This paper was accepted by the
11th International Conference on Applied Cryptography and Network Security (ACNS'13)
(this is the full version)

CloudHKA: A Cryptographic Approach for Hierarchical Access Control in Cloud Computing*

Yi-Ruei Chen[1], Cheng-Kang Chu[2], Wen-Guey Tzeng[3], and Jianying Zhou[4]

[1,3]Department of Computer Science, National Chiao Tung University, Taiwan
[2,4]Institute for Infocomm Research, Singapore
[1]`yrchen.cs98g@nctu.edu.tw`, [2]`ckchu@i2r.a-star.edu.sg`,
[3]`wgtzeng@cs.nctu.edu.tw`, [4]`jyzhou@i2r.a-star.edu.sg`

Abstract. Cloud services are blooming recently. They provide a convenient way for data accessing, sharing, and processing. A key ingredient for successful cloud services is to control data access while considering the specific features of cloud services. The specific features include great quantity of outsourced data, large number of users, honest-but-curious cloud servers, frequently changed user set, dynamic access control policies, and data accessing for light-weight mobile devices. This paper addresses a cryptographic key assignment problem for enforcing a hierarchical access control policy over cloud data.
We propose a new hierarchical key assignment scheme *CloudHKA* that observes the Bell-LaPadula security model and efficiently deals with the user revocation issue practically. We use CloudHKA to encrypt outsourced data so that the data are secure against honest-but-curious cloud servers. CloudHKA possesses almost all advantages of the related schemes, e.g., each user only needs to store one secret key, supporting dynamic user set and access hierarchy, and provably-secure against collusive attacks. In particular, CloudHKA provides the following distinct features that make it more suitable for controlling access of cloud data. (1) A user only needs a constant computation time for each data accessing. (2) The encrypted data are securely updatable so that the user revocation can prevent a revoked user from decrypting newly and previously encrypted data. Notably, the updates can be outsourced by using public information only. (3) CloudHKA is secure against the legal access attack. The attack is launched by an authorized, but malicious, user who pre-downloads the needed information for decrypting data ciphertexts in his authorization period. The user uses the pre-downloaded information for future decryption even after he is revoked. Note that the pre-downloaded information are often a small portion of encrypted data only, e.g. the header-cipher in a hybrid encrypted data ciphertext. (4) Each user can be flexibly authorized the access rights of Write or Read, or both.

Keywords. *Access control, hierarchical key assignment, key management, Bell-LaPadula security model, outsourced data, cloud computing, proxy re-encryption*

1 Introduction

Outsourcing data to cloud server (CS) becomes popular in these years. A data provider (DP) no longer stores a large quantity of data locally. A user can access them from anywhere at any time. However, the outsourced data often contain sensitive information and CS naturally becomes a target of attacks. Even worse, CS itself

* The research was supported in part by projects NSC-101-2221-E-009-074-MY3 (National Science Council, Taiwan) and SecDC-112172014 (A*STAR, Singapore)

could distribute DP's data for illegal profit. Therefore, DP does not want to disclose his data to CS. Furthermore, DP wants to control access to data of different sensitive levels. Only the authorized users can access the data with certain security levels. We want to enforce a designated access control policy for users over cloud data.

This work considers the hierarchical access control (HAC) policy. By the policy, data are organized into security classes SC_1, SC_2, $\ldots$, SC_n, which are partially ordered with a binary relation $\prec$. $SC_j \prec SC_i$ means that the security level of SC_i is higher than that of SC_j. If a user is authorized to read data at SC_i, he is also entitled to read data at SC_j for $SC_j \prec SC_i$. The HAC policy is widely used in various computer systems, e.g., military, government, secure database, and Pay-TV systems.

Hierarchical key assignment (HKA) is a cryptographic method for enforcing HAC policies [1]. An HKA scheme consists of a set of cryptographic keys SK_1, SK_2, $\ldots$, SK_n such that if $SK_j \prec SK_i$, SK_j can be derived by using SK_i. To enforce an HAC policy $\mathcal{P}$ for hierarchical data, a datum at SC_j is encrypted into ciphertext by using SK_j. A user who is authorized to read the data at SC_i is assigned SK_i. Thus, the user can decrypt the data at SC_j, which is lower than SC_i, by using SK_i to derive SK_j.

An important issue in designing an HKA scheme is to revoke an authorized user u from his associated class, say SC_i. DP needs to remove u's access rights for the following two kinds of data:

- *Newly encrypted data at SC_z for $SC_z \preceq SC_i$:* The encrypted data under new encryption keys after revoking u.
- *Previously encrypted data at SC_z for $SC_z \preceq SC_i$:* The encrypted data under previous encryption keys before revoking u.

To prevent u from decrypting newly encrypted data at SC_z, DP can encrypt data by new keys and distribute the new keys to the non-revoked users only. Nevertheless, since non-revoked users needs to access previously encrypted data at SC_z, they should keep all old keys. The key management cost is high if revocation occurs frequently.

To prevent u from decrypting previously encrypted data by using his old keys, DP can decrypt previously encrypted data and encrypt them with new keys, which are distributed to non-revoked users only. Thus, the revoked user u cannot use his old keys to decrypt previously encrypted data. Simultaneously, a non-revoked user needs to keep the newest key of his associated class only. However, since data are of a large quantity, DP needs substantial time in processing them. A common solution is to use the hybrid encryption technique for data encryption. DP randomly chooses a data encryption key K for encrypting data into body-cipher and then encrypts K into header-cipher under a cryptographic key SK_i. In processing data, CS only needs to update the header-cipher and the much larger body-ciphers are no need to be changed. It saves computation time significantly. Nevertheless, the solution causes a new issue, which we call it the *legal access attack*. An authorized, but malicious, user may decrypt all decryptable header-ciphers to obtain K's. The user can use these K's to decrypt body-ciphers in the future even after he is revoked. Furthermore, in processing data, if decryption and encryption operations are done in the CS side, CS gets to know the content of data. Face to the above issues, we want a solution that updates encrypted data without disclosing the content to CS and entailing high overhead for DP and CS. Simultaneously, we hope that the solution is secure against the legal access attack.

We consider the Bell-LaPadula security model [5] for HAC policies. The model consists of two security properties: (1) The *simple security property* requires that a user cannot read the data at a higher security class. (2) The $\star$*(star)-property* requires that a user cannot write data at a lower security class. To observe the security model in an HKA scheme, we separate SK_i into a write- and read-key pair $(\mathtt{WriteK}_i, \mathtt{ReadK}_i)$ for encrypting and decrypting data at SC_i, respectively. A user at SC_i is authorized to obtain $\mathtt{ReadK}_i$, which is used to read (decrypt) the data at SC_z for $SC_z \preceq SC_i$. For data writing (encryption), the user is only authorized to obtain those $\mathtt{WriteK}_z$ of SC_z for $SC_i \preceq SC_z$. The separation provides flexibility in authorizing data access right of READ or WRITE, or both.

Our Contribution. We provide a practical *CloudHKA* scheme for controlling access for encrypted data in cloud computing. CloudHKA is a novel HKA scheme that observes the Bell-LaPadula security model and efficiently deals with the above issues in user revocation. The design of CloudHKA considers the specific features of cloud services. The specific features include great quantity of outsourced data, large number of users, honest-but-curious cloud servers, frequently changed user set, dynamic access control policies, and data accessing for light-weight mobile devices.

In detail, CloudHKA has the following features.

(1) *Optimal secret key size hold by each user.* Each authorized user at SC_i keeps *one* secret distribution-key $\mathtt{DistK}_i$.
(2) *Outsourceable computation in key derivation.* An authorized user can securely outsource computation for deriving a read-key to CS. He needs to do *three* decryption operations only.
(3) *Outsourceable data update in user revocation.* To revoke a user u, DP can outsource data update operations to CS. CS needs to update header-cipher and a small portion (the size is the same as header-cipher) of body-cipher only. After updating previously encrypted data, u cannot decrypt them with his old distribution-keys and the non-revoked users can decrypt them with their newest distribution-keys. In particular, only the distribution-key of u's associated class needs to be updated. It leads that the key re-distribution occurs in u's associated class only.
(4) *Secure against the legal access attack.* CloudHKA enforces that an authorized user cannot pre-download the needed information for decrypting body-cipher by only

accessing a small portion of encrypted data. Therefore, the legal access attack can be prevented by denying uncommon (large traffic) data access from a user.

(5) *Flexible user access right authorization.* Each user can be authorized the access rights of WRITE or READ, or both.

(6) *Provable-security.* CloudHKA is formally shown to be message indistinguishability secure. Even if CS and a set of users collude, they cannot determine the original datum (that is not entitled to be derived by them) from an encrypted datum with non-negligible probability.

