(3) *Encryption Phase*. As shown in Figure 2, the encryption phase consists of two steps. In the first step, Bob uses its encryption key EK_B and an $l \times n$ LSSS access structure (M, ρ) and calls the function $\mathrm{Encrypt}(\mathrm{PK}, \mathrm{EK}_B, (M, \rho))$.

$\mathrm{Encrypt}(\mathrm{PK}, \mathrm{EK}_B, (M, \rho))$ is defined as follows:

(i) First, it generates a random column vector $v = (s, y_2, y_3, \ldots, y_n) \in Z_p^n$ to share the encryption exponent s.

(ii) For each $i \in \{1, 2, \ldots, l\}$, it computes $\lambda_i = M_i \cdot v$ where M_i is the ith row of M.

(iii) Then, it generates

$$\mathrm{preCT} = \left\{ \begin{array}{c} (M, \rho) \\ C = h^s \\ \forall i \in \{1, 2, \ldots, l\}: C_i^{pre} = H_2(\rho(i) \| s_B).\lambda_i \end{array} \right\}. \quad (2)$$

(iv) Finally, it outputs preCT.

Afterward, Bob sends preCT to the ABE Service Provider.

In the second step, the ABE Service Provider executes the function $\mathrm{OutEncrypt}(\mathrm{PK}, \mathrm{UP}_B, \mathrm{preCT})$ after receiving preCT.

$\mathrm{OutEncrypt}(\mathrm{PK}, \mathrm{UP}_B, \mathrm{preCT})$ performs the following instructions:

(i) For each $i \in \{1, 2, \ldots, l\}$, it computes

$$\left\{ \begin{array}{l} C_i = \mathrm{UP}_{B,\rho(i),1}^{C_i^{\mathrm{pre}}} = \left(g^{1/H_2(\rho(i) \| s_B)} \right)^{H_2(\rho(i) \| s_B).\lambda_i} = g^{\lambda_i}, \\ C'_i = \mathrm{UP}_{B,\rho(i),2}^{C_i^{\mathrm{pre}}} = \left(H_1(\rho(i))^{1/H_2(\rho(i) \| s_B)} \right)^{H_2(\rho((i) \| s_B).\lambda_i} = H_1(\rho(i))^{\lambda_i}. \end{array} \right. \quad (3)$$

(ii) Then, it outputs $\mathrm{CT} = \left\{ (M, \rho), C, \{C_i, C'_i\}_{i \in \{1,2,\ldots,l\}} \right\}$.

Finally, the ABE Service Provider sends CT to the Cloud Storage Provider (CSP).

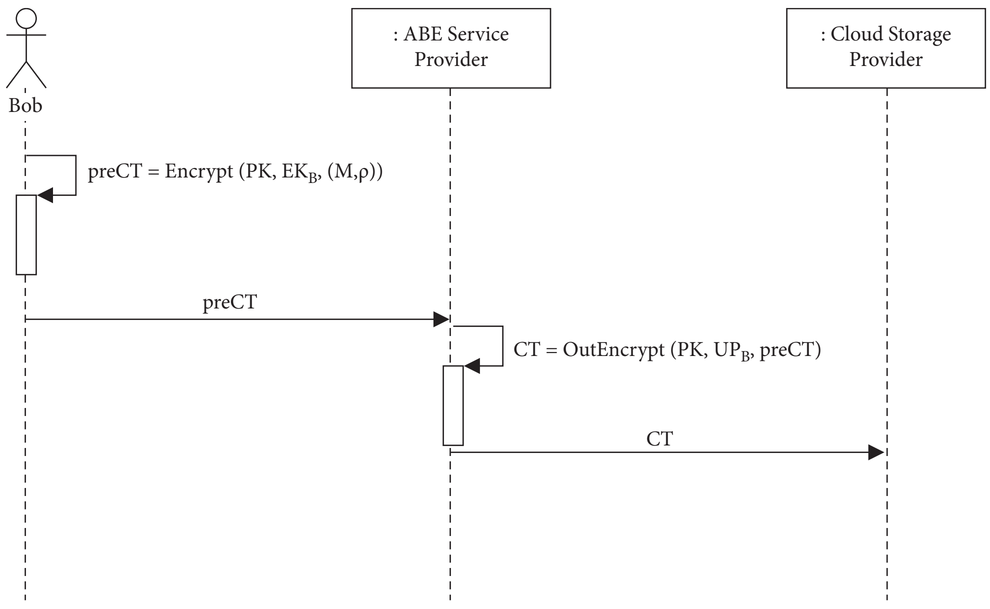
Bob
: ABE Service Provider
: Cloud Storage Provider
preCT = Encrypt (PK, EK_B, (M,ρ))
preCT
CT = OutEncrypt (PK, UP_B, preCT)
CT

Figure 2: Encryption phase's sequence diagram.

(4) *Decryption Phase*. First, as shown in Figure 3, Alice requests the transformed ciphertext transCT from the ABE Service Provider. The ABE Service Provider receives CT from the CSP and executes the function OutDecrypt$(\mathrm{PK}, \mathrm{CT}, \mathrm{TK}_A)$.

OutDecrypt$(\mathrm{PK}, \mathrm{CT}, \mathrm{TK}_A)$ is defined as follows:

(i) If Alice's set of attributes S_A does not satisfy the access structure, then it outputs $\perp$. Otherwise, let $I = \{i : \rho(i) \in S_A\}$ and $\{w_i \in Z_p\}_{i\in I}$ such that $\sum_{i\in I} w_i.M_i = (1, 0, 0, \ldots, 0)$.

(ii) Then, it computes

$$\begin{aligned} A &= \prod_{i\in I}\left(\frac{e\left(D_{A,i}, C_i\right)}{e\left(D'_{A,i}, C'_i\right)}\right)^{w_i} = \prod_{i\in I}\left(\frac{e\left(g^{r_A/z_A}.H_1(\rho(i))^{r_{A,i}/z_A}, g^{\lambda_i}\right)}{e\left(g^{r_{A,i}/z_A}, H_1(\rho(i))^{\lambda_i}\right)}\right)^{w_i} = \prod_{i\in I}\left(\frac{e\left(g^{r_A}.H_1(\rho(i))^{r_{A,i}}, g\right)}{e\left(g^{r_{A,i}}, H_1(\rho(i))\right)}\right)^{w_i.\lambda_i/z_A} \\ &= \prod_{i\in I} e(g,g)^{r_A.w_i.\lambda_i/z_A} = e(g,g)^{r_A/z_A.\sum_{i\in I} w_i.\lambda_i} = e(g,g)^{r_A.s/z_A}. \end{aligned} \tag{4}$$

(iii) Finally, it outputs transCT generated as follows:

$$\mathrm{transCT} = \frac{e(C, D_A)}{A} = \frac{e\left(g^{\beta.s}, g^{(\alpha+r_A)/\beta.z_A}\right)}{e(g,g)^{r_A.s/z_A}} = \left(\frac{e(g,g)^{\alpha}.e(g,g)^{r_A}}{e(g,g)^{r_A}}\right)^{s/z_A} = e(g,g)^{\alpha.s/z_A}. \tag{5}$$

The ABE Service Provider sends transCT to Alice.

