Dynamic Access Policy in Cloud-Based Personal Health Record (PHR) Systems

Xuhui Liu[a], Qin Liu[a], Tao Peng[b], Jie Wu[c]

[a]*College of Computer Science and Electronic Engineering*
Hunan University
Changsha, Hunan Province, P. R. China 410082
[b]*School of Information Science and Engineering*
Central South University
Changsha, Hunan Province, P. R. China 410083
[c]*Department of Computer and Information Sciences*
Temple University
Philadelphia, PA 19122, USA

Abstract

With the development of cloud computing, an increasing number of users are using cloud-based personal health record (PHR) systems. The PHR is closely tied to patient privacy, and thus existing studies suggest encrypting PHRs before outsourcing. Comparison-based encryption (CBE) was the first to implement time comparison in an attribute-based access policy by means of the forward and backward derivation functions. However, CBE cannot be directly applied to cloud-based PHR environments for the following reasons: First, the cost of encryption grows linearly with the number of attributes in the access policy. Second, policy updating incurs high communication and computation costs for the data owner. To efficiently implement a dynamic access policy for PHRs in clouds, we first propose a hierarchical comparison-based encryption (HCBE) scheme that incorporates an attribute hierarchy into CBE. The HCBE scheme encrypts a ciphertext with a small number of generalized attributes at a higher level rather than many specific attributes at a lower level, greatly improving the encryption performance. Using the HCBE scheme as a foundation, we then develop a dynamic policy updating (DPU) scheme by utilizing the proxy re-encryption (PRE) technique. The DPU scheme can avoid the transmission of ciphertexts and minimize the computation overhead on the data owner by delegating the policy updating operations to the cloud. Extensive experiments have been conducted using

a synthetic data set to verify the efficiency of our proposed schemes.

Keywords:
personal health record, cloud computing, comparison-based encryption, attribute hierarchy, dynamic access policy

---

1. Introduction

In recent years, the personal health record (PHR) [25] as a patient-centric model of health information exchange has become popular with an increasing number of users because of the convenience of merging a wide range of health information sources to create a centralized patient profile that can be easily accessed. PHR allows medical practitioners online access to a complete and accurate summary of a patient's medical history, which streamlines care [9]. Cloud computing is a model for enabling ubiquitous and convenient network access to data resources [1]. Because of its overwhelming advantages, such as rapid elasticity, high availability, and low cost, a growing number of patients are deciding to outsource their PHRs to the cloud. The most popular cloud-based PHR systems have included Google Health [32] and Microsoft HealthVault [33], which promise users access to the PHR services at any time and at any place using any kind of device connected to the Internet.

However, a PHR, which includes health data such as allergies or adverse drug reactions, family history, and imaging reports, is closely tied to patient privacy. Allowing a cloud service provider (CSP) such as Amazon, Google, or Microsoft to manage sensitive medical data may raise potential issues. For instance, an untrustworthy CSP may intentionally leak PHRs to medical companies or medical instrument companies for a profit. Existing research suggests encrypting the PHRs before outsourcing in order to preserve patients' privacy [26].

Let us consider the following application scenario: Alice is hospitalized in Hospital A, in need of heart surgery. To help the doctors diagnose the disease better, Alice uploads her encrypted PHR to Google Health, specifying an access policy, as shown in Fig. 1-(a). Unfortunately, during the course of therapy, Alice is diagnosed with hypertension and asthma. To avoid the surgical risks and complications, the relevant attending doctors in Hospital A need to hold a consultation to decide on a surgery program based on careful study of Alice's PHR. For convenience and flexibility, Alice updates the access policy for her encrypted PHR, as shown in Fig. 1-(b).

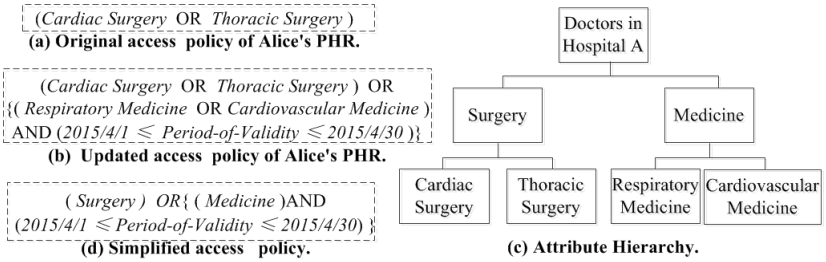


Figure 1: Application scenario.

The access policy can be viewed as a description of attributes and time conditions specifying that only the users whose attributes satisfying the access policy can decrypt the ciphertext during the specified time period. For instance, Fig. 1-(b) stipulates that the cardiologists and the respiratory physicians can view Alice's PHR during the consultation (April 1, 2015, through April 30, 2015), and the cardiac surgeons and the thoracic surgeons can view it at any time. Therefore, the encryption scheme adopted needs to meet the following requirements:

(1) Support an attribute-based access policy. For example, for a given ciphertext associated with access policy $((A_1 \wedge A_2) \vee A_3)$, only the users who possess both attributes A_1 and A_2 or those who possess attribute A_3 can recover it using their own decryption keys.

(2) Support time-based comparison. For example, the time condition of the ciphertext is $(A_1 \wedge [t_x, t_y])$, which means that users possessing attribute A_1 can access the data during time $[t_x, t_y]$.

(3) Support a dynamic access policy. For example, while the access policy is changed from (A_1) to $(A_1 \wedge A_2)$, only the users who possess both attributes A_1 and A_2 can decrypt the ciphertext, and users who possess only attribute A_1 cannot access the data any more.

Comparison-based encryption (CBE) [31] was proposed by Zhu et al. in 2012 as a promising tool facilitating fine-grained access control in cloud computing. CBE utilizes the forward and backward derivation functions to achieve time comparison in attribute-based encryption. For example, suppose that the access policy of the ciphertext is $(A_1 \wedge [t_x, t_y])$, and the time of authorization of a user with attribute A_1 is $[t_a, t_b]$. Then, the user can decrypt the ciphertext only when the current time $(t_c \in [t_x, t_y]) \wedge (t_c \in [t_a, t_b])$. At the same time, the key delegation mechanism was applied to assign a majority of decryption costs to the cloud in order to take full advantage of cloud

resources.

However, CBE cannot be directly applied to cloud-based PHR environments for the following reasons: First, the cost for encryption grows linearly with the number of attributes in the access policy. For a system with a large number of attributes, the cost for encryption may be considerable. Second, because of the lack of local copies of the PHR data, the data owner needs to retrieve the original ciphertext from the cloud, re-encrypt it under the new access policy, and then send the new ciphertext back to the cloud in order to update the access policy. This process will incur high communication and computation costs for the data owner.

In this paper, we first propose a hierarchical comparison-based encryption (HCBE) scheme for efficiently achieving a dynamic access policy in cloud-based PHR systems. The main idea of the HCBE scheme is building a hierarchical structure for attributes, in which an attribute at a higher level is a generalization of attributes at lower levels. Specifically, we encrypt the ciphertext with a small number of generalized attributes at a higher level rather than with many specific attributes at the lower level. For example, if we construct an attribute tree, as shown in Fig. 1-(c), the access policy can be simplified, as shown in Fig. 1-(d), with which the computation cost for encryption may be greatly reduced compared to that in Fig. 1-(b). To implement the attribute hierarchy, we encode each node in an attribute tree with positive-negative depth-first (PNDF) coding. Then, we apply the backward derivation function of CBE to allow the descendant attribute node to deduce the secrets associated with its ancestor attribute nodes. Therefore, users with the specific attributes can decrypt the ciphertext encrypted with the generalized attributes. For example, when the ciphertext is encrypted with access policy $(medicine) \wedge [2015-4-1, 2015-4-30]$, only the cardiologists and the respiratory physicians can decrypt it between April 1, 2015, and April 30, 2015.

Using HCBE as a foundation, we then develop a dynamic policy updating (DPU) scheme by utilizing the proxy re-encryption (PRE) technique [3]. The main idea of the DPU scheme is to allow the data owner to send an update key to the cloud, which will update the access policy associated with the ciphertext without knowing the content of the plaintext. The DPU scheme can avoid the transmission of ciphertext and minimize the computation overhead incurred by the data owner by delegating the policy updating operations to the cloud. Specifically, we express the access policy as an access tree and provide a generalized version (referred to as G-DPU) and an efficient version

(referred to as E-DPU) for the DPU scheme. G-DPU transforms the problem of updating the AND/OR gate to that of updating the threshold gate. For example, the AND gate is transformed to a (t, t) gate, and the OR gate is transformed to a $(1, t)$ gate. Therefore, the updating of the AND gate can be treated as updating the (t, t) gate to a (t', t') gate, and the updating of the OR gate can be treated as updating the $(1, t)$ gate to a $(1, t')$ gate, where $t' = t + 1$ for adding an attribute to the AND/OR gate and $t' = t - 1$ for removing an attribute from the AND/OR gate. The main drawback of G-DPU is that the updating cost will grow linearly with the number of attributes under the gate. Therefore, we provide E-DPU, whose cost remains constant while updating an attribute. Our main contributions in this work are summarized as follows:

1. We are among the first to consider the problem of efficiently achieving dynamic access control in cloud-based PHR systems.
2. We propose the HCBE scheme, which builds an attribute hierarchy and utilizes PNDF coding to improve the encryption performance of the CBE scheme.
3. Using the HCBE scheme as a foundation, we have developed the D-PU scheme , which utilizes the PRE technique to delegate the policy updating operations to the cloud.
4. We analyse the performance and the security of the proposed schemes and have conducted experiments to validate their effectiveness and efficiency.

The rest of this paper is organized as follows. In Section 2, we introduce our models, design goals, and technical preliminaries. Then, we provide an overview of our HCBE scheme in Section 3 and provide its construction in Section 4. Next, we present the construction of the DPU scheme in Section 5. We analyse the security and efficiency of our scheme in Section 6 and describe our experiments in Section 7. Finally, we introduce related work in Section 8, and we conclude the paper in Section 9.

2. Preliminaries

2.1. System Model

As shown in Fig. 2, the system consists of the following parts: the cloud service provider (CSP), the data owner, and the data users. The CSP operates the cloud-based PHR system, which is located on a large number of

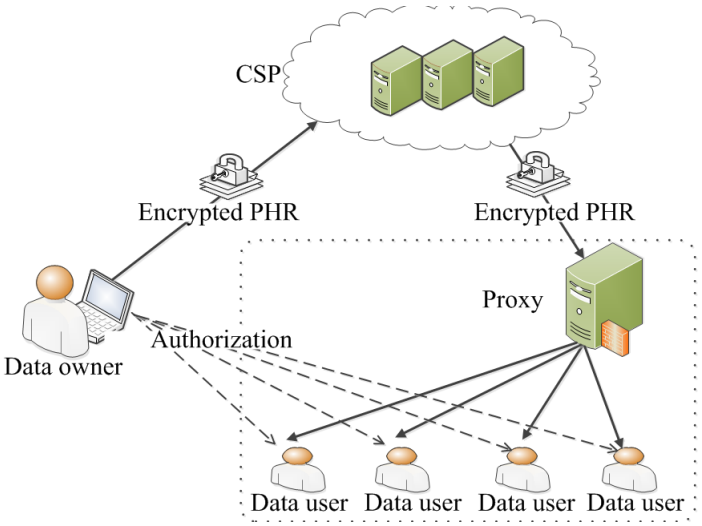


Figure 2: System model.

interconnected cloud servers with abundant hardware resources. The data owner is the individual patient who employs the cloud-based PHR system to manage her PHRs. The data users are the entities who are authorized by the data owner to access the cloud-based PHR system. Taking the application scenario in Fig. 1 as an example, Alice is the data owner, Google is the CSP, and Alice's doctors in Hospital A are the data users. A proxy server responsible for the decryption operation can be deployed internally when all the data users are located in the same trusted domain.

Suppose that the universal attribute set $\mathcal{A} = \{A_1, \ldots, A_M\}$, from which an attribute hierarchy $\widehat{\mathcal{A}}$ of L levels is built. In the tree structure, each attribute A_k contains two hierarchy codes, $\{Pcode_k, Ncode_k\}$, such that the descendant node's codes are larger than those of its ancestors. To efficiently achieve fine-grained access control while using the cloud-based PHR services, our HCBE scheme will be employed as follows. We describe each user with an attribute-based access privilege $\widehat{\mathcal{L}}$, where each attribute $A_k \in \widehat{\mathcal{L}}$, denoted as $A_k(t_a, t_b, Pcode_k, Ncode_k)$, is associated with the authorization time $[t_a, t_b]$ and hierarchy codes $\{Pcode_k, Ncode_k\}$.

The PHR is encrypted with an attribute-based access policy, $\widehat{AP}$, in which each attribute $A_l \in \widehat{AP}$, denoted as $A_l(t_i, t_j, Pcode_l, Ncode_l)$, is also associated with the time condition $[t_i, t_j]$ and hierarchy codes $\{Pcode_l, Ncode_l\}$. The data user can decrypt the PHR only when the following conditions are

simultaneously satisfied:

(1) User attributes satisfy the access policy, denoted $\widehat{\mathcal{L}} \sqsubseteq \widehat{AP}$.

(2) The current time t_c satisfies $(t_c \in [t_x, t_y]) \wedge (t_c \in [t_a, t_b])$.

(3) The attributes in $\widehat{\mathcal{L}}$ are either the same as or more specific than those in $\widehat{AP}$, denoted as $Pcode_k \geq Pcode_l$ and $Ncode_k \geq Ncode_l$.

For the dynamic access policy, the data owner will generate an update key UK, which will be sent to the CSP. On receiving UK, the CSP will update the policy from $\widehat{AP}$ to $\widehat{AP'}$ on behalf of the data owner.

2.2. Adversary Model

Our design goal is to preserve privacy for the data owner while she is using the cloud-based PHR services. There are two main attacks under such a circumstance: *external attacks* initiated by unauthorized outsiders , and *internal attacks* initiated by an *honest but curious CSP* and *malicious data users*. The communication channels are assumed to be secured under existing security protocols such as SSL and SSH , thus we consider only the internal attacks. In the adversary model, the CSP and malicious data users are considered as potential attackers, which are assumed to be more interested in the contents of stored data and the user's private key than in other secret information.

As in [17], we assume that the honest but curious CSP will always correctly execute a given protocol but may try to find out as much secret information as possible using the inputs. The CSP will try to obtain as much prior knowledge as possible to break the ciphertext or forge the private key, which is a form of semantical-security-under-chosen-derivation-key attack (SS-CDA). The malicious data users cannot observe the encrypted data stored in outsourced storage, so they cannot attack the ciphertext directly. However, the malicious data users may collude to access the PHRs outside their permissions, which is a form of collusion privilege (CP) attack. In addition, the malicious data users may increase their attack capabilities by observing the derivation keys directly derived from the valid private keys, which is a form of key-security-under-chosen-derivation-key attack (KS-CDA).

