On the Insecurity of a Method for Providing Secure and Private Fine-Grained Access to Outsourced Data

Alfredo Rial
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
Email: alfredo.rial@uni.lu

Abstract—The protection of sensitive data stored in the cloud is paramount. Among the techniques proposed to provide protection, attribute-based access control, which frequently uses ciphertext-policy attribute-based encryption (CPABE), has received a lot of attention in the last years. Recently, Jahan et al. (IEEE 40th Conference on Local Computer Networks, 2015) propose a scheme based on CPABE where users have reading and writing access to the outsourced data. We analyze the scheme by Jahan et al. and we show that it has several security vulnerabilities. For instance, the cloud server can get information about encrypted messages by using a stored ciphertext and an update of that ciphertext. As another example, users with writing access are able to decrypt all the messages regardless of their attributes. We discuss the security claims made by Jahan et al. and point out the reasons why they do not hold. We also explain that existing schemes can already provide the advantages claimed by Jahan et al.

1. Introduction

The protection of sensitive data stored in the cloud is paramount. Sensitive data must be protected not only from unauthorized users, but also from the cloud servers. Security and privacy issues are among the main reasons why many organizations refrain from outsourcing their private data to the cloud [1], [2], despite the cost savings provided by the cloud.

Previous Work. Different techniques have been proposed to protect data outsourced to the cloud [3], [4], [5]. Among them, attribute-based access control has been given a lot of attention in recent years [6], [7], [8], [9]. In attribute-based access control, users are given attributes, while records of data are associated with access control policies. Only the users whose attributes satisfy the access control policy associated with a record of data can access that record of data. Attribute-based access control is adequate for large environments such as the cloud. Because of the large number of users, a data owner may not know the identities of all the users who should be given access to a record of data, and therefore he is unable to apply user-based access control. Instead, the data owner describes an access control policy that describes the attributes that a user must possess in order to get access to data.

Ciphertext-policy attribute-based encryption (CPABE) [10], [11], [12], [13] is the main technique employed to provide attribute-based access control in a cloud environment. In CPABE, ciphertexts are associated with an access control policy. A user obtains a secret key associated to her attributes and is able to decrypt a ciphertext if her attributes satisfy the access control policy associated with the ciphertext. Different CPABE schemes have been proposed in the literature. In some schemes, the ciphertexts reveal the access control policy to which they are associated, while in other schemes the ciphertexts hide the access control policy.

Several schemes propose the use of attribute-based encryption to protect data outsourced to the cloud. These schemes can be classified into two groups: those that provide users with reading access [6], [7], [8], [9], and those that give users reading and writing access [14], [15], [16]. We will focus on the latter.

Zhao et al. [14] propose a scheme that employs CPABE and attribute-based signatures. In a nutshell, the data owner encrypts a record of data by using CPABE on input an access control policy. Then, the ciphertext is signed by using an attribute-based signature scheme on input a signing policy. To read data, a user whose attributes satisfy the access control policy verifies the signature and decrypts the ciphertext. To write data, a user must possess a secret key for attribute-based signatures that satisfies the signing policy. The cloud server verifies that the signing policy remains unmodified when a new ciphertext is uploaded.

Ruj et al. [15] propose a decentralized version of the scheme in [14]. In [15], when writing data, users can modify both the access control policy associated with the ciphertext and the signing policy associated with the signature. Liu et al. [16] propose a scheme where CPABE is employed along with a hierarchical structure of multiple authorities.

Our Contribution. Recently, Jahan et al. [17] propose a scheme that employs an extension of the CPABE scheme in [10] and any existentially unforgeable signature scheme. The main claimed advantages over the schemes in [14], [15], [16] are that their scheme does not allow a writer to modify the access control policy associated with a ciphertext and
that some computations related to the CPABE scheme can be outsourced.

We show that the scheme in [17] has several security vulnerabilities. Namely, the cloud server can get information about encrypted messages by using a stored ciphertext and an update of that ciphertext. Additionally, users with writing access are able to decrypt all the messages regardless of their attributes and without knowledge of the group secret key. We discuss the security claims made in [17] and point out the reasons why they do not hold.

Furthermore, we show that existing schemes, such as Zhao et al. [14], can apply the restriction that users with writing access are not able to modify the access control policy associated with a ciphertext. In addition, the CPABE computations that can be outsourced are mainly new computations introduced in [17], not those in the original CPABE scheme in [10].