After receiving transCT, Alice decrypts it using its decryption key DK_A by calling the function Decrypt$(\mathrm{PK}, \mathrm{transCT}, \mathrm{DK}_A)$.

Decrypt$(\mathrm{PK}, \mathrm{transCT}, \mathrm{DK}_A)$ executes the following instructions:

(i) It computes

$$\mathrm{Key} = \mathrm{transCT}^{\mathrm{DK}_A} = \left(e(g,g)^{\alpha.s/z_A}\right)^{z_A} = e(g,g)^{\alpha.s}. \tag{6}$$

(ii) Then, it outputs the Key.

4.1.2. Security Analysis

Before starting our security proof, we will create a modified version of Bethencourt et al.'s CP-

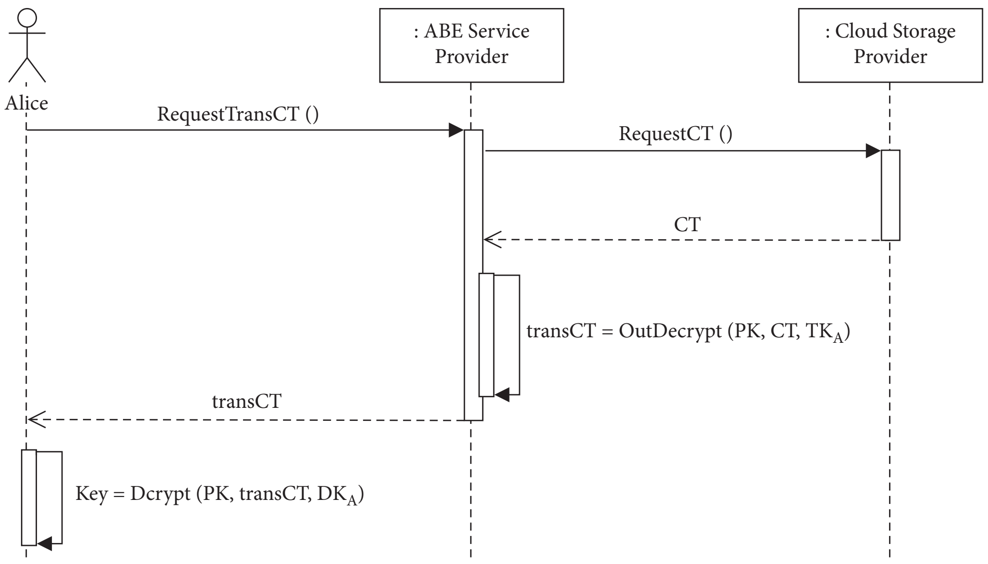

Figure 3: Decryption phase's sequence diagram.

ABE scheme [4] and prove that it achieves the same security level of the original scheme in the random oracle. Then, we will prove that our 1st CP-ABE-OED KEM is selectively CPA-secure in the random oracle if the modified version of Bethencourt et al.'s scheme is selectively CPA-secure in the random oracle. Thus, our 1st CP-ABE-OED KEM is selectively CPA-secure in the random oracle if Bethencourt et al.'s scheme is selectively CPA-secure in the random oracle.

We create the modified Bethencourt et al.'s scheme by modifying the encryption phase as follows:

(i) Instead of using Shamir's Secret Sharing Scheme to build the access policy, we use the LSSS access structure (M, ρ) in the same way as in our scheme and generate the shares λ_i.

(ii) We choose a random number $s_B \in Z_p$ and compute for each row i of M: $\mathscr{R}_i = H_2(\rho(i)||s_B).\lambda_i$ (the hash function $H_2 : \{0,1\}^* \longrightarrow Z_p$ must be defined in the public parameters).

(iii) The generated ciphertext will be defined as follows:

$$\begin{cases} (M,\rho), \\ \widetilde{C} = M.e(g,g)^{\alpha.s} \\ h^s, \\ \forall i = 1,\dots,l : \begin{cases} \mathscr{R}_i = H_2(\rho(i)||s_B).\lambda_i \\ C_i = g^{\lambda_i}, \\ C_i' = H_1(\rho(i))^{\lambda_i}. \end{cases} \end{cases} \quad (7)$$

It is obvious that the modified Bethencourt et al.'s scheme achieves the same security level as the original scheme in the random oracle. That is because if we consider $H_2(\rho(i)||s_B)$ random, then $\mathscr{R}_i$ is random and the attacker cannot compute λ_i from $\mathscr{R}_i$ without knowing $H_2(\rho(i)||s_B)$. Therefore, an attacker cannot distinguish between the distributions $(H_2(\rho(i)||s_B).\lambda_i, g^{\lambda_i}, H_1(\rho(i))^{\lambda_i})$ and $(r, g^{\lambda_i}, H_1(\rho(i))^{\lambda_i})$, where $r \in Z_p$ is a random number.

Now, we prove the following theorem:

Theorem 1. *Our 1st CP-ABE-OED KEM is selectively CPA-secure in the random oracle if the modified Bethencourt et al. scheme is selectively CPA-secure in the random oracle.*

Suppose that we have an adversary $\mathscr{A}$ with non-negligible advantage ε in the selective CPA-security game against our construction. We show how to build a simulator $\mathscr{B}$ that can attack the modified Bethencourt et al. scheme in the selective CPA-security model with advantage ε.

(1) *Init.* The adversary gives the challenge access structure (M^*, ρ^*) to the simulator $\mathscr{B}$. $\mathscr{B}$ sends (M^*, ρ^*) to the challenger.

(2) *Setup.* The simulator $\mathscr{B}$ obtains the public parameters from the challenger:

$$\mathrm{PK}' = \left\{G,\ g, e(\cdot,\cdot), H_1(\cdot),\ H_2(\cdot),\ h = g^{\beta'}, f = g^{1/\beta'}, e(g,g)^{\alpha'}\right\}. \quad (8)$$

The random oracles $H_1(\cdot)$ and $H_2(\cdot)$ are programmed by the challenger.

