Sādhanā (2022)47:20
https://doi.org/10.1007/s12046-021-01776-0

An efficient flexible hierarchical access control scheme enabling real-life exceptions

SUMIT LAL* and NAVEEN KUMAR

Indian Institute of Information Technology, Vadodara, India
e-mail: lal.sumit48@gmail.com; naveen_kumar@iiitvadodara.ac.in

MS received 3 August 2021; accepted 31 October 2021

Abstract. Most organizations today inherently followed hierarchical access control implemented using a hierarchical key assignment scheme (HKAS). An HKAS enforce reflexive, anti-symmetric, and transitive relations among the nodes (or access classes) in the hierarchy. In real life, the organizations come across rare but practical situations such as anti-symmetric exception, the temporary delegation of access to a user, etc. The traditional HKASs are costly and are not readily implement these exceptions. It motivates to relook at redesigning existing HKASs that efficiently incorporate and revokes the exceptions into the existing hierarchy as and when needed. The current similar work uses asymmetric key cryptosystem to implement the flexible HKAS with exceptions. This work proposes a novel and efficient symmetric key-based flexible HKAS that efficiently addresses the flexible hierarchy requirements. To the best of our knowledge, this is the first symmetric key-based flexible HKAS enabling exceptions. The dynamics of the scheme are addressed and compared with the other similar existing schemes. The security of the new HKAS is analyzed formally against a stronger and modern security notion known as key-indistinguishability.

Keywords. Flexible hierarchical access control; key assignment scheme; key-indistinguishability.

1. Introduction

In a hierarchical access control system, the higher privileged users can access the lower privileged users' information, but the reverse is not allowed. A hierarchy is a direct acyclic graph composed of nodes (also known as classes) and directed edges connecting them. A directed edge between two nodes represents the direction of data access. Users and data files are linked with distinct nodes in the hierarchy. A user is allowed to access data files associated with its node and descendant nodes only. The access control in a hierarchy is enforced using a suitable. Hierarchical Access Control Scheme (HKAS [1]). It enables a method for assigning encryption keys to each node in the hierarchy such that it is feasible for a user associated with a node to derive the keys of its descendant nodes in the hierarchy. In contrast, it is infeasible to derive the keys of other nodes. Data belongs to a node are encrypted with the associated key. The best-known examples of the HKAS scheme are the Akl-Taylor [1] and Atallah *et al* [2] schemes.

Consider a hierarchy of n distinct nodes $C_1, C_2, ..., C_n$. Each node C_i is assigned a distinct secret key K_i used to encrypt the node's data. A user knowing K_i can derive any descendant node's key K_j in the hierarchy. The relation between nodes C_i and C_j can be expressed as $C_j \preceq C_i$ where "$\preceq$" is the descendant relation, i.e., node C_j is the descendant of node C_i. An example hierarchy is shown in figure 1. A directed edge in the hierarchy represents the direction of data access (or relation "$\preceq$"). A hierarchy H follows the following three properties on relation "$\preceq$".

1. Reflexive: $C_i \preceq C_i$, where $C_i \in H$.
2. Anti-symmetric: $C_i \preceq C_j$ and $C_j \preceq C_i$ implies $C_i = C_j$, where $C_i, C_j \in H$.
3. Transitive: $C_i \preceq C_j$ and $C_j \preceq C_k$ implies $C_i \preceq C_k$, where $C_i, C_j, C_k \in H$. For example, in figure 1, $C_2 \preceq C_1$ and $C_4 \preceq C_2$ implies $C_4 \preceq C_1$, i.e., a user belongs to C_1 can also access data files associated with C_4.

There are two types of HKAS proposed in the literature based on their key derivation type: Direct and Indirect. In a direct HKAS, a user with a node's key can directly derive a key of a descendant node without computing the key of any intermediate node in the descendant's path [1]. For example, in figure 1, $C_4 \preceq C_1$, therefore a user at C_1 knowing its key K_1 can directly derive K_4 (without computing K_2 of intermediate node C_2) and access data files associated with C_4. Whereas in the case of indirect HKAS, a user can directly derive an immediate successor node's key and therefore recursively compute the key of any descendant node in the path towards the target node [2]. For example, in figure 1, $C_6 \preceq C_1$ therefore a user at C_1 knowing K_1 will

*For correspondence
Published online: 12 January 2022

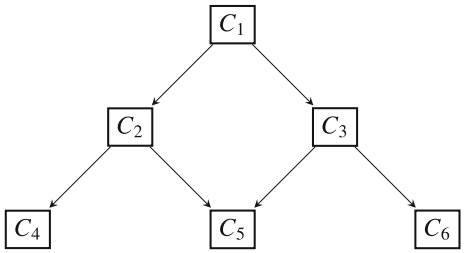

Figure 1. An example hierarchy.

first derive K_3 (an immediate successor node towards C_6) and then derive K_6 of node C_6. This is noted here that an indirect key derivation is not necessarily made the key derivation inefficient as compared to the direct access.

