Computing and Informatics, Vol. 40, 2021, 1136–1159, doi: 10.31577/cai_2021_5_1136

CIPHERTEXT-POLICY ATTRIBUTE BASED ENCRYPTION WITH SELECTIVELY-HIDDEN ACCESS POLICY

Gulmire Arkin

College of Mathematics and System Science
Xinjiang University, China
&
College of Science
Xinjiang Institute of Technology, China
e-mail: 710544651@qq.com

Nurmamat Helil

College of Mathematics and System Science
Xinjiang University, China
e-mail: nur924@sina.com

Abstract. In conventional Ciphertext-Policy Attribute-Based Encryption (CP-ABE), the access policy appears in plaintext form that might reveal confidential user information and violate user privacy. CP-ABE with hidden access policies hides all attributes, but the computational burden increases due to the attribute hiding. In this paper, we present a Linear Secret Sharing Scheme (LSSS) access structure CP-ABE scheme that hides only sensitive attributes, rather than all attributes, in the access policy. We also provide an attribute selection method to choose these sensitive attributes and use an Attribute Bloom Filter (ABF) to hide them. Compared with the existing major CP-ABE schemes with hidden access policies, our proposed scheme is flexible in selecting attributes to hide. This scheme enhances the efficiency of policy hiding while still protecting policy privacy. Test results show that our approach is reasonable and feasible.

Keywords: CP-ABE, sensitive attribute selection, partial attribute hidden

1 INTRODUCTION

Cloud storage technology is quite effective in handling massive data volumes in the big data era. However, security issues such as leakage of user's data and access policy privacy of data need to be properly handled. Attribute-Based Encryption (ABE) has provided flexible and fine-grained access control for outsourced data stored in the cloud [1]. Goyal et al. propose Key-Policy Attribute-Based Encryption (KP-ABE) [2], and Bethencourt et al. introduce Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [3] as variants of ABE. User attributes are essential for both KP-ABE and CP-ABE. To avoid exposing user details from the access policy itself, Nishide et al. [4] propose CP-ABE with hidden policy. Lai et al. [5] establish a completely safe hidden policy CP-ABE building on Nishide's effort. Recently, Yang et al. [6] and Han et al. [7] proposed CP-ABE schemes using Attribute Bloom Filters (ABF). Their schemes incorporate a Linear Secret Sharing Scheme (LSSS) access structure [6, 7, 8, 9, 10]. These schemes have good policy expressivity, high efficiency, and achieve desired security goals under the decisional q-BDHE assumption.

The policy-hiding approaches of the CP-ABE schemes proposed so far fall into two categories. One hides the entire access policy, and the other hides all attributes in the access policy. Hence, the question arises: to reduce the computational cost, can we design a CP-ABE scheme in which we only obfuscate some of the attributes in the access policy rather than all of them? This idea is totally different from the partially hidden access policy of Cui et al. [8], which hides attribute values while leaving attribute names publicly visible.

In general, only a few attributes in the access policy violate user privacy. Therefore, only sensitive attributes that may leak private details should be hidden. Non-sensitive attributes that are not relevant to user privacy do not require protection, and not hiding them lowers computational costs. For example, consider a case in which an electronic medical record is stored on a cloud storage server with the hospital's access policy as:

> (*Alice* **AND** *patient* **AND** *department of cardiology* **AND** *municipal hospital*) **OR** (*tertiary hospitals* **AND** *cardiologist*).

This access policy allows *Alice*, a patient in the department of cardiology in the municipal hospital, or cardiologists from any tertiary hospital to access this record. We observe from the policy that the attributes *department of cardiology* and *cardiologist* reveal that *Alice* is suffering from a heart condition. Other attributes in the access policy provide no privacy information about the patient. Thus, we only need to obfuscate the two sensitive attributes, *department of cardiology* and *cardiologist*, leaving the others in plaintext form in the access policy.

To achieve a better balance between privacy and efficiency, we propose a CP-ABE scheme that hides sensitive attributes. Our contributions are as follows:

1. We use a sensitive attribute selection method to partition attributes in the access policy into two subsets to determine which attributes are sensitive and require obfuscation and which are non-sensitive and do not.
2. We use modified ABF to obfuscate sensitive attributes to prevent revealing privacy or critical information.
3. We determine whether the user can decrypt a given ciphertext in advance with obfuscated sensitive attributes.

We organize our paper as follows. We give the formal definition of a sensitive attribute and present the sensitive attribute selection algorithm in detail in Section 2. In Section 3, we describe the sensitive attribute hiding process through the modified ABF Build and the sensitive attribute verification process through ABF Query. Section 4 provides the system model, assumptions, and security model. The formal construction of our scheme is proposed in Section 5. We give security and efficiency analysis in Section 6. Section 7 shows the experimental results and evaluation of our approach. Section 8 discusses some related works. Finally, we summarize our work in Section 9.

2 SENSITIVE ATTRIBUTE SELECTION

An access policy in plaintext form easily reveals personal information. So, CP-ABE schemes with hidden-policies are proposed to solve the problem. In practice, not all attributes reveal sensitive information. To make the hidden access policy CP-ABE more flexible and efficient while preserving privacy, we selectively hide the sensitive attributes in the access policy. We formally define a sensitive attribute as follows.

Definition 1 (Sensitive attribute)**.** Attributes $X_1, X_2, \ldots, X_k$ in an access policy are sensitive attributes if, for a given threshold δ

$$I(Y; X_1 X_2 \ldots X_k) \geq \delta H(Y) \tag{1}$$

where $I(Y; X_1 X_2 \ldots X_k)$ denotes mutual information between Y and $X_1 X_2 \ldots X_k$, and $H(Y)$ denotes entropy of Y [11, 12], Y denotes a user's privacy, and $\{X_1, X_2, \ldots, X_k\}$ is the smallest set that satisfies the above conditions.

The user's privacy Y in the above definition is an abstract concept. We illustrate Y using the example in Section 1. In that example, we consider a privacy Y that if the patient has heart disease. One can deduce that the patient has heart disease from the attributes *department of cardiology* or *cardiologist* in the access policy. From the perspective of information theory, attributes *department of cardiology* and *cardiologist* provide more information about Y than other attributes. Therefore, attributes *department of cardiology* and *cardiologist* are regarded as sensitive attributes.

2.1 Sensitive Attribute Selection Algorithm

We select and hide sensitive attributes in the access policy to achieve both security and efficiency simultaneously. We base our sensitive attribute selection algorithm on information theory. We determine an attribute set that reveals the greatest amount of private information by measuring the mutual information between the user's privacy and the attributes.

Algorithm 1 describes the sensitive attribute selection process and consists of three steps. In step 1, we select attributes that independently provide more information about Y than the given threshold by calculating the mutual information between the user's privacy and each attribute. In step 2, we choose seed attributes that have closer relations with Y from the remaining attributes. In step 3, we further select attributes that collectively provide more information about Y than the threshold based on step 2.

We note that if the privacy Y can be independently derived from different attribute sets S_1, S_2, $\dots$, S_l, then the returned set of Algorithm 1 is $S = \bigcup_{k=1}^{l} S_k$.

It is possible that multiple privacies Y_1, Y_2, $\dots$, Y_t can be derived from the attributes in a given access policy. In that case, we run the algorithm for each Y_i $(i = 1, 2, \dots, t)$, and obtain corresponding sensitive attribute sets S_1, S_2, $\dots$, S_t. The final sensitive attribute set for the access policy is defined as $S = \bigcup_{i=1}^{t} S_i$.

For a given access structure of a data object, the sensitive attribute set can be obtained before the data owner releases the data object's ciphertext on the cloud.

3 HIDING SENSITIVE ATTRIBUTES AND VERIFICATION

3.1 ABF Build

In the traditional CP-ABE with LSSS access structure, the attribute mapping function ρ is exposed. Since ρ directly reflects the mapping relations between the rows and the attributes in the access matrix, as shown in Figure 1, ρ is the "arch-criminal" of privacy leakage. To protect privacy in the access policy, we remove the attribute mapping relations for sensitive attributes from the access structure and leave the attribute mapping relations for non-sensitive attributes. In this way, we can blur the position of sensitive attributes in the access policy to hide sensitive attributes. Figure 2 shows the hidden and visible parts of the LSSS access structure. We implement the ABF Build here.