As defined in [31], the attackers may initiate CP attacks, KS-CDAs, and SS-CDAs to access unauthorized data files. Therefore, the HCBE scheme is considered to fail if either of the following cases occurs:

- **CASE 1.** Data user $\mathcal{U}$ with access privilege $\widehat{\mathcal{L}} = \{A_k(t_a, t_b, Pcode_k, Ncode_k)\}$ is able to access a PHR with access policy $\widehat{AP} = \{A_l(t_i, t_j, Pcode_l,$

$Ncode_l$)} while any of the following conditions is true:

(1) $\widehat{\mathcal{L}} \not\sqsubseteq \widehat{AP}$.

(2) $[t_a, t_b] \cap [t_i, t_j] = null$;

(3) $Pcode_k < Pcode_l \vee Ncode_k < Ncode_l$.

- **CASE 2.** The CSP can access the PHR without permission.

The DPU scheme is considered to fail if either of the above cases occurs after a policy update.

2.3. Proxy Re-encryption

Let us illustrate the motivation of the PRE scheme [3] by the following example: Alice receives emails encrypted under her public key PK_A via a semi-trusted mail server. When she leaves for vacation, she wants to delegate her email to Bob, whose public key is PK_B, but she does not want to share her secret key SK_A with him. The PRE scheme allows Alice to provide a PRE key $RK_{A \to B}$ to the mail server, with which the mail server can convert a ciphertext that is encrypted under Alice's public key PK_A into another ciphertext that can be decrypted by Bob's secret key SK_B without seeing the underlying plaintext, SK_A, or SK_B.

Note that although the data are encrypted twice, first encrypted with Alice's public key, and then re-encrypted with a PRE key, Bob only needs to execute decryption once to recover the data. The PRE scheme is based on ElGamal encryption [6], and thus the ciphertext is semantically secure, and given the PRE key, the mail server cannot guess either of the secret keys SK_A or SK_B. (Please refer to [3] for more details.) In our DPU scheme, the data owner will send an update key UK to the CSP, which will be delegated to perform policy updating operations in a secure way.

2.4. Composite-Order Bilinear Map

Let p and q be two large primes, and let $N = pq$ be the RSA modulus. Following the work in [5], we define a bilinear map group system $S_N = (N, \mathbb{G}, \mathbb{G}_{\mathrm{T}}, e)$, where $\mathbb{G}$ and $\mathbb{G}_{\mathrm{T}}$ are cyclic groups of prime order $n = sp'q'$ [1], and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}$ is a bilinear map with the following properties:

[1] Let s_1 and s_2 be two secret large primes. We have $n = sn' = s_1 s_2 p' q' | lcm(p+1, q+1)$, where $n' = p'q' | n$, $s = s_1 s_2$, $p = 2p's_1 - 1$, and $q = 2q's_2 - 1$.

- **Bilinearity**: for $a, b \in \mathbb{Z}_n$ and $g_1, g_2 \in \mathbb{G}$, it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
- **Non-degeneracy**: $e(g_1, g_2) \neq 1$, where g_1 and g_2 are the generators of group $\mathbb{G}$;
- **Computability**: $e(g_1, g_2)$ is efficiently computable.

As in [31], we make N public and keep n, s, p', and q' secret in this system. Let $\mathbb{G}_s$ and $\mathbb{G}_{n'}$ denote the subgroups of order s and $n' = p'q'$ in $\mathbb{G}$, respectively. We have $e(g, h) = 1$, when $g \in \mathbb{G}_s$ and $h \in \mathbb{G}_{n'}$.

2.5. Comparison-Based Encryption

In CBE, time is denoted as a set of discrete values $U = \{t_1, t_2, \ldots, t_T\}$, with total ordering $0 \leq t_1 \leq t_2 \leq \ldots \leq t_T \leq Z$, where Z is the maximal integer. Let φ and $\overline{\varphi}$ be two random generators in $\mathbb{G}_{n'}$, where $n' = p'q'$, and p' and q' are two large primes. The functions $(\psi(\cdot), \overline{\psi}(\cdot))$ mapping from integer set $U = \{t_1, t_2, \ldots, t_T\}$ to $V = \{v_{t_1}, \ldots, v_{t_T}\} \in \mathbb{G}_{n'}$ and $\overline{V} = \{\overline{v}_{t_1}, \ldots, \overline{v}_{t_T}\} \in \mathbb{G}_{n'}$ are defined as follows:

$$\begin{aligned} v_{t_i} &\leftarrow \psi(t_i) = \varphi^{\lambda^{t_i}} \\ \overline{v}_{t_i} &\leftarrow \overline{\psi}(t_i) = \overline{\varphi}^{\mu^{Z-t_i}}, \end{aligned} \tag{1}$$

where λ and μ are randomly chosen from $\mathbb{Z}_{n'}^*$.

Using Eq. 1, the forward derivation function (FDF), $f(\cdot)$, and the backward derivation function (BDF), $\overline{f}$, are defined as follows:

$$\begin{aligned} v_{t_j} &\leftarrow f(v_{t_i}) = (v_{t_i})^{\lambda^{t_j - t_i}}, \quad t_i \leq t_j \\ \overline{v}_{t_j} &\leftarrow \overline{f}(\overline{v}_{t_i}) = (\overline{v}_{t_i})^{\mu^{t_i - t_j}}, \quad t_i \geq t_j. \end{aligned} \tag{2}$$

FDF and BDF have the *one-way* property, under the RSA assumption that λ^{-1} and φ^{-1} cannot be efficiently computed because of the secrecy of n'. That is, Eq. 2 is efficiently computable, but it is intractable to obtain v_{t_j} from v_{t_i} while $t_i > t_j$ and to obtain $\overline{v}_{t_j}$ from $\overline{v}_{t_i}$ while $t_i < t_j$. For a given set of attributes $\mathcal{A} = \{A_1, \ldots, A_M\}$, CBE consists of the following algorithms:

- *Setup*$(1^\kappa, \boxed{\mathcal{A}})$: establishes the CBE system on the basis of the flat-structured attribute set $\mathcal{A}$ and generates the master key MK and public key $PK_{\mathcal{A}}$.

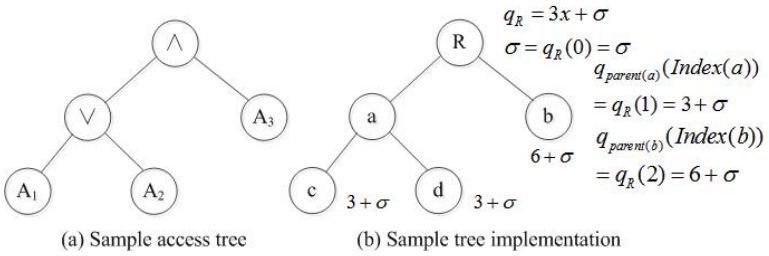


Figure 3: Access tree.

- $GenKey(MK, \mathcal{U}, \boxed{\mathcal{L}})$: generates the private key $SK_{\mathcal{L}}$ on access privilege $\mathcal{L}$ to user $\mathcal{U}$, where each attribute $A_k \in \mathcal{L}$, denoted as $A_k(t_a, t_b)$, is associated with the authorization time $[t_a, t_b]$.

- $Encrypt(PK_{\mathcal{A}}, \boxed{AP})$: takes the public key $PK_{\mathcal{A}}$ and an access policy AP as inputs to generate a session key, ek, and a ciphertext header, $\mathcal{H}_{\mathcal{P}}$, where each attribute $A_l \in AP$, denoted as $A_l(t_i, t_j)$, is associated with the time condition $[t_i, t_j]$.

- $Delegate(\boxed{SK_{\mathcal{L}}}, \mathcal{L}')$: derives a delegation key $SK_{\mathcal{L}'}$ from $SK_{\mathcal{L}}$ for which $\mathcal{L}'$ is less privileged than $\mathcal{L}$, denoted as $\mathcal{L}' \preceq \mathcal{L}$ [2].

- $Decrypt1(\boxed{SK_{\mathcal{L}'}}, \mathcal{H}_{\mathcal{P}})$: converts $\mathcal{H}_{\mathcal{P}}$ into $\mathcal{H}'_{\mathcal{P}}$ with $SK_{\mathcal{L}'}$.

- $Decrypt2(SK_{\mathcal{L}}, \boxed{\mathcal{H}'_{\mathcal{P}}})$: decrypts $\mathcal{H}'_{\mathcal{P}}$ with $SK_{\mathcal{L}}$ to obtain ek.

For improved efficiency, the output of the *Encrypt* algorithm is a random session key ek, which can be used to encrypt the object files using a symmetrical-key cryptosystem. As an improvement on CBE, the HCBE scheme first constructs an attribute hierarchy $\widehat{\mathcal{A}}$ of L levels from $\mathcal{A}$, where leaf nodes are specific attributes and the ancestor node is the generalized attribute of its descendant nodes. Then, we encode each attribute node with the PNDF coding and apply the BDF in CBE to accomplish the attribute hierarchy. We mark the main differences in each algorithm of CBE with boxes for ease of comparison.

[2]Let S and S' denote the set of attributes in $\mathcal{L}$ and $\mathcal{L}'$, respectively. $\mathcal{L}' \preceq \mathcal{L}$ if and only if $S' \subseteq S$, and for each attribute $A_k(t_a, t_b] \in \mathcal{L}$ and $A_k(t_i, t_j] \in \mathcal{L}'$, $t_a \leq t_i$ and $t_b \geq t_j$.

3. Overview of HCBE Scheme

3.1. Access Tree

Following the work in [2], the access policy $\widehat{AP}$ in the HCBE scheme, which is expressed as a Boolean function on AND/OR logic gates, can be depicted as an access tree $\mathcal{T}$, where each interior node is a gate, and the leaves are depicted as attributes . For example, given an access policy $(A_1 \vee A_2) \wedge A_3$, the corresponding access tree $\mathcal{T}$ is as shown in Fig. 3-(a). In the tree, each node a is associated with a threshold value k_a. For the interior node a with N_a children, $k_a = 1$ when a is an OR gate, and $k_a = N_a$ when a is an AND gate. For all leaf nodes, the threshold value is 1.

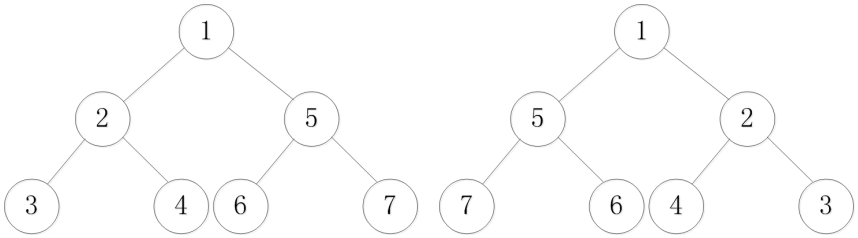
Let function $parent(a)$ denote the parent of node a in access tree $\mathcal{T}$. If a is a leaf node in $\mathcal{T}$, function $att(a)$ is used to denote the attribute associated with a. Furthermore, $\mathcal{T}$ defines an ordering between the children of each node, and the function $index(a)$ returns such a number associated with the children node a. Let $\mathcal{T}$ with root R denote the access tree of $\widehat{AP}$. The policy checking process and the tree implementation process are as follows:

Policy checking. The access privilege $\widehat{\mathcal{L}}$ satisfying the access policy $\widehat{AP}$, denoted as $\widehat{\mathcal{L}} \sqsubseteq \widehat{AP}$, will be calculated by $\mathcal{T}_R(\widehat{\mathcal{L}})$ recursively as follows: Suppose that $\mathcal{T}_a$ denotes the subtree of $\mathcal{T}$ rooted at node a. If a is a non-leaf node, we evaluate $\mathcal{T}_b(\widehat{\mathcal{L}})$ for all children b of node a. $\mathcal{T}_a(\widehat{\mathcal{L}})$ returns 1 if and only if at least k_a children return 1. If a is a leaf node, then $\mathcal{T}_a(\widehat{\mathcal{L}})$ returns 1 if and only if the corresponding attribute is in $\widehat{\mathcal{L}}$.

Tree implementation. To share the secret σ in access tree $\mathcal{T}$, a random polynomial q_R of degree $k_R - 1$ is chosen for $q_R(0) = \sigma$. The rest of the points in q_R are randomly chosen. For each node $a \in \mathcal{T}$, a random polynomial q_a of degree $k_a - 1$ is chosen for $q_a(0) = q_{parent(a)}(index(a))$. The rest of the points in q_a are chosen randomly. To recover the secret σ, the users with sufficient secret shares can perform Lagrange interpolation recursively. For example, Fig. 3-(b) shows the tree implementation process.

3.2. Positive-Negative Depth-First Coding

From the attribute set $\mathcal{A} = \{A_1, \ldots, A_m\}$, we build an attribute hierarchy $\widehat{\mathcal{A}}$ of L levels. In $\widehat{\mathcal{A}}$, an attribute at a higher level is a generalization of the attributes at lower levels. We associate each node with two hierarchical codes, the positive depth-first code (Pcode) and the negative depth-first code (Ncode), by running the PNDF coding algorithm (Alg. 1).



(a) Positive depth-first coding (b) Negative depth-first coding

Figure 4: Sample PNDF coding.

For coding each node, we need two stacks, $PcodeStack$ and $NcodeStack$. Here, the $push(a)$ function will be invoked to push node a onto a stack, the $top()$ function will be invoked to get the top element from a stack, the $empty()$ function will be invoked to determine whether a stack is empty, and the $pop()$ function will be invoked to pop the top element from a stack. First, we push the root node R onto $PcodeStack$ and $NcodeStack$. Line 9 means that the right child of a will be pushed onto $PcodeStack$, and Line 11 means that the left child of a will be pushed onto $PcodeStack$. In contrast, Lines 16 to 19 mean that the children of a will be pushed onto $NcodeStack$ in order from left to right. Therefore, for node a, the left-hand subtree's Pcodes will be larger than those of the right-hand one, and the right-hand subtree's Ncodes will be larger than those of the left-hand one.

Let us take the attribute tree shown in Fig. 1-(c) as an example. The PNDF coding is shown in Fig. 4. Let $Pcode_i$ and $Ncode_i$ denote the Pcode and Ncode of node i, respectively. The PNDF coding has the property that $Pcode_i > Pcode_j$ and $Ncode_i > Ncode_j$ if i is the descendant node of j. For example, the $Pcode$ and $Ncode$ of attribute *Surgery* are 2 and 5, respectively; the $Pcode$ and $Ncode$ of attribute *Cardiac Surgery* are 3 and 7, respectively; and the $Pcode$ and $Ncode$ of attribute *Respiratory Medicine* are 6 and 4, respectively. *Cardiac Surgery* is the descendant of *Surgery*, having both $Pcode$ and $Ncode$ greater than those of *Surgery*. *Respiratory Medicine* is not the descendant of *Surgery*, and its $Ncode$ is less than that of *Surgery*.