In the real scenario, most organizations and institutions follow the multilevel partially ordered hierarchy structure. Although the hierarchical access control is the general access policy in most of the organizations, sometimes they require exceptions to the normal hierarchical access flow such as access delegation [3] or symmetric access (also called anti-symmetric exception [4]) to the resources. Such non-trivial access sometimes becomes necessary due to the reasons such as an employee left the organization or is revoked from the position or another employee belongs to a lower level (or external user) is given temporary responsibility for their work for the smooth functioning of the organization. Such cases are common in an organization and will increase with the increase in the organizational hierarchy structure. Consider a different scenario where the files are associated with different hierarchical users at different security levels. To achieve security and address the high mobility of the defence personals, access to the secret files need to be continuously updated. It requires an exception to the hierarchical access as access to the files need to be continuously shifted from one user to another. To address these issues, we require an effective, efficient, and secure solution to the problem that will minimize the disturbance in a general hierarchical access control system. Some such real scenarios are discussed below.

Some existing works [4–7] address the flexible hierarchical structure with anti-symmetric and transitive exceptions. Few [7] address the access delegation problem and the flexible hierarchical access to the non-descendant nodes in the hierarchy. Although the above schemes address the required exceptions to the normal hierarchical access, the schemes use direct access control methods based on asymmetric cryptography and Akl and Taylor scheme [1]. In this work, we consider a symmetric key based solution to address the above issues considering efficiency reasons compared to the above used asymmetric cryptosystems. This will be the first such scheme that will use a symmetric cryptosystem to address the exceptions in hierarchical access systems to the best of our knowledge. The important contributions of this work are given below.

- We propose a novel symmetric-key based flexible hierarchical access control scheme that uses indirect key derivation [8]. The proposed scheme supports antisymmetric exception.
- Dynamic operations such as insertion, deletion, updating a node, or relation (an edge) will be addressed and are handled locally within the hierarchy (i.e., other nodes are minimally affected).
- Class delegation problem with descendant(s) safety will be addressed. Besides, we propose other related flexible requirements such as class delegation with a descendant(s) access, class delegation to a particular user with a descendant(s) safety, and Class delegation to a particular user with descendant(s) access.
- The security of the proposed scheme is formally analyzed with respect to the modern and stronger security notion named key indistinguishability [9]. It is noted here that the indistinguishability property implies security against key recovery property.

In the proposed scheme, we are not considering the transitive exception because it rules out the hierarchical access control policies and is therefore not suited to the hierarchy-based systems [4]. Instead, we may employ a class delegation mechanism to address the transitive exception related requirements (discussed in section 3.4).

In what follows, section 2 reviews the related work on generic hierarchical access control schemes. Section 3 gives the construction of our proposed scheme. The dynamics of the construction are discussed in section 3.1. Section 3.2 discusses how the proposed construction addressees the Anti-symmetric exception. Class delegation problem is discussed in section 3.3. Section 3.5 formally analyze the security of the proposed construction. The proposed scheme is compared with the other similar existing schemes from the literature in section 4. Section 5 concludes this work.

2. Related work

The initial hierarchical access control scheme was proposed by Akl and Taylor [1]. They proposed a direct key derivation scheme using asymmetric cryptosystems. The main limitation of this scheme was the computation cost that uses exponentiation operation over large primes.

A more flexible key assignment scheme based on Akl and Taylor [1] scheme was proposed by Yeh *et al* [4] addressing anti-symmetric and transitive exception policies. It is motivated by a distributed static database system. Two keys are assigned to each node, one for data encryption and another for

key derivation to enforce exceptions. Yeh [10] uses existing hierarchical access control schemes to enforce non-hierarchical access control policies. Recently, a flexible hierarchical key assignment scheme for time-based access control in data outsourcing is discussed in [11], and Hsiao *et al* discusses a similar scheme [12] in the internet of things.

Another key assignment scheme by Lin *et al* [6] improves system public storage for solving the flexible hierarchical access control problem addressing anti-symmetrical and transitive exceptions. Instead of two public parameters in [4], Lin *et al* [6] uses one public parameter with each node.

Y F Chang [5] proposed another flexible hierarchical access control mechanism enforcing anti-symmetric and transitive exceptions by employing the elliptic curve cryptosystem (ECC [13, 14]). It uses a two-layer hash approach to ensure low computation load and small key size as compared with [4, 6].