The elements of the ABF are specific λ-bit strings connected by two fixed-length strings: the row number i with γ bits, and the attribute att_x with β bits, where $\gamma + \beta = \lambda$.

Before adding the attribute att_o to the ABF, we expand both row number i and att_o to the maximum bit lengths γ and β, respectively, by left-filling the bit strings with zeros. The algorithm takes the λ-bit strings x as input and hashes each att_o by t independent hash functions and finds the position index $h_1(att_o)$, $h_2(att_o)$,

Algorithm 1 Sensitive Attribute Selection Algorithm

```
Require: $A$: all attributes in an access policy; $Y$: the user's privacy; $\delta H(Y), 0 < \delta \leq 1$
Ensure: sensitive attribute set $S$
$B = \varnothing$, $S = \varnothing$, $S_t = \varnothing$;
Step 1:
for all $X_j \in A$ do
    calculate $I(Y; X_j)$;
    if $I(Y; X_j) \geq \delta H(Y)$ then
        $S = S \cup \{X_j\}$; $A = A \setminus \{X_j\}$;
    end if
end for
Step 2:
$B = A$;
for all $X_j \in B$ do
    select $X_j^*$ that $I(Y; X_j^*) = \max\{I(Y; X_j)\}$;
    $S_t = S_t \cup \{X_j^*\}$; $B = B \setminus \{X_j^*\}$;
    if $B == \varnothing$ then
        return $S$;
    end if
end for
Step 3:
for all $X_j \in B$ do
    select $X_j^{**}$ that
            $I(Y; X_j^{**} X_{i_1} X_{i_2} \dots X_{i_{|S_t|}}) \geq I(Y; X_j X_{i_1} X_{i_2} \dots X_{i_{|S_t|}})$;
    if $I(Y; X_j^{**} X_{i_1} X_{i_2} \dots X_{i_{|S_t|}}) \geq \delta H(Y)$ then
        $S_t = S_t \cup \{X_j^{**}\}$; $S = S \cup S_t$; $A = A \setminus S_t$; $S_t = \varnothing$;
        if $B == \varnothing$ then
            return $S$;
        else
            goto Step 2;
        end if
    else $S_t = S_t \cup \{X_j^{**}\}$; $B = B \setminus \{X_j^{**}\}$;
        if $B \neq \varnothing$ then
            goto Step 3;
        else
            return $S$;
        end if
    end if
end for
return $S$;
```

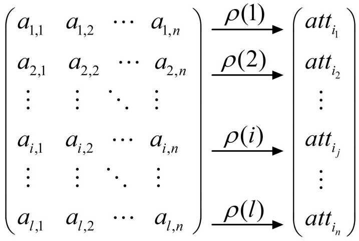

$a_{1,1}$ $a_{1,2}$ $\cdots$ $a_{1,n}$
$a_{2,1}$ $a_{2,2}$ $\cdots$ $a_{2,n}$
$a_{i,1}$ $a_{i,2}$ $\cdots$ $a_{i,n}$
$a_{l,1}$ $a_{l,2}$ $\cdots$ $a_{l,n}$
$\rho(1)$
$\rho(2)$
$\rho(i)$
$\rho(l)$
att_{i_1}
att_{i_2}
att_{i_j}
att_{i_n}

Figure 1. Traditional LSSS access structure

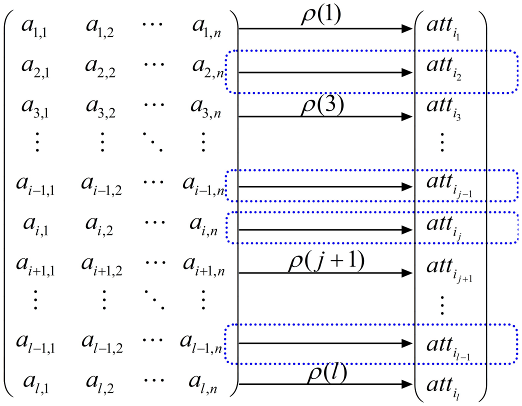

$a_{1,1}$ $a_{1,2}$ $\cdots$ $a_{1,n}$
$a_{2,1}$ $a_{2,2}$ $\cdots$ $a_{2,n}$
$a_{3,1}$ $a_{3,2}$ $\cdots$ $a_{3,n}$
$a_{i-1,1}$ $a_{i-1,2}$ $\cdots$ $a_{i-1,n}$
$a_{i,1}$ $a_{i,2}$ $\cdots$ $a_{i,n}$
$a_{i+1,1}$ $a_{i+1,2}$ $\cdots$ $a_{i+1,n}$
$a_{l-1,1}$ $a_{l-1,2}$ $\cdots$ $a_{l-1,n}$
$a_{l,1}$ $a_{l,2}$ $\cdots$ $a_{l,n}$
$\rho(1)$
$\rho(3)$
$\rho(j+1)$
$\rho(l)$
att_{i_1}
att_{i_2}
att_{i_3}
$att_{i_{j-1}}$
att_{i_j}
$att_{i_{j+1}}$
$att_{i_{l-1}}$
att_{i_l}

Figure 2. LSSS access structure after removing the mapping relations for sensitive attributes (mapping relations for attributes in the blue dotted rectangle are marked for removal)

$\dots$, $h_t(att_o)$. Then, it randomly selects $t-1$ x-bit strings r_1, r_2, $\dots$, r_{t-1}, where $r_t = r_1 \oplus r_2 \oplus \cdots \oplus r_{t-1} \oplus x$. Finally, it saves r_i by $h_i(att_o)$ in the ABF.

In our scheme, we use the ABF only for sensitive attributes. Non-sensitive attributes are exposed in LSSS access structure as usual, as shown in Figure 3.

This algorithm consists of the following steps.

Step 1: Obtain the set of sensitive attributes $S = \{att_{i_1}, att_{i_2}, \dots, att_{i_k}\}$ in the access structure.

Step 2: Obtain $att_o \in S$, define element $x = (i \parallel att_o)$ in the ABF, where i represents the γ-bit row number and att_o is the β-bit attribute string.

Step 3: Hash attribute att_o by t independent hash functions and obtain the position index $h_1(att_o)$, $h_2(att_o)$, $\dots$, $h_t(att_o)$.

Step 4: Randomly choose the $(\gamma+\beta)$-bit string r_1, r_2,$\dots$, r_{t-1} and make r_1, r_2, $\dots$, r_{t-1} satisfy $r_t = r_1 \oplus r_2 \oplus \cdots \oplus r_{t-1} \oplus x$, where r_1, r_2, $\dots$, r_t are t secret shares of the $x = (i \parallel att_o)$.