Then, $\mathscr{B}$ sends the public parameters

$$\mathrm{PK} = \left\{G,\ g, e(\cdot,\cdot), H_1(\cdot), H_2(\cdot),\ h = g^{\beta'}, e(g,g)^{\alpha'}\right\}, \quad (9)$$

to the adversary $\mathscr{A}$

(3) *Phase I.* The adversary sends request queries of sets of attributes S that do not satisfy the challenge access structure (M^*, ρ^*) to $\mathscr{B}$. The simulator $\mathscr{B}$ calls the challenger's key generation oracle on S to obtain the key

$$\mathrm{SK}' = \left\{ \begin{array}{c} S \\ D' = g^{(\alpha'+r)/\beta'} \\ \forall j \in S_i\colon \left\{ \begin{array}{c} D'_j = g^r . H_1(j)^{r_j} \\ D'^{,}_j = g^{r_j} \end{array} \right\} \end{array} \right\}. \quad (10)$$

The simulator chooses a random value $z \in Z_p$ and sets the decryption key as DK $= z$ and the transformation key as

$$\mathrm{TK} = \left\{ \begin{array}{c} S \\ D = D'^{1/z} \\ \forall j \in S_i\colon \left\{ \begin{array}{c} D_j = D'^{1/z}_j \\ D'_j = D'^{,1/z}_j \end{array} \right\} \end{array} \right\}. \quad (11)$$

(4) *Challenge*. The simulator sends two distinct random messages m_0 and m_1 to the challenger. The challenger flips a coin $\pi \in \{0, 1\}$ and creates

$$\mathrm{CT}' = \left\{ \begin{array}{l} (M^*, \rho^*), \\ \mathbb{C}' = m_\pi . e(g,g)^{\alpha' s'}, \\ C' = h^{s'}, \\ \forall i = 1, \ldots, l\colon \left\{ \begin{array}{l} \mathscr{R}_i = H_2(\rho(i) \| s_B) . \lambda'_i, \\ C'_i = g^{\lambda'_i}, \\ C'^{,}_i = H_1(\rho(i))^{\lambda'_i}. \end{array} \right. \end{array} \right. \quad (12)$$

Then, the challenger sends CT$^{'}$ to the simulator.
Later on, the simulator computes

$$\mathrm{UP}^* = \left\{ \begin{array}{c} \left\{ \begin{array}{l} \mathrm{UP}_{u,1} = (C'_i)^{1/\mathscr{R}_i} \\ \mathrm{UP}_{u,2} = (C'_i)^{1/\mathscr{R}_i} \end{array} \right\}_{\forall u \in U \cap Y} \\ \left\{ \begin{array}{l} \mathrm{UP}_{u,1} = g^{1/t_u} \\ \mathrm{UP}_{u,2} = x_u^{1/t_u} \end{array} \right\}_{\forall u \in U \setminus Y} \end{array} \right\}. \quad (13)$$

where $t_u \in Z_p$ and $x_u \in G$ are random numbers.
Then, $\mathscr{B}$ creates CT* as follows:

$$\mathrm{CT}^* = \left\{ \begin{array}{l} (M^*, \rho^*), \\ C = C', \\ C_i^{\mathrm{pre}} = \mathscr{R}_i, \end{array} \right. \quad (14)$$

Finally, the simulator flips a coin $b \in \{0, 1\}$ and computes Key$^* = \mathbb{C}'/\mathrm{m_b}$ and then sends CT*, UP*, and Key* to the adversary.

(5) *Phase 2*. The simulator continues to answer queries as in Phase 1.

(6) *Guess*. The adversary will eventually output a guess b' of b. The adversary outputs 0 to guess that Key* is random, and outputs 1 to guess that Key$^* = e(g,g)^{\alpha' s'}$. The simulator outputs b if $b' = 1$; otherwise it outputs $\bar{b}$. Thus, if the adversary wins the selective CPA-security game with a non-negligible advantage, then $\mathscr{B}$ can break the security of the modified Bethencourt et al.'s scheme with the same advantage.

4.2. The 2nd Proposed CP-ABE-OED Key Encapsulation Mechanism

4.2.1. The Construction. In this scheme, we consider the ABE Service Provider semi-trusted, which means that it cannot collude with illegitimate users to reveal the plaintext.

We will only describe the modified methods that are different from the previous scheme.

(1) KeyGen(U, MK, S_i)

(i) First, it generates the encryption key EK$_i = s_i$ where s_i is picked randomly in Z_p, and the decryption key DK$_i = z_i$ where z_i is a random number in Z_p.

(ii) Afterward, it computes the user's parameters as follows: $\mathrm{UP}_i = \left\{ \begin{array}{c} \mathrm{UP}_{i,1} = g^{1/s_i}, \\ \forall u \in U\colon \mathrm{UP}_{i,u,2} = H_1(u)^{1/s_i}. \end{array} \right\}$

(iii) It also computes the Transformation Key (TK$_i$). First, it chooses a random number $r_i \in Z_p$ and for each $j \in S_i$ it picks $r_{i,j} \in Z_p$ randomly. Then, it computes TK$_i$ = $\left\{ \begin{array}{c} S_i \\ D_i = g^{(\alpha + r_i)/\beta . z_i} \\ \forall j \in S_i\colon \left\{ \begin{array}{c} D_{i,j} = g^{r_i/z_i} . H_1(j)^{r_{i,j}/z_i} \\ D^{,}_{i,j} = g^{r_{i,j}/z_i} \end{array} \right\} \end{array} \right\}$.

(iv) Finally, it outputs (EK$_i$, DK$_i$, UP$_i$, TK$_i$).

(2) Encrypt(PK, EK$_B$, (M, ρ))

(i) It picks a random number $s \in Z_p$ and computes

$$\mathrm{preCT} = \left\{ \begin{array}{c} (M, \rho), \\ C = h^s, \\ C^{\mathrm{pre}} = s_B . s \end{array} \right\}. \quad (15)$$

(ii) Then, it outputs preCT.

(3) OutEncrypt(PK, UP$_B$, preCT)

(i) First, it chooses a random column vector $v = (C^{\mathrm{pre}}, y_2, y_3, \ldots, y_n) \in Z_p^n$.

(ii) For each $i \in \{1, 2, \ldots, l\}$, it computes $\lambda_i = M_i . v$ where M_i is the ith row of M.

(iii) Then, for each $i \in \{1, 2, \ldots, l\}$, it computes

$$\left\{ \begin{array}{l} C_i = \mathrm{UP}_{B,1}^{\lambda_i} = g^{\lambda_i/s_B} \\ C'_i = \mathrm{UP}_{B,\rho(i),2}^{\lambda_i} = H_1(\rho(i))^{\lambda_i/s_B}, \end{array} \right. \quad (16)$$

(iv) Then, it outputs CT $= \{(M, \rho), C, \{C_i, C'_i\}_{i \in \{1,2,\ldots,l\}}\}$.

The decryption phase will perform in the same way as in the previous scheme; however, we will describe it here to show the correctness of our scheme.

(4) OutDecrypt(PK, CT, TK$_A$).