Recently, Pareek *et al* [7] gives a flexible hierarchical key assignment scheme with constant computation cost for key derivation and without depending on the hierarchy depth. The class delegation is addressed with a descendant(s) safety and is enforced using group proxy re-encryption that supports proxy re-encryption between two classes in a dynamic hierarchy.

Barka and Sandhu [15] give a detailed framework for role-based delegation models. This work defines how to identify the characteristics related to the delegation problem and how it can be used in building use cases used to build delegation models. Jason Crampton [3, 16] studies the delegation of roles in role-based access control models and gives use cases. Ray *et al* [17] formalized a trust-based fine-grained access control model with delegation. Ferrara *et al* [18] proposed verifiable HKAS using an MLE scheme as a building block.

All the above schemes have used asymmetric cryptosystems based on Akl and Taylor scheme [1]. Atallah *et al* [8] propose indirect HKAS based on symmetric key cryptosystem and lightweight hash function. Dynamic operations in [8] are efficiently addressed as compared to the Akl and Taylor scheme. A distinct symmetric random key is assigned to every node in the hierarchy. An efficient hash function is used for key derivation purposes. The scheme requires less public and private storage as compare to the above asymmetric schemes. Only one symmetric key is to be stored with each user's private storage. Also, the dynamic operations (such as deleting or adding a node or an edge into the hierarchy) are handled locally without disturbing the whole hierarchy as in the case of the above discussed Akl and Taylor [1] based systems. The suitability of different symmetric key-based hierarchical key assignment schemes for the data outsourcing scenario is recently analyzed in [19].

This work proposes a symmetric key-based HKAS based on Atallah *et al* [2] scheme addressing antisymmetric exception and class delegation, also dynamic operations are handled locally. The proposed scheme requires less public and private storage and low computational load than other HKAS with exceptions.

3. Proposed hierarchical key assignment scheme

This section describes the proposed hierarchical key assignment scheme in detail. We consider a cryptographic hash function $H:\{0,1\}^* \rightarrow \{0,1\}^\rho$ where ρ is the security parameter. A trusted Central Authority (CA) is assumed to be present similar to other existing HKASs in the literature. It involved hierarchy creation, key assignment to each hierarchy node, key distribution to the system users, and the hierarchy's maintenance. In the proposed scheme, CA assigns a distinct random public label $l_i \leftarrow \{0,1\}^\rho$ to each node C_i in the hierarchy.

3.1 *Key assignment*

The CA assigns a distinct random secret key $K_i^d \leftarrow \{0,1\}^\rho$ to each node C_i in the hierarchy. We call the key a derivation key as it is used for key derivation purposes in the hierarchy. A derivation key K_i^d assigned to a node C_i is used to derive derivation keys associated with its descendant nodes in the hierarchy.

Each node C_i is also assigned an encryption key K_i^e. The derivation key K_i^d assigned to the node C_i is used to compute the node's encryption key $K_i^e = H(K_i^d||l_i) \, mod \, 2^\rho$. Data associated with each node is encrypted with their respective encryption key K_i^e. We consider symmetric key cryptosystem to encrypt the data and uses AES-128 as an encryption algorithm. An authorized user is assigned a respective node's derivation key using which it can derive derivation keys associated with any of its descendant nodes and hence the respective encryption key. An authorized writer will derive and use the encryption key to encrypt the newly created data before uploading it to the server. An authorized reader will download the encrypted data, derive the required encryption key, and use it to decrypt the downloaded data before using it.

3.2 *Public information*

Along with a public label l_i assigned to each node C_i in the hierarchy, the CA computes and assigns a public token $Y_{i,j} = K_j^d \oplus H(K_i^d||l_j)$ to each directed edge $e(i, j)$ from node C_i to C_j. The public tokens will help in key derivation along a path in the hierarchy to derive a descendant node's derivation key. All the labels and tokens are published.

3.3 *Key derivation*

An authorized user knowing a node's derivation key can derive the derivation key of any descendant node in the hierarchy. To compute an authorized target node's decryption key, a user will use the known derivation key and public information to derive the immediate descendant node's derivation key in the path towards the target node. Following the same procedure of deriving the derivation key of the immediate successor node in the path towards the target node, the user will compute the derivation key of the target node. The target node's decryption key is then computed using the target node's derivation key and public information.

For example, consider an example HKAS given in figure 2 containing six nodes. Each node C_i $(1 \leq i \leq 6)$ is assigned a public label l_i, derivation key K_i^d, and encryption key K_i^e. Each edge $e(i, j)$ from node C_i to C_j is assigned a public token $Y_{i,j} = K_j^d \oplus H(K_i^d||l_j)$. Let a user associated with node C_1 is assigned a respective derivation key K_1^d wants to access node C_4's data. Knowing key K_1^d and public information, the user can compute the derivation key of its immediate descendant nodes $K_2^d = K_1^d \oplus Y_{1,2}$ (in the path towards the target node C_4), and then compute K_4^d in the same way. Now, the user can compute the decryption key $K_4^e = H(K_4^d||l_4)\ mod\ 2^\rho$ of the target node C_4 using K_4^d. The user can now access the required data associated with the target node C_4 from the cloud, decrypt it using the decryption key K_4^e, and use it.