Figure 1 shows the system overview of CloudHKA. The detailed construction is illustrated in Section 3. The system consists of CS, DP, and users. CS is operated by cloud service providers. It is assumed to have bountiful storage space and computation power. DP outsources his data to CS with a self-defined HAC policy $\mathcal{P}$. DP is free to add or delete data in CS and change the access control policy. DP can execute his code over CS to manage his data. A user can be authorized to read or write data in CS. Typically, a user is assumed to have limited storage space and computing power. We assume that CS is always on-line, but DP and users are only on-line when necessary.

Related works. Akl and Taylor [1] first addressed the problem of assigning cryptographic keys in an access hierarchy. They proposed an HKA scheme to enforce an HAC policy. After that, many researches proposed methods for improving performance, supporting dynamic access control policies, or providing distinct features [2, 3, 12, 16, 19, 20, 24, 26, 30]. Atallah et al. formalized the security requirement for HKA schemes and provided an efficient and provably-secure HKA scheme against *key recovery* attacks [3]. Recently, they proposed another scheme with security against *key-indistinguishability* attacks [2]. They also addressed the problem of reducing key derivation time for each user in a deep access hierarchy. The result is obtained by maintaining extra public system information.

Sahai and Waters [23] proposed an attribute-based encryption (ABE) scheme that provides fine-grained data access control. Most ABE schemes enforce monotone access policies over encrypted data [6, 13, 15, 17, 22, 23, 29]. An ABE scheme allows a user to encrypt data into ciphertexts according to a policy. Only the users with a set of attributes that satisfy the policy can decrypt the ciphertexts. Nevertheless, many ABE schemes do not address the issue of dynamic user set and dynamic access policy. Boldyreva et al. [8] addressed the issue of revoking a user with time. They periodically distribute the updated keys to non-revoked users for decrypting newly encrypted data. Yu et al. [29] proposed a revocable ABE scheme for revoking a user immediately. In contrast, Hur and Noh [17] proposed a revocable ABE scheme with immediate attribute and user revocation capability. Sahai et al. [22] proposed the revocable storage ABE scheme that deals with the issue of efficiently preventing a revoked user from decrypting previously encrypted data. In addition to the user revocation issue, decryption time of the existing ABE schemes grows with the depth of access formula. Green et al. [15] proposed a method of uotsoucing the overhead for

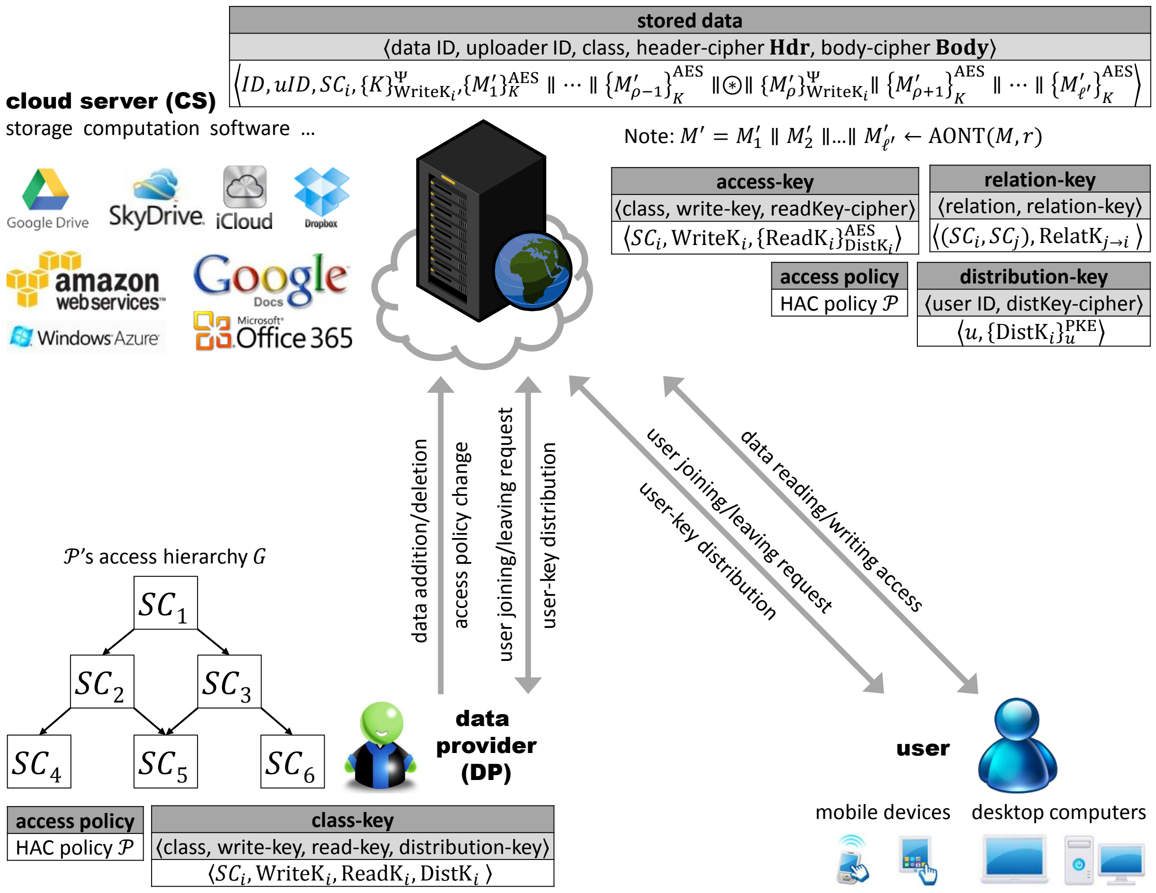
stored data
⟨data ID, uploader ID, class, header-cipher **Hdr**, body-cipher **Body**⟩
$\langle ID, uID, SC_i, \{K\}^{\Psi}_{\mathrm{WriteK}_i}, \{M'_1\}^{\mathrm{AES}}_K \parallel \cdots \parallel \{M'_{\rho-1}\}^{\mathrm{AES}}_K \parallel \circledast \parallel \{M'_\rho\}^{\Psi}_{\mathrm{WriteK}_i} \parallel \{M'_{\rho+1}\}^{\mathrm{AES}}_K \parallel \cdots \parallel \{M'_{\ell'}\}^{\mathrm{AES}}_K \rangle$
cloud server (CS)
storage computation software ...
Google Drive
SkyDrive
iCloud
Dropbox
amazon web services
Google Docs
Windows Azure
Microsoft Office 365
Note: $M' = M'_1 \parallel M'_2 \parallel \ldots \parallel M'_{\ell'} \leftarrow \mathrm{AONT}(M, r)$
access-key
⟨class, write-key, readKey-cipher⟩
$\langle SC_i, \mathrm{WriteK}_i, \{\mathrm{ReadK}_i\}^{\mathrm{AES}}_{\mathrm{DistK}_i} \rangle$
relation-key
⟨relation, relation-key⟩
$\langle (SC_i, SC_j), \mathrm{RelatK}_{j \to i} \rangle$
access policy
HAC policy $\mathcal{P}$
distribution-key
⟨user ID, distKey-cipher⟩
$\langle u, \{\mathrm{DistK}_i\}^{\mathrm{PKE}}_u \rangle$
data addition/deletion
access policy change
user joining/leaving request
user-key distribution
user joining/leaving request
user-key distribution
data reading/writing access
$\mathcal{P}$'s access hierarchy G
SC_1
SC_2
SC_3
SC_4
SC_5
SC_6
data provider (DP)
user
mobile devices
desktop computers
access policy
HAC policy $\mathcal{P}$
class-key
⟨class, write-key, read-key, distribution-key⟩
$\langle SC_i, \mathrm{WriteK}_i, \mathrm{ReadK}_i, \mathrm{DistK}_i \rangle$

Fig. 1. A system overview of our CloudHKA.

users in decryption. Additionally, the size of user secret key or ciphertext in existing ABE schemes grows proportionally in the number of associated attributes. Designing an ABE scheme with a constant size of a user secret key and a ciphertext is still an open problem.

2 Preliminaries

2.1 HAC Policy with the Bell-LaPadula Security Model

An HAC policy $\mathcal{P}$ is a 5-tuple $(\mathcal{SC}, \prec, \mathcal{U}, \mathcal{D}, \lambda)$, where $\mathcal{SC} = \{SC_i : 1 \le i \le n\}$ is a set of security classes, $\prec$ is a binary relation over $\mathcal{SC} \times \mathcal{SC}$, $\mathcal{U}$ is a set of users, $\mathcal{D}$ is a set of data, and $\lambda : \mathcal{U} \cup \mathcal{D} \to \mathcal{SC}$ is a security function that associates each user and datum with a security class. $(\mathcal{SC}, \prec)$ forms a partial order set (poset), where $SC_j \prec SC_i$ means that the security level of class SC_i is higher than that of SC_j. To observe the Bell-LaPadula security model, $\mathcal{P}$ requires the following two properties.

1) *Simple security property*: A user $U \in \mathcal{U}$ cannot read a datum $D \in \mathcal{D}$ if $\lambda(U) \prec \lambda(D)$.
2) $\star$*-property*: A user $U \in \mathcal{U}$ cannot write a datum $D \in \mathcal{D}$ if $\lambda(D) \prec \lambda(U)$.