(i) If Alice's set of attributes S_A does not satisfy the access structure, then it outputs $\perp$. Otherwise, let $I = \{i : \rho(i) \in S_A\}$ and $\{w_i \in Z_p\}_{i\in I}$ such that $\sum_{i\in I} w_i.M_i = (1,0,0,\ldots,0)$.

(ii) Then, it computes

$$A = \prod_{i\in I}\left(\frac{e\left(D_{A,i}, C_i\right)}{e\left(D^{'}_{A,i}, C'_i\right)}\right)^{w_i} = \prod_{i\in I}\left(\frac{e\left(g^{r_A/z_A}.H_1(\rho(i))^{r_{A,i}/z_A}, g^{\lambda_i/s_B}\right)}{e\left(g^{r_{A,i}/z_A}, H_1(\rho(i))^{\lambda_i/s_B}\right)}\right)^{w_i} = \prod_{i\in I}\left(\frac{e\left(g^{r_A}.H_1(\rho(i))^{r_{A,i}}, g\right)}{e\left(g^{r_{A,i}}, H_1(\rho(i))\right)}\right)^{w_i.\lambda_i/z_A.s_B}$$
$$= \prod_{i\in I} e(g,g)^{r_A.w_i.\lambda_i/z_A.s_B} = e(g,g)^{r_A/z_A.s_B.\sum_{i\in I} w_i.\lambda_i} = e(g,g)^{r_A.s_B.s/z_A.s_B} = e(g,g)^{r_A.s/z_A}. \quad (17)$$

(iii) Finally, it outputs transCT generated as follows:

$$\text{transCT} = \frac{e(C, D_A)}{A} = \frac{e\left(g^{\beta.s}, g^{(\alpha+r_A)/\beta.z_A}\right)}{e(g,g)^{r_A.s/z_A}} = \left(\frac{e(g,g)^{\alpha}.e(g,g)^{r_A}}{e(g,g)^{r_A}}\right)^{s/z_A} = e(g,g)^{\alpha.s/z_A}. \quad (18)$$

(5) Decrypt(PK, transCT, DK_A)

(i) It computes

$$\text{Key} = \text{transCT}^{\text{DK}_A} = \left(e(g,g)^{\alpha.s/z_A}\right)^{z_A} = e(g,g)^{\alpha.s} \quad (19)$$

(ii) Then, it outputs the Key.

4.2.2. Security Analysis. In this security proof, we will consider two types of adversaries:

(i) Type-1 adversary: which refers to illegitimate users trying to break our scheme

(ii) Type-2 adversary: which refers to a curious ABE cloud provider trying to reveal sensitive information

For the Type-1 adversary, our scheme is viewed as Bethencourt et al.'s scheme [4] with outsourced decryption.

Now, we prove the following theorem:

Theorem 2. *Our 2nd CP-ABE-OED KEM is selectively CPA-secure in the random oracle against Type-1 adversaries if Bethencourt et al.'s scheme [4] is selectively CPA-secure in the random oracle.*

Suppose we have an adversary $\mathscr{A}$ with non-negligible advantage ε in the selective CPA-security game against our construction. We show how to build a simulator $\mathscr{B}$ that can attack Bethencourt et al.'s scheme [4] in the selective CPA-security model with advantage ε.

(1) *Init.* The adversary gives the challenge access structure (M^*, ρ^*) to the simulator $\mathscr{B}$. $\mathscr{B}$ sends the challenge access structure to the challenger.

(2) *Setup.* The simulator $\mathscr{B}$ obtains Bethencourt et al.'s [4] public parameters

$$\text{PK}^{'} = \left\{G, g, e(\cdot,\cdot), H_1(\cdot), h = g^{\beta'}, f = g^{1/\beta'}, e(g,g)^{\alpha'}\right\}. \quad (20)$$

and forwards them to the adversary $\mathscr{A}$.

(3) *Phase I.* The adversary sends request queries of sets of attributes S that do not satisfy the challenge access structure (M^*, ρ^*) to $\mathscr{B}$. The simulator $\mathscr{B}$ calls Bethencourt et al.'s [4] key generation oracle on S to obtain the key

$$\text{SK}' = \left\{\begin{array}{c} S \\ D' = g^{(\alpha'+r)/\beta'} \\ \forall j \in S_i : \left\{\begin{array}{c} D'_j = g^{r}.H_1(j)^{r_j} \\ D^{''}_j = g^{r_j} \end{array}\right\} \end{array}\right\}. \quad (21)$$

The simulator chooses a random value $z \in Z_p$ and sets the decryption key as DK $= z$ and the transformation key as

$$\text{TK} = \left\{\begin{array}{c} S \\ D = D'^{1/z} \\ \forall j \in S_i : \left\{\begin{array}{c} D_j = D^{'1/z}_j \\ D'_j = D^{''1/z}_j \end{array}\right\} \end{array}\right\}. \quad (22)$$

(4) *Challenge.* The simulator sends two distinct random messages m_0 and m_1 to the challenger. The challenger flips a coin $\pi \in \{0,1\}$ and creates

$$\text{CT}' = \left\{\begin{array}{l} (M^*, \rho^*), \\ \mathbb{C}' = m_\pi.\text{Key}^{'}, \\ C' = h^{s'}, \\ \forall i = 1,\ldots,l : \left\{\begin{array}{l} C'_i = g^{\lambda'_i}, \\ C'_i = H_1(\rho(i))^{\lambda'_i} \end{array}\right. \end{array}\right. \quad (23)$$

Table 1: Notations used in the performance results and analysis section.

Notation	Definition
$\|x\|$	Number of elements in x
l	Number of rows in the LSSS access structure or number of leaf nodes in the access tree
N_l	An upper bound greater than l of all the access policies
S	User's set of attributes
U	Universe of attributes
M_{eG}	Modular exponentiation in G
M_{eG_T}	Modular exponentiation in G_T
M_G	Multiplication in G
M_{G_T}	Multiplication in G_T
M_Z	Multiplication in Z_p
H_G	Hashing in G
H_Z	Hashing in Z_p
G	Element in G
G_T	Element in G_T
Z_p	Element in Z_p

Then, the challenger sends CT$^{'}$ to the simulator.

Later on, the simulator computes

$$\mathrm{UP}^* = \left\{ \left\{ \begin{array}{c} \mathrm{UP}_{u,1} = g^{1/t_u} \\ \mathrm{UP}_{u,2} = H_1(u)^{1/t_u} \end{array} \right\}_{\forall u \in U} \right\}. \quad (24)$$

where $t_u \in Z_p$ are random numbers.

Later on, the simulator constructs CT* as follows:

$$\mathrm{CT}^* = \begin{cases} (M^*, \rho^*), \\ C^* = C', \\ \forall i = 1, \dots, l: \begin{cases} C_i^* = C_i', \\ C'^*_i = C_i^{''}. \end{cases} \end{cases} \quad (25)$$

Finally, the simulator flips a coin $b \in \{0,1\}$ and computes Key$^* = \mathbb{C}'/m_b$, then sends CT*, UP*, and Key* to the adversary.