3.3. Definition of HCBE Scheme

Suppose that the number of nodes in the attribute hierarchy is m. In HCBE, the hierarchical codes are denoted as a set of discrete values $U_m = \{(Pcode_1, Ncode_1), \ldots, (Pcode_k, Ncode_k), \ldots, (Pcode_m, Ncode_m)\}$, with to-

Algorithm 1 PNDF Coding

1: *stack* $PcodeStack$, $NcodeStack$;
2: $PcodeStack.push(R)$, $NcodeStack.push(R)$;
3: $i = 0$, $j = 0$;
4: **while** (!$PcodeStack.empty()$) **do**
5: $a = PcodeStack.top()$;
6: $a \rightarrow Pcode = i++$;
7: $PcodeStack.pop()$;
8: **if** $a \rightarrow rchild$ **then**
9: $PcodeStack.push(a \rightarrow rchild)$
10: **if** $a \rightarrow lchild$ **then**
11: $PcodeStack.push(a \rightarrow lchild)$
12: **while** (!$NcodeStack.empty()$) **do**
13: $a = NcodeStack.top()$;
14: $a \rightarrow Ncode = j++$;
15: $NcodeStack.pop()$;
16: **if** $a \rightarrow lchild$ **then**
17: $NcodeStack.push(a \rightarrow lchild)$
18: **if** $a \rightarrow rchild$ **then**
19: $NcodeStack.push(a \rightarrow rchild)$

tal ordering $0 \leq Pcode_1 \leq Pcode_2 \leq \ldots \leq Pcode_m \leq Z_m$ and $0 \leq Ncode_1 \leq Ncode_2 \leq \ldots \leq Ncode_m \leq Z_m$, where Z_m is the maximal integer.

We apply BDF to establish the attribute hierarchy. Let $\mathbb{G}_{n'}$ be a multiplicative group of RSA-type composite order $n' = p'q'$, where p' and q' are two large primes. First, we choose random generators φ_1 and φ_2 in $\mathbb{G}_{n'}$ and random numbers θ_1 and θ_2 in $\mathbb{Z}^*_{n'}$, where the orders of θ_1 and θ_2 are sufficiently large in $\mathbb{Z}^*_{n'}$. Next, we define mapping functions $\psi_1(.)$ and $\psi_2(.)$ from an integer set $U_m = \{(Pcode_1, Ncode_1), \ldots, (Pcode_k, Ncode_k), \ldots, (Pcode_m, Ncode_m)\}$ into $V_m = \{(v_{Pcode_1}, v_{Ncode_1}), \ldots, (v_{Pcode_k}, v_{Ncode_k}), \ldots, (v_{Pcode_m}, v_{Ncode_m})\}$:

$$\begin{aligned} v_{Pcode_k} &= \varphi_1^{\theta_1^{Z_m - Pcode_k}} \\ v_{Ncode_k} &= \varphi_2^{\theta_2^{Z_m - Ncode_k}} \end{aligned} . \tag{3}$$

According to the definitions of $\psi_1(.)$ and $\psi_2(.)$, it is easy to define BDFs

Table 1: Summary of Notations

Notation	Description
$PK_{\widehat{\mathcal{A}}}, MK$	System public key, master key
$\mathcal{L}$,$\mathcal{L}'$	The access privilege assigned to the user's certificate
$SK_{\widehat{\mathcal{L}}}, SK_{\widehat{\mathcal{L}'}}$	The user's private key, the derivation privacy key
$\widehat{AP}, \widehat{AP'}$	The previous/updated access policy
$\mathcal{T}, \mathcal{T}'$	The previous/updated access policy tree
$\widehat{\mathcal{H}}_{\mathcal{P}}, \widetilde{\mathcal{H}}_{\mathcal{P}}$	The previous/updated ciphertext headers
UK	Update key for updating $\widehat{\mathcal{H}}_{\mathcal{P}}$ to $\widetilde{\mathcal{H}}_{\mathcal{P}}$
ek	The session key
Δ	A set of secret shares for nodes in access tree $\mathcal{T}$

$f_1(.)$ and $f_2(.)$ as follows:

$$\begin{aligned} v_{Pcode_l} &\leftarrow f_1(v_{Pcode_k}) = (v_{Pcode_k})^{\theta_1{}^{Pcode_k - Pcode_l}}, \quad Pcode_k \geq Pcode_l \\ v_{Ncode_l} &\leftarrow f_2(v_{Ncode_k}) = (v_{Ncode_k})^{\theta_2{}^{Ncode_k - Ncode_l}}, \quad Ncode_k \geq Ncode_l. \end{aligned} \tag{4}$$

The most commonly used notations are shown in Table 1. As shown in Fig. 5, the HCBE scheme consists of the following algorithms:

- $Setup(1^\kappa, \widehat{\mathcal{A}}) \rightarrow (MK, PK_{\widehat{\mathcal{A}}})$: The data owner takes a security parameter κ and the attribute hierarchy $\widehat{\mathcal{A}}$ as inputs, and outputs the master key MK and the system public key $PK_{\widehat{\mathcal{A}}}$.

- $GenKey(MK, \mathcal{U}, \widehat{\mathcal{L}}) \rightarrow SK_{\widehat{\mathcal{L}}}$: The data owner utilizes her master key MK to generate a private key $SK_{\widehat{\mathcal{L}}}$ on an access privilege $\widehat{\mathcal{L}}$ for user $\mathcal{U}$, wherein each attribute $A_k \in \widehat{\mathcal{L}}$, denoted as $A_k(t_a, t_b, Pcode_k, Ncode_k)$, is associated with the authorization time $[t_a, t_b]$ and hierarchy codes $\{Pcode_k, Ncode_k\}$.

- $Encrypt(PK_{\widehat{\mathcal{A}}}, \widehat{AP}) \rightarrow (\widehat{\mathcal{H}_{\mathcal{P}}}, ek)$: The data owner takes the public key $PK_{\widehat{\mathcal{A}}}$ and an access policy $\widehat{AP}$ as inputs to generate a session key ek and a ciphertext header $\widehat{\mathcal{H}_{\mathcal{P}}}$, wherein each attribute $A_l \in \widehat{AP}$, denoted as $A_l(t_i, t_j, Pcode_l, Ncode_l)$, is associated with the time condition $[t_i, t_j]$ and hierarchy codes $\{Pcode_l, Ncode_l\}$.

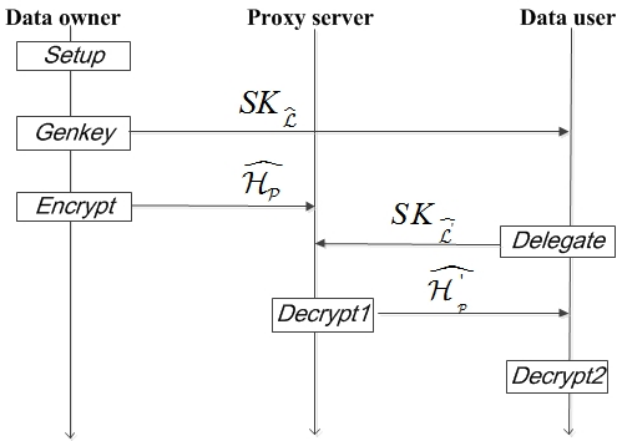


Figure 5: Working process of the HCBE scheme.

- $Delegate(SK_{\widehat{\mathcal{L}}}, \widehat{\mathcal{L}'}) \rightarrow SK_{\widehat{\mathcal{L}'}}$: The data user takes the private key $SK_{\widehat{\mathcal{L}}}$ and an access privilege $\widehat{\mathcal{L}'}$ as inputs to generate a derived private key $SK_{\widehat{\mathcal{L}'}}$ for the proxy server if $\widehat{\mathcal{L}'} \preceq \widehat{\mathcal{L}}$ [3].

- $Decrypt1(SK_{\widehat{\mathcal{L}'}}, \widehat{\mathcal{H}_{\mathcal{P}}}) \rightarrow \widehat{\mathcal{H}'_{\mathcal{P}}}$: The proxy server takes the derived private key $SK_{\widehat{\mathcal{L}'}}$ and a ciphertext header $\widehat{\mathcal{H}_{\mathcal{P}}}$ as inputs, and outputs a new ciphertext header $\widehat{\mathcal{H}'_{\mathcal{P}}}$ if $\widehat{\mathcal{L}'}$ satisfies $\widehat{AP}$.

- $Decrypt2(SK_{\widehat{\mathcal{L}}}, \widehat{\mathcal{H}'_{\mathcal{P}}}) \rightarrow ek$: The data user takes the private key $SK_{\widehat{\mathcal{L}}}$ and the new ciphertext header $\widehat{\mathcal{H}'_{\mathcal{P}}}$ as inputs, and outputs a session key ek, which can be used to decrypt the stored data.

4. Construction of HCBE Scheme

$Setup(1^{\kappa}, \widehat{\mathcal{A}}) \rightarrow (MK, PK_{\widehat{\mathcal{A}}})$: Given a bilinear map system $S_N = (N = pq, \mathbb{G}, \mathbb{G}_{\mathrm{T}}, e)$, where $\mathbb{G}, \mathbb{G}_{\mathrm{T}}$ are cyclic groups of composite order $n = sn'$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathrm{T}}$, this algorithm first chooses the random generators $\omega \in \mathbb{G}$, $g \in \mathbb{G}_s$, and $\varphi, \overline{\varphi}, \varphi_1, \varphi_2 \in \mathbb{G}_{n'}$, where $\mathbb{G}_s$ and $\mathbb{G}_{n'}$ are two subgroups of $\mathbb{G}$.

[3] Let S and S' denote the set of attributes in $\widehat{\mathcal{L}}$ and $\widehat{\mathcal{L}'}$, respectively. $\widehat{\mathcal{L}'} \preceq \widehat{\mathcal{L}}$ if and only if $S' \subseteq S$, and for each attribute $A_k(t_a, t_b, Pcode_k, Ncode_k) \in \widehat{\mathcal{L}}$ and $A_l(t_i, t_j, Pcode_l, Ncode_l) \in \widehat{\mathcal{L}'}$, $t_a \leq t_i$, $t_b \geq t_j$, $Pcode_k \geq Pcode_l$, and $Ncode_k \geq Ncode_l$.

Thus, we have $e(g,\varphi) = e(g,\overline{\varphi}) = e(g,\varphi_1,) = e(g,\varphi_2) = 1$, but $e(g,\omega) \neq 1$. Then, it chooses four random numbers $\lambda, \mu, \theta_1, \theta_2 \in \mathbb{Z}_n^*$ and employs a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$, mapping the root attribute, R, described as a binary string, to a random group element. Next, it chooses two random exponents $\alpha, \beta \in \mathbb{Z}_n^*$ and sets $h = \omega^\beta$, $\eta = g^{1/\beta}$, and $\zeta = e(g,\omega)^\alpha$. The master key is set as $MK = (g^\alpha, \beta, p, q, n')$, and the public key is set as

$$PK_{\widehat{\mathcal{A}}} = (S_N, \omega, g, \varphi, \overline{\varphi}, \varphi_1, \varphi_2, h, \eta, \zeta, \lambda, \mu, \theta_1, \theta_2, H). \tag{5}$$

GenKey$(MK, \mathcal{U}, \widehat{\mathcal{L}}) \rightarrow SK_{\widehat{\mathcal{L}}}$: Given a user $\mathcal{U}$ with license $\widehat{\mathcal{L}}$, this algorithm chooses two random numbers $\tau_{\mathcal{U}}, r \in \mathbb{Z}$, and then for each attribute $A_k(t_a, t_b, Pcode_k, Ncode_k) \in \widehat{\mathcal{L}}$, it calculates:

$$\begin{aligned} D_{A_k} &= (D_t, D'_{t_a}, \overline{D'}_{t_b}, D''_t, D_{Pcode_k}, D_{Ncode_k}) \\ &= (g^{\tau_{\mathcal{U}}} {H_{A_k}}^r, (v_{t_a})^r, (\overline{v}_{t_b})^r, \omega^r, (v_{Pcode_k})^r, (v_{Ncode_k})^r), \end{aligned} \tag{6}$$

where $H_{A_k} = H(R) \cdot v_{Pcode_k} \cdot v_{Ncode_k}$, $v_{t_a} = \varphi^{\lambda^{t_a}}$, $\overline{v}_{t_b} = \overline{\varphi}^{\mu^{Z-t_b}}$, $v_{Pcode_k} = \varphi_1^{\theta_1^{Zm-Pcode_k}}$, and $v_{Ncode_k} = \varphi_2^{\theta_2^{Zm-Ncode_k}}$. Then, the private key of $\mathcal{U}$ is set as

$$SK_{\widehat{\mathcal{L}}} = (D = g^{(\alpha+\tau_{\mathcal{U}})/\beta}, \{D_{A_k}\}_{A_k \in \widehat{\mathcal{L}}}). \tag{7}$$

Encrypt$(PK_{\widehat{\mathcal{A}}}, \widehat{AP}) \rightarrow (\widehat{\mathcal{H}_{\mathcal{P}}}, ek)$: Given an access policy tree $\mathcal{T}$ over access policy $\widehat{AP}$, the ciphertext header $\widehat{\mathcal{H}_{\mathcal{P}}}$ can be calculated using

$$\widehat{H_P} = (\mathcal{T}, C = h^\sigma, \{C_l = (C_{1,l}, C_{2,l}, C_{3,l}, C_{4,l})\}_{A_l \in \mathcal{T}}). \tag{8}$$

Here, each component of C_l is set as follows:

$$\begin{aligned} C_{1,l} &= (\bar{E}_{t_i}, {E'}_{t_i}) = (\bar{v}_{t_i}\omega)^x, H_{A_l}^x), \\ C_{2,l} &= (E_{t_j}, {E'}_{t_j}) = ((v_{t_j}\omega)^y, H_{A_l}^y), \\ C_{3,l} &= (E_{Pcode_l}, {E'}_{Pcode_l}) = ((v_{Pcode_l} \cdot \omega)^{z_1}, H_{A_l}^{z_1}), \\ C_{4,l} &= (E_{Ncode_l}, {E'}_{Ncode_l}) = ((v_{Ncode_l} \cdot \omega)^{z_2}, H_{A_l}^{z_2}), \end{aligned} \tag{9}$$

where $H_{A_l} = H(R) \cdot v_{Pcode_l} \cdot v_{Ncode_l}$. The session key ek is set as $\zeta^\sigma = e(g^\alpha, \omega)^\sigma$, where σ is a main secret in $\mathbb{Z}_n$ for tree $\mathcal{T}$, and $\triangle_\sigma(A_l) = x + y + z_1 + z_2$ is the secret share of σ in the tree $\mathcal{T}$ for an attribute A_l (see Ref. [2]). In order to implement dynamic policy updating for the encrypted data, the data owner should preserve each secret share $\Delta_\sigma(a)$ of σ for each node a in $\mathcal{T}$.