The poset $(\mathcal{SC}, \prec)$ is represented as a directed graph (access hierarchy) G. Each class SC_i is a node and the relation $SC_j \prec SC_i$ is represented by the directed edge (SC_i, SC_j) in G. G can be simplified by eliminating the edges that are implied by the transitive closure property. For example, Figure 1 has an access hierarchy G with the nodes SC_1, SC_2, ..., SC_6 and edges (SC_1, SC_2), (SC_1, SC_3), (SC_2, SC_4), (SC_2, SC_5), (SC_3, SC_5), and (SC_3, SC_6).

2.2 Proxy Re-Encryption (PRE) Scheme

A proxy re-encryption (PRE) scheme delegates a proxy to re-encrypt a ciphertext under key ek_A into another ciphertext under key ek_B by using the re-encryption key $rk_{A \to B}$ without revealing the plaintext [4, 7, 9, 14, 18, 25, 27]. A PRE scheme Ψ consists of the following six poly-time algorithms:

– $\mathsf{Setup}(\tau) \to (sp, \mathcal{MK})$. On input a security parameter κ, Setup outputs the public system parameter sp (which is explicit used in other algorithms) and master secret key set $\mathcal{MK}$.
– $\mathsf{KeyGen}(\mathcal{MK}, i) \to (ek_i, dk_i)$. On input the master secret key set $\mathcal{MK}$ and an index i, KeyGen outputs a pair of encryption and decryption keys (ek_i, dk_i).
– $\mathsf{ReKeyGen}((ek_i, dk_i), (ek_j, dk_j))^1 \to rk_{i \to j}$. On input two pairs of encryption and decryption key (ek_i, dk_i) and (ek_j, dk_j), $\mathsf{ReKeyGen}$ outputs a re-encryption key $rk_{i \to j}$.
– $\mathsf{Enc}(ek_i, m) \to c_i$. On input an encryption key ek_i and a plaintext m, Enc output a ciphertext c_i.
– $\mathsf{ReEnc}(rk_{i \to j}, c_i) \to c_j$. On input a re-encryption key $rk_{i \to j}$ and ciphertext c_i, ReEnc output a ciphertext c_j under ek_j.
– $\mathsf{Dec}(dk_i, c_i) \to m$. On input a decryption key dk_i and ciphertext c_i, Dec outputs a plaintext m.

These algorithms satisfy the following two requirements.

– For all $(ek_i, dk_i) \leftarrow \mathsf{KeyGen}(\mathcal{MK}, i)$, $\mathsf{Dec}(dk_i, \mathsf{Enc}(ek_i, m)) = m$,
– For all $rk_{i \to j} \leftarrow \mathsf{ReKeyGen}((ek_i, dk_i), (ek_j, dk_j))$, $\mathsf{Dec}(dk_j, \mathsf{ReEnc}(rk_{i \to j}, \mathsf{Enc}(ek_i, m))) = m$.

Ψ is *uni-directional* if $rk_{j \to i}$ cannot be derived from $rk_{i \to j}$. It is *multi-hop* if a ciphertext can be re-encrypted many times in a sequence.

[1] For the construction of a PRE scheme, it is preferable to compute $rk_{i \to j}$ in a *non-interactive* way, that is, without using the secret key dk_j. While using PRE scheme as a building block in our scheme, an interactive PRE scheme is also suitable.

For security, a uni-directional PRE scheme is IND-CPA secure[2] if, for a given ciphertext, a collusive set of malicious entities cannot determine which message, m_0 or m_1, is encrypted under an uncorrupted ek_i. A malicious entity is the proxy, a non-user, or an authorized user with a partial set of decryption keys. The corresponding security game between a challenger $\mathcal{C}_{\text{PRE}}$ and an adversary $\mathcal{A}_{\text{PRE}}$ is described as follows.

IND-CPA Security Game. In the beginning, $\mathcal{C}_{\text{PRE}}$ runs $\text{SETUP}(\tau)$ to generate $(sp, \mathcal{MK})$ and gives sp to $\mathcal{A}_{\text{PRE}}$. $\mathcal{A}_{\text{PRE}}$ determines which message, m_0 or m_1, corresponds to a challenged ciphertext $c_{i^*} = \mathsf{Enc}(ek_{i^*}, m_b)$ for $b \in \{0,1\}$, where ek_{i^*} is the encryption key of a target index i^* chosen by $\mathcal{C}_{\text{PRE}}$ and m_0, m_1 are two plaintexts chosen by $\mathcal{A}_{\text{PRE}}$. $\mathcal{A}_{\text{PRE}}$ is allowed to query the following oracles in an arbitrary order. We say that a ciphertext c_i is re-encryptable into another ciphertext c_j if all re-encryption keys on the path from index i to j are returned for queries. A ciphertext c_i is decryptable if the decryption key dk_i is corrupted or c_i is re-encryptable into a decryptable ciphertext.

- $\mathcal{O}_{\text{uc}}(i) \to ek_i$. For the query of an index i, the uncorrupted key extraction oracle $\mathcal{O}_{\text{uc}}$ returns ek_i.
- $\mathcal{O}_{\text{c}}(i) \to \{(ek_i, dk_i), \perp\}$. For the query of an index i, the corrupted key extraction oracle $\mathcal{O}_{\text{c}}$ returns the key pair (ek_i, dk_i) if $i \neq i^*$ and c_{i^*} cannot be re-encrypted into a ciphertext under ek_i. Otherwise, $\mathcal{O}_{\text{c}}$ returns nothing.
- $\mathcal{O}_{\text{rk}}(i,j) \to \{rk_{i \to j}, \perp\}$. For the query of an index pair (i,j), the re-encryption extraction oracle $\mathcal{O}_{\text{rk}}$ returns $rk_{i \to j}$ if c_{i^*} cannot be re-encrypted into a decryptable ciphertext. Otherwise, $\mathcal{O}_{\text{rk}}$ returns nothing.

We say that $\mathcal{A}_{\text{PRE}}$ wins the IND-CPA security game if he can determine which m_b for $b \in \{0,1\}$ is the original message of c_{i^*} under ek_{i^*}. A PRE scheme is IND-CPA secure if, for all poly-time $\mathcal{A}_{\text{PRE}}$, the advantage of $\mathcal{A}_{\text{PRE}}$ for winning the IND-CPA security game is negligible in τ.

2.3 All-Or-Nothing Transformation

All-or-nothing transformation (AONT) AONT is an unkeyed and randomized function with the property that it is hard to compute the whole message unless the entire function output is known [21]. AONT maps an ℓ-block message $X = X_1||X_2||\cdots||X_\ell$ and a random string r to an ℓ'-block string $Y = Y_1||Y_2||\cdots||Y_{\ell'}$. AONT satisfies the following properties:

- Given X and r, $Y \leftarrow \mathsf{AONT}(X, r)$ can be computed efficiently.
- Given Y, $X \leftarrow \mathsf{AONT}^{-1}(Y)$ can be computed efficiently.
- If any block of Y is lost, it is infeasible to recover X.

[2] A stronger security notion of a PRE scheme is IND-CCA secure. The adversaries are given access to a decryption oracle. Constructing an IND-CCA secure PRE scheme is considered harder. Our scheme needs only an IND-CPA secure PRE scheme.

3 Our CloudHKA

3.1 Overview

The construction of CloudHKA is based on a uni-directional and multi-hop PRE scheme Ψ. Assume that the given HAC policy is $\mathcal{P}$, which is represented by a directed graph $G = (V, E)$. For each class $SC_i \in V$, DP generates a pair of write- and read-key ($\mathtt{WriteK}_i, \mathtt{ReadK}_i$). A message that is encrypted by using $\mathtt{WriteK}_i$ can be decrypted by using $\mathtt{ReadK}_i$. A user who obtains the write-key $\mathtt{WriteK}_i$ is authorized the WRITE right for SC_i. A user who obtains the read-key $\mathtt{ReadK}_i$ is authorized the READ right for SC_i and its lower classes. Although the pair of write- and read-key is like the pair of public- and private-key of a public-key system, neither of them can be published to a public domain in CloudHKA. A write-key $\mathtt{WriteK}_z$ is given to a user at SC_i (through a secure channel) when he requests to write data into SC_z for $SC_i \preceq SC_z$.