(5). *Phase 2.* The simulator continues to answer queries as in Phase 1.

(6). *Guess.* The adversary will eventually output a guess b' of b. The adversary outputs 0 to guess that Key* is random, and outputs 1 to guess thatKey$^* = e(g,g)^{\alpha' s'}$. The simulator outputs b if $b' = 1$; otherwise it outputs $\overline{b}$. Thus, if the adversary wins the selective CPA-security game with a non-negligible advantage, then $\mathscr{B}$ can break the security of Bethencourt et al.'s scheme with the same advantage.

The Type-2 adversary is not allowed to collude with unauthorized users. Thus, he can request only the transformation keys and not the decryption keys from the key generation oracle.

Now, we prove the following theorem:

Theorem 3. *Our 2nd CP-ABE-OED KEM is selectively CPA-secure in the random oracle against Type-2 adversaries if our 2nd CP-ABE-OED KEM is selectively CPA-secure in the random oracle against Type-1 adversaries.*

It is obvious that Type-2 adversary cannot distinguish between two pre-ciphertexts preCT$_0^*$ and preCT$_1^*$ where

$$\mathrm{preCT}_b^* = \left\{ \begin{array}{c} (M^*, \rho^*), \\ \mathrm{Key}_b^*, \\ C^* = h^s, \\ C^{\mathrm{pre}*} = s_B.s \end{array} \right\}. \quad (26)$$

and $\mathrm{Key}_b^* = \begin{cases} e(g,g)^{\alpha.s}, & b = 0, \\ e(g,g)^R, & b = 1. \end{cases}$ where $R \in Z_p$ is a random number.

That is because $C^{\mathrm{pre}*}$ is random since s_B is random, and the adversary cannot retrieve s without knowing s_B. Thus, Type-2 adversary has no advantage over Type-1 adversary since the only additional information ($C^{\mathrm{pre}*} = s_B \cdot s$) he has compared to the Type-1 adversary is not useful. Hence, Theorem 3 is proved.

5. Performance Results and Analysis

5.1. Theoretical Results and Analysis. In this section, we theoretically compare the user's computation, communication, and storage costs between our two schemes and the following schemes:

(i) The CPA-secure construction of [12].
(ii) The CPA-secure construction of [21] based on [12] using Pedersen Commitment as a commitment scheme.
(iii) Zhou et al.'s scheme [23].
(iv) Zhang et al.'s scheme [26].
(v) Li et al.'s scheme [27].

We normalized all the schemes based on the following rules:

(i) The transformation key is created by TA and not the user.
(ii) We will consider all the schemes as Key Encapsulation Mechanisms (KEMs), meaning that we neglect the part where the message m is encrypted (e.g., $C = m \cdot e(g,g)^{\alpha.s}$) and leave only the parts responsible for sharing the key $e(g,g)^{\alpha.s}$.

Table 2: User-side computation cost comparison.

Phase	CP-ABE-OED1	CP-ABE-OED2	[12]	[21]	[23]	[26]	[27]
KeyGen	0	0	0	0	0	0	0
Encrypt	$1.M_{eG} + l.M_Z + l.H_Z$	$1.M_{eG} + 1.M_Z$	$(2 + 3.l).M_{eG} + l.H_G$	$(3 + 3.l).M_{eG} + (2 + l).H_G$	$3.M_{eG} + 1.H_G$	$1.M_{eG} + (1 + 3.N_l).M_G + l.M_Z$	$3.M_{eG} + l.M_G + 1.H_Z$
Decrypt	$1.M_{eG_T}$	$1.M_{eG_T}$	$1.M_{eG_T}$	$3.M_{eG_T}$	$1.M_{eG_T} + 1.M_{G_T}$	$1.M_{eGT}$	$1.M_{eG_T}$

Table 3: Running times (in milliseconds) of the main operations using JPBC Library [32] on a Windows 8.1 Core i7 2 GHz PC with 8 Go of RAM.

M_{eG}	M_{eG_T}	M_G	M_{G_T}	M_Z	H_G	H_Z
14.5 ms	0.924 ms	0.073 ms	0.0072 ms	0.0014 ms	31.07 ms	0.118 ms

(iii) We consider that each user has the ability to encrypt and decrypt.

(iv) We ignored the access structure $\mathbb{A}$ and the set of attributes S when computing the size of the ciphertexts and the keys since they are common elements between all the schemes.

In Table 1, we define the notations used in this section.

In Table 2, we compare the number of operations executed in each phase (registration phase, encryption phase, and decryption phase) between our proposed schemes and the reviewed schemes.

Obviously, the user is not involved in the computations of the registration phase in all the schemes.

In [12, 21], the user-side encryption cost is very expensive, because the encryption in these schemes is not outsourced. Based on the results in Table 3, which were computed using a Type A curve of the JPBC Library [32] on a Windows 8.1 Core i7 2 GHz PC with 8 GB of RAM, we have the following:

(i) $M_G = 52M_Z$

(ii) $M_{eG} = 10357M_Z$

(iii) $H_G = 22193M_Z$

(iv) $H_Z = 84M_Z$

We mention that the hashes were computed using the Element.setFromHash() method based on SHA-256.

If we convert the encryption costs of the reviewed schemes, we get the following:

(i) $1.M_{eG} + l.M_Z + l.H_Z = (10357 + 85.l).M_Z$ for our 1st CP-ABE-OED KEM

(ii) $1.M_{eG} + 1.M_Z = 10358.M_Z$ for our 2nd CP-ABE-OED KEM

(iii) $3.M_{eG} + 1.H_G = 53264.M_Z$ for [23]

(iv) $1.M_{eG} + (1 + 3.N_l).M_G + l.M_Z = (10409 + 156.N_l + l).M_Z$ for [26]

(v) $3.M_{eG} + l.M_G + 1.H_Z = (52.l + 31155).M_Z$ for [27]

We observe that, for access policies smaller than 425 leaf nodes, the user-side encryption in [27] is more efficient than [23].

If $N_l = l$, which is the smallest value N_l can take, the user-side encryption in [26] will be more efficient than [27] (respectively, [23]) for access policies with less than 200 leaf nodes (respectively, 270 leaf nodes). If $N_l = 5.l$, [27] (respectively, [23]) will achieve better efficiency than [26] for all the access policies bigger than 30 leaf nodes (respectively, 50 leaf nodes). Overall, we can say that [26] is more efficient than [23, 27] for small access policies; however, [23, 27] are more efficient for large access policies.

Our 1st CP-ABE-OED KEM achieves a higher user-side encryption efficiency than [26] for all the access policy sizes. It also achieves higher efficiency than [23] for access policies with less than 500 leaf nodes, and higher efficiency than [27] for access policies with less than 630 leaf nodes.