**Outline of the paper.** The remainder of this paper is organized as follows. In Section 2, we describe the concept of CPABE and its security properties, and we recall the CPABE scheme in [10]. We also describe existentially unforgeable signature schemes. In Section 3, we describe in detail the scheme proposed in [17]. In Section 4, we describe the security vulnerabilities of the scheme proposed in [17], and in Section 5 we explain why the security claims made in [17] do not hold. We conclude in Section 6.

## 2. Technical Background

### 2.1. Bilinear Maps

Let $G$, $\tilde{G}$ and $G_t$ be groups of prime order $p$. A map $e : G \times \tilde{G} \rightarrow G_t$ must satisfy bilinearity, i.e., $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$; non-degeneracy, i.e., for all generators $g \in G$ and $\tilde{g} \in \tilde{G}$, $e(g, \tilde{g})$ generates $G_t$; and efficiency, i.e., there exists an efficient algorithm $\tilde{G}(k)$ that outputs the pairing group setup $(p, G, \tilde{G}, e, g, \tilde{g})$ and an efficient algorithm to compute $e(a, b)$ for any $a \in G$, $b \in \tilde{G}$. If $G = \tilde{G}$ the map is symmetric and otherwise asymmetric.

### 2.2. Access Structures and Access Trees

Let $\{s_1, s_2, \ldots, s_n\}$ be a set of attributes. A collection $P \subseteq 2^{\{s_1, s_2, \ldots, s_n\}}$ is monotone if, $\forall B$ and $C$, if $B \in P$ and $B \subseteq C$ then $C \in P$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $P$ of non-empty subsets of $\{s_1, s_2, \ldots, s_n\}$, i.e., $P \subseteq 2^{\{s_1, s_2, \ldots, s_n\}} \setminus \emptyset$. The sets in $P$ are called the authorized sets, and the sets not in $P$ are called the unauthorized sets.

Let $T$ be a tree representing an access structure. Each non-leaf node $x$ represents a threshold gate described by an amount $n_x$ of children and a threshold value $k_x$ such that $0 < k_x \leq n_x$. (When $k_x = 1$, $x$ represents an OR gate and, when $k_x = n_x$, $x$ represents an AND gate.) Each leaf node $x$ is described by an attribute.

We denote by $parent(x)$ the parent of node $x$, and by $att(x)$ the attribute associated with the leaf node $x$. The children of every node are ordered from 1 to $n_x$, $index(x)$ returns such a number associated with the node $x$.

Let $T$ be an access tree with root $r$, and $T_x$ the subtree of $T$ rooted at $x$ (hence $T$ equals $T_r$). $T_x(\mathcal{A})$ is a function that returns 1 if the set of attributes $\mathcal{A}$ satisfies $T_x$. $T_x(\mathcal{A})$ is computed recursively as follows. If $x$ is a non-leaf node, evaluate $T_x$ for all children $x'$ of node $x$. $T_x(\mathcal{A})$ returns 1 when at least $k_x$ children return 1. If $x$ is a leaf node, $T_x(\mathcal{A})$ returns 1 when $att(x) \in \mathcal{A}$.

### 2.3. Ciphertext-Policy Attribute-based Encryption

A ciphertext-policy attribute-based encryption (CPABE) scheme [10] possesses the algorithms (ABESsetup, ABEEext, ABEEnc, ABEDec). The key generation center ($KGC$) executes $ABESsetup(1^n)$ to output public parameters $par$ and a master secret key $msk$. A user $U$ with a set of attributes $\mathcal{A}$ queries $KGC$, which runs $ABEEext(msk, \mathcal{A})$ and returns a secret key $spk$. On input a message $m$ and an access structure $P$, $ABEEnc(par, m, P)$ computes a ciphertext $ct$ that can only be decrypted by a user $U$ that possesses a set of attributes $\mathcal{A}$ that satisfies $P$ ($ct$ implicitly contains $P$). Algorithm $ABEDec(par, ct, spk)$, on input a ciphertext $ct$ and a secret key $spk$, outputs the message $m$ if $\mathcal{A}$ satisfies $P$.

**Definition 2.1 (Secure CPABE).** Security for CPABE is defined through the following game between a challenger $C$ and an adversary $A$.