In data outsourcing, a datum M at SC_i is transformed into

$$M' = M'_1||M'_2||\dots||M'_{\ell'} \leftarrow \mathsf{AONT}(M, r)$$

and then encrypted in the form

$$\begin{aligned}&\langle \text{data ID, uploader ID, class, header-cipher, body-cipher,} \rangle \\ &= \langle ID, uID, SC_i, \mathtt{Hdr}^{SC_i}_{ID} = \{K\}^{\Psi}_{\mathtt{WriteK}_i}, \mathtt{Body}^{SC_i}_{ID} = \{M'_1\}^{\mathsf{AES}}_K||\dots||\{M'_{\rho-1}\}^{\mathsf{AES}}_K|| \\ &\qquad \circledast ||\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_i}||\{M'_{\rho+1}\}^{\mathsf{AES}}_K||\dots||\{M'_{\ell'}\}^{\mathsf{AES}}_K\rangle, \end{aligned} \tag{1}$$

where

- uID is a user who stores (uploads) his data into CS,
- $\{K\}^{\Psi}_{\mathtt{WriteK}_i}$ and $\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_i}$ are respectively the ciphertexts of a randomly chosen AES encryption key K and ρ-th block of M' under $\mathtt{WriteK}_i$,
- $\rho \in \{1, 2, \cdots, \ell'\}$ only known by CS and DP,
- $\circledast$ is a special symbol for marking the start position of $\mathtt{Body}^{SC_i}_{ID}[\rho]$, and
- $\{M'_\omega\}^{\mathsf{AES}}_K$ is the ciphertext of M'_ω for $\omega \in \{1, 2, \dots, \ell'\} \setminus \{\rho\}$ under K.

Before CS storing an encrypted datum into SC_i, he should authenticate that the associated class of a data uploader is no lower than SC_i. It observes the $\star$-property.

For each relation $(SC_j, SC_i) \in E$, DP generates a (public) relation-key $\mathtt{RelatK}_{i\to j}$ that is used to re-encrypt a header-cipher and body-cipher (ρ-th block) of SC_i into that of SC_j. Assume that a user who is authorized the READ right of SC_j wants to read (decrypt) datum ID encrypted as (1). CS re-encrypts $\{K\}^{\Psi}_{\mathtt{WriteK}_i}$ and $\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_i}$ into $\{K\}^{\Psi}_{\mathtt{WriteK}_j}$ and $\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_j}$ by using the relation-key $\mathtt{RelatK}_{i\to j}$. The user then decrypts $\{K\}^{\Psi}_{\mathtt{WriteK}_j}$ and $\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_j}$ to obtain K and M'_ρ by using $\mathtt{ReadK}_j$. By using K to decrypt $\{M'_\omega\}^{\mathsf{AES}}_K$ for $\omega \in \{1, 2, \dots, \ell'\} \setminus \{\rho\}$, the user obtains M'_ω and combines it with M'_ρ to recover $M \leftarrow \mathsf{AONT}^{-1}(M')$. The concept can be easily extended for the case with $d = dst_G(SC_j, SC_i) > 1$, where $dst_G(SC_j, SC_i)$ is the distance between SC_j and SC_i in the access hierarchy G.

To revoke a user u at SC_i, DP does the following procedures.

- Removing Write right: DP simply removes u from his SC_i in $\mathcal{P}$. Then, u's Write right of SC_z for $SC_i \preceq SC_z$ is removed since he cannot pass CS's authentication in data writing.
- Removing Read right: This part can be separated into two cases.
  (1) Preventing u from decrypting newly encrypted data at SC_z for $SC_z \preceq SC_i$: DP re-generates SC_z's key pair and related relation-keys. Then, the new data at SC_z will be encrypted under the new write-key of SC_z. The new read-key of SC_z is distributed to the non-revoked users only.
  (2) Preventing u from decrypting previously encrypted data at SC_z for $SC_z \preceq SC_i$: DP sends CS a (public) transform-key $\mathtt{TranK}_z$ for transforming (re-encrypting) SC_z's header-ciphers and body-ciphers under the old write-key into the new one under the new write-key. Thus, only the non-revoked users who obtain the new read-keys can decrypt the updated header-ciphers and body-ciphers.

Remark. The data encryption form in (1) enforces a user accesses the whole body-cipher for decryption. Assume that an authorized user u at SC_i wants to access a datum ID in (1). u needs to obtain K and M'_ρ by using $\mathtt{ReadK}_i$ so that he can recover M. To obtain M'_ρ, u needs to find the start position of $\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_i}$. Since u does not know ρ, he needs to find $\circledast$ by accessing whole $\mathtt{Body}^{SC_i}_{ID}$ (or the part before meeting $\circledast$.) This design effectively prevents the legal access attack. In the legal access attack, u pre-downloads K and M'_ρ for each datum. However, u does not know the position of $\circledast$ until the whole body-cipher is retrieved. In CloudHKA, a authorized, but malicious, user needs to access a large portion of a data for pre-downloading the needed information for decryption. A large collection of pre-downloaded information will cause traffic in accessing. A traffic limitation mechanism (with a specified policy according to the system) can easily deny the legal access attack. For example, in a protected database, the amount of transmitted data for each user in a time period is often limited.

3.2 The Construction

Let $\Psi = (\Psi.\mathsf{Setup}, \Psi.\mathsf{KeyGen}, \Psi.\mathsf{ReKeyGen}, \Psi.\mathsf{Enc}, \Psi.\mathsf{ReEnc}, \Psi.\mathsf{Dec})$ be a uni-directional and multi-hop PRE scheme. Let AES be a symmetric key encryption scheme with key generation, encryption, and decryption algorithms ($\mathsf{AES.G}$, $\mathsf{AES.E}$, $\mathsf{AES.D}$). Let PKE be an asymmetric key (or public-key) encryption scheme with key generation, encryption, and decryption algorithms ($\mathsf{PKE.G}$, $\mathsf{PKE.E}$, $\mathsf{PKE.D}$). Let AONT be an all-or-nothing transformation function that maps an ℓ-block message and a random string to an ℓ'-block string.

To simplify the description of our scheme, we assume that two system entities of CS, DP, and users can authenticate the identity of each other. The integrity and correctness of messages or data transmitted between two system entities can be verified by each other.

System Setup. DP defines an initial HAC policy $\mathcal{P} = (\mathcal{SC}, \prec, \mathcal{U}, \mathcal{D}, \lambda)$ with n security classes SC_1, SC_2, …, SC_n. Assume that $\mathcal{P}$ is represented as an access hierarchy $G = (V, E)$. Then, DP generates $(sp, \mathcal{MK}) \leftarrow \Psi.\mathsf{Setup}(\kappa)$ with a given security parameter κ and associates each $SC_i \in V$ with the following keys and tokens.

- Write- and read-key pair $(\mathtt{WriteK}_i, \mathtt{ReadK}_i) \leftarrow \Psi.\mathsf{KeyGen}(\mathcal{MK}, i)$.
- Distributed-key $\mathtt{DistK}_i \leftarrow \mathsf{AES.G}(\kappa)$.
- ReadKey-cipher $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i} \leftarrow \mathsf{AES.E}(\mathtt{DistK}_i, \mathtt{ReadK}_i)$.

DP associates each relation $(SC_j, SC_i) \in E$ a relation-key

$$\mathtt{RelatK}_{i\to j} \leftarrow \Psi.\mathsf{ReKeyGen}((\mathtt{WriteK}_i, \mathtt{ReadK}_i), (\mathtt{WriteK}_j, \mathtt{ReadK}_j)).$$

Finally, DP uploads $\mathcal{P}$, $\langle SC_i$, $\mathtt{WriteK}_i$, $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i}\rangle$ for $SC_i \in V$, and $\langle (SC_j, SC_i)$, $\mathtt{RelatK}_{i\to j}\rangle$ for $(SC_j, SC_i) \in E$ to CS. DP keeps $\mathcal{P}$ and $\langle SC_i$, $\mathtt{WriteK}_i$, $\mathtt{ReadK}_i$, $\mathtt{DistK}_i\rangle$ for $SC_i \in V$ locally. Each user u of the system generates his public- and private-key pair (pk_u, sk_u).

Access Right Authorization. Assume that DP associates a user u with a class SC_i in $\mathcal{P}$. To authorize u the READ right of SC_z for $SC_z \prec SC_i$, DP uses u's public-key pk_u to encrypt the distribution-key $\mathtt{DistK}_i$ as a distKey-cipher

$$\{\mathtt{DistK}_i\}^{\mathsf{PKE}}_u \leftarrow \mathsf{PKE.E}(pk_u, \mathtt{DistK}_i)$$

and uploads $\langle u, \{\mathtt{DistK}_i\}^{\mathsf{PKE}}_u\rangle$ to CS. CS forwards $\{\mathtt{DistK}_i\}^{\mathsf{PKE}}_u$ to u and u decrypts it to obtain $\mathtt{DistK}_i$. To observe the $\star$-property, CS gives the current $\mathtt{WriteK}_z$ to u when u requests to write data into SC_z for $SC_i \preceq SC_z$.