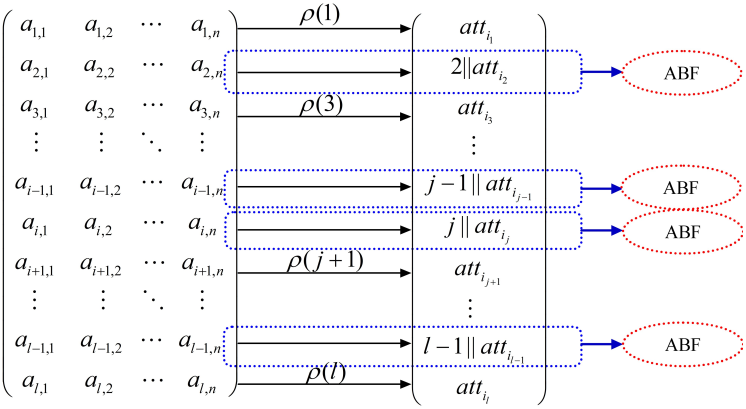

$a_{1,1}$ $a_{1,2}$ ⋯ $a_{1,n}$
$a_{2,1}$ $a_{2,2}$ ⋯ $a_{2,n}$
$a_{3,1}$ $a_{3,2}$ ⋯ $a_{3,n}$
$a_{i-1,1}$ $a_{i-1,2}$ ⋯ $a_{i-1,n}$
$a_{i,1}$ $a_{i,2}$ ⋯ $a_{i,n}$
$a_{i+1,1}$ $a_{i+1,2}$ ⋯ $a_{i+1,n}$
$a_{l-1,1}$ $a_{l-1,2}$ ⋯ $a_{l-1,n}$
$a_{l,1}$ $a_{l,2}$ ⋯ $a_{l,n}$
$\rho(1)$
$\rho(3)$
$\rho(j+1)$
$\rho(l)$
att_{i_1}
$2\|att_{i_2}$
att_{i_3}
$j-1\|att_{i_{j-1}}$
$j\|att_{i_j}$
$att_{i_{j+1}}$
$l-1\|att_{i_{l-1}}$
att_{i_l}
ABF
ABF
ABF
ABF

Figure 3. The LSSS access structure after using ABF to hide sensitive attributes (Blue dotted rectangles represent the mapping relations for sensitive attributes that are to be hidden by ABF)

Step 5: Store r_i by $h_i(att_o)$ in the ABF as

$$r_1 \longrightarrow h_1(att_o),$$

$$r_2 \longrightarrow h_2(att_o),$$

$$\cdots$$

$$r_t \longrightarrow h_t(att_o).$$

Step 6: Set $S = S \backslash \{att_o\}$. Repeat steps 2 through 5 until $S = \varnothing$.

3.2 ABF Query

Since the attribute mapping relations for non-sensitive attributes are unchanged in the scheme, it is easy for a user to determine if his non-sensitive attributes are in the access structure. However, removal of the attribute mapping relations for sensitive attributes necessitates an additional verification process to determine if the user's attributes are in the access structure as being sensitive attributes.

We implement an ABF query to check whether a user's attributes are in the access structure $(\mathbf{M}, \rho)$ by comparing the user's attribute att_u with the β-bit string of element $x = (i \parallel att_o)$ in the ABF output. If att_u is the same as the β-bit string of $x = (i \parallel att_o)$, the user's attribute att_u is in $(\mathbf{M}, \rho)$. Otherwise, this attribute is not in $(\mathbf{M}, \rho)$. To provide an example, we suppose att_o is the same as att_u and carry out the specific process by the following steps.

Step 1: Put user attribute att_u into the ABF.

Step 2: Obtain $h_1(att_u), h_2(att_u), \dots, h_t(att_u)$ by hashing att_u with the t independent hash functions used in the ABF.

Step 3: According to the position index $h_1(att_u), h_2(att_u), \dots, h_t(att_u)$ obtain t secret shares of the attribute att_u as follows:

$$\begin{aligned} h_1(att_u) &:= r_1(att_u), \\ h_2(att_u) &:= r_2(att_u), \\ &\cdots \\ h_t(att_u) &:= r_t(att_u). \end{aligned}$$

Step 4: According to the t secret shares of att_u, reconstruct the element $x = (i \parallel att_o) = r_1(att_u) \oplus r_2(att_u) \oplus \cdots \oplus r_t(att_u)$, which is formed after the user's attribute att_u goes through ABF.

Step 5: Compare the user attribute att_u with the β- bit string of $x = (i \parallel att_o)$ output by the ABF.

Step 6: If the user's attribute is in $(\mathbf{M}, \rho)$, it will obtain the remaining γ-bit string i from $x = (i \parallel att_o)$. The remaining γ-bit string i denotes the position of att_u in the access matrix. If the results in step 5 are different, att_u is not in $(\mathbf{M}, \rho)$, and the algorithm outputs a random string. Thus, the user cannot obtain any sensitive information from the access policy.

4 PRELIMINARIES

4.1 Liner Secret Sharing Scheme (LSSS)

We use the LSSS proposed by Cui et al. [8], so we omit its detailed description here.

4.2 Bilinear Pairings

G, G_T are two multiplicative groups with the same prime order p. A bilinear mapping $\hat{e} : G \times G \longrightarrow G_T$ has the following properties.

1. Bilinearity: $\forall a, b \in Z_p$ and $g \in G$ $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab}$;
2. Non-Degeneracy: $\hat{e}(g_1, g_2) \neq 1$;
3. Computability: $\hat{e}(u, v)$ is comptationally efficient for all u, v.

4.3 Decisional q-BDHE Assumption

The decisional q-bilinear Diffie-Hellman exponent (Decisional q-BDHE) problem is defined as follows.

Definition 2. Choose a group G of prime order p according to the security parameter κ. Choose $a, s \in \mathbb{Z}_p$ at random and let g be a generator of G, with g_i denoting g^{a_i}. Given $\overrightarrow{y} = (g, g_1, \dots, g_q, g_{q+2}, \dots, g_{2q}, g^s)$, the adversary must distinguish $\hat{e}(g,g)^{a^{q+1}s} \in G_T$ from the random element $R \in G_T$. An algorithm has advantage in solving the decisional q-BDHE problem in G if $|Pr[B(\overrightarrow{y}, T = \hat{e}(g,g)^{a^{q+1}s}) = 0] - Pr[B(\overrightarrow{y}, T = R) = 0]| \geq \varepsilon$.

Definition 3. We say that the decisional q-BDHE assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the q-BDHE problem.

4.4 CP-ABE Scheme with Partial Attributes Hidden

As shown in Figure 4, the whole system consists of four entities: the cloud storage server (CSS), the key generation center (KGC), the data owner, and the user. The roles are as follows.

- The CSS provides computing, storage, and other related services for the whole system. It is semi-trusted.
- The KGC generates public keys, the master key, and users' private keys for the system. It is fully trusted.
- The data owner establishes access policies for data objects and encrypts them under these access policies. The user also preprocesses these access policies by performing the sensitive attribute selection algorithm. The user partially hides access policies using ABF.
- The user accesses data objects in the CSS. However, only users whose attributes match the access policy can access the object.

Our scheme includes the following algorithms.

Sensitive Attribute Selection Algorithm: The data owner executes the sensitive attribute selection algorithm with inputs A and δ, where A is the set of all the attributes in the access policy and δ is the threshold value. It outputs the sensitive attribute set S.

Setup: The KGC takes the security parameter κ as input and produces the public key PK and master secret key MK as outputs.

Key Generation Algorithm: The KGC uses PK, MK, and a set of attributes S_u as inputs to the algorithm and generates SK_u for the user u as output.

Encryption Algorithm: The encryption algorithm has three phases.

1. *Offline encryption:* This phase takes PK, the message M as input and outputs IT, an intermediate ciphertext.
2. *Online encryption:* The online encryption phase takes PK, intermediate ciphertext IT and the access structure $(\mathbf{M}, \rho)$ as inputs and outputs a ciphertext CT.

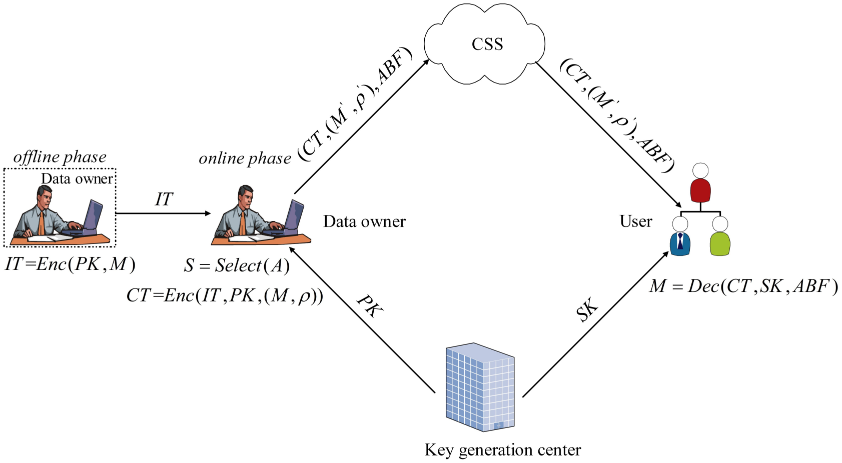


Figure 4. System framework

3. *ABF Build:* This phase receives the sensitive attribute set S as input and outputs the ABF.