- **Setup.** $C$ runs $ABESsetup(1^n)$, keeps $msk$ and sends $par$ to $A$.
- **Phase 1.** $A$ requests secret keys for sets of attributes $\mathcal{A}_1, \ldots, \mathcal{A}_{q_1}$.
- **Challenge.** $A$ sends two equal length messages $m_0$ and $m_1$ and an access structure $P^*$ such that none of the sets $\mathcal{A}_1, \ldots, \mathcal{A}_{q_1}$ satisfy $P^*$. $C$ picks a random bit $b$, runs $ct^* = ABEEnc(par, m_b, P^*)$ and sends $ct^*$ to $A$.
- **Phase 2.** $A$ requests secret keys for sets of attributes $\mathcal{A}_{q_1+1}, \ldots, \mathcal{A}_q$ that do not satisfy $P^*$.
- **Guess.** $A$ outputs a guess $b'$ of $b$.

A CPABE scheme is secure if, for all p.p.t. adversaries $A$, $|Pr[b' = b] - \frac{1}{2}|$ is negligible.

In Figure 1, we recall the CPABE scheme proposed in [10]. Let $H : \{0, 1\}^* \rightarrow G$ be a hash function modelled as a random oracle.

### 2.4. Signature Schemes

A signature scheme consists of the algorithms KeyGen, Sign, and VfSig. Algorithm $KeyGen(1^n)$ outputs a secret key $ssk$ and a public key $spk$, which include a description of the message space $M$. $Sign(ssk, m)$ outputs a signature $s$ on message $m \in M$. $VfSig(spk, s, m)$ outputs 1 if $s$ is a valid signature on $m$ and 0 otherwise.
Definition 2.3 (Existential Unforgeability). Let \( O \) output a signature on a message without knowledge of \( \beta \). Existential unforgeability ensures that it is not feasible to:

- ppt adversary \( A \) contains the messages sent to \( O \) of a set \( M \).

KeyGen uses the pairing setup \( \mathbb{P} \), \( A \) with output parameters \( \mathbb{P} \) and \( \mathcal{A} \). Pick random \( r \in \mathbb{Z}_p \) and, for each \( j \in \mathcal{A} \), pick random \( r_j \in \mathbb{Z}_p \). Compute the key as \( sk_\mathcal{A} = (D_j = g^{r_j \cdot H(i)}/j^\alpha_j, D_j^* = g^{r_j})_{j \in \mathcal{A}} \).

VfSig accepts the signatures created by the \( \mathcal{A} \) and chooses other \( d_R \) points randomly to define \( q_\mathcal{A} \) completely. For any other node \( x \), set \( \gamma_x(0) = \gamma_{\text{parent}}(x)(\text{index}(x)) \) and choose other \( d_x \) points randomly to define \( q_x \) completely. Let \( Y \) be the set of leaf nodes in \( \mathcal{T} \). The ciphertext consists of \( c_t = (\mathcal{T}, C = m \cdot U^*, C = h^s, \{C_y = g^{\gamma_y(0)}, C_y^* = H(\text{index}(y))^{\gamma_y(0)}\} \in Y) \).

ABEDec defines a recursive algorithm ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, x \)). If \( x \) is a leaf node and \( i = \text{index}(x) \) belongs to the set of attributes \( \mathcal{A} \) associated with \( sk_\mathcal{A} \). ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, x \)) = \( e(D_i, C_x)/e(D'_i, C'_x) = e(g, g)^{q_x(0)} \). If \( i \notin \mathcal{A} \), then ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, x \)) = \( \bot \).

If \( x \) is not a leaf node, then proceed as follows. For all the children \( z \) of \( x \), call ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, z \)) and store the output as \( F_z \). Let \( S_z \) be an arbitrary \( k \)-sized set of child nodes \( z \) such that \( F_z \neq \bot \). If no such set exists ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, z \)) = \( \bot \). Otherwise, let \( i = \text{index}(z) \) and \( S'_z = \{\text{index}(z_i) \in S_z \} \), and, for \( i \in \mathbb{Z}_p \) and a set \( S \) of elements in \( \mathbb{Z}_p \), define the Lagrange coefficient \( \Delta_i(S/x) = \prod_{y \neq z \in S} x - y \). Compute \( F_x = \prod_{z \in S_x} F_z^{\Delta_i(S'_z/x)} = e(g, g)^{q_x(0)} \). ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A} \)) executes ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, R \)), where \( R \) is the root node. If the tree \( \mathcal{T} \) is satisfied by the set of attributes \( \mathcal{A} \), ABEDecNode(\( \mathcal{A}, ct, sk_\mathcal{A}, R \)) returns \( \hat{A} = e(g, g)^{r_A} \). Finally, output \( m = \hat{C}/(e(C, D)/A) \).

A signature scheme must fulfill the following correctness and existential unforgeability properties [18].