Delegate$(SK_{\widehat{\mathcal{L}}}, \widehat{\mathcal{L}'}) \rightarrow SK_{\widehat{\mathcal{L}'}}$: Given a specified access privilege $\widehat{\mathcal{L}'}$ and the private key $SK_{\widehat{\mathcal{L}}} = (D, \{D_{A_k}\}_{A_k \in \widehat{\mathcal{L}}})$, this algorithm checks for each attribute

$A_l(t_i, t_j, Pcode_l, Ncode_l) \in \widehat{\mathcal{L}'}$ to ascertain whether A_l is a generalized attribute of A_k, $t_a \leq t_j$, and $t_b \geq t_i$. If so, this algorithm uses Eq. 2 and Eq. 4 to compute

$$\begin{aligned}
D'_t &\leftarrow g^{\tau_{\mathcal{U}}} {H_{A_k}}^r \cdot \tfrac{f_1(D_{Pcode_k}) \cdot f_2(D_{Ncode_k})}{D_{Pcode_k} \cdot D_{Ncode_k}} \\
&= g^{\tau_{\mathcal{U}}} (H(R) \cdot v_{Pcode_k} \cdot v_{Ncode_k})^r \cdot \tfrac{f_1((v_{Pcode_k})^r) \cdot f_2((v_{Ncode_k})^r)}{(v_{Pcode_k})^r \cdot (v_{Pcode_k})^r} \\
&= g^{\tau_{\mathcal{U}}} H(R)^r \cdot {v_{Pcode_l}}^r \cdot {v_{Ncode_l}}^r = g^{\tau_{\mathcal{U}}} H^r_{A_l} \\
D'_{t_j} &\leftarrow f(D'_{t_a}) \cdot D''_t = f((v_{t_a})^r) \cdot \omega^r = (v_{t_j})^r \cdot \omega^r, \\
\overline{D}'_{t_i} &\leftarrow \overline{f}(\overline{D}'_{t_b}) \cdot D''_t = \overline{f}((\overline{v}_{t_b})^r) \cdot \omega^r = (\overline{v}_{t_i})^r \cdot \omega^r, \\
D'_{Pcode_l} &\leftarrow f_1(D_{Pcode_k}) \cdot D''_t = f_1((v_{Pcode_k})^r) \cdot \omega^r = (v_{Pcode_l})^r \cdot \omega^r, \\
D'_{Ncode_l} &\leftarrow f_2(D_{Ncode_k}) \cdot D''_t = f_2((v_{Ncode_k})^r) \cdot \omega^r = (v_{Ncode_l})^r \cdot \omega^r,
\end{aligned} \tag{10}$$

where

$$\begin{aligned}
&f((v_{t_a})^r) = (\varphi^{r\lambda^{t_a}})^{\lambda^{t_j - t_a}} = \varphi^{r\lambda^{t_j}} = (v_{t_j})^r, \\
&\overline{f}((\overline{v}_{t_b})^r) = (\overline{\varphi}^{r\mu^{Z-t_b}})^{\mu^{t_b - t_i}} = \overline{\varphi}^{r\mu^{Z-t_i}} = (\overline{v}_{t_i})^r, \\
&f_1((v_{Pcode_k})^r) = (\varphi_1^{r\theta_1^{Z_m - Pcode_k}})^{\theta_1^{Pcode_k - Pcode_l}} = \varphi_1^{r\theta_1^{Z_m - Pcode_l}} = (v_{Pcode_l})^r, \\
&f_2((v_{Ncode_k})^r) = (\varphi_2^{r\theta_2^{Z_m - Ncode_k}})^{\theta_2^{Ncode_k - Ncode_l}} = \varphi_2^{r\theta_2^{Z_m - Ncode_l}} = (v_{Ncode_l})^r.
\end{aligned} \tag{11}$$

Next, it chooses a random $\delta \in \mathbb{Z}$ and computes

$$\begin{aligned}
&\widetilde{D}_t = D'_t \cdot (gH_{A_l})^\delta = g^{\tau_{\mathcal{U}}} {H_{A_l}}^r \cdot (gH_{A_l})^\delta = g^{\tau_{\mathcal{U}} + \delta} H^{r+\delta}_{A_l} = g^{\tau'_k} H^{r'}_{A_l}, \\
&\widetilde{D}'_{t_j} = D'_{t_j} \cdot (v_{t_j}\omega)^\delta = (v_{t_j}\omega)^{r+\delta} = (v_{t_j}\omega)^{r'}, \\
&\widetilde{\overline{D}'}_{t_i} = \overline{D}'_{t_i} \cdot (\overline{v}_{t_i}\omega)^\delta = (\overline{v}_{t_i}\omega)^{r+\delta} = (\overline{v}_{t_i}\omega)^{r'}, \\
&\widetilde{D}'_{Pcode_l} = D'_{Pcode_l} \cdot (v_{Pcode_l}\omega)^\delta = (v_{Pcode_l}\omega)^{r+\delta} = (v_{Pcode_l}\omega)^{r'}, \\
&\widetilde{D}'_{Ncode_l} = D'_{Ncode_l} \cdot (v_{Ncode_l}\omega)^\delta = (v_{Ncode_l}\omega)^{r+\delta} = (v_{Ncode_l}\omega)^{r'},
\end{aligned} \tag{12}$$

where $H_{A_l} = H(R) \cdot v_{Pcode_l} \cdot v_{Ncode_l}$, $\tau'_k = \tau_{\mathcal{U}} + \delta$, and $r' = r + \delta$. Finally, the derivation privacy key is set as $SK_{\widehat{\mathcal{L}'}} = \{\widetilde{D}_t, \widetilde{D}'_{t_j}, \widetilde{\overline{D}'}_{t_i}, \widetilde{D}'_{Pcode_l}, \widetilde{D}'_{Ncode_l}\}_{A_l \in \mathcal{L}'}$.

$Decrypt1(SK_{\widehat{\mathcal{L}'}}, \widehat{\mathcal{H}_{\mathcal{P}}}) \rightarrow \widehat{\mathcal{H}'_{\mathcal{P}}}$: Given the private key $SK_{\widehat{\mathcal{L}'}}$ and a ciphertext header $\widehat{\mathcal{H}_{\mathcal{P}}}$, we check whether each attribute $A_l(t_i, t_j, Pcode_l, Ncode_l) \in \widehat{\mathcal{L}'}$ is consistent with $A_l(t_i, t_j, Pcode_l, Ncode_l) \in \widehat{AP}$. If true, the secret share $\triangle_\sigma(A_l)$ of σ over $\mathbb{G}_{\mathrm{T}}$ is reconstructed by using

$$\begin{aligned}
F_1 &\leftarrow \tfrac{e(\widetilde{D}_t, \overline{E}_{t_i})}{e(\widetilde{\overline{D}'}_{t_i}, E'_{t_i})} = \tfrac{e(g^{\tau'_k} H^{r'}_{A_l}, (\overline{v}_{t_i}\omega)^x)}{e((\overline{v}_{t_i}\omega)^{r'}, H^x_{A_l})} \\
&= e(g^{\tau'_k}, \overline{v}^x_{t_i}) \cdot e(g^{\tau'_k}, \omega^x) = e(g^{\tau'_k}, \omega)^x
\end{aligned} \tag{13}$$

$$F_2 \leftarrow \frac{e(\widetilde{D}_t, \overline{E}_{t_j})}{e(\widetilde{D}'_{t_j}, E'_{t_j})} = \frac{e(g^{\tau'_k} H_{A_l}^{r'}, (v_{t_j}\omega)^y)}{e((v_{t_j}\omega)^{r'}, H_{A_l}^y)} \\ = e(g^{\tau'_k}, v_{t_j}^y) \cdot e(g^{\tau'_k}, \omega^y) = e(g^{\tau'_k}, \omega)^y \tag{14}$$

$$F_3 \leftarrow \frac{e(\widetilde{D}_t, E_{Pcode_l})}{e(\widetilde{D}'_{Pcode_l}, E'_{Pcode_l})} = \frac{e(g^{\tau'_k} H_{A_l}^{r'}, (v_{Pcode_l}\omega)^{z_1})}{e((v_{Pcode_l}\omega)^{r'}, H_{A_l}^{z_1})} \\ = \frac{e(g^{\tau'_k}, (v_{Pcode_l}\omega)^{z_1}) \cdot e(H_{A_l}^{r'}, (v_{Pcode_l}\omega)^{z_1})}{e((v_{Pcode_l}\omega)^{r'}, H_{A_l}^{z_1})} \\ = e(g^{\tau'_k}, v_{Pcode_l}^{z_1}) \cdot e(g^{\tau'_k}, \omega^{z_1}) = e(g^{\tau'_k}, \omega)^{z_1} \tag{15}$$

$$F_4 \leftarrow \frac{e(\widetilde{D}_t, E_{Ncode_l})}{e(\widetilde{D}'_{Ncode_l}, E'_{Ncode_l})} = \frac{e(g^{\tau'_k} H_{A_l}^{r'}, (v_{Ncode_l}\omega)^{z_2})}{e((v_{Ncode_l}\omega)^{r'}, H_{A_l}^{z_2})} \\ = \frac{e(g^{\tau'_k}, (v_{Ncode_l}\omega)^{z_2}) \cdot e(H_{A_l}^{r'}, (v_{Ncode_l}\omega)^{z_2})}{e((v_{Ncode_l}\omega)^{r'}, H_{A_l}^{z_2})} \\ = e(g^{\tau'_k}, v_{Ncode_l}^{z_2}) \cdot e(g^{\tau'_k}, \omega^{z_2}) = e(g^{\tau'_k}, \omega)^{z_2} \tag{16}$$

$$F_t = F_1 \cdot F_2 \cdot F_3 \cdot F_4 = e(g^{\tau'_k}, \omega)^{\triangle_\sigma(A_l)}, \tag{17}$$

in which $H_{A_l} = H(R) \cdot v_{Pcode_l} \cdot v_{Ncode_l}$. We have $e(g^{\tau'_k}, \overline{v}_{t_i}^x) = e(g^{\tau'_k}, v_{t_j}^y) = e(g^{\tau'_k}, v_{Pcode_l}^{z_1}) = e(g^{\tau'_k}, v_{Ncode_l}^{z_2}) = 1$ since $g^{\tau'_k} \in \mathbb{G}_s$ and $v_{t_j}^x, \overline{v}_{t_i}^y, v_{Pcode_l}^{z_1}, v_{Ncode_l}^{z_2} \in \mathbb{G}_{n'}$. Next, the value $C_2 = e(g^{\tau'_k}, \omega)^\sigma$ is computed from $\{e(g^{\tau'_k}, \omega)^{\triangle_\sigma(A_l)}\}_{A_l \in \mathcal{T}}$ by using the aggregation algorithm (see Ref. [2]). Finally, the new ciphertext header $\widehat{\mathcal{H}'_\mathcal{P}} = (C = h^\sigma, C_2)$ is returned.

Decrypt2$(SK_{\widehat{\mathcal{L}}}, \widehat{\mathcal{H}'_\mathcal{P}}) \rightarrow ek$: After receiving $\widehat{\mathcal{H}'_\mathcal{P}} = (C, C_2) = (\omega^{\beta\sigma}, e(g^{\tau'_k}, \omega)^\sigma)$, the data user uses the secret δ to compute

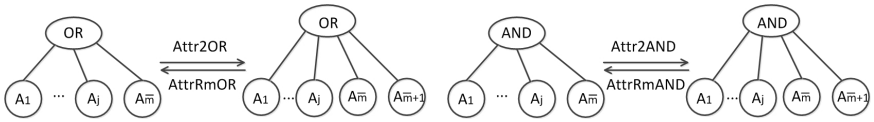
$$D' = D \cdot \eta^\delta = g^{(\alpha+\tau_\mathcal{U})/\beta} g^{\delta/\beta} = g^{(\alpha+\tau_\mathcal{U}+\delta)/\beta} = g^{(\alpha+\tau'_k)/\beta}. \tag{18}$$

Next, the session key is computed by

$$ek = \frac{e(D', C)}{C_2} = \frac{e(g^{(\alpha+\tau'_k)/\beta}, (\omega^\beta)^\sigma)}{e(g^{\tau'_k}, \omega)^\sigma} = e(g^\alpha, \omega)^\sigma. \tag{19}$$

5. DPU Scheme

To reduce the communication and computation costs incurred by the data owner, the DPU scheme utilizes the PRE technique to allow the data owner to delegate the policy updating operations to the CSP. Inspired by the



(a) Attr2OR and AttrRmOR (b) Attr2AND and AttrRmAND

Figure 6: Operations of policy updating.

work in [27], we consider four basic operations involved in policy updating as shown in Fig. 6, where the non-leaf nodes are AND and OR gates, and the leaf nodes correspond to attributes. Specifically, $Attr2OR$ denotes the adding of an attribute to an OR gate, $Attr2AND$ denotes the adding of an attribute to an AND gate, $AttrRmOR$ denotes the removing of an attribute from an OR gate, and $AttrRmAND$ denotes the removing of an attribute from an AND gate.

Given access tree $\mathcal{T}$ over access policy $\widehat{AP}$, the data owner needs to preserve $\Delta = \{\Delta_\sigma(x)\}_{x\in\mathcal{T}}$ while running the *Encrypt* algorithm of the HCBE scheme, where $\Delta_\sigma(x)$ is the share of secret σ for node x in $\mathcal{T}$. Suppose that $A_1, \ldots, A_{\overline{m}}$ are the attributes under the AND/OR gate node. We substitute notation $\Delta_\sigma(A_i)$ for notation $\Delta_\sigma(x)$, where x is a leaf node representing attribute A_i with $i \in [1, \overline{m}]$.

Let $\widehat{\mathcal{H}_\mathcal{P}} = (\mathcal{T}, C, \{C_i\}_{A_i\in\mathcal{T}})$ and $\widetilde{\mathcal{H}_\mathcal{P}} = (\mathcal{T}', C, \{C_i\}_{A_i\in\mathcal{T}'})$ denote the previous and new ciphertext headers for session key ek, respectively. The DPU scheme consists mainly of the following three algorithms:

- $UKGen(\{C_i\}_{A_i\in\mathcal{T}}, \Delta) \rightarrow (UK)$: The data owner takes a part of ciphertext header $\{C_i\}_{A_i\in\mathcal{T}}$ and secret shares Δ as inputs, and outputs an update key UK.