Decryption Algorithm: The decryption algorithm consists of two phases.

1. *ABF Query:* This phase takes S_u and the ABF produced by the ABF Build algorithm above. It outputs the sensitive attributes and their positions in the access matrix.
2. *Decryption:* The decryption phase receives SK and CT as input and returns the message M as output if the user satisfies the access policy.

4.5 Game Between Adversary and Challenger

The game between adversary and challenger is based on selectively chosen plaintext attacks. The details are described in Figure 5.

If the adversary outputs b' and $b' = b$, the adversary wins the game.

In polynomial time, the adversary's advantage in this game is defined as $\varepsilon = |Pr[m = m_b] - \frac{1}{2}|$, where $b \in \{0, 1\}$ and $\varepsilon \in \mathbb{R}^+$.

5 THE CP-ABE SCHEME WITH PARTIAL ATTRIBUTES HIDDEN

5.1 Our construction

We introduce partial attribute hiding and employ the sensitive attribute selection algorithm to ensure the access policy does not violate user privacy. Our method runs the sensitive attribute selection algorithm to extract sensitive attributes in

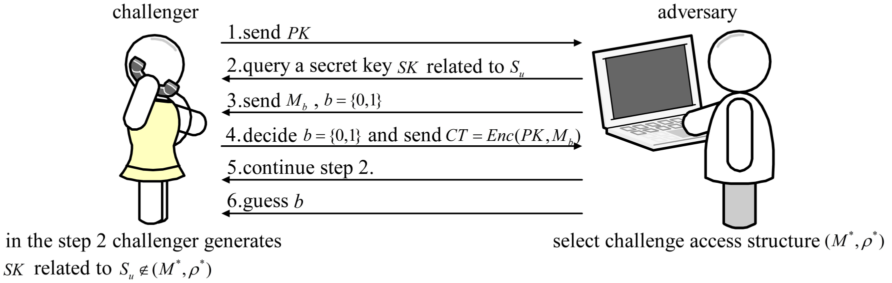

challenger
adversary
1.send PK
2.query a secret key SK related to S_u
3.send M_b , $b=\{0,1\}$
4.decide $b=\{0,1\}$ and send $CT=Enc(PK,M_b)$
5.continue step 2.
6.guess b
in the step 2 challenger generates
SK related to $S_u \notin (M^*,\rho^*)$
select challenge access structure (M^*,ρ^*)

Figure 5. The game between adversary and challenger

the access policy and hides them by establishing the ABF. In this section, we detail the four phases of our scheme: setup, key generation, encryption, and decryption.

Setup: In this phase, the KGC executes the setup algorithm using the security parameter κ and chooses two multiplicative cyclic groups G and G_T of prime order p. g is the generator of G, and $\hat{e} : G \times G \longrightarrow G_T$ is a bilinear map. The KGC randomly chooses $h, u, \upsilon, \omega \in G, \alpha \in \mathbb{Z}_p$, and outputs the public key $PK = (G, p, g, h, u, \upsilon, \omega, \hat{e}(g, g)^{\alpha})$ and master key $MK = (\alpha)$.

Key Generation: The algorithm takes the user's attribute set $S_u = \{att_1, att_2, \ldots, att_q\}$ as input. The KGC selects random exponents $a, a_1, a_2, \ldots, a_q \in \mathbb{Z}_p$, where a is a unique secret for each user and a_i, $i = 1, 2, \ldots, q$ is a unique secret for each attribute $att_i \in S_u$. The KGC then generates the private key

$$SK = (K_0 = g^{\alpha}\omega^{a}, K_1 = g^{a}, \{K_{i,2} = g^{a_i}, K_{i,3} = (u^{att_i}h)^{a_i}\upsilon^{-a}\}_{i=1,2,\ldots,q}).$$

Encryption Algorithm: The encryption algorithm is divided into the following three phases.

1. *Offline encryption:* We define N as the maximum row number of $(\mathbf{M}, \rho)$. This phase takes the public key PK as input. The data owner selects $s \in \mathbb{Z}_p$ and computes $\tilde{C} = M \cdot \hat{e}(g, g)^{\alpha s}$, $C_0 = g^s$. Then, the data owner randomly chooses $\lambda'_j, x_j, t_j \in \mathbb{Z}_p$ and computes $C_{j,1} = \omega^{\lambda'_j}\upsilon^{t_j}$, $C_{j,2} = (u^{x_j}h)^{-t_j}$, $C_{j,3} = g^{t_j}$, $j \in 1, 2, \ldots, N$, where λ'_j is a random share of s and x_j is a random attribute in the access structure. The algorithm outputs the intermediate ciphertext $IT = (\tilde{C}, s, C_0, \{\lambda'_j, x_j, t_j, C_{j,1}, C_{j,2}, C_{j,3}\}_{j\in 1,2,\ldots,N})$.
2. *Online encryption:* This algorithm receives the public key PK, an intermediate ciphertext IT, and the access structure $(\mathbf{M}, \rho)$ as inputs, where $\mathbf{M}$ is $l \times n$ matrix and $l < N$. It outputs the ciphertext CT. The algorithm randomly chooses the vector $\nu = (s, y_2, y_3, \ldots, y_n) \in \mathbb{Z}_p^n$, where s is a secret of being shared, and computes a vector of the share of s as $\lambda_i = \mathbf{M}_i \cdot \nu$, $i = 1, 2, \ldots, l$. For $j = 1, 2, \ldots, l$, the algorithm computes $C_{j,4} = \lambda_j - \lambda'_j$,$C_{j,5} = t_j(\rho(j) - x_j)$.

Finally, the algorithm outputs the ciphertext CT:

$$CT = ((\mathbf{M}, \rho), C_0, C_{j,1}, C_{j,2}, C_{j,3}, C_{j,4}, C_{j,5}).$$

3. *ABF Build:* The algorithm takes the sensitive attribute set S from the sensitive attribute selection algorithm as input, and stores each sensitive attribute in the ABF. Refer to Section 3 for the procedure for storing each sensitive attribute in ABF. After all the sensitive attributes are placed in the ABF, the data owner stores the above ABF with the partially displayed access structure $(\mathbf{M}', \rho')$ and the ciphertext CT in the cloud.

Decryption Algorithm: The decryption algorithm has two steps. The first step determines whether the user's attributes satisfy the access structure via the ABF Query if necessary. The second step performs the decryption.

Before the execution of the decryption algorithm, the user knows the non-sensitive attributes in the access structure. Although some of the attributes in the access structure are visible, some of the attributes in the access structure may be hidden in the ABF, which makes it difficult for a user to determine whether the ciphertext can be decrypted directly. The ABF Query assists with this determination.

1. *ABF Query:* This algorithm compares the β-bit strings of the element $x = (i \parallel att_o)$ formed by the user attribute through the ABF with the user's attribute att_u. If the β-bit string of $x = (i \parallel att_o)$ is the same as att_u, then att_u is in $(\mathbf{M}, \rho)$. Otherwise, att_u does not exist in the access policy. (Refer to Section 3 for details.) If the query succeeds, the position of user attribute att_u in the access matrix can be obtained. Otherwise, it outputs random strings; the user could not get any sensitive information in the access structure from the output.
2. *Decrypt:* Only the user whose attributes satisfy the access policy can execute the decryption algorithm. Assume that u is the user passed the ABF Query, the attributes of u satisfying $(\mathbf{M}', \rho')$, it can find coefficients $\sum_{i \in I} \omega_i \lambda_i = s$, where $\{\omega_i \in \mathbb{Z}_N\}_{i \in I}$. Then the user

$$\hat{e}(g,g)^{\alpha s} = \frac{\hat{e}(C_0, K_0)}{\hat{e}(\omega^{\sum_{i \in I} C_{j,4}\omega_i}, K_1)} \cdot \frac{1}{\Pi_{i \in I} \hat{e}(C_{j,1}, K_1)\hat{e}(C_{j,2} \cdot u^{c_{j,5}}, K_{i,2})\hat{e}(C_{j,3}, K_{i,3})^{\omega_i}} \tag{2}$$

and recovers the message as $M = \tilde{C}/\hat{e}(g,g)^{\alpha s}$. Otherwise, it outputs $\perp$.