Data Writing. To write a datum M into SC_i, an uploader uID computes $M' \leftarrow \mathsf{AONT}(M, r)$, where r is a random string. uID then generates a data encryption key $K \leftarrow \mathsf{AES.G}(\kappa)$, randomly chooses an index $\rho \in \{1, 2, \ldots, \ell'\}$, and sends CS the encrypted data in the form

$$\begin{aligned}\langle \rho, SC_i, \mathtt{C}_1, \mathtt{C}_2\rangle = \langle \rho, SC_i, \{K\}^{\Psi}_{\mathtt{WriteK}_i}, \{M'_1\}^{\mathsf{AES}}_K || \ldots || \{M'_{\rho-1}\}^{\mathsf{AES}}_K || \\ \circledast || \{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_i} || \{M'_{\rho+1}\}^{\mathsf{AES}}_K || \ldots || \{M'_{\ell'}\}^{\mathsf{AES}}_K\rangle.\end{aligned}$$

After receiving the data, CS checks the validity of the writing request from uID. If uID is associated with SC_z for $SC_i \preceq SC_z$, CS selects a unique data identity ID and stores the data with the format

$$\begin{aligned}&\langle \text{data ID, uploader ID, class, header-cipher, body-cipher}\rangle \\ &= \langle ID, uID, SC_i, \mathtt{Hdr}^{SC_i}_{ID} = \mathtt{C}_1, \mathtt{Body}^{SC_i}_{ID} = \mathtt{C}_2\rangle. \qquad (2)\end{aligned}$$

CS keeps ρ as a secret. The value will be used when CS needs to update body-ciphers.

Data Reading. Assume that an authorized user u at SC_j wants to read a datum encrypted as (2). If $SC_i \preceq SC_j$, CS re-encrypts the header-cipher and body-cipher as follows. Let $d = dst_G(SC_j, SC_i)$.

- Extract the relation-keys on the path from SC_i to SC_j as $\texttt{RelatK}_{v_1 \to v_2}$, $\texttt{RelatK}_{v_2 \to v_3}$, ..., $\texttt{RelatK}_{v_d \to v_{d+1}}$, where $v_1 = i$ and $v_{d+1} = j$.
- For each v_z from v_1 to v_d, replace $\{K\}^{\Psi}_{\texttt{WriteK}_{v_z}}$ and $\{M'_\rho\}^{\Psi}_{\texttt{WriteK}_{v_z}}$ as

$$\{K\}^{\Psi}_{\texttt{WriteK}_{v_{z+1}}} \leftarrow \Psi.\mathsf{ReEnc}(\texttt{RelatK}_{v_z \to v_{z+1}}, \{K\}^{\Psi}_{\texttt{WriteK}_{v_z}}),$$

$$\{M'_\rho\}^{\Psi}_{\texttt{WriteK}_{v_{z+1}}} \leftarrow \Psi.\mathsf{ReEnc}(\texttt{RelatK}_{v_z \to v_{z+1}}, \{M'_\rho\}^{\Psi}_{\texttt{WriteK}_{v_z}}).$$

CS returns $\langle \texttt{C}_0, \texttt{C}_1, \texttt{C}_2 \rangle = \langle \{\texttt{ReadK}_j\}^{\mathsf{AES}}_{\texttt{DistK}_j}, \texttt{Hdr}^{SC_j}_{ID}, \texttt{Body}^{SC_j}_{ID} \rangle$ to u. After receiving the ciphertexts, u decrypts $\texttt{C}_0$ to obtain $\texttt{ReadK}_j$ by using his (newest) $\texttt{DistK}_j$. u finds $\circledast$ to extract $\{M'_\rho\}^{\Psi}_{\texttt{WriteK}_j}$ from $\texttt{C}_2$. Then, u decrypts $\texttt{C}_1$ and $\{M'_\rho\}^{\Psi}_{\texttt{WriteK}_j}$ to obtain K and M'_ρ by using $\texttt{ReadK}_j$. u then decrypts the other blocks of $\texttt{C}_2$ to obtain M'_ω for $\omega \in \{1, 2, \ldots, \ell'\} \setminus \{\rho\}$ by using K. Finally, u combines M'_ρ and M'_ω's as M' and recovers $M \leftarrow \mathsf{AONT}^{-1}(M')$.

Data Deletion. A datum can be deleted by its uploader only. To delete a datum ID, its uploader with identity uID sends a deletion request of ID to CS. CS deletes the datum ID and its associated information.

User Revocation with Outsourceable Data Update. Assume that DP wants to revoke a user u from SC_i.

- Removing u's WRITE right: DP simply updates his HAC policy. Hereafter, when u wants to write data into SC_z for $SC_i \preceq SC_z$, he cannot pass CS's validity check in data writing.
- Removing u's READ right:
  (1) To remove u's READ right for newly encrypted data at SC_z for $SC_z \preceq SC_i$: DP re-generates the key pair of SC_z as $(\texttt{WriteK}'_z, \texttt{ReadK}'_z)$ and affected relation-keys. DP then updates the affected readKey-ciphers as follows.
  - For $SC_z \prec SC_i$, DP updates SC_z's readKey-cipher as $\{\texttt{ReadK}'_z\}^{\mathsf{AES}}_{\texttt{DistK}_z}$.
  - For SC_i, DP updates SC_i's distribution-key as $\texttt{DistK}'_i$ and readKey-cipher as $\{\texttt{ReadK}'_i\}^{\mathsf{AES}}_{\texttt{DistK}'_i}$.

  DP distributes the updated distribution-key $\texttt{DistK}'_i$ to the non-revoked users at SC_i. For each non-revoked user $\bar{u}$, DP updates $\langle \bar{u}, \{\texttt{DistK}_i\}^{\mathsf{PKE}}_{\bar{u}} \rangle$ as $\langle \bar{u}, \{\texttt{DistK}'_i\}^{\mathsf{PKE}}_{\bar{u}} \rangle$.

  (2) To remove u's READ right for previously encrypted data at SC_z for $SC_z \preceq SC_i$: DP sends CS a transform-key

$$\texttt{TranK}_z \leftarrow \Psi.\mathsf{ReKeyGen}((\texttt{WriteK}_z, \texttt{ReadK}_z), (\texttt{WriteK}'_z, \texttt{ReadK}'_z)).$$

  CS uses $\texttt{TranK}_z$ to update each SC_z's header-cipher and ρ-th block of body-cipher as

$$\{K\}^{\Psi}_{\texttt{WriteK}'_z} \leftarrow \Psi.\mathsf{ReEnc}(\texttt{TranK}_z, \{K\}^{\Psi}_{\texttt{WriteK}_z}),$$

$$\{M'_\rho\}^{\Psi}_{\mathtt{WriteK}'_z} \leftarrow \Psi.\mathsf{ReEnc}(\mathtt{TranK}_z, \{M'_\rho\}^{\Psi}_{\mathtt{WriteK}_z}).$$

Updates of Access Hierarchy. The update operations include relation insertion, relation deletion, class insertion, and class deletion.

- *Relation insertion.* To insert a new relation (SC_j, SC_i), DP generates a new $\mathtt{RelatK}_{i\to j} \leftarrow \Psi.\mathsf{ReKeyGen}((\mathtt{WriteK}_i, \mathtt{ReadK}_i), (\mathtt{WriteK}_j, \mathtt{ReadK}_j))$ and uploads the updated HAC policy and $\langle (SC_j, SC_i), \mathtt{RelatK}_{i\to j}\rangle$ to CS.
- *Relation deletion.* To delete a relation (SC_j, SC_i), DP needs to prevent the users at SC_i from re-encrypting the header-ciphers and body-cipher of SC_z into that of SC_j for $SC_z \preceq SC_i$. The procedure is like to revoke a "psuedo-user" from SC_i. The differences are that DP does not need to re-generate (1) SC_i's distribution-key and readKey-cipher and (2) (SC_j, SC_i)'s relation-key. There is no need to distribute the new distribution-key of SC_i.
- *Class insertion.* To insert a class SC_i, DP generates $(\mathtt{WriteK}_i, \mathtt{ReadK}_i)$, $\mathtt{DistK}_i$, and $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i}$ and uploads the updated HAC policy and $\langle SC_i$, $\mathtt{WriteK}_i$, $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i}\rangle$ to CS. DP then runs the relation insertion procedure to insert the incoming and outgoing relations of SC_i.
- *Class deletion.* To delete a class SC_i, DP deletes SC_i's associated parameters in CS and runs the relation deletion procedure for every SC_i's incoming and outgoing relations.

4 Analysis

4.1 Performance Analysis

This section illustrates the performance of CloudHKA. We compare CloudHKA with the first HKA scheme [1] and recent two HKA schemes [2, 3] in Table 1. To our best knowledge, the schemes in [2, 3] provide most features up to now and are provably-secure.

Storage cost. In CloudHKA, each user at SC_i stores the distribution-key $\mathtt{DistK}_i$. In the other three schemes, the secret key size for each user is also one.