Obviously, our 2nd CP-ABE-OED KEM is more efficient than all the schemes for all the access policy sizes.

The decryption phase costs are almost the same (one modular exponentiation) in all the schemes since they all use the same key blinding technique used in [12]. In [21], the user performs 2 more modular exponentiations to reveal the commitment.

Table 4 shows the communication costs generated in the registration phase, the encryption phase, and the decryption phase between our proposed schemes and the reviewed schemes. In [12, 21, 23, 26], TA sends TK and DK to the user in the registration phase, which costs $(2 + 2.|S|)$ elements in G and $(2 + |S|)$ elements in Z_p for [26], $(1 + 2.|S|)$ elements in G and one element in Z_p for [23], and $(2 + |S|)$ elements in G and one Z_p element for [12, 21]. In [27], TA sends the encryption transformation key ETK and DK to the user, which costs two Z_p elements and $|U|$ elements in G. However, in our proposed schemes, only two elements in Z_p (EK and DK) are communicated between TA and the user. The reason is that TK in our proposed schemes is transferred by TA directly to the ABE Service Provider.

In the encryption phase, the user in [26] receives two Intermediate Ciphertexts (ITs) from the ABE Service Provider offline, each of them containing $(1 + 3.N_l)$ elements in G and $(1 + 3.N_l)$ elements in Z_p, and sends the ciphertext CT to CSP, which costs $(1 + 3.l)$ elements in G and $2.l$ elements in Z_p. This makes [26] the most expensive scheme for the users in terms of communication cost produced in the encryption phase. In [12, 21], the user communicates CT to CSP; this costs $(1 + 2.l)$ elements in G for [12] and an additional element in G for [21] generated by the commitment element cm. The user in [27] sends the partially encrypted ciphertext preCT and the outsourcing parameters to the ABE service provider. This costs $(2 + l)$ elements in G and $(l + 1)$ elements in Z_p, which makes [27] slightly more efficient than [12, 21]. In our 1st CP-ABE-OED KEM, the user sends preCT to the ABE Service Provider, which costs him one G element and l elements in Z_p. In [23], the user sends preCT that costs only 3 elements in G and one element in Z_p to the ABE Service Provider. In our 2nd CP-ABE-OED KEM, the transfer of preCT to the ABE Service Provider costs only one G element and one Z_p element, which makes it the most efficient scheme in terms of user's communication cost in the encryption phase.

In the decryption phase, the user receives the transformed ciphertext transCT from the ABE Service Provider in all the schemes, which costs two G_T elements in [23] and

Table 4: User-side communication cost comparison.

Phase	KeyGen	Encrypt			Decrypt		
Link	TA => Bob/Alice	Bob => CSP	ABE Service => Bob	Bob => ABE Service	CSP => Alice	ABE Service => Alice	Alice => ABE Service
CP-ABE-OED1	$\|EK\| + \|DK\| = 2.Z_p$	0	0	$\|preCT\| = 1.G + l.Z_p$	0	$\|transCT\| = 1.G_T$	0
CP-ABE-OED2	$\|EK\| + \|DK\| = 2.Z_p$	0	0	$\|preCT\| = 1.G + 1.Z_p$	0	$\|transCT\| = 1.G_T$	0
[12]	$\|TK\| + \|DK\| = (2 + \|S\|).G + 1.Z_p$	$\|CT\| = (1 + 2.l).G$	0	0	0	$\|transCT\| = 1.G_T$	$\|TK\| = (2 + \|S\|).G$
[21]	$\|TK\| + \|DK\| = (2 + \|S\|).G + 1.Z_p$	$\|CT\| = (1 + 2.l).G$	0	0	1.G	$\|transCT\| = 1.G_T$	$\|TK\| = (2 + \|S\|).G$
[23]	$\|TK\| + \|DK\| = (1 + 2.\|S\|).G + 1.Z_p$	0	0	$\|preCT\| = 3.G + 1.Z_p$	0	$\|transCT\| = 2.G_T$	$\|TK\| = (1 + 2.\|S\|).G$
[26]	$\|TK\| + \|DK\| = (2 + 2.\|S\|).G + (2 + \|S\|).Z_p$	$\|CT\| = (1 + 3.l).G + 2.l.Z_p$	$2.\|IT\| = 2.((1 + 3.N_1).G + (1 + 3.N_1).Z_p)$	0	0	$\|transCT\| = 1.G_T$	$\|TK\| = (2 + 2.\|S\|).G + (1 + \|S\|).Z_p$
[27]	$\|ETK\| + \|DK\| = \|U\|.G + 2.Z_p$	0	0	$\|preCT\| + \|OP\| = (2 + l).G + (l + 1).Z_p$	0	$\|transCT\| = 1.G_T$	$\|TK\| = (2 + \|S\|).G$

Table 5: User-side storage cost comparison.

	CP-ABE-OED1	CP-ABE-OED2	[12]	[21]	[23]	[26]	[27]
Bob/ Alice	$\lvert EK\rvert + \lvert DK\rvert = 2.Z_p$	$\lvert EK\rvert + \lvert DK\rvert = 2.Z_p$	$\lvert TK\rvert + \lvert DK\rvert = (2 + \lvert S\rvert).G + 1.Z_p$	$\lvert TK\rvert + \lvert DK\rvert = (2 + \lvert S\rvert).G + 1.Z_p$	$\lvert TK\rvert + \lvert DK\rvert = (1 + 2.\lvert S\rvert).G + 1.Z_p$	$\lvert TK\rvert + \lvert DK\rvert + 2.\lvert IT\rvert = (4 + 2.\lvert S\rvert + 6.N_l).G + (4 + \lvert S\rvert + 6.N_l).Z_p$	$\lvert ETK\rvert + \lvert DK\rvert = \lvert U\rvert.G + 2.Z_p$