- $CTGen(PK_{\widehat{\mathcal{A}}}, \Delta_\sigma(x)) \rightarrow (C_{\overline{m}+1})$: To add an attribute $A_{\overline{m}+1}$ under node x, the data owner takes system public key $PK_{\widehat{\mathcal{A}}}$ and x's share $\Delta_\sigma(x)$ as inputs, and outputs the new ciphertext $C_{\overline{m}+1}$ for attribute $A_{\overline{m}+1}$.

- $CTUpdate(\{C_i\}_{A_i\in\mathcal{T}}, UK) \rightarrow \{C_i'\}_{A_i\in\mathcal{T}'}$: The CSP takes a part of ciphertext header $\{C_i\}_{A_i\in\mathcal{T}}$ and the update key UK as inputs, and outputs a new ciphertext header $\{C_i'\}_{A_i\in\mathcal{T}'}$.

In general, the data owner first generates an update key UK by running the $UKGen$ algorithm and then sends UK to the CSP. Once UK is received

from the data owner, the CSP runs the $CTUpdate$ algorithm to update the previous access policy $\widehat{AP}$ to the new access policy $\widehat{AP'}$. In addition, for the $Attr2OR$ and $Attr2AND$ operations, the data owner needs to run the $CTGen$ algorithm to generate a new ciphertext $C_{\overline{m}+1}$ for the newly added attribute $A_{\overline{m}+1}$ under the AND/OR gate, and then sends $C_{\overline{m}+1}$ to the CSP, which will incorporate $C_{\overline{m}+1}$ into the new ciphertext header $\widetilde{\mathcal{H}_{\mathcal{P}}}$.

Here, we provide a generalized version (G-DPU) and an efficient version (E-DPU) for the DPU scheme. G-DPU is a generalized policy update scheme that transforms the updating of the AND/OR gating to the updating of a threshold gate from a (t, n)-gate to a (t', n')-gate. However, G-DPU needs to regenerate new secret shares for all attributes under the updating gate while adding or deleting an attribute node. Therefore, the updating cost will grow linearly with the number of attributes under the gate. In order to remedy this defect, we provide E-GPU, which regenerates only new secret shares for one attribute under the updating gate and the newly added attribute. Therefore, E-GPU incurs only a constant cost for updating an attribute.

5.1. The Generalized DPU

In this subsection, we provide G-DPU, which transforms the operations to the updating of the threshold gate, as shown in Fig. 6. Let node x denote the AND/OR gate being updated, where $A_1, \ldots, A_{\overline{m}}$ are the original attributes under node x, and let $\Delta_\sigma(x)$ and $\Delta_\sigma(A_j)$ denote the share associated with node x and attribute A_j for $j \in [1, \overline{m}]$, respectively. The basic policy-updating operations include adding a new attribute $A_{\overline{m}+1}$ under node x and removing attribute A_j for $j \in [1, \overline{m}]$ from node x. In our scheme, $\Delta_\sigma(x)$ will not be changed in policy updating in order to save the computation cost for data owners. Given a fixed $\Delta_\sigma(x)$, the data owner first runs the secret-sharing algorithm, as shown in Alg. 2, to regenerate shares for $\overline{m}'$ children of node x, denoted as $\Delta'_\sigma(A_1), \ldots, \Delta'_\sigma(A_{\overline{m}'})$.

1)Attr2OR: This operation is transformed to the updating of a $(1, \overline{m})$ gate to a $(1, \overline{m}')$ gate, where $\overline{m}' = \overline{m}+1$. After running Alg. 2, we have $\Delta'_\sigma(A_1) = \ldots = \Delta'_\sigma(A_{\overline{m}}) = \Delta'_\sigma(A_{\overline{m}+1}) = \Delta_\sigma(x)$. Suppose that the previous ciphertext header $\widehat{\mathcal{H}_{\mathcal{P}}} = (\mathcal{T}, C, \{C_i\}_{A_i \in \mathcal{T}})$. The data owner only needs to construct the new ciphertext component $C_{\overline{m}+1} = (C_{1,\overline{m}+1}, C_{2,\overline{m}+1}, C_{3,\overline{m}+1}, C_{4,\overline{m}+1})$ for $A_{\overline{m}+1}$

Algorithm 2 Secret-Sharing Scheme

Input: x //Node x from an access tree $\mathcal{T}$
Input: $\Delta_\sigma(x)$ // The secret to be shared
Output: A set of shares $\Delta'_\sigma(A_1), \ldots, \Delta'_\sigma(A_{\overline{m}'})$

1: Let q_x be a polynomial for node x;
2: Set $q_x(0) := \Delta_\sigma(x)$;
3: Set degree $d_x := k_x - 1$; //k_x is the threshold value of the node x
4: Let rest of points in q_x be randomly chosen;
5: **for** the ith child A_i of node x, $i \in [1, \overline{m}']$ **do**
6: Set $\Delta_\sigma(A_i) = q_x(i)$; //$i$ is the index of attribute A_i

as follows:

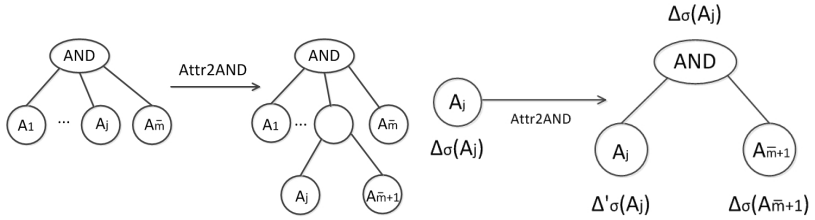
$$\begin{aligned} C_{1,\overline{m}+1} &= ((\bar{v}_{t_i}\omega)^{x_{\overline{m}+1}}, H_{A_{\overline{m}+1}}^{x_{\overline{m}+1}}), \\ C_{2,\overline{m}+1} &= ((v_{t_k}\omega)^{y_{\overline{m}+1}}, H_{A_{\overline{m}+1}}^{y_{\overline{m}+1}}), \\ C_{3,\overline{m}+1} &= ((v_{Pcode_{\overline{m}+1}} \cdot \omega)^{z_{1,\overline{m}+1}}, H_{A_{\overline{m}+1}}^{z_{1,\overline{m}+1}}), \\ C_{4,\overline{m}+1} &= ((v_{Ncode_{\overline{m}+1}} \cdot \omega)^{z_{2,\overline{m}+1}}, H_{A_{\overline{m}+1}}^{z_{2,\overline{m}+1}}), \end{aligned} \tag{20}$$

where $x_{\overline{m}+1}, y_{\overline{m}+1}, z_{1,\overline{m}+1}, z_{2,\overline{m}+1} \in \mathbb{Z}$ such that $x_{\overline{m}+1} + y_{\overline{m}+1} + z_{1,\overline{m}+1} + z_{2,\overline{m}+1} = \Delta_\sigma(x)$.

2)AttrRmOR: This operation is transformed to the updating of a $(1, \overline{m})$ gate to a $(1, \overline{m}')$ gate, where $\overline{m}' = \overline{m} - 1$. From the output of Alg. 2, we know that $\Delta'_\sigma(A_1) = \ldots = \Delta'_\sigma(A_{\overline{m}-1}) = \Delta_\sigma(x)$. Therefore, in order to remove an attribute A_j from an OR gate, the data owner only needs to send a tuple $(AttrRmOR, \widehat{\mathcal{H}_\mathcal{P}}, j)$ to request the server to delete the corresponding ciphertext component C_j from the ciphertext header .

3) Attr2AND: This operation is transformed to the updating of a $(\overline{m}, \overline{m})$ gate to a $(\overline{m}', \overline{m}')$ gate, where $\overline{m}' = \overline{m} + 1$. The data owner first calls Alg. 2 to output a set of shares $\Delta'_\sigma(A_1), \ldots, \Delta'_\sigma(A_{\overline{m}+1})$. Then, for $j \in [1, \overline{m}]$, the data owner runs the $UKGen$ algorithm to construct the update key UK as follows:

$$\begin{aligned} UK = \{ UK_1 &= ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_j}^{x_j})^{\frac{\Delta'_\sigma(A_j) - \Delta_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ UK_2 &= ((v_{t_k}\omega)^{y_j}, H_{A_j}^{y_j})^{\frac{\Delta'_\sigma(A_j) - \Delta_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ UK_3 &= ((v_{Pcode_j} \cdot \omega)^{z_{1,j}}, H_{A_j}^{z_{1,j}})^{\frac{\Delta'_\sigma(A_j) - \Delta_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ UK_4 &= ((v_{Ncode_j} \cdot \omega)^{z_{2,j}}, H_{A_j}^{z_{2,j}})^{\frac{\Delta'_\sigma(A_j) - \Delta_\sigma(A_j)}{\Delta_\sigma(A_j)}} \}. \end{aligned} \tag{21}$$



(a) Attr2AND (b) New secret-share scheme

Figure 7: Converting an attribute to an AND gate.

Furthermore, the data owner needs to run the $CTGen$ algorithm to generate the new ciphertext $C_{\overline{m}+1}$ for the newly added attribute $A_{\overline{m}+1}$, where $C_{\overline{m}+1}$ is constructed in the same way as Eq. 20. The main difference is that $x_{\overline{m}+1}, y_{\overline{m}+1}, z_{1,\overline{m}+1}, z_{2,\overline{m}+1} \in \mathbb{Z}$ have to satisfy the requirement of $x_{\overline{m}+1} + y_{\overline{m}+1} + z_{1,\overline{m}+1} + z_{2,\overline{m}+1} = \Delta'_\sigma(A_{\overline{m}+1})$. Next, the data owner will send the tuple $(Attr2AND, C_{\overline{m}+1}, UK)$ to the CSP, which will update the ciphertext components $\{C_j\}_{j\in[1,\overline{m}]}$ with UK and add $C_{\overline{m}+1}$ to form the new ciphertext header $\widetilde{\mathcal{H}_\mathcal{P}}$.

Upon receiving the update key UK, for each attribute A_j with $j \in [1, \overline{m}]$ the CSP runs the $CTUpdate$ algorithm to update corresponding ciphertext component C_j to C'_j as follows:

$$\begin{aligned} C'_j = \{ & C'_{1,j} = C_{1,j} \cdot UK_1 = ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_j}^{x_j})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{2,j} = C_{2,j} \cdot UK_2 = ((v_{t_k}\omega)^{y_j}, H_{A_j}^{y_j})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{3,j} = C_{3,j} \cdot UK_3 = ((v_{Pcode_j} \cdot \omega)^{z_{1,j}}, H_{A_j}^{z_{1,j}})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{4,j} = C_{4,j} \cdot UK_4 = ((v_{Ncode_j} \cdot \omega)^{z_{2,j}}, H_{A_j}^{z_{2,j}})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}.\} \end{aligned} \tag{22}$$

4) AttrRmAND: This operation is transformed to the updating of a $(\overline{m}, \overline{m})$ gate to a $(\overline{m}', \overline{m}')$ gate, where $\overline{m}' = \overline{m} - 1$. The data owner first calls Alg. 2 to output a set of shares $\Delta'_\sigma(A_1), \ldots, \Delta'_\sigma(A_{\overline{m}-1})$. Then, for $j \in [1, \overline{m} - 1]$, the data owner runs the $UKGen$ algorithm to construct the update key UK with Eq. 21. Next, the data owner will send the tuple $(AttrRmAND, j, UK)$ to the CSP, which will remove C_j from the ciphertext header and run the $CTUpdate$ algorithm to form the new ciphertext header $\widetilde{\mathcal{H}_\mathcal{P}}$. Specifically, for $j \in [1, \overline{m} - 1]$, the updated ciphertext C'_j is constructed with Eq. 22.

5.2. The Efficient DPU

The main drawback of G-DPU is that the cost for updating an AND gate will grow linearly with the number of attributes under the gate. Therefore, we provide E-DPU, which incurs only a constant cost for updating an attribute.

As shown in Fig. 7(a), in terms of the $Attr2AND$ operation, the combination of the new attribute $A_{\overline{m}+1}$ and the attribute A_j in the new access policy plays the same role as the attribute A_j in the previous policy. Therefore, we can modify the previous ciphertext component C_j corresponding to A_j into a new version C'_j and construct the new ciphertext component for $A_{\overline{m}+1}$.

As shown in Fig. 7(b), we take $\Delta_\sigma(A_j)$ as the secret to be shared. To generate shares $\Delta'_\sigma(A_j)$ and $\Delta_\sigma(A_{\overline{m}+1})$ for attributes A_j and $A_{\overline{m}+1}$, the data owner randomly chooses $\varepsilon \in \mathbb{Z}$ such that $\Delta_\sigma(A_{\overline{m}+1}) = \Delta_\sigma(A_j) + 2\varepsilon$, and $\Delta'_\sigma(A_j) = \Delta_\sigma(A_j) + \varepsilon$. Then, she runs the $UKGen$ algorithm to generate the update key UK as follows:

$$\begin{aligned} UK = \{&UK_1 = ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_j}^{x_j})^{\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &UK_2 = ((v_{t_k}\omega)^{y_j}, H_{A_j}^{y_j})^{\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &UK_3 = ((v_{Pcode_j}\cdot\omega)^{z_{1,j}}, H_{A_j}^{z_{1,j}})^{\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &UK_4 = ((v_{Ncode_j}\cdot\omega)^{z_{2,j}}, H_{A_j}^{z_{2,j}})^{\frac{\varepsilon}{\Delta_\sigma(A_j)}}\}. \end{aligned} \tag{23}$$

In addition, the data owner runs the $CTGen$ algorithm to generate the new ciphertext component $C_{\overline{m}+1}$ as follows:

$$\begin{aligned} C_{\overline{m}+1} = \{&C_{1,\overline{m}+1} = ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_{\overline{m}+1}}^{x_j})^{1+\frac{2\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C_{2,\overline{m}+1} = ((v_{t_k}\omega)^{y_j}, H_{A_{\overline{m}+1}}^{y_j})^{1+\frac{2\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C_{3,\overline{m}+1} = ((v_{Pcode_{\overline{m}+1}}\cdot\omega)^{z_{1,j}}, H_{A_{\overline{m}+1}}^{z_{1,j}})^{1+\frac{2\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C_{4,\overline{m}+1} = ((v_{Ncode_{\overline{m}+1}}\cdot\omega)^{z_{2,j}}, H_{A_{\overline{m}+1}}^{z_{2,j}})^{1+\frac{2\varepsilon}{\Delta_\sigma(A_j)}}\}. \end{aligned} \tag{24}$$