Key derivation cost. In CloudHKA, when a user u at SC_j requests to read a datum at SC_i for $SC_i \preceq SC_j$, CS runs $d = dst_G(SC_j, SC_i)$ times of $\Psi.\mathsf{ReEnc}$ to re-encrypt the header-cipher under $\mathtt{WriteK}_i$ into the header-cipher under $\mathtt{WriteK}_j$. u then runs one $\mathsf{AES.D}$ to obtain $\mathtt{ReadK}_j$ and two $\Psi.\mathsf{Dec}$ to obtain K and M'_ρ. The total key derivation cost of the other three schemes are also linear in d. Nevertheless, only CloudHKA can outsource most of the computation operations to CS so that a user only needs constant computation time in key derivation. Note that in [1], although the computation operation only contains a modular exponentiation, the size of the used group equals to the size of the multiplication of d large co-prime numbers. The computation time in key derivation is still linear to d.

User revocation cost. In CloudHKA, to revoke a user u at SC_i, the rekey operation for DP contains: (1) $|V_i|$ times of $\Psi.\mathsf{KeyGen}$, $\Psi.\mathsf{ReKeyGen}$, and $\mathsf{AES.E}$, (2)

Table 1. A comparison of our CloudHKA with previous HKA schemes

	AT [1]	AFB [3]	ABFF [2]	**CloudHKA**
	Storage cost			
#(user secret key)	1	1	1	1
	Key derivation cost (for a user u at SC_j to derive a key of SC_i)			
Full computation	t_{Exp}	$d \cdot (t_H + t_{\mathrm{XOR}})$	$2d \cdot (t_H + t_{\mathsf{AES.D}}) + t_H$	$2d \cdot t_{\Psi.\mathsf{ReEnc}} + 2t_{\Psi.\mathsf{Dec}} + t_{\mathsf{AES.D}}$
Outsourceable computation	-	-	-	$2d \cdot t_{\Psi.\mathsf{ReEnc}}$
	User revocation cost (revoking a user u from SC_i)			
Rekey	-	$O(\|E_i\| + \sum_{SC_z \in V_i} n_z)$	$O(\|V_i\| + \|E_i\| + n_i)$	$O(\|V_i\| + \|E_i\| + n_i)$
Full data update	-	$\#c(u) \cdot (t_{\mathsf{AES.D}} + t_{\mathsf{AES.E}})$	$\#c(u) \cdot (t_{\mathsf{AES.D}} + t_{\mathsf{AES.E}})$	$\|V_i\| \cdot t_{\Psi.\mathsf{ReKeyGen}} + 2 \cdot \#c(u) \cdot t_{\Psi.\mathsf{ReEnc}}$
Outsourceable data update	-	-	-	$2 \cdot \#c(u) \cdot t_{\Psi.\mathsf{ReEnc}}$
	User access right authorization			
Read-Write	✓	✓	✓	✓
Read-only	-	-	-	✓
Write-only	-	-	-	✓
	Security			
Security game	-	Key-Recovery	Key-Indistinguishability	Message-Indistinguishability
Building block	-	PRF family	PRF family and AES	Uni-directional PRE

† Exp: A modular exponentiation over a large group.
† H: A cryptographic hash function.
† t_f: The computation time of function f.
† $\#c(u)$: The number of decryptable data ciphertexts of u.
† $G = (V, E)$: An access hierarchy.
† $d = dst_G(SC_j, SC_i)$: The distance between SC_j and SC_i in the access hierarchy G.
† V_i: The set of SC_i and its lower classes, i.e., $V_i = \{SC_z : SC_z \preceq SC_i\}$.
† E_i: The set of relations related to the classes in V_i, i.e., $E_i = \{(SC_v, SC_z) : SC_z \in V_i\}$.
† n_i: The number of users associated with SC_i.

one $\mathsf{AES.G}$, (3) $|E_i|$ times of $\Psi.\mathsf{ReKeyGen}$, (4) n_i times of $\mathsf{PKE.E}$, and (5) $2 \cdot \#c(u)$ times of $\Psi.\mathsf{ReEnc}$, where $V_i = \{SC_z : SC_z \preceq SC_i\}$ is the set of SC_i and its lower classes, $E_i = \{(SC_\xi, SC_z) : SC_z \in V_i\}$ is the set of relations related to the classes in V_i, n_i is the number of users (excluding u) at SC_i, and $\#c(u)$ is the number of decryptable data ciphertexts of u. The distribution-key update only occurs in the class SC_i, the distribution of the new distribution-key is needed for the non-revoked users at SC_i only. Note that the extended HKA scheme in [2] is the first HKA scheme supporting this kind of *local key re-distribution* property. To let the non-revoked users decrypt previously encrypted data, in CloudHKA, DP only needs to run $|V_i|$ times of $\Psi.\mathsf{ReKeyGen}$ to generate the needed transform-keys to CS. CS can update every u's decryptable header-cipher into the one under the new write-key by using $\Psi.\mathsf{ReEnc}$. In other three HKA schemes, to update all u's decryptable ciphertexts, DP needs to download them, decrypt them with old data encryption keys, encrypt them with new data encryption keys, and then upload them to CS.

4.2 Bell-LaPadula Security Model Observation

Our CloudHKA observes the simple security property and $\star$-property. The uni-directional property of Ψ ensures that a relation-key $\mathtt{RelatK}_{j \to i}$ cannot be reversed. Thus, it is not possible to compute the inverted header-cipher and body-cipher re-encryptions from class SC_i to its lower class SC_j. Therefore, CloudHKA observes the simple security property. The $\star$-property is observed in CloudHKA since CS only allows a user at SC_i to write data into SC_i and its higher classes. Note that giving all write-keys to CS does not violate the $\star$-property since CS does not have the Read right of any class in the policy.

4.3 Security Analysis

In this section, we formally show that CloudHKA ensures data confidentiality based on the security of PRE schemes. We also demonstrate that the user revocation mechanism in CloudHKA removes the access rights of a revoked user.

To simplify our security analysis, we assume that the encryption schemes AES and PKE are IND-CPA secure. For example, AES with CBC mode and ElGamal suit our need, respectively. The IND-CPA security of an encryption scheme ensures that an unauthorized user cannot distinguish an encrypted distribution-key, read-key, or datum from an encrypted random string. By the assumption, CloudHKA ensures that only an authorized user can obtain legal distribution-keys and read-keys. Then, the security of our CloudHKA only relies on the security of PRE scheme Ψ for protecting (K, M'_ρ).

4.3.1 User- and Read-Key Authorization

In CloudHKA, DP stores $\mathtt{DistK}_i$ as $\{\mathtt{DistK}_i\}_u^{\mathsf{PKE}}$ under user u's individual public-key pk_u for a user u at SC_i. Only u can decrypt $\{\mathtt{DistK}_i\}_u^{\mathsf{PKE}}$ to obtain $\mathtt{DistK}_i$. DP stores $\mathtt{ReadK}_i$ as $\{\mathtt{ReadK}_i\}_{\mathtt{DistK}_i}^{\mathsf{AES}}$. Only an authorized user who is assigned $\mathtt{DistK}_i$ can obtain $\mathtt{ReadK}_i$.

4.3.2 Data Confidentiality

Our goal is to show that even if CS and a set of malicious users collude, for a given SC_{i^*}'s header-cipher and ρ-th block body-cipher pair $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$ that encrypts either $m_0 = (K_0, M'_{\rho,0})$ or $m_1 = (K_1, M'_{\rho,1})$, it is hard for the collusive entities to determine the original message of the ciphertext pair. The original messages m_0 and m_1 are chosen by the collusive entities. A malicious user can be a non-user, a revoked user, or an authorized user. They are not authorized to read the data at SC_z for $SC_{i^*} \preceq SC_z$. The corresponding security game between a challenger $\mathcal{C}_{\text{CloudHKA}}$ and an adversary $\mathcal{A}_{\text{CloudHKA}}$ is described as follows.

Message-Indistingushability (MI) Game of CloudHKA. In the beginning, $\mathcal{C}_{\text{CloudHKA}}$ sets up the system and gives the public information (as the information uploaded to CS) to $\mathcal{A}_{\text{CloudHKA}}$. $\mathcal{A}_{\text{CloudHKA}}$ determines which message, $m_0 = (K_0, M'_{\rho,0})$ or $m_1 = (K_1, M'_{\rho,1})$, corresponds to a challenged ciphertext pair $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$, where SC_{i^*} is a target class chosen by $\mathcal{C}_{\text{CloudHKA}}$ and m_0, m_1 are chosen by $\mathcal{A}_{\text{CloudHKA}}$. $\mathcal{A}_{\text{CloudHKA}}$ is allowed to query a distribution-key corruption oracle

$\mathcal{O}_{\text{distKey}}(SC_z)$ to obtain the distribution-key $\mathtt{DistK}_z$. Nevertheless, $\mathcal{A}_{\text{CloudHKA}}$ is not allowed to query $\mathcal{O}_{\text{userKey}}(SC_z)$ for $SC_{i^*} \preceq SC_z$.

We say that $\mathcal{A}_{\text{CloudHKA}}$ wins the MI game if he can determine which m_b for $b \in \{0,1\}$ is the original message pair of $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$. Our CloudHKA is message-indistinguishable if, for all poly-time $\mathcal{A}_{\text{CloudHKA}}$, the advantage of $\mathcal{A}_{\text{CloudHKA}}$ for winning the CPA-security game is negligible in κ.