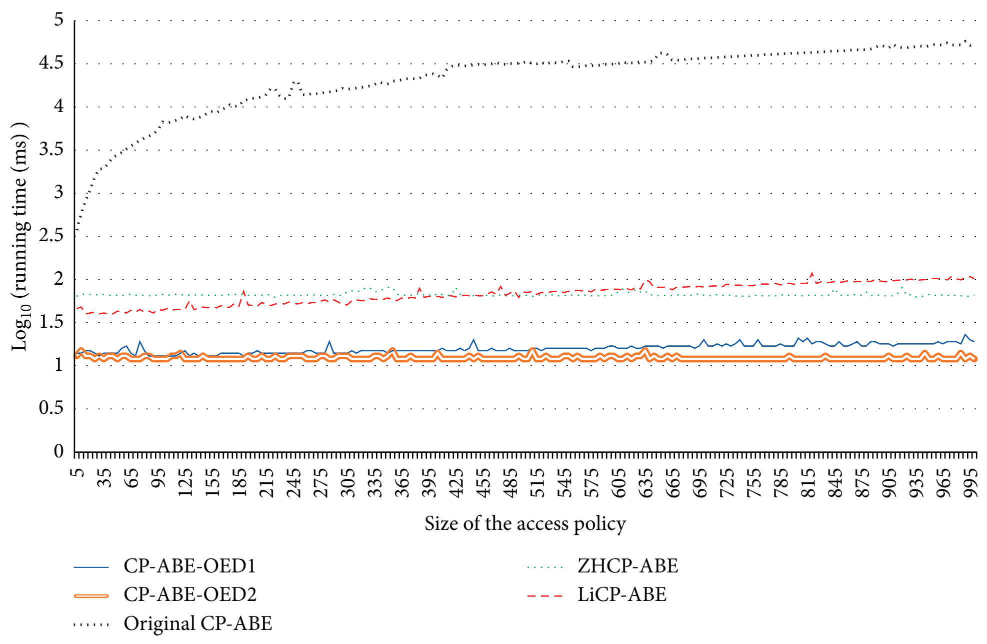

Figure 4: User-side encryption running time comparison.

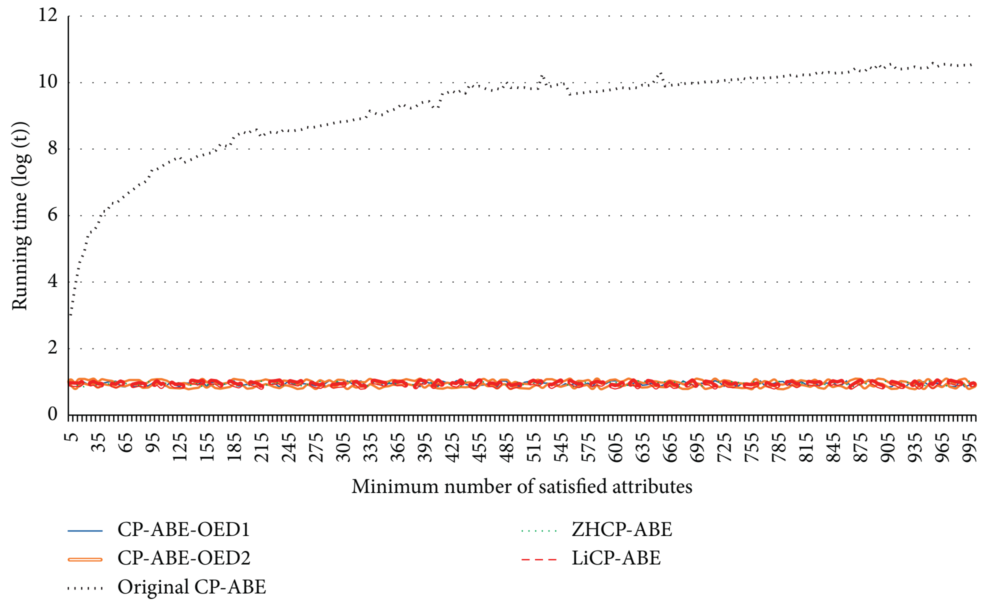

Figure 5: User-side decryption running time comparison.

only one G_T element in the other schemes. In addition, the user should first communicate his TK to the ABE Service Provider, which costs $(2 + 2.|S|)$ elements in G and $(1 + |S|)$ elements in Z_p for [26], $(1 + 2.|S|)$ elements in G for [23], and $(2 + |S|)$ elements in G for [12, 21, 27]. In [21], the user also receives the commitment from CSP, which costs one G element. In our proposed schemes, the user does not need to send or receive anything from CSP or the ABE Service Provider except transCT; TK is already sent by TA to the ABE Service Provider in the Registration Phase. Thus, our proposed schemes are the most efficient schemes in terms of user's communication cost produced in the decryption phase.

Table 5 compares the user's storage cost for each scheme. In general, [26] is the scheme that requires the biggest user-side storage space to store TK, DK, and two Intermediate Ciphertextx (ITs). The user in [27] stores $|U|$ elements in G and one Z_p element for the encryption transformation key ETK, and one Z_p element for the decryption key DK. Thus, for large universe applications, [27] is considered the most storage space consuming scheme for users. In [12, 21, 23], the user stores TK and DK. DK costs one Z_p element in all the schemes and TK costs $(1 + 2.|S|)$ elements in G for [23] and $(2 + |S|)$ elements in G for [12, 21]. In our proposed schemes, the user stores the encryption key EK and the decryption key DK; each of them costs only one Z_p element. Therefore, our proposed schemes are the most lightweight schemes in terms of user-side storage.

5.2. Experimental Results and Analysis. In this section, we will experimentally compare the running times of the user-side encryption and decryption of our 1st outsourced CP-ABE scheme (CP-ABE-OED1), our 2nd outsourced CP-ABE scheme (CP-ABE-OED2), the original CP-ABE scheme [4], ZHCP-ABE [23], and LiCP-ABE [27]. The implementations of the studied schemes were developed in Java using the JPBC Library [32] and the hashes were computed using setFromHash method of the *Element* class based on *SHA-256*.

We run 200 experiments for each $N = \{5, 10, 15, 20, 25, \ldots, 1000\}$ on a Windows 8.1 Core i7 2 GHz PC with 8 GB of RAM where the access policy is defined as follows (A_1 AND A_2 AND $A_3 \ldots$ AND A_N) and the user's set of attributes is $\{A_1, A_2, A_3, \ldots, A_N\}$. This approach simulates the worst-case scenario where the decryption phase depends on all the access policy's components. For each N, we repeat the experiment 10 times and calculate the average running time in milliseconds to smooth any experimental variability.

In Figure 4, the x-axis represents the size of the access policy and the y-axis represents the Log_{10} of the user-side encryption running time in milliseconds.

The experimental results confirmed our theoretical results. Besides, the theoretical results showed that CP-ABE-OED1 is more efficient than ZHCP-ABE [23] (respectively, LiCP-ABE [27]) for access policies with less than 500 leaf nodes (respectively, for access policies with less than 630 leaf nodes). However, the experimental results showed that CP-ABE-OED1 is more efficient than ZHCP-ABE [23] and LiCP-ABE [27] for all the access policy sizes up to 1000.

We observe that the difference in running time between CP-ABE-OED1 and CP-ABE-OED2 is linearly increasing with a relatively small slope, and this is due to the number of multiplications and hashing operations performed in CP-ABE-OED1 that is linear to the size of the access policy.

In Figure 5, the x-axis represents the size of the user's set of attributes and the y-axis represents the Log_2 of the user-side decryption running time in milliseconds.