**Definition 2.2 (Correctness).** Correctness ensures that the algorithm VfSig accepts the signatures created by the algorithm KeyGen on input a secret key computed by an algorithm KeyGen. More formally, correctness is defined as follows.

\[
\Pr \left[ (ssk, spk) \xleftarrow{\$} \text{KeyGen}(1^k); \ m \xleftarrow{\$} \mathcal{M}; \ s \xleftarrow{\$} \text{Sign}(ssk, m); \ 1 = \text{VfSig}(spk, s, m) \right] = 1
\]

**Definition 2.3 (Existential Unforgeability).** The property of existential unforgeability ensures that it is not feasible to output a signature on a message without knowledge of the secret key or of another signature on that message. Let \( O_k \) be an oracle that, on input \( ssk \) and a message \( m \in \mathcal{M} \), outputs \( \text{Sign}(ssk, m) \), and let \( S_s \) be a set that contains the messages sent to \( O_k \). More formally, for any ppt adversary \( A \), existential unforgeability is defined as follows.

\[
\Pr \left[ (ssk, spk) \xleftarrow{\$} \text{KeyGen}(1^k); \ (m, s) \xleftarrow{\$} A(spk)^O(\mathcal{A}(ssk)); \ 1 = \text{VfSig}(spk, s, m) \wedge \ m \in \mathcal{M} \wedge \ m \notin S_s \right] \leq \epsilon(k)
\]


In this section, we describe the protocol proposed in [17]. The protocol is executed by the following entities.

- **Data Owner.** The data owner stores records of data in the cloud. For each record of data, the data owner specifies the users that have reading access and those that have writing access.

- **Manager.** The manager is a trusted server running inside the data owner’s network and acts as a key generation center for ciphertext-policy attribute-based encryption.

- **Cloud Server.** The cloud server stores records of data sent by the data owner. The data owner gives users reading and writing access to the records of data. The cloud server is assumed to be honest-but-curious.

- **Users.** A user can access records of data stored in the cloud server if the data owner has granted her reading access. Additionally, a user can modify a record of data stored in the cloud server if the data owner has granted her writing access.

The protocol consists of eight steps.

**Data Owner Setup.** The data owner executes the following steps:
Create a bilinear group setup \((p, \mathbb{G}, \mathbb{G}_t, e, g) \leftarrow \mathbb{G}(1^k)\), where \(1^k\) is the security parameter.

Create a data owner secret key \(osk \leftarrow \beta\), where \(\beta\) is picked randomly by doing \(\beta \leftarrow \mathbb{Z}_p\), and a data owner public key \(opk \leftarrow (g^\beta, g^{1/\beta})\).

Create a group secret key \(gsk \leftarrow u_0\), where \(u_0\) is picked randomly by doing \(u_0 \leftarrow \mathbb{Z}_p\), and a group public key \(gpk \leftarrow g^{u_0}\). The group secret key will be known by the group of users with writing access.

For each record of data to be outsourced to the cloud server, create a signing key pair \((ssk, spk) \leftarrow \text{KeyGen}(1^k)\).

**Manager Setup.** The manager receives \((p, \mathbb{G}, \mathbb{G}_t, e, g), opk\) and \(gpk\) as input. The manager executes the following steps:

- Pick random \(\alpha_1, \alpha_2 \leftarrow \mathbb{Z}_p\), set \(\alpha \leftarrow \alpha_1 + \alpha_2\), and create a manager secret key \(mgsk \leftarrow (\alpha_1, \alpha_2, \alpha, g^\alpha)\).
- Parse \(opk\) as \((g^\beta, g^{1/\beta})\) and set a cloud key \(ck \leftarrow (g^{\alpha_2/\beta}, V_{no})\), where \(V_{no}\) is the version number.
- Compute \(e(g, g)^\alpha\), set \(h \leftarrow g^\beta\) and \(f \leftarrow g^{1/\beta}\) and create a public key \(pk \leftarrow (\mathbb{G}, g, h, f, e(g, g)^\alpha, gpk, V_{no})\).

**Key Distribution.** The data owner sends the group secret key \(gsk\) to the group of users with writing access by using a secure communication channel. The data owner sends the signing verification keys \(spk\) to the cloud server, to the users with reading access and to the group of users with writing access. The data owner sends the signing secret key \(ssk\) to the group of users with writing access by using a secure communication channel.

**Key Generation.** The manager receives as input the public key \(pk\), the manager secret key \(mgsk\), and a set of user attributes \(A\). The manager executes the following steps:

- Pick random \(r \in \mathbb{Z}_p\). For each \(j \in A\), pick random \(r_j \in \mathbb{Z}_p\).
- Compute the key as \(sk_k = (D = g^{(\alpha_1+r)/\beta}, \{D_j = g^r \cdot H(j)^{r_j}, D_j' = g^{r_j}\}_{j \in A})\).