Without loss of generality, we consider a static adversary only since an adaptive adversary is no more powerful than a static one in our case [2, 3]. A static adversary $\mathcal{A}'_{\text{CloudHKA}}$ guesses a target class, say SC_{i^*}, which is the challenged class in the original MI game of CloudHKA. Then $\mathcal{A}'_{\text{CloudHKA}}$ is directly given the challenged ciphertext pair $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$ along with the maximum amount of corrupted distribution-keys, i.e., the distribution-keys $\{\mathtt{DistK}_z : SC_{i^*} \not\preceq SC_z\}$. Since the guess of SC_{i^*} is correct with $1/|V|$ probability, it only affects $\mathcal{A}_{\text{CloudHKA}}$'s winning advantage by a factor of $|V|$. We describe the modified MI game between a challenger $\mathcal{C}'_{\text{CloudHKA}}$ and an adversary $\mathcal{A}'_{\text{CloudHKA}}$ as follows.

Modified MI Game of CloudHKA. In the beginning, $\mathcal{A}'_{\text{CloudHKA}}$ chooses two messages $m_0 = (K_0, M'_{\rho,0})$ and $m_1 = (K_1, M'_{\rho,1})$ to $\mathcal{C}'_{\text{CloudHKA}}$. $\mathcal{C}'_{\text{CloudHKA}}$ chooses a target SC_{i^*}, sets up the system, and gives $\mathcal{A}'$ the public information and a challenged ciphertext pair $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$ alongs with the corrupted distribution-keys $\{\mathtt{DistK}_z : SC_{i^*} \not\preceq SC_z\}$. $\mathcal{A}_{\text{CloudHKA}}$ determines which message, m_0 or m_1, corresponds to $(\mathtt{Hdr}_{ID}^{SC_{i^*}}, \mathtt{Body}_{ID}^{SC_{i^*}}[\rho])$.

The following theorwm shows that our CloudHKA is message-indistinguishable based on the modified MI game of CloudHKA.

Theorem 1. *Our CloudHKA is message-indistinguishable if the underlying PRE scheme Ψ is IND-CPA secure.*

Proof. The proof is a standard reduction argument. Assume that there exists a poly-time adversary $\mathcal{A}'_{\text{CloudHKA}}$ who wins CloudHKA's modified MI game with non-negligible advantage ϵ. We can employ $\mathcal{A}'_{\text{CloudHKA}}$ to construct a poly-time algorithm $\mathcal{B}$ for breaking Ψ's IND-CPA security with advantage ϵ. In the reduction, $\mathcal{B}$ is treated as the challenger $\mathcal{C}'_{\text{CloudHKA}}$ in CloudHKA's modified MI game and Ψ's adversary $\mathcal{A}_{\text{PRE}}$ in PRE's IND-CPA security game.

In the beginning, $\mathcal{A}'_{\text{CloudHKA}}$ chooses two messages $m_0 = (K_0, M'_{\rho,0})$ and $m_1 = (K_1, M'_{\rho,1})$. $\mathcal{B}$ takes an HAC policy $\mathcal{P}$ with an access hierarchy $G = (V, E)$, a target class SC_{i^*}, a security parameter κ, and (m_0, m_1) as inputs. At the same time, $\mathcal{B}$ takes $\mathcal{A}'_{\text{CloudHKA}}$, $\mathcal{C}_{\text{PRE}}$, $\mathcal{O}_{\text{uc}}$, $\mathcal{O}_{\text{c}}$, $\mathcal{O}_{\text{rk}}$, $\mathsf{AES.G}$, and $\mathsf{AES.E}$ as oracles. Let $x \xleftarrow{\$} \mathcal{X}$ denote that x is chosen from the set $\mathcal{X}$ randomly.

$\mathcal{B}^{\mathcal{A}'_{\text{CloudHKA}}, \mathcal{C}_{\text{PRE}}, \mathcal{O}_{\text{uc}}, \mathcal{O}_{\text{c}}, \mathcal{O}_{\text{rk}}, \mathsf{AES.G}, \mathsf{AES.E}}(\mathcal{P}, SC_{i^*}, \kappa, (m_0, m_1))$

Step 1. For each $SC_i \in V$:

– If $SC_{i^*} \preceq SC_z$: Set $(\mathtt{WriteK}_i, \mathtt{ReadK}_i) \leftarrow (\mathcal{O}_{\mathrm{uc}}(i), \lambda)$, $\mathtt{DistK}_i \leftarrow \lambda$, and $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i} \xleftarrow{\$} \mathsf{Range}(\mathsf{AES.E})$, where λ is an empty string and $\mathsf{Range}(\mathsf{AES.E})$ is the ciphertext space of AES.
– If $SC_{i^*} \npreceq SC_z$: Set $(\mathtt{WriteK}_i, \mathtt{ReadK}_i) \leftarrow \mathcal{O}_{\mathrm{c}}(i)$, $\mathtt{DistK}_i \leftarrow \mathsf{AES.G}(\kappa)$, and $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i} \leftarrow \mathsf{AES.E}(\mathtt{DistK}_i, \mathtt{ReadK}_i)$.

For each $(SC_j, SC_i) \in E$: Set $\mathtt{RelatK}_{i \to j} \leftarrow \mathcal{O}_{\mathrm{rk}}(i, j)$.

Step 2. Get $c_{i^*} \leftarrow \mathcal{C}_{\mathrm{PRE}}(\langle i^*, (m_0, m_1)\rangle)$. Give $\mathcal{A}'_{\mathrm{CloudHKA}}$ the public information and a challenged $(\mathtt{Hdr}^{SC_{i^*}}_{ID}, \mathtt{Body}^{SC_{i^*}}_{ID}[\rho]) \leftarrow c_{i^*}$ alongs with the corrupted distribution-keys $\{\mathtt{DistK}_z : SC_{i^*} \npreceq SC_z\}$. The public information contains $\mathcal{P}$, $\langle SC_i$, $\mathtt{WriteK}_i$, $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i}\rangle$ and $\langle (SC_j, SC_i)$, $\mathtt{RelatK}_{i \to j}\rangle$ for $SC_i \in V$ and $(SC_j, SC_i) \in E$.

Step 3. Return the output of $\mathcal{A}'_{\mathrm{CloudHKA}}$.

In $\mathcal{B}$, the distribution of the transcripts for $\mathcal{A}'_{\mathrm{CloudHKA}}$ is indistinguishable from the distribution of the information for $\mathcal{A}'_{\mathrm{CloudHKA}}$ in the modified MI game:

– *Public information.* Each $\mathtt{WriteK}_i$ is generated by querying $\mathcal{O}_{\mathrm{uc}}$ or $\mathcal{O}_{\mathrm{c}}$. Each $\{\mathtt{ReadK}_i\}^{\mathsf{AES}}_{\mathtt{DistK}_i}$ is generated by running $\mathsf{AES.E}(\mathtt{DistK}_i, \mathtt{ReadK}_i)$ or randomly choosing from $\mathsf{Range}(\mathsf{AES.E})$.
– *Challenged ciphertext pair.* The index i^* submitted to $\mathcal{C}_{\mathrm{PRE}}$ is valid since in $\mathcal{C}_{\mathrm{PRE}}$'s view. The queries $\mathcal{O}_{\mathrm{c}}(i)$ and $\mathcal{O}_{\mathrm{rk}}(i, j)$ do not lead c_{i^*} to become a decryptable ciphertext. $\mathcal{C}_{\mathrm{PRE}}$ returns the encrypted m_b under $\mathtt{WriteK}_{i^*}$ for $b \xleftarrow{\$} \{0, 1\}$. Thus, $(\mathtt{Hdr}^{SC_{i^*}}_{ID}, \mathtt{Body}^{SC_{i^*}}_{ID}[\rho])$ is m_b's ciphertext under $\mathtt{WriteK}_{i^*}$ for $b \in \{0, 1\}$.
– *Corrupted distribution-keys.* The corrupted distribution-keys $\{\mathtt{DistK}_z : SC_{i^*} \npreceq SC_z\}$ are valid keys that are generated by running $\mathsf{AES.G}(\kappa)$. The distribution-keys $\mathtt{DistK}_z$ for $SC_{i^*} \preceq SC_z$ need not to be specified since they are not allowed to be corrupted in CloudHKA's modified MI game.

The above illustration shows that $\mathcal{B}$ successfully simulates the expected distribution in feeding the needed information for $\mathcal{A}'_{\mathrm{CloudHKA}}$. By our assumption of $\mathcal{A}'_{\mathrm{CloudHKA}}$, $\mathcal{B}$ breaks Ψ's IND-CPA security game with non-negligible advantage ϵ. It leads to a contradiction. □

4.3.3 Revocation of Access Rights

We illustrate that the user revocation mechanism in CloudHKA removes the Write and Read rights of a revoked user.