As expected, the running times of the user-side decryption in all the studied outsourced CP-ABE schemes are constant and equivalent; that is because they all used the same decryption outsourcing technique firstly proposed by [12]. The user needs only about 2 ms (since $\text{Log}_2(t) = 1$ according to Figure 5) to decrypt a ciphertext regardless the size of the access policy or the length of her set of attributes.

6. Conclusion

In this paper, we proposed two efficient CP-ABE Key Encapsulation Mechanisms that can be provided as services in the cloud, minimizing the user-side computation, communication, and storage costs. The first scheme is suitable for applications where the ABE Service Provider is untrusted, whereas the second scheme, which is more efficient, requires the ABE Service Provider to be at least semi-trusted. Both schemes are proved to be selectively CPA-secure in the random oracle. However, our systems support only one TA that is responsible for the registration of all the users. Hence, our systems will face a bottleneck problem if TA does not use a very powerful device or if the registration requests are very frequent. Therefore, in the future, it will be interesting to extend our schemes to use a multi-authority architecture to handle this problem. Converting our schemes to support a multi-authority architecture might also improve the security of the systems by preventing the key-escrow problem produced when attackers compromise the TA's master key. In a multi-authority approach, compromising some authorities' master keys by attackers will not break the system.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Center for Scientific and Technical Research (CNRST) (scholarship number: 4UIZ2017).

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "A View of Cloud Computing: Clearing the clouds away from the true potential and obstacles posed by this computing capability," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Aarhus, Denmark, May 2005.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security—CCS '06*, p. 89, Alexandria, VA, USA, October 2006.

[4] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the 2017—IEEE Symposium on Security and Privacy*, pp. 321–334, Berkeley, CA, USA, May 2007.

[5] Y. Yacobi, "A fast attribute based encryption," *IACR Cryptology ePrint Archive*, vol. 304, 2016.

[6] S. Hohenberger and B. Waters, "Attribute-based encryption with fast decryption," in *Proceedings of the International Conference on Public-Key Cryptography—PKC 2013*, pp. 162–179, Nara, Japan, February 2013.

[7] K. Zhang, J. Ma, J. Zhang, Z. Ying, T. Zhang, and X. Liu, "Online/offline traceable attribute-based encryption," *Journal*

of Computer Research and Development, vol. 55, pp. 216–224, 2018.

[8] S. Hohenberger and B. Waters, *Online/Offline Attribute-Based Encryption*, in *Proceedings of the IACR International Conference on Public-Key Cryptography*, Buenos Aires, Argentina, March 2014.

[9] S. Ding, C. Li, and H. Li, "A novel efficient pairing-free CP-ABE based on elliptic curve cryptography for IoT," *IEEE Access*, vol. 6, pp. 27336–27345, 2018.

[10] V. Odelu and A. K. Das, "Design of a new CP-ABE with constant-size secret keys for lightweight devices using elliptic curve cryptography," *Security and Communication Networks*, vol. 9, no. 17, pp. 4048–4059, 2016.

[11] V. Odelu, A. K. Das, and A. Goswami, "An efficient CP-ABE with constant size secret keys using ECC for lightweight devices," *IEEE Transactions on Consumer Electronics*, vol. 62, pp. 1–15, 2016.

[12] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of ABE ciphertexts," in *Proceedings of the 20th USENIX Conference on Security*, p. 34, Berkeley, CA, USA, August 2011.

[13] B. Waters, "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization," in *Proceedings of the IACR International Conference on Public-Key Cryptography*, vol. 6571, pp. 1–25, Taormina, Italy, March 2011.

[14] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469–472, 1985.

[15] M. Green, A. Akinyele, and M. Rushanan, *Libfenc*, The Functional Encryption Library.

[16] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Proceedings of the Annual International Cryptology Conference*, pp. 537–554, Santa Barbara, CA, USA, August 1999.

[17] J. Lai, R. H. Deng, C. Guan, and J. Weng, "Attribute-based encryption with verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 1343–1354, 2013.

[18] Q. Li, J. Ma, R. Li, X. Liu, J. Xiong, and D. Chen, "Secure, efficient and revocable multi-authority access control system in cloud storage," *Computers & Security*, vol. 59, pp. 45–59, 2016.

[19] B. Qin, R. H. Deng, S. Liu, and S. Ma, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 1384–1393, 2015.

[20] S. Lin, R. Zhang, H. Ma, and M. Wang, "Revisiting attribute-based encryption with verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 10, pp. 2119–2130, 2015.

[21] X. Mao, J. Lai, Q. Mei, K. Chen, and J. Weng, "Generic and efficient constructions of attribute-based encryption with verifiable outsourced decryption," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 533–546, 2016.

[22] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proceedings of the 12th Annual International Cryptology Conference*, pp. 129–140, Santa Barbara, CA, USA, August 1992.

[23] Z. Zhou and D. Huang, "Efficient and secure data storage operations for mobile cloud computing," in *Proceeding of the 2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pp. 37–45, Las Vegas, NV, USA, October 2012.

[24] J. Li, C. Jia, J. Li, and X. Chen, "Outsourcing encryption of attribute-based encryption with MapReduce," in *in Proceedings of the 14th International Conference, ICICS 2012,*, pp. 191–201, Hong Kong, China, October 2012.

[25] M. Asim, M. Petković, and T. Ignatenko, "Attribute-based encryption with encryption and decryption outsourcing," in *Proceedings of the 12th Australian Information Security Management Conference.*, pp. 21–28, Perth, Western Australia, December 2014.

[26] R. Zhang, H. Ma, and Y. Lu, "Fine-grained access control system based on fully outsourced attribute-based encryption," *Journal of Systems and Software*, vol. 125, pp. 344–353, 2017.

[27] J. Li, X. Li, L. Wang, D. He, H. Ahmad, and X. Niu, "Fuzzy encryption in cloud computation: efficient verifiable outsourced attribute-based encryption," *Soft Computing*, vol. 22, no. 3, pp. 707–714, 2018.

[28] P. Zhang, Z. Chen, J. K. Liu, K. Liang, and H. Liu, "An efficient access control scheme with outsourcing capability and attribute update for fog computing," *Future Generation Computer Systems*, vol. 78, pp. 753–762, 2018.

[29] J. Blömer, P. Günther, V. Krummel, and N. Löken, "Attribute-based encryption as a service for access control in large-scale organizations," in *Proceedings of the 11th International Symposium,Foundations and Practice of Security*, pp. 3–17, Montreal, QC, Canada, November 2018.

[30] A. Beimel, "Secure schemes for secret sharing and key distribution," *Tech. Inst. Technol. Fac. Comput. Sci.*, 1996.

[31] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of the 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 568–588, Tallinn, Estonia, May 2011.

[32] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 850–855, Kerkyra, Corfu, Greece, June 28 - July 1 2011.