Next, the data owner will send the tuple $(Attr2AND, UK, C_{\overline{m}+1})$ to the CSP and ask the CSP to update the ciphertext header from $\widehat{\mathcal{H}_\mathcal{P}}$ to $\widetilde{\mathcal{H}_\mathcal{P}}$. The CSP first adds $C_{\overline{m}+1}$ to $\widetilde{\mathcal{H}_\mathcal{P}}$ and then runs the $CTUpdate$ algorithm to update the previous ciphertext component C_j to the new version C'_j as follows:

$$\begin{aligned} C'_j = \{&C'_{1,j} = C_{1,j}\cdot UK_1 = ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_j}^{x_j})^{1+\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C'_{2,j} = C_{2,j}\cdot UK_2 = ((\bar{v}_{t_i}\omega)^{y_j}, H_{A_j}^{y_j})^{1+\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C'_{3,j} = C_{3,j}\cdot UK_3 = ((v_{Pcode_j}\cdot\omega)^{z_{1,j}}, H_{A_j}^{z_{1,j}})^{1+\frac{\varepsilon}{\Delta_\sigma(A_j)}}, \\ &C'_{4,j} = C_{4,j}\cdot UK_4 = ((v_{Ncode_j}\cdot\omega)^{z_{2,j}}, H_{A_j}^{z_{2,j}})^{1+\frac{\varepsilon}{\Delta_\sigma(A_j)}}\}. \end{aligned} \tag{25}$$

Algorithm 3 New Secret-Sharing Scheme

Input: $\Delta_\sigma(x)$ //Secret share associated with parent of A_j
Input: $\{\Delta_\sigma(A_i)\}_{i\in[1,\overline{m}]\wedge i\neq j}$ //Secret shares associated with $\overline{m}-1$ children
Output: $\Delta'_\sigma(A_j)$ //New secret share for attribute A_j

1: Generate a polynomial $L(x)$ by performing Lagrange interpolation;
2: $\Delta'_\sigma(A_j) = L(j)$; //$j$ is the index of attribute A_j

Similarly, we also provide the construction for the $AttrRmAND$ operation in E-DPU. This operation involves converting an AND gate, $A_j \bigwedge A_{\overline{m}+1}$, into A_j by removing an attribute $A_{\overline{m}+1}$. The data owner first runs Alg. 3 to generate the new share associated with attribute A_j, $\Delta'_\sigma(A_j)$. Specifically, given the secret share associated with A_j's parent node x, denoted as $\Delta_\sigma(x)$, and the secret shares associated with other attributes under node x, denoted as $\{\Delta_\sigma(A_i)\}_{i\in[1,\overline{m}]\wedge i\neq j}$, Alg. 3 generates a Lagrange interpolation polynomial, $L(x)$, and then takes j, the index of A_j under node x, as input, and outputs a new secret share $\Delta'_\sigma(A_j)$.

Next, the data owner runs the $UKGen$ algorithm to generate the update key UK with Eq. 21. Then, the data owner will send the tuple ($AttrRmAND$, $\overline{m}+1, UK$) to the CSP, which will update the ciphertext header from $\widehat{\mathcal{H}_\mathcal{P}}$ to $\widetilde{\mathcal{H}_\mathcal{P}}$. Specifically, the CSP first deletes the ciphertext component $C_{\overline{m}+1}$ from $\widehat{\mathcal{H}_\mathcal{P}}$ and then runs the $CTUpdate$ algorithm to update the previous ciphertext component C_j corresponding to A_j to the new version C'_j with Eq. 22.

5.3. Correctness Proof

To verify the correctness of the DPU scheme, we need to prove that the new ciphertext $\widetilde{\mathcal{H}_\mathcal{P}}$, generated from $Attr2OR$, $AttrRmOR$, $Attr2AND$, and $AttrRmAND$ operations in both G-DPU and E-DPU, can be used to recover the session key ek by running the $Decrypt2$ algorithm. Because of space limitations, we provide only the correct proof for the $Attr2AND$ operation in G-DPU.

As shown in Eq. 22, for each attribute A_j with $j \in [1, \overline{m}]$, the updated

ciphertext component C'_j is constructed as follows:

$$\begin{aligned} C'_j = \{ & C'_{1,j} = (\bar{E}_{t_i}, E'_{t_i}) = ((\bar{v}_{t_i}\omega)^{x_j}, H_{A_j}^{x_j})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{2,j} = (E_{t_k}, E'_{t_k}) = ((v_{t_k}\omega)^{y_j}, H_{A_j}^{y_j})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{3,j} = (E_{Pcode_j}, E'_{Pcode_j}) = ((v_{Pcode_j}\cdot\omega)^{z_{1,j}}, H_{A_j}^{z_{1,j}})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}, \\ & C'_{4,j} = (E_{Ncode_j}, E'_{Ncode_j}) = ((v_{Ncode_j}\cdot\omega)^{z_{2,j}}, H_{A_j}^{z_{2,j}})^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}}\}. \end{aligned} \tag{26}$$

Let $\overline{\kappa}$ denote $\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)}$. Then, for $j \in [1, \overline{m}]$, the outputs of the *Decrypt*1 algorithm of the HCBE scheme are as follows:

$$\begin{aligned} F_1 &\leftarrow \frac{e(\widetilde{D}_t, \overline{E}_{t_i}))}{e(\widetilde{D'}_{t_i}, E'_{t_i})} = \frac{e(g^{\tau'_k} H_{A_j}^{r'}, (\overline{v}_{t_i}\omega)^{\overline{\kappa}x_j})}{e((\overline{v}_{t_i}\omega)^{r'}, H_{A_j}^{\overline{\kappa}x_j})} \\ &= e(g^{\tau'_k}, \overline{v}_{t_k}^{\overline{\kappa}x_j}) \cdot e(g^{\tau'_k}, \omega^{\overline{\kappa}x_j}) = e(g^{\tau'_k}, \omega)^{\overline{\kappa}x_j} \end{aligned} \tag{27}$$

$$\begin{aligned} F_2 &\leftarrow \frac{e(\widetilde{D}_t, E_{t_k})}{e(\widetilde{D}'_{t_k}, E'_{t_k})} = \frac{e(g^{\tau'_k} H_{A_j}^{r'}, (v_{t_k}\omega)^{\overline{\kappa}y_j})}{e((v_{t_k}\omega)^{r'}, H_{A_j}^{\overline{\kappa}y_j})} \\ &= e(g^{\tau'_k}, v_{t_k}^{\overline{\kappa}y_j}) \cdot e(g^{\tau'_k}, \omega^{\overline{\kappa}y_j}) = e(g^{\tau'_k}, \omega)^{\overline{\kappa}y_j} \end{aligned} \tag{28}$$

$$\begin{aligned} F_3 &\leftarrow \frac{e(\widetilde{D}_t, E_{Pcode_j})}{e(\widetilde{D}'_{Pcode_j}, E'_{Pcode_j})} = \frac{e(g^{\tau'_k} H_{A_j}^{r'}, (v_{Pcode_j}\omega)^{\overline{\kappa}z_{1,j}})}{e((v_{Pcode_j}\omega)^{r'}, H_{A_j}^{\overline{\kappa}z_{1,j}})} \\ &= e(g^{\tau'_k}, v_{Pcode_j}^{\overline{\kappa}z_{1,j}}) \cdot e(g^{\tau'_k}, \omega^{\overline{\kappa}z_{1,j}}) = e(g^{\tau'_k}, \omega)^{\overline{\kappa}z_{1,j}} \end{aligned} \tag{29}$$

$$\begin{aligned} F_4 &\leftarrow \frac{e(\widetilde{D}_t, E_{Ncode_j})}{e(\widetilde{D}'_{Ncode_l}, E'_{Ncode_j})} = \frac{e(g^{\tau'_k} H_{A_j}^{r'}, (v_{Ncode_j}\omega)^{\overline{\kappa}z_{2,j}})}{e((v_{Ncode_j}\omega)^{r'}, H_{A_j}^{\overline{\kappa}z_{2,j}})} \\ &= e(g^{\tau'_k}, v_{Ncode_j}^{\overline{\kappa}z_{2,j}}) \cdot e(g^{\tau'_k}, \omega^{\overline{\kappa}z_{2,j}}) = e(g^{\tau'_k}, \omega)^{\overline{\kappa}z_{2,j}} \end{aligned} \tag{30}$$

$$\begin{aligned} F_t &= F_1 \cdot F_2 \cdot F_3 \cdot F_4 \\ &= e(g^{\tau'_k}, \omega)^{\overline{\kappa}(x_j + y_j + z_{1,j} + z_{2,j})} \\ &= e(g^{\tau'_k}, \omega)^{\frac{\Delta'_\sigma(A_j)}{\Delta_\sigma(A_j)} \cdot \Delta_\sigma(A_j)} \\ &= e(g^{\tau'_k}, \omega)^{\Delta'_\sigma(A_j)}. \end{aligned} \tag{31}$$

The new ciphertext component $C_{\overline{m}+1}$ corresponding to $A_{\overline{m}+1}$ is shown in Eq. 20. Therefore, the outputs of the *Decrypt*1 algorithm of the HCBE scheme are as follows:

$$F_1 \leftarrow \frac{e(\widetilde{D}_t, \overline{E}_{t_i})}{e(\widetilde{\overline{D}'}_{t_i}, E'_{t_i})} = \frac{e(g^{\tau'_k} H^{r'}_{A_{\overline{m}+1}}, (\overline{v}_{t_i}\omega)^{x_{\overline{m}+1}})}{e((\overline{v}_{t_i}\omega)^{r'}, H^{x_{\overline{m}+1}}_{A_{\overline{m}+1}})} \tag{32}$$
$$= e(g^{\tau'_k}, \overline{v}^{x_{\overline{m}+1}}_{t_i}) \cdot e(g^{\tau'_k}, \omega^{x_{\overline{m}+1}}) = e(g^{\tau'_k}, \omega)^{x_{\overline{m}+1}}$$

$$F_2 \leftarrow \frac{e(\widetilde{D}_t, E_{t_k})}{e(\widetilde{D}'_{t_k}, E'_{t_k})} = \frac{e(g^{\tau'_k} H^{r'}_{A_{\overline{m}+1}}, (v_{t_k}\omega)^{y_{\overline{m}+1}})}{e((v_{t_k}\omega)^{r'}, H^{y_{\overline{m}+1}}_{A_{\overline{m}+1}})} \tag{33}$$
$$= e(g^{\tau'_k}, v^{y_{\overline{m}+1}}_{t_k}) \cdot e(g^{\tau'_k}, \omega^{y_{\overline{m}+1}}) = e(g^{\tau'_k}, \omega)^{y_{\overline{m}+1}}$$

$$F_3 \leftarrow \frac{e(\widetilde{D}_t, E_{Pcode_{\overline{m}+1}})}{e(\widetilde{D}'_{Pcode_{\overline{m}+1}}, E'_{Pcode_{\overline{m}+1}})} = \frac{e(g^{\tau'_k} H^{r'}_{A_{\overline{m}+1}}, (v_{Pcode_{\overline{m}+1}}\omega)^{z_{1,\overline{m}+1}})}{e((v_{Pcode_{\overline{m}+1}}\omega)^{r'}, H^{z_{1,\overline{m}+1}}_{A_{\overline{m}+1}})} \tag{34}$$
$$= e(g^{\tau'_k}, v^{z_{1,\overline{m}+1}}_{Pcode_{\overline{m}+1}}) \cdot e(g^{\tau'_k}, \omega^{z_{1,\overline{m}+1}}) = e(g^{\tau'_k}, \omega)^{z_{1,\overline{m}+1}}$$

$$F_4 \leftarrow \frac{e(\widetilde{D}_t, E_{Ncode_{\overline{m}+1}})}{e(\widetilde{D}'_{Ncode_{\overline{m}+1}}, E'_{Ncode_{\overline{m}+1}})} = \frac{e(g^{\tau'_k} H^{r'}_{A_{\overline{m}+1}}, (v_{Ncode_{\overline{m}+1}}\omega)^{z_{2,\overline{m}+1}})}{e((v_{Ncode_{\overline{m}+1}}\omega)^{r'}, H^{z_{2,\overline{m}+1}}_{A_{\overline{m}+1}})} \tag{35}$$
$$= e(g^{\tau'_k}, v^{z_{2,\overline{m}+1}}_{Ncode_{\overline{m}+1}}) \cdot e(g^{\tau'_k}, \omega^{z_{2,\overline{m}+1}}) = e(g^{\tau'_k}, \omega)^{z_{2,\overline{m}+1}}$$

$$\begin{aligned} F_t &= F_1 \cdot F_2 \cdot F_3 \cdot F_4 \\ &= e(g^{\tau'_k}, \omega)^{(x_{\overline{m}+1} + y_{\overline{m}+1} + z_{1,\overline{m}+1} + z_{2,\overline{m}+1})} \\ &= e(g^{\tau'_k}, \omega)^{\Delta_\sigma(A_{\overline{m}+1})} \end{aligned} . \tag{36}$$

Therefore, the value $C'_2 = e(g^{\tau'_k}, \omega)^\sigma$ is computed from $\{e(g^{\tau'_k}, \omega)^{\triangle_\sigma(A_l)}\}_{A_l \in \mathcal{T}'}$ by using the aggregation algorithm, and with the new ciphertext header $\widetilde{\mathcal{H}_\mathcal{P}} = (C = h^\sigma, C'_2)$, a data user can run the *Decrypt*2 algorithm to recover the session key ek. ■

6. Analysis

6.1. Performance Analysis

In this subsection, we analyse the complexity of the CBE scheme and of our scheme. For simplification, we provide the following notations to denote the times for various operations in both schemes: $E(\mathbb{G})$ and $E(\mathbb{G}_\mathrm{T})$ are used to denote the exponentiation in $\mathbb{G}$ and $\mathbb{G}_\mathrm{T}$, respectively. B is used to denote the bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$. For the CBE scheme, we neglect the operations in $\mathbb{Z}_N$, the hash function $H : \{0,1\}^* \to \mathbb{G}$, and the multiplication in $\mathbb{G}$ and $\mathbb{G}_\mathrm{T}$ since they are significantly less expensive than other exponentiation and paring operations.

Table 2 and Table 3 show the computation and communication complexity for each phase in the CBE scheme and the HCBE scheme, respectively.