5.2 Correctness

If the user attribute set S_u satisfies the access matrix, then the user obtains $\sum_{i\in I}\omega_i\lambda_i = s$. Therefore,

$$\begin{aligned}\hat{e}(g,g)^{\alpha s} &= \frac{\hat{e}(C_0,K_0)}{\hat{e}(\omega^{\sum_{i\in I}C_{j,4}\omega_i},K_1)}\cdot\frac{1}{\Pi_{i\in I}\hat{e}(C_{j,1},K_1)\hat{e}(C_{j,2}\cdot u^{c_{j,5}},K_{i,2})\hat{e}(C_{j,3},K_{i,3})^{\omega_i}}\\ &= \frac{\hat{e}(g,g)^{\alpha s}\hat{e}(g,g)^{as}}{\hat{e}(\omega^{\sum_{i\in I}(\lambda_j-\lambda'_j)\omega_i},g^a)}\\ &\quad\cdot\frac{1}{\Pi_{i\in I}\hat{e}(\omega^{\lambda'_j}\upsilon^{t_j},g^a)\hat{e}((u^{x_j}h)^{-t_j}\cdot u^{t_j(\rho(j)-x_j)},g^{a_i})\hat{e}(g^{t_j},(u^{att_i}h)^{a_i}\upsilon^{-a})^{\omega_i}}\\ &= \frac{\hat{e}(g,g)^{\alpha s}\hat{e}(g,g)^{as}}{\hat{e}(g,\omega)^{a\sum_{i\in I}\omega_i\lambda_i}}\\ &= \hat{e}(g,g)^{\alpha s}.\end{aligned}\tag{3}$$

5.3 A Case Study

In this section, we provide an example to illustrate the process of the entire scheme. The case study considers a hospital that stores electronic medical records in a CSS and specifies the following access policy:

> (*Alice* **AND** *Patient* **AND** *Department of cardiology* **AND** *Municipal hospital*) **OR** (*Tertiary hospitals* **AND** *Cardiologist*).

We define our attributes as

att_1: Tertiary hospitals,

att_2: Cardiologist,

att_3: Municipal hospital,

att_4: Alice,

att_5: Department of cardiology,

att_6: Patient.

We define the attribute map as ρ : $\rho(1) = att_2$, $\rho(2) = att_1$, $\rho(3) = att_5$, $\rho(4) = att_3$, $\rho(5) = att_4$, and $\rho(6) = att_6$. The access matrix corresponding to the above access policy is:

To protect sensitive information in an access policy, the data owner performs the sensitive attribute selection algorithm to select sensitive attributes that need to be hidden. We assume that the sensitive attributes chosen by the algorithm are att_2 and att_5.

Then, the KGC generates PK for the data owner. The data owner obtains PK and prepares to encrypt the data objects, assuming that the secret sharing value

$$\boldsymbol{M}=\begin{pmatrix}1&1&0&0&0\\0&1&0&0&0\\1&0&1&1&1\\0&0&0&0&1\\0&0&1&1&0\\0&0&1&0&0\end{pmatrix}\begin{matrix}\rightarrow\\\rightarrow\\\rightarrow\\\rightarrow\\\rightarrow\\\rightarrow\end{matrix}\begin{pmatrix}att_2\\att_1\\att_5\\att_3\\att_4\\att_6\end{pmatrix}$$

$s = 2$ and the vector $\nu = (2, 3, 1, 4, 6)$. Then the secret shares corresponding to each attribute is the set $\boldsymbol{M} \cdot \nu^{\top} = (5, 3, 13, 6, 5, 1)^{\top}$.

After encrypting M, the sensitive attributes att_2 and att_5 are hidden by the ABF Build algorithm. User u with attribute set $S_u = \{att_1, att_2\}$ sends his attribute set to the KGC. The KGC then sends the corresponding private key to the user based on $S_u = \{att_1, att_2\}$. After receiving the private key, the user verifies whether he can decrypt the ciphertext through the ABF Query. Since the user's attribute att_1 is in plaintext form in the access structure, it does not need to be validated by the ABF Query. However, att_2 is obscured, and the system must perform an ABF Query for att_2. If the ABF Query validates the attribute att_2, the user will receive the location in the access matrix. Otherwise, the user receives a random string. If the user u can pass the validation, he gets the position of the attribute att_2 in the access matrix, that is the 1st row of access matrix.

Therefore, the user obtains

$$\boldsymbol{M}_u^{\mathrm{T}} = \begin{pmatrix}0&1\\1&1\\0&0\\0&0\\0&0\end{pmatrix}$$

from the ABF Query algorithm and finds the coefficients ω_i such that $\sum_{i\in I} \omega_i \cdot \boldsymbol{M}_i = (1, 0, \ldots, 0)$. The coefficients are $\omega_1 = -1$ and $\omega_2 = 1$.

Then, the user obtains

$$s = \begin{pmatrix}3\\5\end{pmatrix}\begin{pmatrix}-1 & 1\end{pmatrix} = 2$$

by the ω_i and the secret shares of att_i, $i = 1, 2$.

Finally, the system computes

$$\hat{e}(g,g)^{\alpha s} = \frac{\hat{e}(C_0, K_0)}{\hat{e}(\omega^{\sum_{i\in I} C_{j,4}\omega_i}, K_1)\Pi_{i\in I}\hat{e}(C_{j,1}, K_1)\hat{e}(C_{j,2} \cdot u^{c_{j,5}}, K_{i,2})\hat{e}(C_{j,3}, K_{i,3})^{\omega_i}} \tag{4}$$

and recovers the message as $M = \tilde{C}/\hat{e}(g,g)^{\alpha s}$.

6 SECURITY, POLICY PRIVACY, EFFICIENCY

6.1 Security

Theorem 1. No adversary can selectively break our scheme under the decisional q-BDHE assumption in polynomial time.

Proof. In the scenario, the plaintext message m was encrypted under the access structure $(\mathbf{M}, \rho)$. If an adversary wants to decrypt a ciphertext CT, he must obtain the private key SK. According to the security model of the scheme, the attribute set of the adversary does not satisfy the access structure $(\mathbf{M}^*, \rho^*)$ chosen in the initialization phase. Therefore, the adversary cannot obtain the private key. The only thing the adversary can do here is to attack the chosen plaintext attack mentioned in Section 6, and the adversary's advantage $\varepsilon = |\Pr[m = m_b] - \frac{1}{2}|$ is negligible. Hence, this scheme is safe under the decisional q-DBHE assumption. □

6.2 Policy Privacy

In our scheme, the sensitive attributes in the access structure $(\mathbf{M}, \rho)$ are hidden by the ABF. According to the analysis by Dong et al. [13], ABF produces far fewer false positives than traditional bloom filters. However, with the increase of element content in the ABF, the false positive will still increase. The sensitive attribute selection algorithm in our scheme reduces the number of attributes that need to be hidden by selecting sensitive attributes in the access structure and, thus, reduces the false positive produced by ABF to some extent. Moreover, when query the element att_u, we collect all bit strings that are in $h_i(att_u)$ and "XOR" together. If the result is att_i, then att_u is in S. Otherwise att_u is not in S, and the ABF outputs random strings. The threshold δ in the algorithm reflects the importance of the access policy to the data owner. If δ takes a small value, the attributes that have a little relationship with the private information Y are selected by the algorithm. If δ takes a larger value, the algorithm selects the attributes that are very closely related to the privacy information Y. The data owner can flexibly set the threshold δ according to his own needs in our scheme.