In what follows, we discuss the required dynamic operations for the proposed HKAS to make the scheme complete and useful for generic dynamic applications. Later we discuss how it can be extended to a more generic hierarchy to enforcing anti-symmetric exception and class delegation.

3.4 *Dynamic operations*

Here we discuss the essential dynamic operations associated with the proposed static HKAS. CA has Hierarchy System HS along with all necessary information associated with the hierarchy.

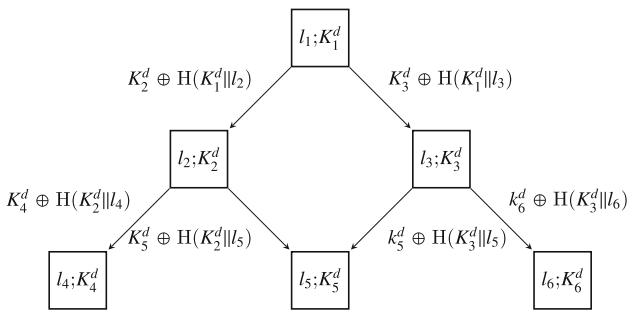

Figure 2. An example of proposed HKAS.

3.4.1 *Revoking an encryption key*

It is addressed using a rekeying operation which assigns a new key for further reference. Rekeying a node's encryption key in an HKAS will replace the old key with the new key and update all the related references. The rekeying of a node's encryption key is essential in a situation when the key is required to be revoked due to the reasons such as the key is leaked. We assume that the respective derivation is not leaked. Rekeying a node's encryption key in an HKAS will replace the old encryption key with the new key and update all the related references. An encryption key is used to access data files belongs to the node. The rekeying of the encryption key will only affect the given node in the proposed construction. The rekeying procedure of a node encryption key is given in algorithm 1 and is described below.

Rekeying of encryption key of node C_i requires a new public label l_i (step 1). Store encryption key K_i^e in a temporary variable K (step 2). Re-compute encryption key K_i^e using updated key K_i^d (step 3). For each file f associated with node C_i, download it from the cloud store, decrypt it with key K (step 5), encrypt it with updated key K_i^e (step 6), and store it back to the cloud storage. Now, for each incoming edge at node C_i compute the respective public token Y and update it at the cloud storage (steps 8-9).

Revoking an encryption key requires the cost of updating the encryption key. The respective node files will be decrypted using the existing key and then re-encrypted with the updated key. Corresponding public information will be updated. The updated key is then distributed among all the authorized users who possess the node's encryption key. Users with the node's derivation key can derive it using public information and not send the encryption key.

Algorithm 1 Revoke_EncKey($HS, C_i, H()$)

Input: Hierarchy system HS, node $C_i \in HS$ whose encryption needs to be revoked and hash function $H()$.
Output: Public tokens in HS.

1: $l_i \leftarrow \{0,1\}^\rho$
2: $K \leftarrow K_i^e$
3: $K_i^e \leftarrow H(K_i^d||l_i)\ mod\ 2^\rho$
4: for (each file f associated with C_i) do
5: $\quad f' \leftarrow decrypt\ f$ with key K
6: $\quad$ encrypt f' with K_i^e and upload it back to the server
7: end for
8: for (each predecessor node C_p of C_i, $C_p \in HS$) do
9: $\quad$ Compute and update $Y_{p,i} = K_i^d \oplus H(K_p^d||l_i)$
10: end for

3.4.2 *Revoking a derivation key*

Revoking a derivation key is essential when the node's key is required to be updated due to reasons such as the key was leaked, the user with the

key was revoked from the organization, etc. It is also addressed using a rekeying operation that assigns a new derivation key to the node for further reference. Rekeying a node's derivation key in an HKAS will replace the old derivation key with the new key and update all the related references.

A derivation key in the proposed scheme is used to access data files belonging to all the hierarchy's descendant nodes. Therefore, the rekeying operation here will only affect the given node and its descendants. So, It requires the cost of updating all the derivation keys and revocation of the encryption key in the descendant hierarchy. The revocation procedure of a node's derivation key is given in algorithm 2 and is described below.