Table 2: Performance Analysis of CBE

	Computation Complexity	Communication Complexity
Setup	$1 \cdot B + 3 \cdot E(\mathbb{G})$	$6 \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_\mathrm{T}} + 2 \cdot l_{\mathbb{Z}_n}$
GenKey	$(1 + 4 \cdot \lvert S \rvert) \cdot E(\mathbb{G})$	$(1 + 4 \cdot \lvert S \rvert) \cdot l_{\mathbb{G}}$
Encrypt	$(1 + 4 \cdot \lvert \mathcal{T} \rvert) \cdot E(\mathbb{G}) + 1 \cdot E(\mathbb{G}_\mathrm{T})$	$4 \cdot \lvert \mathcal{T} \rvert \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_\mathrm{T}}$
Delegate	$(1 + 5 \cdot \lvert S \rvert) \cdot E(\mathbb{G})$	$3 \cdot \lvert S \rvert \cdot l_{\mathbb{G}}$
Decrypt1	$2 \cdot \lvert S \rvert \cdot B + \lvert \mathcal{T} \rvert \cdot E(\mathbb{G}_T)$	$1 \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_\mathrm{T}}$
Decrypt2	$1 \cdot B + 1 \cdot E(\mathbb{G})$	

Table 3: Performance Analysis of HCBE

	Computation Complexity	Communication Complexity
Setup	$1 \cdot B + 3 \cdot E(\mathbb{G})$	$8 \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_T} + 4 \cdot l_{\mathbb{Z}_n}$
GenKey	$(1 + 6 \cdot \lvert S \rvert) \cdot E(\mathbb{G})$	$(1 + 6 \cdot \lvert S \rvert) \cdot l_{\mathbb{G}}$
Encrypt	$(1 + 8 \cdot \lvert \mathcal{TL} \rvert) \cdot E(\mathbb{G}) + 1 \cdot E(\mathbb{G}_\mathrm{T})$	$8 \cdot \lvert \mathcal{TL} \rvert \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_\mathrm{T}}$
Delegate	$(1 + 8 \cdot \lvert S \rvert) \cdot E(\mathbb{G})$	$5 \cdot \lvert S \rvert \cdot l_{\mathbb{G}}$
Decrypt1	$4 \cdot \lvert S \rvert \cdot B + \lvert \mathcal{TL} \rvert \cdot E(\mathbb{G}_\mathrm{T})$	$1 \cdot l_{\mathbb{G}} + 1 \cdot l_{\mathbb{G}_\mathrm{T}}$
Decrypt2	$1 \cdot B + 1 \cdot E(\mathbb{G})$	

Here, $|\mathcal{TL}|$ denotes the number of generalized attributes in the access tree $\mathcal{T}$, $|\mathcal{T}|$ denotes the number of specific attributes in the access tree $\mathcal{T}$, S denotes the set of attributes of the data owner and the data user, and $l_{\mathbb{Z}_n}$, $l_{\mathbb{G}}$, and $l_{\mathbb{G}_\mathrm{T}}$ denote the lengths of elements in $\mathbb{Z}_n^*$, $\mathbb{G}$, and $\mathbb{G}_\mathrm{T}$, respectively.

From the perspective of the data owner, the encryption cost in the HCBE scheme is impacted primarily by the number of generalized attributes $|\mathcal{TL}|$. With a well-designed attribute hierarchy, the encryption cost can largely be avoided. For example, if $|\mathcal{T}| = 2 \cdot |\mathcal{TL}|$, then the computation cost will be reduced by 50%. Our *GenKey* algorithm may cost a little more than the CBE scheme, since the HCBE scheme needs to generate the key components on the Pcode and Ncode.

For a data user, our decryption cost is the same as that of the CBE scheme, but the delegation process may be more expensive. However, for a given attribute hierarchy of L levels, the Pcode/Ncode derivation process will be calculated at least L times, unlike the time derivation process, which needs to be performed continually. Therefore, in the long run, the cost of our *Delegation* algorithm is similar to that of the CBE scheme. Finally, our *Decrypt*1 algorithm will be more expensive than the CBE scheme. However,

this part of the decryption operation is delegated to the proxy server, which may be located in a private cloud platform. Therefore, the HCBE scheme is acceptable for application in the cloud-based PHR environment.

We also provide an analysis of the performance of the DPU scheme. Let $\overline{m}$ denote the number of attributes under an AND/OR gate node that is to be updated. Since the $Attr2OR$ and $AttrRmOR$ operations will not change the secret shares of the remaining attributes, their cost is almost constant and will not be listed. Here, we only compare the performance of the $Attr2AND$ and $AttrRmAND$ operations for the two versions of the DPU scheme. From Table 4 and Table 5, we can see that under an AND gate, the cost of G-DPU grows linearly with the number of attributes in the gate node.

Table 4: Performance Analysis of G-DPU

	Computation Complexity	Communication Complexity
UKGen	$8\overline{m} \cdot E(\mathbb{G})$	$8\overline{m} \cdot l_{\mathbb{G}}$
CTGen	$8 \cdot E(\mathbb{G})$	$8 \cdot l_{\mathbb{G}}$
CTUpdate	$8\overline{m} \cdot E(\mathbb{G})$	0

Table 5: Performance Analysis of E-DPU

	Computation Complexity	Communication Complexity
UKGen	$8 \cdot E(\mathbb{G})$	$8 \cdot l_{\mathbb{G}}$
CTGen	$8 \cdot E(\mathbb{G})$	$8 \cdot l_{\mathbb{G}}$
CTUpdate	$8 \cdot E(\mathbb{G})$	0

6.2. Security Analysis

As described in Section 2, the HCBE scheme fails if either CASE 1 or CASE 2 occurs. In this subsection, we sketch the security of our scheme as follows:

The data file stored in the cloud is encrypted with a session key $ek = e(g^{\alpha}, \omega)^{\sigma}$. For ease of illustration, we assume that ek is encrypted with the access policy $\widehat{AP} = A_l(t_i, t_j, Pcode_l, Ncode_l) \wedge A_x(t_i, t_j, Pcode_x, Ncode_x)$. We consider the first condition in CASE 1 to be true if user $\mathcal{U}_1$, whose access privilege $\widehat{\mathcal{L}_1} = A_l(t_i, t_j, Pcode_l, Ncode_l)$, can recover ek by colluding with $\mathcal{U}_2$, whose access privilege $\widehat{\mathcal{L}_2} = A_x(t_i, t_j, Pcode_x, Ncode_x)$. The construction of the HCBE scheme allows them to recover $F_{t1} = e(g^{\tau'_{k1}}, \omega)^{\Delta_{\sigma}(A_l)}$ and $F_{t2} =$

$e(g^{\tau'_{k2}}, \omega)^{\Delta_\sigma(A_x)}$ with private keys $SK_{\widehat{\mathcal{L}_1}}$ and $SK_{\widehat{\mathcal{L}_2}}$, respectively. However, τ'_{k1} and τ'_{k2} are uniquely chosen to distinguish different users. Therefore, with F_{t1} and F_{t2}, they cannot obtain either $T1 = e(g^{\tau'_{k1}}, \omega)^\sigma$ or $T2 = e(g^{\tau'_{k2}}, \omega)^\sigma$ to recover ek, and the first condition in CASE 1 is false.

Next, we assume that ek is simply encrypted with the access policy $\widehat{AP} = A_l(t_i, t_j, Pcode_l, Ncode_l)$. We consider the second condition in CASE 1 to be true if user $\mathcal{U}_1$, whose access privilege $\widehat{\mathcal{L}_1} = A_l(t_a, t_b, Pcode_l, Ncode_l)$, can recover ek while $t_j < t_a$ (or $t_i > t_b$). Note that, because of the one-way property of the FDF and BDF in CBE, $\mathcal{U}_1$ cannot derive D'_{t_j} and $\overline{D}'_{t_i}$ from D'_{t_a} and $\overline{D}'_{t_b}$ while $t_j < t_a$ (or $t_i > t_b$). Therefore, $\mathcal{U}_1$ cannot obtain F_1 and F_2 to recover ek, and the second condition in CASE 1 is false.

Finally, we consider the third condition in CASE 1 to be true if user $\mathcal{U}_1$, whose access privilege $\widehat{\mathcal{L}_1} = A_l(t_a, t_b, Pcode_l, Ncode_l)$, can recover ek while access policy $\widehat{AP} = A_x(t_a, t_b, Pcode_x, Ncode_x)$ while $Pcode_x > Pcode_l$ or $Ncode_x > Ncode_l$. Note that, because of the one-way property of the BDF in CBE, $\mathcal{U}_1$ cannot derive D'_{Pcode_x} from D'_{Pcode_l} while $Pcode_x > Pcode_l$. The same situation holds for $Ncode_x > Ncode_l$. Therefore, the third condition in CASE 1 is false, and CASE 1 will not occur. ■

The proof of CASE 2 is similar to that of CASE 1. To obtain ek, the CSP needs to calculate $F_1 \cdot F_2 \cdot F_3 \cdot F_4$ to obtain $e(g^{\tau'_k}, \omega)^{\Delta_\sigma(A_l)}$ for a sufficient number of attributes $A_l \in \mathcal{T}$. Since the CSP is not allowed to access the PHR system, it cannot obtain a sufficient number of private keys. As proven in CASE 1, the entities that do not meet the access policy specifications cannot recover ek. Therefore, CASE 2 will not occur. ■

As described in Section 2, the DPU scheme is considered to fail if either CASE 1 or CASE 2 occurs after policy updating. In Subsection 5.3, we proved that the new ciphertext $\widetilde{\mathcal{H}_\mathcal{P}}$ generated from our DPU scheme is equal to the output of the *Delegate* algorithm under the updated access policy. Therefore, because of the security of the PRE technique, the security of the DPU scheme can be attributed to the HCBE scheme. ■

7. Experimental Results

In this section, we first compare our HCBE scheme with the CBE scheme in terms of computation cost. Then, we describe the experiments we conducted to test the performance of the DPU scheme.

Table 6: Computational Time for Various Operations

Operation	Time(ms)
The bilinear pairing operation	$1,102$
The power operation in group $\mathbb{G}$	865
The power operation in group $\mathbb{G}_T$	128
The power operation in group $\mathbb{G}_s$	868
The power operation in group $\mathbb{G}_{n'}$	862
$H : \{0,1\}^* \rightarrow \mathbb{G}$	176

7.1. Computation Cost in HCBE

Our experiments were conducted with the Java Pairing-Based Cryptography library. We used a symmetric elliptic curve of 160-bit group order, a1-curve , for which the base field size is 1024 bits and the embedding degree is 2. We implemented our scheme in a stand-alone mode on a PC with an Intel Core i3 CPU running at 2.3 GHz with 2 GB of memory. Let $\mathbb{G}$ and $\mathbb{G}_T$ denote cyclic groups of composite order n, where $n = sn'$, and let $\mathbb{G}_s$ and $\mathbb{G}_{n'}$ denote the subgroups of order s and n' in $\mathbb{G}$. Table 6 shows the computation overhead of pairing operations, the power operations in different groups, and hash operations.

The computational costs of algorithms $Setup$, $GenKey$, $Encrypt$, $Delegate$, $Decrypt1$, and $Decrypt2$ in the CBE and HCBE schemes are shown in Fig. 8–Fig. 13. Let M and m denote the number of specific attributes in an access policy $\widehat{AP}$ and the number of nodes in an access policy tree $\mathcal{T}$, respectively. In our experiments, M was set to 10, and m was set to $[50, 100]$. Given the maximal integer Z ranging from 7 to 70,000, we generated a private key with privilege $[t_1, t_2]$, where $t_1 \in_R [1, Z/4]$ and $t_2 \in_R [3Z/4, Z]$, for a certain comparison range $[1, Z]$. The message was encrypted by the time condition $t \in_R [Z/4, 3Z/4]$ to ensure that $\max(t - t_1, t_2 - t) \geq Z/4$.

From Fig. 8–Fig. 13, we observe that the growth of time overhead was insignificant as the value of Z increases for both the CBE and HCBE schemes. Meanwhile, in our scheme, the growth of time overhead was insignificant, whereas m grew from 50 to 100. Because of the introduction of an attribute hierarchy, the computational overhead of the $Setup$ and $GenKey$ algorithms in our scheme was greater than that for the CBE scheme. However, the difference is minor. For example, as shown in Fig. 8, the computation time of our $Setup$ algorithm grew from 5.43 s to 5.46 s under the setting of $m = 50$, and the computation time of the $Setup$ algorithm in the CBE scheme grew

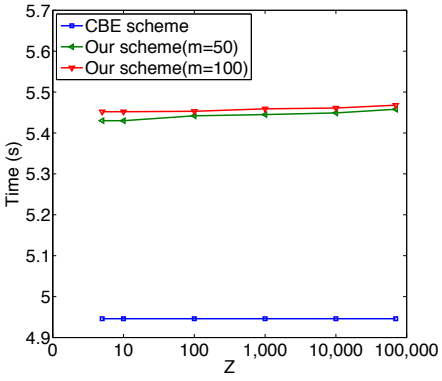


Figure 8: Computation cost of *Setup*.

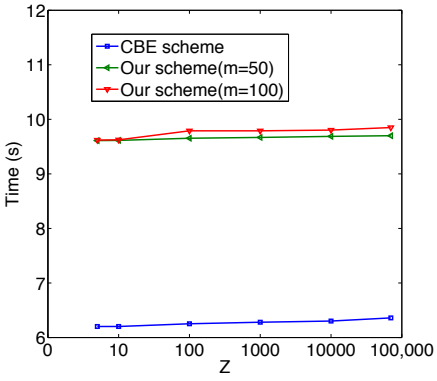


Figure 9: Computation cost of $GenKey$.

from 4.94 s to 4.95 s as Z ranged from 7 to 70, 000; as shown in Fig. 9, the computation time of our $GenKey$ algorithm grew from 9.62 s to 9.85 s under the setting of $m = 100$, and the computation time of the $GenKey$ algorithm in the CBE scheme grew from 6.20 s to 6.36 s as Z ranged from 7 to 70, 000.

In the experiments, we employed $M = 10$ specific attributes in the CBE scheme, and we employed 5 generalized attributes in our scheme for better comparison. As shown in Fig. 10, the encryption time in our scheme is much lower than that in the CBE scheme. For example, the computation time of our *Encrypt* algorithm grew from 25.75 s to 25.98 s under the setting of $m = 50$, and the computation time of the *Encrypt* algorithm in the CBE scheme grew from 38.95 s to 40.00 s as Z ranged from 7 to 70, 000. Furthermore, with the decrease in the value of m, our scheme had better performance.

The algorithms run by the data user included *Delegate* and *Decrypt*2. From Fig. 11 and Fig. 13, we observe that our scheme incurred a bit more computation cost for the data user than did the CBE scheme. We also show the comparison of time overhead of the *Decrypt*1 algorithm, executed by the proxy server, in Fig. 12. The above experimental results are consistent with our theoretical analysis in Section 6.

7.2. Computation Cost in DPU

In this subsection, we evaluate the computation time for each type of operation in both G-DPU and E-DPU. Let $\overline{m}$ denote the number of attributes

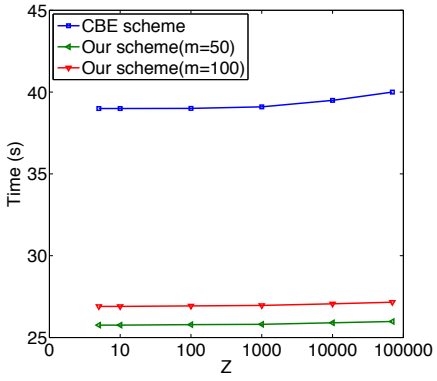


Figure 10: Computation cost of *Encrypt*.