6.3 Efficiency

We use the attribute selection algorithm to extract sensitive attributes in the access policy. Although the attribute selection process has some cost impact on the encryption phase, this process can be performed before the encryption starts. In the ABF Build process, we only hide the sensitive attributes in the access policy, and we only verify sensitive attributes in the access policy in the ABF query. The efficiency of our approach is between the methods from Han et al. [7] and Hohenberger et al. [14]. Compared with both of these, our scheme does not only reduce the computational cost but also effectively protects the access policy. The δ value directly affects the sensitive attribute selection algorithm. Our proposal offers flexible protection of the policy privacy. The efficiency of our scheme is shown in Table 1.

Scheme	Scheme [14]	Scheme [7]	Our Scheme
Attributes hidden	No	Total	Partial
System setup	$\hat{e} + \exp$	$\hat{e} + \exp$	$\hat{e} + \exp$
Key generation	$(4p + 3)\exp$ $+ (2p + 1)M$	$(3pn + p + 3)\exp$ $+ (2pn + 1)M$	$(4p + 3)\exp$ $+ (2p + 1)M$
Offline-Encrypt	$(5p + 2)\exp + (2p)M$	$(5p + 2)\exp + (2p)M$	$(5p + 2)\exp + (2p)M$
Online-encrypt	M	M	M
ABF build	–	$(lt)H$	$(kt)H$
ABF Query	–	$(pt)H$	$(qt)H$
Decrypt	$(3p + 1)\hat{e} +$ $(2p + 1)\exp + (3p)M$	$(3p + 1)\hat{e} +$ $(2p + 1)\exp + (3p)M$	$(3p + 1)\hat{e} +$ $(2p + 1)\exp + (3p)M$

Table 1. Comparison of different schemes ($k \leq l$ and $q \leq p$)

- k: The number of sensitives attributes in the access policy,
- l: The number of attributes in the access policy,
- p: The number of attributes of the user,
- q: The number of sensitive attributes of the user,
- n: The number of attributes type in the system,
- t: The number of the hash function in the ABF.

7 EXPERIMENTAL RESULTS AND EVALUATION

To evaluate the feasibility and efficiency of our scheme, we first implemented our attribute selection algorithm on the Heart Disease Data Set[1]. We used 14 attributes:

1. age,
2. sex,
3. cp (chest pain type),
4. trestbps (resting blood pressure),
5. chol (serum cholestoral in mg/dl),
6. fbs (fasting blood sugar > 120 mg/dl),
7. restecg (resting electrocardiographic results),
8. thalach (maximum heart rate achieved),
9. exang (exercise induced angina),
10. oldpeak (ST depression induced by exercise relative to rest),

[1] `https://archive.ics.uci.edu/ml/datasets/Heart+Disease`

11. slope (the slope of the peak exercise ST segment),
12. ca (number of major vessels (0-3) colored by fluoroscopy),
13. thal (normal; fixed defect; reversible defect),
14. num (diagnosis of heart disease, the predicted attribute).

The last attribute "num" is the user's privacy Y; that is, if the user has heart disease. From the data set, we calculated the entropy $H(Y) = 0.995$. We set different threshold δ values to select sensitive attributes according to Definition 1, retrieving different sensitive attribute sets, as shown in Table 2.

Threshold	Final Sensitive Attribute Set (to Be Hidden)	Sensitive Attribute Sets
$\delta = 1.0$	$\varnothing$	/
$\delta = 0.95$	$S = \{13, 12, 3, 1, 8, 11, 9\}$	$S_1 = \{13, 12, 3, 1, 8, 11, 9\}$
$\delta = 0.90$	$S = \{13, 12, 3, 1, 8, 11\}$	$S_1 = \{13, 12, 3, 1, 8, 11\}$
$\delta = 0.85$	$S = \{13, 12, 3, 1, 8, 11\}$	$S_1 = \{13, 12, 3, 1, 8, 11\}$
$\delta = 0.80$	$S = \{13, 12, 3, 1, 8\}$	$S_1 = \{13, 12, 3, 1, 8\}$
$\delta = 0.75$	$S =${13, 12, 3, 1, 8}	$S_1 =${13, 12, 3, 1, 8}
$\delta = 0.70$	$S = \{13, 12, 3, 1, 9, 11, 2, 5, 8, 4, 7, 6, 10\}$	$S_1 = \{13, 12, 3, 1\}$, $S_2 = \{9, 11, 2, 5, 8, 4, 7, 6, 10\}$
$\delta = 0.65$	$S = \{13, 12, 3, 1, 9, 11, 2, 5, 8, 4, 7, 6\}$	$S_1 = \{13, 12, 3, 1\}$, $S_2 = \{9, 11, 2, 5, 8, 4, 7, 6\}$
$\delta = 0.60$	$S = \{13, 12, 3, 1, 9, 11, 2, 5, 8, 4, 7\}$	$S_1 = \{13, 12, 3, 1\}$, $S_2 = \{9, 11, 2, 5, 8, 4, 7\}$
$\delta = 0.55$	$S = \{13, 12, 3, 9, 11, 2, 1, 5, 8\}$	$S_1 = \{13, 12, 3\}$, $S_2 = \{9, 11, 2, 1, 5, 8\}$
$\delta = 0.50$	$S = \{13, 12, 3, 9, 11, 2, 1, 5, 8\}$	$S_1 = \{13, 12, 3\}$, $S_2 = \{9, 11, 2, 1, 5, 8\}$

Table 2. Sensitive attribute selection results

In Table 2 we enumerated the sensitive attribute sets as the sequence constructed by Algorithm 1. We enumerated the sensitive attributes as the selected sequences of them by the algorithm 1 also. When $\delta = 1.0$, no sensitive attribute set was constructed. Consequently, no attribute was selected as a sensitive attribute because $I(Y; X_1 X_2 \dots X_{13}) < H(Y)$. For other cases, the numbers of elements of two constructed sensitive attribute sets S_1 and S_2 decreased along with the decrease of δ. When $\delta = [0.7, 0.5]$, there appeared two sensitive attribute sets, that means attributes both in S_1 and S_2 independently bring enough information about Y with respect to the threshold δ. We propose that the finally selected attributes needing to be hidden are $S = S_1 \cup S_2$.

Since our work focuses on selectively hiding attributes in the access structure and follows Han et al.'s scheme [7], we designed experiments to evaluate the cost of the ABF build algorithm in the encryption process and the ABF query in the decryption process. We used a sytem with an Intel Core i5-7300 CPU at 2.60 GHz 2.71 GHz