Revoking the derivation key of a node C_i requires assigning a new derivation key K_i^d (step 1). Store encryption key K_i^e in a temporary variable K (step 2). Re-compute encryption key K_i^e using updated key K_i^d. For each file f associated with node C_i, download it from the cloud store, decrypt it with key K (step 5), encrypt it with updated key K_i^e (step 6), and store it back to the cloud storage. Now, for each incoming edge at node C_i compute the respective public token Y and update it at the cloud storage (steps 8-9). For each successor node of C_i, repeat the above procedure to update all the descendant nodes in the hierarchy (steps 11-12).

Algorithm 2 Revoke_DerivationKey($HS, C_i, H()$)

Input: Hierarchy system HS, node $C_i \in HS$ whose derivation key needs to be revoked and hash function $H()$.
Output: Derivation key for node C_i is updated.

1: $K_i^d \leftarrow \{0,1\}^\rho$
2: $K \leftarrow K_i^e$
3: $K_i^e \leftarrow H(K_i^d \| l_i)\ mod\ 2^\rho$
4: for (each file f associated with C_i) do
5: $f' \leftarrow$ *decrypt* f with key K
6: encrypt f' with K_i^e and upload it back to the server
7: end for
8: for (each predecessor node C_p of C_i, $C_p \in HS$) do
9: Compute and update $Y_{p,i} = K_i^d \oplus H(K_p^d \| l_i)$
10: end for
11: for (each successor node C_s of C_i, $C_s \in HS$) do
12: Call *Revoke_DerivationKey*($HS, C_s, H()$)
13: end for

3.4.3 *Node insertion* A new node insertion is needed when a new security class is created as part of the hierarchy that contains a distinct set of data files. For example, assume a node represents a department within an organization, and a new department is created. It requires a node as the department needs to be inserted within the organizational hierarchy.

Node insertion in HKAS will insert a new node in a hierarchy system by assigning a derivation key and public label to it. Now update all the incoming and outgoing edges by defining a new public token with respect to each edge. It will only affect the given node in the proposed construction. The node insertion procedure is given in algorithm 3 and is described below.

The CA generates and assigns distinct random derivation key K_i^d and distinct random label l_i to node C_i (steps 1-2). Add a public token (or an edge) for each predecessor and successor node of node C_i in the hierarchy (steps 3-8).

Algorithm 3 *Node_Insertion*($HS, C_i, H()$)

Input: Hierarchy system HS, node C_i which is to be inserted in HS and hash function $H()$.
Output: The new node C_i is updated in the hierarchy.

1: $K_i^d \leftarrow \{0,1\}^\rho$
2: $l_i \leftarrow \{0,1\}^\rho$
3: for each node C_p immediate predecessor of C_i, $C_p \in HS$ do
4: *Edge_Insertion*($HS, C_p, C_i, H()$)
5: end for
6: for each node C_s immediate successor of C_i, $C_s \in HS$ do
7: *Edge_Insertion*($HS, C_i, C_s, H()$)
8: end for

3.4.4 *Node deletion* A node needs to be deleted from a hierarchy when a security class becomes irrelevant in the hierarchy. For example, assume a node represents a department within an organization, and an existing department needs to be demolished or deleted, considering no user remains at the department. The node as the department needs to be deleted from the organizational hierarchy.

A node deletion from an HKAS will remove the node and update all the related references. It will only affect the given node and its connected edges in the proposed construction. The node deletion procedure is given in algorithm 4 and is described below.

To delete a node C_i from a given hierarchy, the data owner inserts an edge from each immediate predecessor node of C_i to each of its successor nodes (steps 1-5). Now delete each incoming and outgoing edge from node C_i (steps 6-11). It will free the node C_i from the hierarchy without affecting the users at ascendant nodes.

Algorithm 4 *Node_Deletion*$(HS, C_i, H())$

Input: Hierarchy system HS, node $C_i \in HS$ which is to be deleted from HS and hash function $H()$.
Output: The node is deleted from the hierarchy.

```
for (each immediate predecessor node C_p of C_i, C_p ∈ HS) do
    for each immediate successor node C_s of C_i, C_s ∈ HS do
        Edge_Insertion(HS, C_p, C_s, H())
    end for
end for
for (each incoming edge from C_p to C_i, C_p ∈ HS) do
    Edge_Deletion(HS, C_p, C_i, H())
end for
for (each outgoing edge C_i to C_s, C_s ∈ HS) do
    Edge_Deletion(HS, C_i, C_s, H())
end for
```

3.4.5 *Edge insertion* Edge insertion is needed when a security class is given access to another security class. This operation is trivial and straightforward. An edge can be inserted subject to the condition that it will not form an acyclic graph (or hierarchy). An edge insertion will not affect any other node in the proposed hierarchy construction. The edge insertion procedure for adding a node C_i into the given hierarchy is given in algorithm 5. It requires creating a public edge token $Y_{i,j}$ between the source and destination nodes C_i and C_j, respectively (step 1). Then published the computed token to the outsourced store (step 2).