– *Preventing a revoked user from writing data.* To revoke a user from SC_i, DP removes u from SC_i in his HAC policy directly. Then, the request of writing operations from u will not pass CS's validity check. u is no longer allowed to write data into SC_z for $SC_i \preceq SC_z$.

– *Preventing a revoked user from reading newly encrypted data.* The rekey operation for revoking u ensures that u cannot decrypt newly encrypted data. We give an illustration with the following three parts:
  - *u cannot obtain the updated* $\mathtt{ReadK}'_i$. The readKey-cipher of SC_i is updated as $\{\mathtt{ReadK}'_i\}^{\mathsf{AES}}_{\mathtt{DistK}'_i}$. Only the non-revoked users at SC_i can update the distribution-key as $\mathtt{DistK}'_i$ for decrypting $\{\mathtt{ReadK}'_i\}^{\mathsf{AES}}_{\mathtt{DistK}'_i}$ to obtain $\mathtt{ReadK}'_i$.
  - *u no longer decrypts new ciphertext pair* $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$ *for* $SC_z \preceq SC_i$. Since u cannot obtain $\mathtt{ReadK}'_i$, u cannot decrypt $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$. The relation-keys $\mathtt{RelatK}_{z\to\xi}$ for $SC_z \preceq SC_i$ are re-generated by using the updated key pairs. u cannot use the new (or old) relation-keys to re-encrypt new $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$ into the old one under $\mathtt{WriteK}_z$. Thus, u cannot derive the original message in $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$.
  - *u no longer decrypts new body-ciphers* $\mathtt{Body}'^{SC_z}_{ID}$ for $SC_z \preceq SC_i$. Since u cannot decrypt the new $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$ for $SC_z \preceq SC_i$ to obtain (K, M'_ρ), he cannot recover M by computing $\mathsf{AONT}^{-1}(M')$.

– *Preventing a revoked user from reading previously encrypted data.* In revoking a user u from SC_i, DP sends a transform-key $\mathtt{TranK}_z$ for each SC_z, $SC_z \preceq SC_i$. CS uses $\mathtt{TranK}_z$ to update (re-encrypt) each old $(\mathtt{Hdr}^{SC_z}_{ID}, \mathtt{Body}^{SC_z}_{ID}[\rho])$ as a new $(\mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}[\rho])$. Hereafter, when u requests to read old datum ID, CS returns the new $\langle\{\mathtt{ReadK}'_z\}^{\mathsf{AES}}_{\mathtt{DistK}'_z}, \mathtt{Hdr}'^{SC_z}_{ID}, \mathtt{Body}'^{SC_z}_{ID}\rangle$. Since u cannot obtain $\mathtt{DistK}'_z$, he cannot obtain (K, M'_ρ) and recover M.

5 Discussion

This section introduces some existing desirable PRE schemes for CloudHKA. Then, we demonstrate that CloudHKA can be slightly extended for dealing with some extra issues in practical system.

5.1 Concrete PRE Schemes

A suitable PRE scheme Ψ for our CloudHKA scheme should be uni-directional, multi-hop, and IND-CPA secure. The uni-directional property is required for realizing the simple security property. The multi-hop property is required since the height of an access hierarchy is usually larger than one. The IND-CPA security of Ψ is required to show that our CloudHKA is message-indistinguishable. Additionally, for each key pair in Ψ, the decryption key cannot derive its corresponding encryption key and vice versa. It ensure that the access rights of Write and Read can be separated. To our best knowledge, there are four desirable PRE schemes in the literature [14, 18, 25, 27]. These PRE schemes can be applied to our CloudHKA directly. In particular, when the height of access hierarchies is large, we recommend to apply the one proposed by Luo et al. [18]. It is the only PRE scheme without ciphertext size growth in re-encrypting ciphertexts up to now. While applying a PRE scheme with growing ciphertext to CloudHKA, the bandwidth cost for DP in responding a data reading request and the computation cost for each user in key derivation are linear in the number of re-encryption operations.

5.2 Preventing Dishonest or Unreliable Data Storage

The outsourced data stored in CS may be altered by unexpected bit flips from system errors or accidentally deleted by CS. One solution is to apply a data integrity check scheme for ensuring correctness of outsourced data. We can use the hash-then-sign technique. Before storing data in CloudHKA, a data uploader hashes a data-cipher into a short string and computes a signature of the string by his individual signing key. After receiving an encrypted data with its signature, a user can check correctness of its body-cipher by the uploader's public verification key.

5.3 Preventing Invalid Ciphertext Re-Encryption or Transformation

The re-encryption operations in key derivation and ciphertext update may cause some unexpected errors. One solution is to apply an IND-CCA secure PRE scheme in CloudHKA. The IND-CCA security of a PRE scheme can prevent a proxy from re-encrypting ciphertexts in an invalid way. After applying such PRE scheme to CloudHKA, a user can verify the validity of re-encrypted header-ciphers and body-ciphers (in key derivation or ciphertext update) returned by CS. The PRE schemes proposed by Shoa et al. [25] and Wang et al. [27] achieve the IND-CCA security.

5.4 Scalable Rekey Mechanism for Large Number of Users

The rekey cost in computation and communication for distributing a new distribution-key of SC_i is linear in the number of users at SC_i. The number of users is often large in cloud-based services. One solution to deal with this issue is to apply a tree-based group key management (GKM) scheme [10, 11, 28] to maintain the common distribution-key among a dynamic set of users at each class. The GKM scheme reduces the computation and communication cost from $O(n_i)$ to $O(\lg n_i)$, where n_i is the number of users in SC_i. In particular, the GKM schemes in [10, 11] have an efficient rekey mechanism for a user who may miss key update messages in his off-line period. The methods are suitable in handing frequent rekey operations. While applying these two methods to CloudHKA, each user needs to store one extra personal secret key only.

6 Conclusion

In this paper we propose a practical CloudHKA for controlling data access in cloud computing. CloudHKA observes the Bell-Lapadula security model. We use ciphertext re-encryption technique to minimize the computation cost for a user in key derivation and for DP and CS in ciphertext update. CloudHKA deals with the user revocation issue practically and provides flexible authorization of data access rights. Simultaneously, CloudHKA is secure against the legal access attack. The proposed CloudHKA is formally shown to be message-indistinguishable by assuming IND-CPA security of the underlying PRE scheme.

References

1. Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.
2. Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security*, 12(3), 2009.
3. Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 190–202, 2005.
4. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security*, 9(1):1–30, 2006.
5. David E. Bell and Leonard J. Lapadula. Secure computer systems: Unified exposition and multics interpretation. In *Technical Report MTR-2997, Mitre Corporation, Bedford, Massachusetts*, 1976.
6. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the IEEE Symposium on Security and Privacy (S & P)*, pages 321–334, 2007.
7. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of EUROCRYPT*, pages 127–144, 1998.
8. Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 417–426, 2008.
9. Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 185–194, 2007.
10. Yi-Ruei Chen, J. D. Tygar, and Wen-Guey Tzeng. Secure group key management using uni-directional proxy re-encryption schemes. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1952–1960, 2011.
11. Kuei-Yi Chou, Yi-Ruei Chen, and Wen-Guey Tzeng. An efficient and secure group key management scheme supporting frequent key updates on pay-tv systems. In *Proceedings of the IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–8, 2011.
12. Jason Crampton, Keith M. Martin, and Peter R. Wild. On key assignment for hierarchical access control. In *Proceedings of the IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–111, 2006.
13. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 89–98, 2006.
14. Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *Proceedings of Applied Cryptography and Network Security (ACNS)*, pages 288–306, 2007.
15. Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the USENIX Security Symposium*, 2011.
16. Lein Harn and Hung-Yu Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, 1990.
17. Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.
18. Song Luo, Qingni Shen, and Zhong Chen. Fully secure unidirectional identity-based proxy re-encryption. In *Proceedings of International Conference on Information Security and Cryptology (ICISC)*, pages 109–126, 2011.

19. Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, 34(9):797–802, 1985.
20. Indrakshi Ray, Indrajit Ray, and Natu Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 65–73, 2002.
21. Ronald L. Rivest. All-or-nothing encryption and the package transform. In *Proceedings of the International Workshop on Fast Software Encryption (FSE)*, pages 210–218, 1997.
22. Amit Sahai, Hakan Seyalioglu, and Brent Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In *Proceedings of CRYPTO*, pages 199–217, 2012.
23. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proceedings of EUROCRYPT*, pages 457–473, 2005.
24. Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. *Theoretical Computer Science*, 412(41):5684–5699, 2011.
25. Jun Shao, Peng Liu, Zhenfu Cao, and Guiyi Wei. Multi-use unidirectional proxy re-encryption. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 1–5, 2011.
26. Wen-Guey Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(1):182–188, 2002.
27. Hongbing Wang, Zhenfu Cao, and Licheng Wang. Multi-use and unidirectional identity-based proxy re-encryption schemes. *Information Sciences*, 180(20):4042–4059, 2010.
28. Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Network*, 8(1):16–30, 2000.
29. Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 534–542, 2010.
30. Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.