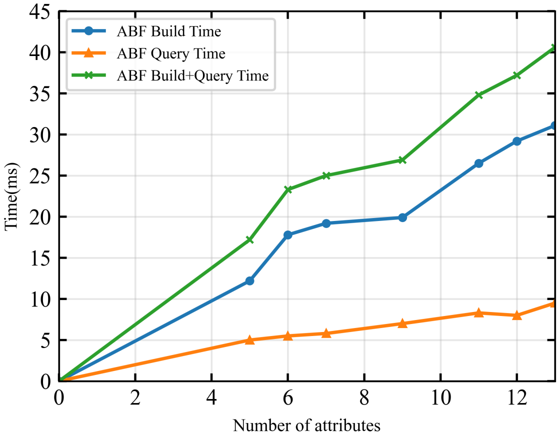


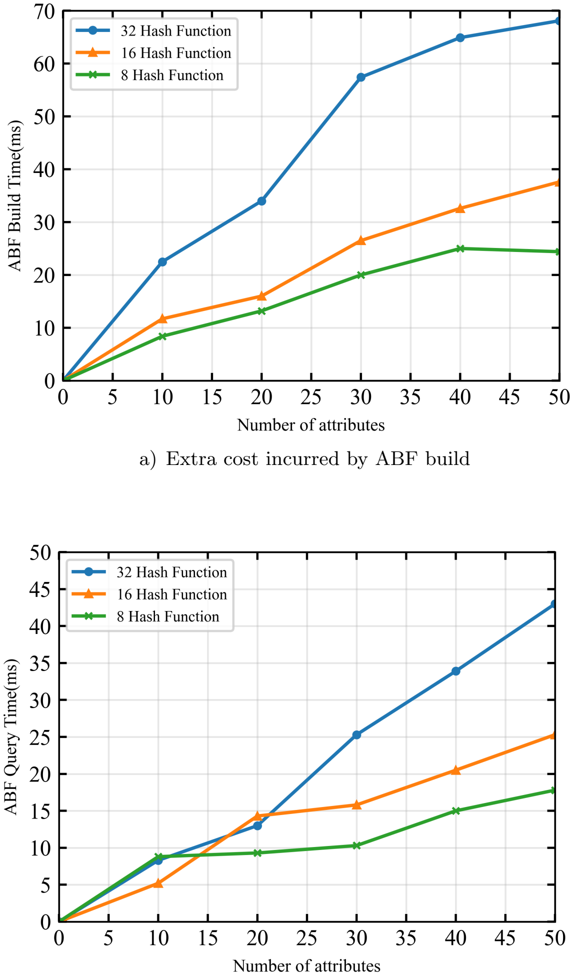
Figure 6. Extra cost incurred by ABF Build and Query (Heart Disease Data Set)

with 8 GB RAM running Windows 10 Professional to execute the ABF build and ABF query algorithms for randomly generated attributes in different numbers. We wrote the two algorithms in Java and ran them with Java Development Kit (JDK) version 1.8.0. All test results are the averaged times of 10 tests. Figure 6 shows the cost impact caused by ABF build and query. As shown in Table 2, if $\delta = 0.8$, the data owner hides five attributes instead of 13 in the balance between privacy and efficiency. As a result, the data owner reduces the cost of the ABF build process from 31.1 milliseconds to 12.2 milliseconds. The ABF query cost fell from 9.5 milliseconds to 5 milliseconds. The overall cost fell from 40.6 milliseconds to 17.2 milliseconds. This test used 32 hash functions.

In another experiment, we randomly generated 50 attributes, hid some of them, and measured the cost impacts. Figure 7 shows the costs of ABF build, ABF query, and the overall cost for hiding different numbers of attributes. The costs of comparison of plaintext attributes are negligible in all tests.

8 RELATED WORK

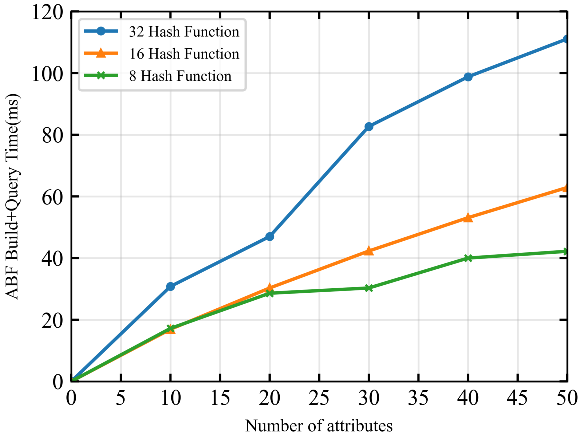
The identity-based encryption (IBE) method proposed by Sahai and Waters [1] is the predecessor of attribute-based encryption (ABE). In the original ABE [2, 15, 16], the private key and ciphertext are both associated with the user's attributes. Later, KP-ABE [2] and CP-ABE [3] expanded ABE. In CP-ABE, private keys are associated with attributes, and access policies are embedded into the ciphertext to achieve fine-grained access control. However, the original CP-ABE stored the access structure in plaintext form in the cloud, ignoring the privacy of the access policy.



a) Extra cost incurred by ABF build

b) Extra cost incurred by ABF query

Nishide et al. [4] introduced the concept of hiding the access policy for the first time, considering the confidentiality of the access structure. Soon after, several other CP-ABE schemes [5, 17] with hidden access policies were put forward. However, all these schemes inherit the "AND" gate access structure of [4] and the dependency of the ciphertext length on the number of attributes. So, there are not only limitations in policy expression but also a large computational overhead. Later, new schemes were proposed to reduce the computational cost [18, 19, 20], but the "AND" gate access structure is limits policy expression. Compared to the "AND" gate access structure, the LSSS access matrix is less restrictive in policy expression,



c) Overall cost

Figure 7. Extra cost incurred by ABF build and query

so Cui et al. and Belguith et al. [8, 9, 21] proposed the fully secure CP-ABE scheme with an LSSS access structure. In Cui's efforts [8, 9], the attribute is composed of the attribute name and attribute value. In their scheme, the attributes values are hidden with attribute names in plaintext. In the CP-ABE schemes using an access tree [5, 22, 23], attribute anonymity technology protects the security of the policy by anonymous operation of the leaf nodes in the access tree. However, none of those schemes solve the linear growth relation between the ciphertext and the number of attributes. In the CP-ABE scheme with hidden access policy [24, 25], the linear growth relation is controlled effectively. Our scheme also uses the LSSS access structure, but we focus more on protecting sensitive attributes in the access policy to prevent privacy disclosure while reducing redundant computation.

The diversity of access structures in different CP-ABE schemes leads to different policy hiding methods. The tree access structure uses attribute anonymity technology to hide leaf nodes [22], while CP-ABE [10] with its LSSS access structure directly hides the whole access matrix to protect privacy in the access policy and applies hybrid encryption for determining whether the user's attributes satisfy the access policy. Recently, a new approach has been adopted to realize policy hiding [6, 7]. In these schemes, the attribute mapping relationship is hidden, and all the attributes in the access policy are hidden with ABF. Before the user executes the decryption algorithm, the user is verified by the ABF Query. The user passes the verification if his attributes satisfy the access policy and then can perform the decryption algorithm. Otherwise, the user obtains no information related to the access policy. So, these schemes achieve the goal of hiding access policy dexterously. However, not all attributes in the access policy disclose the user's privacy.

Thus, Han's method [7] of hiding all the attributes increases the computational overhead. Therefore, we consider extracting sensitive attributes in access policy via an attribute selection algorithm and obscure only those. Our work reduces the computational overhead of the encryption and decryption process while effectively protecting access policy privacy to prevent the leakage of sensitive information.

9 CONCLUSION

In this paper, we have proposed a CP-ABE scheme with some attributes hidden. In contrast to the existing approaches that hide all attributes in the access policy, we have introduced "partial hiding." To achieve partial hiding of attributes, we have presented a sensitive attribute selection algorithm to partition attributes in the access policy into two sets. According to his own need, the data owner chooses the partition criteria and can select the sensitive attributes before data encryption work starts. We hide selected sensitive attributes using ABF and leave the remainder exposed in plaintext. Our security analysis showed that our scheme is selectively secure against chosen plaintext attacks. Since we hide only some of the attributes instead of all of them, our method is more efficient while still protecting sensitive information in the access policy. Experimental results show that the presented attribute selection algorithm is reasonable and feasible. In addition, by hiding some of attributes, we reduce the cost during the ABF build and query processes. Our attribute selection algorithm has flexibility in that δ reflects how important the policy privacy is for the data owner.

There are some challenges to our proposal. There is no unique answer for selecting sensitive attributes. Defining the privacy information Y and assessing the privacy parameter δ is challenging. In our future work, we will further consider more reasonable and feasible attribute selection algorithms.

Acknowledgements

This research work was supported by National Natural Science Foundation of China (Grant No. 61862059 and 61562085).

10 CONFLICT OF INTEREST

We declare that we have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] Sahai, A.—Waters, B.: Fuzzy Identity-Based Encryption In: Cramer, R. (Ed.): Advances in Cryptology – EUROCRYPT 2005. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3494, 2005, pp. 457–473, doi: 10.1007/11426639_27.