As can be seen, the key generation corresponds to executing the algorithm ABEExt of the CPABE scheme in [10], where \(\alpha_1\) is employed as secret key.

**Encrypt.** The data owner receives as input a public key \(pk\), a group public key \(gpk\), a record of data \(m\) and an access control policy \(P\), which is defined by an access tree \(T\). Let \(H : \{0, 1\}^* \to \mathbb{G}\) be a hash function modelled as a random oracle. The data owner executes the following steps:

- For each node \(x\) in the tree \(T\), choose a polynomial \(q_x\) of degree \(k_x - 1\), where \(k_x\) is the threshold value of \(x\). Polynomials \(q_x\) are chosen in a top-down manner, starting from the root node \(R\). Pick random \(s \leftarrow \mathbb{Z}_p\) and set \(q_R(0) = s\), and choose other \(d_R\) points randomly to define \(q_R\) completely. For any other node \(x\), set \(q_x(0) = q_{\text{parent}(x)}(\text{index}(x))\) and choose other \(d_x\) points randomly to define \(q_x\) completely.
- Set a time stamp \(\tau\) and a version number \(v \leftarrow 0\).
- Compute \(C \leftarrow m \cdot e(g, g)^{\alpha x}, C_0 \leftarrow g^{v \alpha x},\) and \(C'' \leftarrow g^{\beta}\).
- Let \(Y\) be the set of leaf nodes in \(T\). For all \(y \in Y\), compute \(C_y \leftarrow g^{\beta s_y}(0)\) and \(C'_y \leftarrow h(\text{att}(y))^{\beta s_y}(0)\).
- Set the ciphertext \(ct \leftarrow (v, \tau, T, C, C_0, C'', \{C_y, C'_y\}_{y \in Y})\).

The manager shares \(\alpha\) with the group members by encrypting it using \(gpk\). The cloud server computes \(e(g, g)^{\nu s_k}\) by using \(C_0\). The final ciphertext generated by the data owner is \(ct \leftarrow (v, \tau, T', C' \leftarrow m \cdot e(g, g)^{\alpha y}, e(g, g)^{\alpha s_k}, C, C_0, C'', \{C_y, C'_y\}_{y \in Y})\). Finally, the data owner signs the ciphertext \(ct\) by running the algorithm Sign on input the signing secret key \(ssk\) and \(ct\).

**Upload Ciphertext to Cloud.** The data owner or a user with writing access sends the ciphertext \(ct\) to the cloud server. The cloud server executes the following steps:

- Verify \(\tau\).
- Verify the signature on the ciphertext \(ct\) by running \(\text{VfSig}\) on input the signing verification key \(spk\), the signature and \(ct\).
- If \(ct\) is uploaded by a user with writing access, for all \(y \in Y\), check whether \(C_y\) and \(C'_y\) in the stored ciphertext are equal to \(C_y\) and \(C'_y\) in the uploaded ciphertext. This check ensures that the access tree \(T\) is the same in the stored and in the uploaded ciphertext. If this condition is satisfied, the uploaded ciphertext is accepted and stored.

**Decrypt.** A user who wishes to get access to a ciphertext \(ct\) sends the signing verification key \(spk\) to the cloud server. The cloud server compares \(spk\) with the signing verification key stored. If they are equal, then the cloud server sends the ciphertext \(ct\) and the cloud key \(ck\) to the user. The user executes the following steps:

- Verify the signature on the ciphertext \(ct\) by running \(\text{VfSig}\) on input the signing verification key \(spk\), the signature and \(ct\).
- Parse \(ct\) as \((v, \tau, T, C', C_0, C'', \{C_y, C'_y\}_{y \in Y})\) and set \(ct'\) as \((T, C, C'', \{C_y, C'_y\}_{y \in Y})\).
- Run the algorithm ABEDEcnNode described in Figure 1 on input the ciphertext \(ct'\), the secret key \(sk_k\) for the user’s attributes, and the root node \(R\). The result is \(A = e(g, g)^{\nu s_k}\).
- Parse \(sk_k\) as \((D, \{D_j, D'_j\}_{j \in A})\) and \(ck\) as \((g^{\alpha_2/\beta}, V_{no})\).
- If the user has reading access (i.e., the user does not know the signing secret key \(ssk\)), the user decrypts the ciphertext by computing \(m \leftarrow C \cdot A \cdot e(C'', D) \cdot e(C', g^{\alpha_2/\beta})\). The manager computes \(e(C'', D)\), while the user computes \(e(C', g^{\alpha_2/\beta})\) and \(e(C'', D) \cdot e(C', g^{\alpha_2/\beta})\).
- If the user has writing access, the user decrypts the ciphertext by computing \(m \leftarrow C' \cdot A \cdot e(C'', D) \cdot e(C', g^{\alpha_2/\beta})\), where \(A = e(g, g)^{\nu s_k}\).
A user with writing access and the cloud server execute the following steps to compute an updated ciphertext for a record of data \( m' \).