Algorithm 5 *Edge_Insertion*$(HS, C_i, C_j, H())$

Input: Hierarchy system HS, directed edge source node C_i, destination node C_j, $C_i, C_j \in HS$, and hash function $H()$.
Output: The edge (C_i, C_j) is inserted in HS.

1: Compute $Y_{i,j} = K_j^d \oplus H(K_i^d \| l_j)$
2: Publish $Y_{i,j}$ in cloud store

3.4.6 *Edge deletion* Edge deletion is required when access for a security class or node needs to be revoked from a given class. Deleting an edge will disallow any further access to the descendant node data files from the users at the deleted edge source node. It is addressed using access right revocation operation, which revokes the public token with respect to the deleted edge and applies re-keying in the descendant hierarchy nodes. It prevents the deleted node's user(s) from accessing the data files associated with an old descendant(s). This operation will only affect the given node and its descendant(s). The edge deletion procedure from a node C_i to C_j is given in algorithm 6 and is described as follows.

Removing an edge (C_i, C_j) from the given hierarchy requires removing the public token $Y_{i,j}$ (step 1) and call the procedure Revoke_Derivation Key() in algorithm 2 to update the keys in the descendant hierarchy (step 2). Step 2 will disallow any further access to the descendant nodes data files whose connecting edge is now revoked.

Algorithm 6 *Edge_Deletion*$(HS, C_i, C_j, H())$

Input: Hierarchy system HS, source node C_i, destination node C_j, $C_i, C_j \in HS$, and hash function $H()$.
Output: The edge (C_i, C_j) is deleted from HS.

1: Delete public token $Y_{i,j}$ from outsourced store
2: Call *Revoke_DerivationKey*$(HS, C_j, H())$

In the next section, we discuss how the anti-symmetric exception and delegation problem will be addressed in the given hierarchy construction.

3.5 *Enforcing anti-symmetric exception*

Anti-symmetric is the fundamental relation in any direct acyclic graph. Many organizations are implicitly following the direct acyclic relationships among different departments within the organizations where each department can be represented as a distinct security class. Although it mostly follows the direct acyclic graph relationship, sometimes exceptions become necessary in the hierarchy to handle extraordinary circumstances.

One such possible exception is anti-symmetric. The anti-symmetric exception to the node is giving access right of the ancestor node. While accessing the ancestor node, it is also essential that the ancestor node's descendant will remain safe, and the node is infeasible to compute the keys of the descendant(s) of the ancestor node. It means the access should be given for only the desired ancestor node.

For example, consider the hierarchy system where a node C_i is the ancestor of node C_j. To insert the anti-symmetric relation between the nodes, i.e., insert an edge (C_j, C_i) such that users at node C_j can access the data files at ancestor node C_i. The procedure to enforce an anti-symmetric exception for a node C_j to access an ancestor C_i in a given hierarchy is given in algorithm 7. In the proposed scheme, this procedure only requires to compute and store a public edge token $Z_{j,i}$. No other node is get affected in the hierarchy. It makes the operation efficient as only one hash, and one xor operation needs to be computed. On the other hand, the existing schemes require exponential operation computation. Algorithm 7 is discussed below.

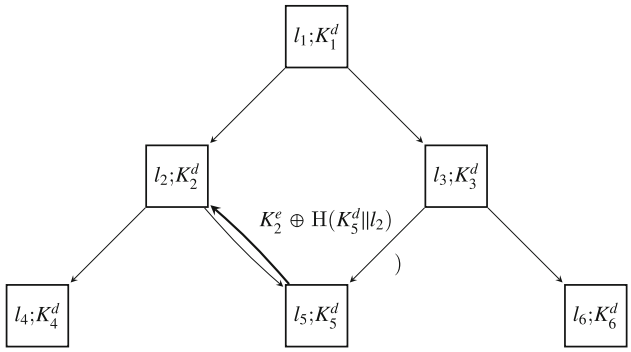

Figure 3. An example of proposed HKAS with anti-symmetric exception.

Create a public edge token $Z_{j,i} = K_i^e \oplus H(K_j^d||l_i)$ between C_J and C_i (step 1). Now, publish the new token $Z_{j,i}$ to the outsourced store (step 2). The $Z_{j,i}$ is different than $Y_{j,i}$ as it uses the encryption key K_i^e of the destination node instead of derivation key K_i^d in the computation of $Y_{j,i}$. This is to restrict the source node (C_j) users to access node C_i' data only. It will not allow accessing the descendant nodes data files as required. For example, consider the hierarchy in figure 3. A public token $Z_{5,2} = K_2^e \oplus H(K_5^d||l_2)$ is used to derive encryption key K_1^e of the target node C_1. Using K_2^e, one cannot derive keys associated with node C_4 due to the use of a one-way function between K_2^d and K_2^e. Therefore, the public token $Z_{5,2}$ will restrict access to node C_2 only. Due to the transitive property, any ancestor of node C_5 can also access the nodes that are accessible by node C_5, any ancestor of node C_5 can access node C_2.