[2] Goyal, V.—Pandey, O.—Sahai, A.—Waters, B.: Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06), 2006, pp. 89–98, doi: 10.1145/1180405.1180418.

[3] Bethencourt, J.—Sahai, A.—Waters B.: Ciphertext-Policy Attribute-Based Encryption. 2007 IEEE Symposium on Security and Privacy (SP '07), 2007, pp. 321–334, doi: 10.1109/sp.2007.11.

[4] Nishide, T.—Yoneyama, K.—Ohta, K.: Attribute-Based Encryption with Partially Hidden Encryptor-Specified Access Structures. In: Bellovin, S. M., Gennaro, R., Keromytis, A., Yung, M. (Eds.): Applied Cryptography and Network Security (ACNS 2008). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5037, 2008, pp. 111–129, doi: 10.1007/978-3-540-68914-0_7.

[5] Lai, J.—Deng, R. H.—Li, Y.: Fully Secure Ciphertext-Policy Hiding CP-ABE. In: Bao, F., Weng, J. (Eds.): Information Security Practice and Experience (ISPEC 2011). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6672, 2011, pp. 24–39, doi: 10.1007/978-3-642-21031-0_3.

[6] Yang, K.—Han, Q.—Li, H.—Zheng, K.—Shen, X.: An Efficient and Fine-Grained Big Data Access Control Scheme with Privacy-Preserving Policy. IEEE Internet of Things Journal, Vol. 4, 2017, No. 2, pp. 563–571, doi: 10.1109/jiot.2016.2571718.

[7] Han, Q.—Zhang, Y.—Li, H.: Efficient and Robust Attribute-Based Encryption Supporting Access Policy Hiding in Internet of Things. Future Generation Computer Systems, Vol. 83, 2018, pp. 269–277, doi: 10.1016/j.future.2018.01.019.

[8] Cui, H.—Deng, R. H.—Wu, G.—Lai, J.: An Efficient and Expressive Ciphertext-Policy Attribute-Based Encryption Scheme with Partially Hidden Access Structures. In: Chen, L., Han, J. (Eds.): Provable Security (ProvSec 2016). Springer, Cham, Lecture Notes in Computer Science, Vol. 10005, 2016, pp. 19–38, doi: 10.1007/978-3-319-47422-9_2.

[9] Cui, H.—Deng, R. H.—Lai, J.—Yi, X.—Nepal, S.: An Efficient and Expressive Ciphertext-Policy Attribute-Based Encryption Scheme with Partially Hidden Access Structures, Revisited. Computer Networks, Vol. 133, 2016, No. 2018, pp. 157–165, doi: 10.1016/j.comnet.2018.01.034.

[10] Khan, F.—Li, H.—Zhang, L.—Shen, J.: An Expressive Hidden Access Policy CP-ABE. 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), 2017, pp. 178–186, doi: 10.1109/dsc.2017.29.

[11] Borda, M.: Fundamentals in Information Theory and Coding. Springer, Berlin, Heidelberg, 2011, pp. 11–23, doi: 10.1007/978-3-642-20347-3.

[12] Cover, T. M.—Thomas, J. A.: Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing). Wiley-Interscience, 2001, pp. 13–23, doi: 10.1002/0471200611.ch2.

[13] Dong, C.—Chen, L.—Wen, Z.: When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol. Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, 2013, pp. 789–800, doi: 10.1145/2508859.2516701.

[14] Hohenberger, S.—Waters, B.: Online/Offline Attribute-Based Encryption. In: Krawczyk, H. (Ed.): Public-Key Cryptography – PKC 2014. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8383, 2014, pp. 293–310, doi: 10.1007/978-3-642-54631-0_17.

[15] Lewko, A.—Okamoto, T.—Sahai, A.—Takashima, K.—Waters, B.: Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. Advances in Cryptology – EUROCRYPT 2010. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6110, 2010, pp. 62–91, doi: 10.1007/978-3-642-13190-5_4.

[16] Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (Eds.): Public Key Cryptography – PKC 2011. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6571, 2011, pp. 53–70, doi: 10.1007/978-3-642-19379-8_4.

[17] Li, X.—Gu, D.—Ren, Y.—Ding, N.—Yuan, K.: Efficient Ciphertext-Policy Attribute-Based Encryption with Hidden Policy. In: Xiang, Y., Pathan, M., Tao, X., Wang, H. (Eds.): Internet and Distributed Computing Systems (IDCS 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7646, 2012, pp. 146–159, doi: 10.1007/978-3-642-34883-9_12.

[18] Zhang, L.—Cui, Y.—Mu, Y.: Improving Privacy-Preserving CP-ABE with Hidden Access Policy. In: Sun, X., Pan, Z., Bertino, E. (Eds.): Cloud Computing and Security (ICCCS 2018). Springer, Cham, Lecture Notes in Computer Science, Vol. 11065, 2018, pp. 596–605, doi: 10.1007/978-3-030-00012-7_54.

[19] Huang, C.—Yan, K.—Wei, S.—Zhang, G.—Lee, D. H.: Efficient Anonymous Attribute-Based Encryption with Access Policy Hidden for Cloud Computing. 2017 IEEE International Conference on Progress in Informatics and Computing (PIC 2017), 2017, pp. 266–270, doi: 10.1109/PIC.2017.8359555.

[20] Qiu, S.—Liu, J.—Shi, Y.—Zhang, R.: Hidden Policy Ciphertext-Policy Attribute-Based Encryption with Keyword Search Against Keyword Guessing Attack. Science China (Information Sciences), Vol. 60, 2017, Art. No. 052105, pp. 1–12, doi: 10.1007/s11432-015-5449-9.

[21] Belguith, S.—Kaaniche, N.—Laurent, M.—Jemai, A.—Attia, R.: PHOABE: Securely Outsourcing Multi-Authority Attribute Based Encryption with Policy Hidden for Cloud Assisted IoT. Computer Networks, Vol. 133, 2018, pp. 141–156, doi: 10.1016/j.comnet.2018.01.036.

[22] Hur, J.: Attribute-Based Secure Data Sharing with Hidden Policies in Smart Grid. IEEE Transactions on Parallel and Distributed Systems, Vol. 24, 2013, No. 11, pp. 2171–2180, doi: 10.1109/tpds.2012.61.

[23] Sabitha, S.—Rajasree, M. S.: Access Control Based Privacy Preserving Secure Data Sharing with Hidden Access Policies in Cloud. Journal of Systems Architecture, Vol. 75, 2017, No. 2018, pp. 50–58, doi: 10.1016/j.sysarc.2017.03.002.

[24] Phuong, T. V. X.—Yang, G.—Susilo, W.: Hidden Ciphertext-Policy Attribute-Based Encryption Under Standard Assumptions. IEEE Transactions on Information Forensics and Security, Vol. 11, 2016, No. 1, pp. 35–45, doi: 10.1109/tifs.2015.2475723.

[25] Susilo, W.—Yang, G.—Guo, F.—Huang, Q.: Constant-Size Ciphertexts in Threshold Attribute-Based Encryption without Dummy Attributes. Information Sciences, Vol. 429, 2018, pp. 349–360, doi: 10.1016/j.ins.2017.11.037.

Gulmire Arkin received her M.Sc. degree in the School of Mathematics and System Science, Xinjiang University in 2020. She works in the College of Science, Xinjiang Institute of Technology as Full Time Teacher. Her current research interests include information system security, access control, and cloud storage security.

Nurmamat Helil received his B.Sc., M.Sc. and Ph.D. degrees in the School of Mathematical Sciences, Peking University in 2000, 2003 and 2008, respectively. He is Full Professor of the College of Mathematics and System Science, Xinjiang University, China. From April 2010 to April 2011, he worked as Post-Doctor in the School of Computer Science and Engineering, Chung-Ang University, Korea. From April 2016 to April 2017, he worked as Visiting Research Scholar in the Department of Computer Science and Engineering, University of Minnesota Twin Cities, USA. His research interests include information system security, access control, and cloud storage security.