- The user computes \( e(C'', g^{1/\beta}) = e(g, g)^s \), where \( C'' \) is in \( ct \) and \( g^{1/\beta} \) is in the public key.
- The user computes \( (e(g, g)^s)^s = e(g, g)^{\alpha s} \).
- The cloud server computes \( e(C_0, g) = e(g, g)^{u_0} s \).
- The user computes \( C'' \leftarrow m \cdot e(g, g)^{u_0} s \cdot e(g, g)^{\alpha s} \) and \( C \leftarrow m \cdot e(g, g)^{\alpha s} \).
- The user selects a time stamp \( \tau \) and increments the version number \( v \leftarrow v + 1 \).
- The user replaces \( \tau, v, C'' \) and \( C \) in the ciphertext \( ct \) by the new values.
- The user signs the new ciphertext \( ct \) by running the algorithm Sign on input the signing secret key \( s\text{sk} \) and \( ct \).

**Key Regeneration.** When a member in the group is revoked, a new user is added or a key is compromised, the manager and the data owner execute the following steps.

- The manager picks random \( \alpha' \leftarrow \mathbb{Z}_p \) and chooses a new cloud key \( ck \leftarrow g^{\alpha'2/\beta} \), where \( \alpha' = \alpha' - \alpha_1 \), \( \alpha_1 \) is not modified to avoid the need of updating all the user secret keys \( s\text{sk} \).
- The manager updates the public key \( pk \leftarrow (G, g, h, f, e(g, g)^{\alpha'}, gpk, V_{no}) \), the cloud key \( ck \leftarrow (g^{\alpha'2/\beta}, V_{no}) \), and the manager secret key \( n\text{sk} \leftarrow (\alpha_1, \alpha_2, \alpha', g^{\alpha''}) \), where \( V_{no} \) is incremented by 1.
- The data owner updates the group secret key \( gsk \leftarrow u_Z \), where \( u_Z \leftarrow \mathbb{Z}_p \), and the group public key \( gpk \leftarrow g^{\alpha'2} \). \( gsk \) is sent to the group members by using a secure channel.
- The manager encrypts \( \alpha' \) by using \( gpk \) and sends the encrypted \( \alpha' \) to the group members.

For each encrypted record of data, the data owner and the cloud server execute the following steps.

- For each encrypted record of data, the data owner computes \( C \leftarrow m \cdot e(g, g)^{\alpha''} s \) and \( C_x \leftarrow g^{\alpha''} s \) and sends \( C \) and \( C_x \) to the cloud.
- The data owner and the holder of \( gsk \) compute a re-encryption key \( g^{\alpha''} \) and send it to the cloud.
- The cloud computes \( C_x \leftarrow C e(C_0, g^{\alpha''} u_0) = m \cdot e(g, g)^{\alpha''} s e(g, g)^{u_0} s \).
- The data owner sets the re-encrypted ciphertext as \( ct \leftarrow (v \leftarrow x, \tau, T_c, C, C_x, C'', \{ C_y, C'_y \}_{y \in Y}) \).
- The data owner computes a new key pair \( (s\text{sk}, spk) \leftarrow \text{KeyGen}(1^k) \) whenever there is a change in group membership.

### 4. Analysis of the Method in [17]

In this section, we discuss the method in [17] and describe its security vulnerabilities.

**The cloud server can get information about encrypted messages.** A ciphertext for a message \( m \) is of the form \( ct \leftarrow (v, \tau, T_c, C, C_0, C', \{ C_y, C'_y \}_{y \in Y}) \). When a user with writing access updates the ciphertext to encrypt a new message \( m' \), the user updates the values \( C' \) and \( C \) to new values \( C'_{up} \) and \( C_{up} \). As can be seen, the cloud server can compute \( m'/m = C'_{up}/C' \) and \( m'/m = C_{up}/C \). This violates the security properties of a CPABE scheme. For instance, if \( 1 = m'/m \), the cloud server learns that the writer did not modify the stored encrypted message.