If we want to enforce an anti-symmetric exception relation between two nodes, C_i and C_j, i.e., insert an edge (C_j, C_i) in the hierarchy where a node C_i is the ancestor of node C_j. It is such that the users at node C_j can be able to access the data files of ancestor node C_i and no other ancestor node of C_j can access C_i knowing public tokens and derivation key K_j^d. Then there will be two possible cases.

First, the CA will assign a distinct random secret key $K_j^a = \{0,1\}^\rho$ to node C_j. We call the key an ancestor derivation key as it is used for key derivation purposes for an ancestor node in the hierarchy. Create a public edge token $Z_{j,i} = K_i^e + H(K_j^a||l_i)$ between C_j and C_i (step 1). Now, publish the new token $Z_{j,i}$ to the outsourced store (step 2). The $Z_{j,i}$ is different than $Y_{j,i}$ as it uses the ancestor derivation key and encryption key K_i^e of the destination node instead of the derivation key K_i^d in the computation of $Y_{j,i}$. This is to restrict the source node C_j's users to access the node C_i's data. It will not allow accessing the descendant nodes data files. Also, no other ancestor node of C_j can access C_i as they only know public tokens and derivation key K_j^d.

Second, using Class delegation with descendant(s) safety, the CA delegates access rights of a node C_i to node C_j such that C_j can now access C_i but cannot be able to access descendants of C_i. The CA delegates the decryption rights of a node C_i to node C_j, by providing the encryption key of C_i. The node C_j can access C_i, but knowledge of the encryption key does not allow computation of the derivation key. Thus C_j cannot access any of C_i's descendant(s) nodes. Also, no other ancestor node of C_j can access C_i.

Algorithm 7 *AntiSymm_Exception*$(HS, C_j, C_i, H())$

Input: Hierarchy system HS, directed edge source node C_j, destination node C_i, $C_i, C_j \in HS$, and hash function $H()$.

Output: The anti-symmetric edge (C_j, C_i) is inserted in HS.

1: $K_j^a \leftarrow \{0,1\}^\rho$
2: Compute public token $Z_{j,i} = K_i^e \oplus H(K_j^d||l_i)$
3: Publish $Z_{j,i}$ in cloud store

3.6 *Class delegation*

A class delegation problem requires immediate class access by other class users who initially do not have access to it. This is a temporary solution. For example, consider a situation where a department (as a class or node) is temporally given extra load (or access) of another independent department within the organization. This is different from the anti-symmetric exception because the users may also be able to access the node, which is neither its descendant nor ancestor.

In the proposed scheme during the class delegation process, the CA plays an important role. The CA is responsible for securely transmitting the access rights as per the requirement of the delegation. Also, if the delegated access is revoked, the CA needs to perform the necessary operations to revoke the access rights. In class delegation, the data owner temporarily delegates the decryption rights of a node to another unauthorized node by securely transmitting the key. Also, if the data owner wishes, the delegation can be revoked by updating the key while preserving dynamic operations' security. In what follows, we discuss different types of class delegation and its functioning.

3.6.1 *Class delegation with descendant(s) safety*

The data owner temporarily delegates access rights of a node C_i to node C_j such that C_j can now access C_i but cannot access descendants of C_i. The CA delegates the decryption rights of a node C_i to another node C_j, by providing the encryption key of C_i. The node C_j can access C_i, but knowledge of the encryption key does not allow computation of the derivation key. Thus C_j cannot access

any of C_i's descendant(s) nodes. After a specific duration, the decryption rights of node C_i by node C_j is revoked by using the procedure for Revoke_EncKey() in algorithm 1.

For example, in figure 2, node C_3 requests the CA to delegate decryption rights of a node C_2, after verifying the request the CA will securely provide the encryption key K_2^e to C_3. Now, C_3 can access C_2, but knowledge of the encryption key does not contribute to the feasible calculation of the derivation key. Thus C_3 cannot be able to access C_4 and C_5. After a specific duration, the decryption rights of node C_2 by node C_3 is revoked by using the procedure of Revoke_EncKey() in algorithm 1.

3.6.2 *Class delegation to a particular user with descendant(s) safety* The data owner temporarily delegates the access rights of a node C_i to a user U ($U \in C_j$) of node C_j such that U can access C_i but cannot be able to access C_i's descendant(s) node.