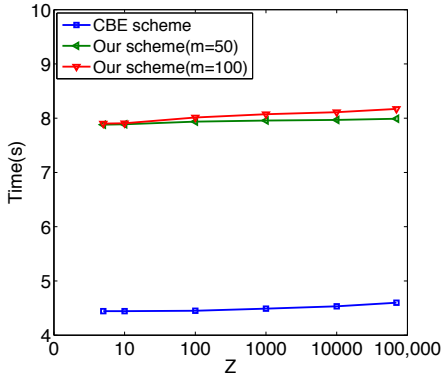


Figure 11: Computation cost of *Delegate*.

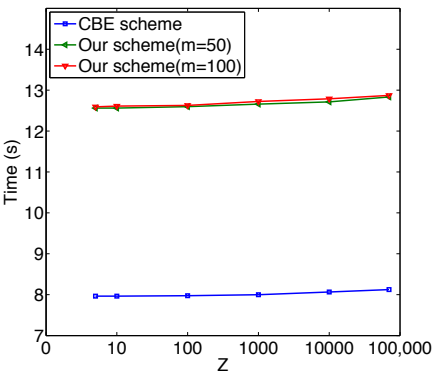


Figure 12: Computation cost of *Decrypt*1.

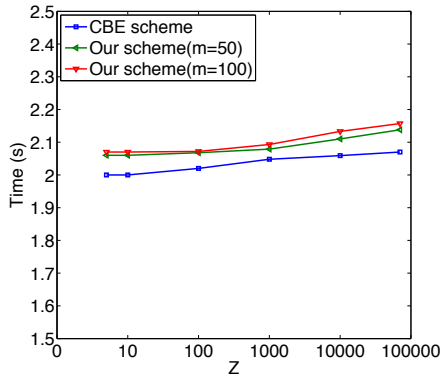
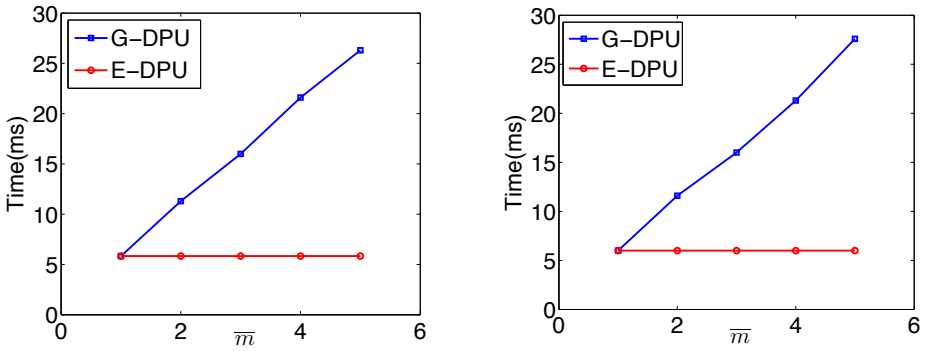


Figure 13: Computation cost of *Decrypt*2.

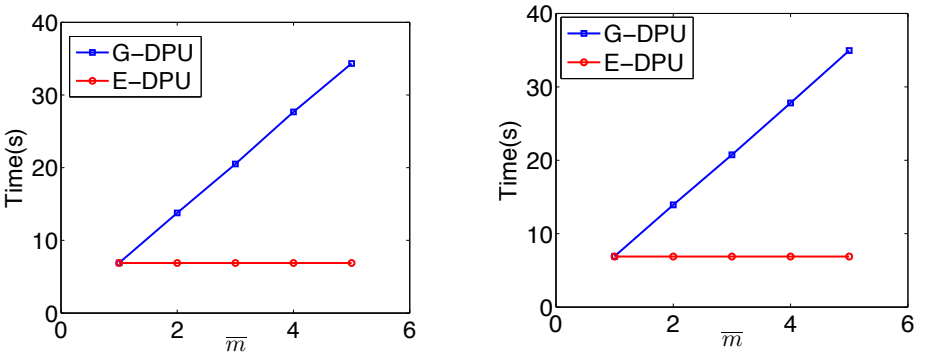
under an AND/OR gate node to be updated, where $\overline{m}$ ranged from 1 to 5 in our experiments. For the $Attr2AND$ and $AttrRmAND$ operations, the time overhead of the $CTUpdate$ algorithm is shown in Fig. 14, from which we can see that G-DPU incurred more computational cost for the CSP compared with E-DPU. As for the $Attr2OR$ and $AttrRmOR$ operations, the secret shares associated with the remaining attributes does not change , and thus the time overhead incurred by the CSP for both versions approached 0.

Fig. 15 shows the cost of the $UKGen$ algorithm in $Attr2AND$ and $AttrRmAND$ operations. From the figure, we can see that the time cost



(a) Attr2AND (b) AttrRmAND

Figure 14: Computation cost of $CTUpdate$.



(a) Attr2AND (b) AttrRmAND

Figure 15: Computation cost of $UKGen$.

of G-DPU is proportional to $\overline{m}$, the number of attributes under the updating gate. Similarly, the $Attr2OR$ and $AttrRmOR$ operations do not change the secret shares associated with the remaining attributes, and the time cost of the $UKGen$ algorithm is 0 in the above two operations. Furthermore, the $CTGen$ algorithm is run by the data owner while performing $Attr2OR$ and $Attr2AND$ operations. In the experiments, the cost of generating the new ciphertext component for a newly added attribute was about 11 s. Therefore, G-DPU incurs a much heavier workload for the data owner during policy updating compared with E-DPU.

From the above experiment results, we also can see that the cost of the $Encrypt$ algorithm is far greater than that of the $UKGen$ algorithm. If the

data owner performs the re-encryption operation on her own, the overhead will be extensive. In our proposed schemes, the data owner's time overhead will be greatly reduced by delegating the re-encryption operation to the CSP, thereby getting a better service experience.

8. Related Work

Today, many CSPs such as Amazon, Google, and Microsoft have provided PHR services. PHRs contain a significant amount of sensitive information, thus creating the problem of how to preserve individual privacy while using a cloud-based PHR system [9]. To prevent the exposure of information to unauthorized individuals, cryptographic tools and access control mechanisms are proposed as promising solutions [16]. Furthermore, forensic techniques [22] are useful tools for recovering sensitive user data.

For example, Gondkar et al. [7] proposed a novel framework for secure sharing of PHRs in cloud computing. Liu et al. [15] provided an efficient and safe access management mechanism to solve the security problems associated with the implementation of PHR in a cloud environment. Yao et al. [29] utilized order-preserving symmetric encryption (OPSE) [4] for preserving data privacy in multi-source personal health record clouds. Li et al. [13] utilized predicate encryption [20] in order to achieve authorized search on PHRs in cloud computing. Nepal et al. [19] discussed the challenges and possible solutions for achieving trustworthy processing of healthcare big data in hybrid clouds. Liu et al. [18] proposed a Ciphertext-Policy Attribute-Based Signcryption (CP-ABSC) scheme for the fine-grained access control of PHRs in cloud computing.

Most existing work has adopted ABE [8, 21, 2] as the cryptographic tool to achieve fine-grained access control in cloud-based PHR systems. The original ABE systems only support monotone access policy and assume the existence of a single private key generator (PKG). Much research has been done to achieve more expressive access policies [14, 11] and distributed key management [10, 12]. On the basis of the ABE scheme, Zhu et al. [31] proposed the CBE scheme by making use of the forward and backward derivation functions and applied CBE to the cloud environment. However, the encryption cost of the CBE scheme grows linearly with the number of attributes in the access policy. To solve this problem, we have proposed the HCBE scheme by incorporating the attribute hierarchy into the CBE scheme.

To achieve dynamic access control in cloud computing, Wang et al. [26] and Yu et al. [30] applied the proxy re-encryption (PRE) technique [3] to ABE. Shi et al. [24] proposed a dubbed directly revocable key-policy ABE with verifiable ciphertext delegation (drvuKPABE) scheme to achieve direct revocation and verifiable ciphertext delegation. Liu et al. [17] proposed a time-based proxy re-encryption scheme, which allowed the cloud to automatically re-encrypt the ciphertexts based on time . Yang et al. [28] presented a conditional proxy re-encryption scheme to achieve cloud-enabled user revocation. However, these previous schemes focused on how to revoke a specific user from the system rather than on updating of the attribute-based access policy in the ciphertext. Rezaeibagha and Mu [23] presented an access control mechanism for an EHR system with a hybrid cloud structure; the system features dynamic policy transformation based on some useful cryptographic building blocks. Yang et al. [27] proposed a novel scheme that enables efficient access control with dynamic policy updating in cloud computing. They designed policy updating algorithms for different types of access policies so as to simultaneously achieve correctness and completeness and meet security requirements.

Inspired by the work in [27], we have developed the DPU scheme to efficiently achieve dynamic policy updating in cloud-based PHR environments.

9. Conclusion and Future Work

In this paper, we have proposed an HCBE scheme and a DPU scheme for achieving dynamic access control in cloud-based PHR systems. The HCBE scheme supports time comparison in attribute-based encryption in an efficient way by incorporating attribute hierarchy into CBE. The DPU scheme utilizes the PRE technique to delegate the policy updating operations to the cloud. However, because of space limitations, we only provide a sketch of the security of the HCBE scheme. In our future work, we will endeavour to prove that the HCBE scheme is secure against CP attacks, KS-CDAs, and SS-CDAs.

Acknowledgements

This work was supported in part by NSFC Grants 61402161 ; by the Hunan Provincial Natural Science Foundation of China (Grant No. 2015JJ3046); and by NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167.

[1] M. Armbrust, A. Fox, R. Griffith, et al, "A view of cloud computing", in *Communications of the ACM*, 2010, vol. 53. no. 2, pp. 50-58.

[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute based encryption", in *Proceedings of IEEE S&P*, 2007, pp.321-349.

[3] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proceedings of EUROCRYPT*, 1998, pp. 127-144.

[4] A. Boldyreva, N. Chenette, and A. O. Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions", in *Advances in Cryptology-CRYPTO*, 2011, vol. 6841, pp. 578-595.

[5] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing", in *Advances in Cryptology–CRYPTO*, 2001, vol. 2139, pp.213-229.

[6] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO*, 1984, pp. 10-18.

[7] D.Gondkar, V S.Kadam, et al, "Attribute based encryption for securing personal health record on cloud," in *Proceedings of IEEE ICDCS*, 2014, pp. 1-5.

[8] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data", in *Proceedings of ACM CCS*, 2006, pp.89-98.

[9] L. Guo, C. Zhang, J. Sun, et al, "PAAS: A privacy-preserving attribute-based authentication system for ehealth networks", in *Proceedings of IEEE ICDCS*, 2012, pp. 224-233.

[10] J. Han, W. Susilo, Y. Mu, J. Zhou, "Improving privacy and security in decentralized ciphertext-policy attribute-based encryption", in *IEEE Transactions on Information Forensics and Security*, 2015, vol.10, pp.665-678.

[11] T. Jung, X.Y. Li, Z. Wan, M. Wan, "Control cloud data access privilege and anonymity with fully anonymous attribute-based encryption", in *IEEE Transactions on Information Forensics and Security*, 2015, vol.10, pp.190-199.

[12] A. Lewko and B. Waters, "Decentralizing attribute-based encryption", in *Advances in Cryptology–EUROCRYPT*, 2011, vol. 6632, pp. 568-588.

[13] M. Li, S. Yu, N. Cao, et al, "Authorized private keyword search over encrypted data in cloud computing", in *Proceedings of IEEE ICDCS*, 2011, pp. 383-392.

[14] K. Liang, M.H. Au, et al, "A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing", in *Future Generation Computer Systems*, 2015, vol.52, pp.95-108.

[15] C.H. Liu, F.Q. Lin, C.S. Chen, "Design of secure access control scheme for personal health record-based cloud healthcare service", in *Security and Communication Networks*, 2015, vol.8, pp.1332-1346.

[16] X. Liu, et al, "Efficient and privacy-preserving outsourced calculation of rational numbers", in *Proceedings of IEEE TDSC*, 2016, pp. 1.

[17] Q. Liu, G. Wang, J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," in *Information Sciences*, 2014, vol. 258, pp. 355-370.

[18] J. Liu, X. Huang, J. K. Liu, "Secure sharing of personal health records in cloud computing: ciphertext-policy attribute-based signcryption," in *Future Generation Computer Systems*, 2015, vol.52, pp.67-76.

[19] S. Nepal, R. Ranjan, K. K. R. Choo, "Trustworthy processing of healthcare big data in hybrid clouds", in *IEEE Cloud Computing*, 2015, vol. 2, pp. 78-74.

[20] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products", in *Advances in Cryptology–ASIACRYPT*, 2009, vol. 5912, pp. 214-231.

[21] R. Ostrovsky, A. Sahai, B. Waters, "Attribute-based encryption with non-monotonic access structures", in *Proceedings of ACM CCS*, 2007, pp.195-203.

[22] D. Quick and K. K. R. Choo, "Google Drive: Forensic analysis of data remnants," in *Journal of Network and Computer Applications*, 2014, vol. 40, pp. 179-193.

[23] F. Rezaeibagha, Y. Mu, "Distributed clinical data sharing via dynamic access-control policy transformation", in *International journal of medical informatics*, 2016, vol.89, pp. 25-31.

[24] Y. Shi, Q. Zheng, et al, "Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation", in *Information Sciences*, 2015, vol.295, pp. 221-231.

[25] P. Tang, J. Ash, D. Bates, et al, "Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption", in *Journal of the American Medical Informatics Association*, 2006, vol. 13, no. 2, pp. 121-126.

[26] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services", in *Proceedings of ACM CCS*, 2010, pp. 735-737.

[27] K. Yang, X. Jia, K. Ren, et al, "Enabling efficient access control with dynamic policy updating for big data in the cloud," in *Proceedings of IEEE INFOCOM*, 2014, pp. 2013-2021.

[28] Y. Yang, H. Zhu, et al, "Cloud based data sharing with fine-grained proxy re-encryption", in *Pervasive and Mobile Computing*, 2015.

[29] X. Yao, Y. Lin, Q. Liu, et al, "Efficient and privacy-preserving search in multi-source personal health record clouds", in *Proceedings of IEEE ISCC*, 2015, pp. 803-808.

[30] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure,scalable, and fine-grained data access control in cloud computing", in *Proceedings of IEEE INFOCOM*, 2010, pp. 534-542.

[31] Y. Zhu, H. Hu, G. Ahn, et al, "Comparison-based encryption for fine-grained access control in clouds", in *Proceedings of ACM CODASPY*, 2012, pp. 105-116.

[32] Googlehealth. https://www.google.com/health/.

[33] Healthvault. http://www.healthvault.com/.