The reason of this vulnerability is that the stored ciphertext and the updated ciphertext use the same randomness. To securely update the encrypted message, it is necessary to compute a new ciphertext with fresh randomness.

We believe the reason why the scheme under analysis does not compute new ciphertexts with fresh randomness is to prevent a user with writing access from changing the access control policy of a ciphertext. When a ciphertext is uploaded to the cloud by a user with writing access, the cloud server checks that the elements \( \{ C_y, C'_y \}_{y \in Y} \) of the updated ciphertext are equal to those of the stored ciphertext. If the new ciphertext employs fresh randomness, comparing the elements \( \{ C_y, C'_y \}_{y \in Y} \) is not possible.

However, we note that the ciphertexts in the CPABE scheme proposed in [10], unlike those in other schemes [13], reveal the access control policy. Therefore, the cloud server is able to check that the updated ciphertext and the stored ciphertext have the same access control policy even if the updated ciphertext is a completely new ciphertext with fresh randomness.

**A user with writing access can decrypt all the ciphertexts regardless of her attributes.** A user with writing access has knowledge of \( \alpha \). Given a ciphertext \( ct \leftarrow (v, \tau, T_c, C, C_0, C'', \{ C_y, C'_y \}_{y \in Y}) \), a user with writing access can decrypt the ciphertext by using \( m \leftarrow C/e(C''\alpha, f) \), where \( f \) is in the public key \( pk \). As can be seen, the user can obtain the message by using only \( \alpha \), i.e., without using the secret key \( s\text{sk} \) related to her attributes.

The reason of this vulnerability is that \( \alpha \) is the master secret key of the CPABE scheme. With knowledge of \( \alpha \), it is possible to decrypt any ciphertext.

We note that the reason why users with writing access are given \( \alpha \) is to update ciphertexts. Concretely, they employ \( \alpha \) to compute \( e(g, g)^{\alpha} = e(g, g)^{\alpha s} \). However, as explained above, for the scheme to be secure, new ciphertexts with fresh randomness need to be computed, and the users with writing access do not need to know \( \alpha \) to compute new ciphertexts.

**The scheme provides both user-based and attribute-based access control.** We also note that a user needs to present the signing verification key related to a ciphertext.
in order to be able to download it. Therefore, even if a user with writing access can decrypt all the ciphertexts because she knows $\alpha$, she can only download those for which she possesses the corresponding signing verification key.

Here we observe that the scheme under analysis provides simultaneously user-based access control and attribute-based access control. As can be seen, the data owner encrypts a message under an access control policy that determines the attributes that a user must possess in order to be able to obtain the message. Additionally, for each record of data, the data owner computes a signing key pair and sends the public key to all the users with reading access, whereas the secret key is sent to the users that have writing access.

Implementing simultaneously user-based and attribute-based access control is inefficient and counterproductive. Attribute-based access control is ideal for a cloud environment where the number of users is large and the data owner does not know the identities of the users who should be given access to his records of data. Instead, the data owner describes the attributes of the user that must possess. In contrast, user-based access control is ideal for smaller environments where the data owner knows the identities of all the users that should have access to the records of data. User-based access control is less flexible than attribute-based access control and is only applicable in smaller settings, but it allows the use of encryption schemes that are more efficient than CPABE, such as identity-based broadcast encryption [19]. Given that the scheme under analysis requires the data owner to know the identities of the users that must have reading or writing access to each of the records of data, using CPABE to encrypt the messages does not make sense.

The users with writing access can decrypt messages without knowledge of the group secret key. A ciphertext is of the form $ct \leftarrow (v, \tau, T, C', C, C_0, C'_0, \{C_y, C'_y\}_{y \in Y})$. The method in [17] explains that $C \leftarrow m \cdot e(g, g)^{u_0}$ and $C' \leftarrow m \cdot e(g, g)^{u_0} e(g, g)^{s_0}$ are the encrypted messages for the readers and for the writers respectively.

In the decryption method described for the writers, the writers do $m \leftarrow C'' \cdot A / e(C'', D) e(C'', g^{s_0}) e(C'', g^{u_0})$. To do that computation, the writers need to know the group secret key $u_0$ in order to compute $g^{u_0}$. However, the writers can also decrypt the ciphertext without knowledge of $u_0$. First, we point out that, in principle, nothing prevents a writer from decrypting $C$ instead of $C'$. To decrypt $C$, a writer can use the same computations as a reader. We note that both readers and writers possess the signing verification key $spk$ required to download a ciphertext from the cloud server, and thus at this stage the cloud server does not distinguish whether the user is a reader or a writer.

Furthermore, even if we assume that the cloud server distinguishes a reader from a writer and gives a writer only the value $C'$, the writer can still decrypt without knowledge of $u_0$. As can be seen, the writer can use the element $C_0$ to compute $m \leftarrow C'' \cdot A / e(C'', D) e(g, C_0) e(C'', g^{s_2})$. In fact, the element $C_0$ in the ciphertext is unnecessary since it is not used at all in the decryption process, and as we have shown it allows writers to decrypt $C'$ without knowledge of $u_0$.

5. Discussion

In Section 4, we have described several security vulnerabilities in the method proposed in [17]. In [17], a security analysis is provided and some claims on the security of the scheme are made. We note that, in [17], formal security definitions that state accurately the security properties of the scheme are not provided. Consequently, the security claims made in [17] are not formally proven. In this section, we discuss the security claims made in [17].

The cloud server and the manager cannot decrypt ciphertexts. In [17], it is claimed that the cloud server cannot decrypt ciphertexts because the cloud server cannot obtain secret keys $sk_h$, whereas the manager cannot decrypt ciphertexts because the manager is not given access to them. However, as described in Section 4, the cloud server, given a ciphertext that encrypts $m$ and an updated ciphertext that encrypts $m'$, can compute $m' / m$, which violates the security properties of CPABE defined in Section 2.3. With respect to the manager, we note that the manager performs the tasks of a key generation center in a CPABE scheme, and key generation centers in CPABE are trusted.

Collusion resistance. In [17], it is claimed that the method benefits from the collusion resistance property of the CPABE scheme in [10], and thereby a group of users cannot decrypt a ciphertext if one user in the group is not able to do so on her own. However, as described in Section 4, users with writing access are given $\alpha$ and therefore they can use $\alpha$ to decrypt every ciphertext regardless of their attributes.

Preventative measure against compromise of $gsk$ or $\alpha$. In [17], update mechanisms for $gsk$ and $\alpha$ are proposed for the case that $gsk$ or $\alpha$ are compromised. As described in Section 4, for the scheme to be secure, an updated ciphertext must be a new ciphertext for fresh randomness. Therefore, $\alpha$ can be kept secret by the manager, thereby reducing greatly the risk of compromise. For $gsk$, we have shown that a user with writing access can decrypt ciphertexts without knowledge of $gsk$.

The cloud server, the manager and the readers cannot perform the write operation. In [17], it is claimed that the cloud server, the manager and the readers cannot perform a write operation because they do not know the secret signing key $ssk$. It is also noted that, if a user attempts to upload a ciphertext that she previously downloaded, the cloud server would reject it because the time stamp is invalid. The method in [17] requires the data owner to create a signing key pair for each record of data and to distribute the signing secret key to the users with writing access via a secure channel. The concrete properties of the secure
channel are not described. We note that both authentication and confidentiality are essential to prevent impersonation attacks and disclosure of the secret signing key. On the other hand, we note that computing and distributing new signing keys for each record of data is very inefficient.

As can be seen, most of the security claims made in [17] can be disproved. We think that defining security formally, i.e., defining precisely the security properties of the scheme and the capabilities of the adversary, is key to conduct a formal security analysis. The lack of security definitions makes it unclear what the security properties of the scheme are and leads to unproven security claims that, as in [17], do not hold.

Additionally, we think that fixing the method in [17] is unnecessary. The two main advantages claimed in [17] over previous work are, first, that their method prevents a writer from modifying the access control policy of a ciphertext, and second, that some computations can be outsourced. Nevertheless, as we have explained, the ciphertexts in the CPABE scheme in [10] reveal the access control policy. Therefore, in other methods that provide reading and writing access to the cloud based on the CPABE scheme in [10], such as [14], the cloud server can check that the access control policy has not been modified. As for outsourcing some computations, we note that the method in [17] is computationally more expensive than the original CPABE scheme in [10]. Moreover, the computations that can be outsourced in [17] are mainly the additional ones introduced in [17].

6. Conclusion

Recently, Jahan et al. [17] have proposed a scheme to outsource data securely to the cloud based on ciphertext-policy attribute-based encryption. We have analyzed the scheme proposed in [17] and we have shown that it has several security vulnerabilities. We have also discussed why the security claims made in [17] do not hold. Particularly, we have highlighted the need of providing formal security definitions and a formal security analysis. Furthermore, we have explained that existing schemes can already offer the advantages claimed by [17].

References