The CA delegates the decryption rights of a node C_i to a user U_t ($U_t \in C_j$) of another node C_j, by providing the encryption key of C_i. The user U_t can access C_i, but knowledge of the encryption key does not contribute to the computation of the derivation key; thus, U_t cannot access any of C_i's descendant(s) nodes. After a specific duration, decryption rights of node C_i by the user U_t of node C_j are revoked using the procedure for Revoke_EncKey() in algorithm 1.

For example, as shown in figure 2, suppose a user U_1 of node C_6 request the CA to delegate the decryption rights of node C_2. After verifying the request, the CA will securely provide the encryption key K_2^e to the user U_1 of node C_6. Now, the user U_1 can access C_2, but knowledge of the encryption key does not contribute to the feasible calculation of the derivation key. Thus the user U_1 of node C_6 cannot access C_4 and C_5. After a specific duration, the decryption rights of node C_2 by the user U_1 are revoked using the procedure for Revoke_EncKey() in algorithm 1.

3.6.3 *Class delegation with descendant(s) access* The data owner temporarily delegates the access rights of a node C_i to node C_j such that C_j can access C_i and any of C_i's descendant(s) class.

The CA delegates the decryption rights of a node C_i to another node C_j by providing the derivation key of C_i. The users at C_j can access C_i, also by derivation the key of C_i, they can access any of C_i's descendant(s) nodes. After a specific duration, the decryption rights of node C_i by node C_j are revoked by using the procedure *Revoke_DerivationKey*() in algorithm 2.

For example, in figure 2, node C_3 request the CA to delegate decryption rights of a node C_2 After verifying the request, the CA will securely provide the derivation key K_2^d to C_3. Now, users at C_3 can access C_2. Also, with the derivation key of node C_2, C_3 can access C_4 and C_5. After a specific duration, the decryption rights of node C_2 by node C_3 is revoked by using the procedure for Revoke_DerivationKey() in algorithm 2.

3.6.4 *Class delegation to a particular user with descendant(s) access* The data owner temporarily delegates the decryption rights of a node C_i to a user $U_i(U_i \in C_j)$ of node C_j such that C_j can access C_i and all of its descendant classes.

The CA delegates decryption rights of a node C_i to a user $U_t(U_t \in C_j)$ of another node C_j, by providing the derivation key of C_i. such that U_t can access C_i, also by derivation key of C_i, U_t can access any of C_i's descendant(s) node. After a specific duration, decryption rights of node C_i by the user U_t of node C_j is revoked by using the procedure for Revoke_DerivationKey() in algorithm 2.

For example, as shown in figure 2, suppose a user U_1 of node C_6 request the CA to delegate the decryption rights of node C_2. After verifying the request, CA will securely provide the derivation key K_2^d to the user U_1 of node C_6. Now, the user U_1 of node C_6 can access C_2. Also, with the derivation key of node C_2, the user U_1 of node C_6 can derive keys of C_4 and C_5. After a specific duration, the decryption rights of node C_2 by the user U_1 is revoked by using the procedure for Revoke_DerivationKey() in algorithm 2.

3.7 *Securing distributed database system using class delegation*

Yeh *et al* [4] consider transitive exception in hierarchical access control. They considered a fixed database with some tables and confidential records where the users can only request a specific query processor to retrieve some aggregate information in these tables. A user is not allowed to access the confidential records directly. He or she can send the query requests to the query processor, which reads the tables and respond to the user's query, as shown in figure 4.

In the proposed scheme, a database that contains tables with confidential records is stored in a separate root node without having access right to any other node except to the query processor node, which has delegation right of that node. The user in the system can make query requests to the query processor, which looks into the table as it has

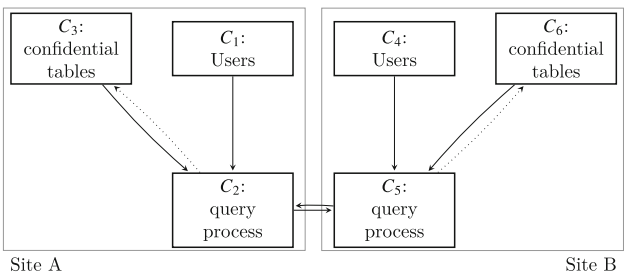

Figure 4. A two-site distributed database system.

the decryption key of the node having confidential tables and answers the user's query. Since the confidential records are stored in the root node, no other node has the node's access.

3.8 *Security analysis*

This section gives the security analyses of the proposed scheme. The analysis shows the completeness and soundness of the proposed scheme.

Definition 1 Completeness: Every node should be assigned a key used to compute all the necessary keys used to access the data associated with all of its authorized nodes in the hierarchy.

Theorem 1 *The proposed scheme is complete.*

Proof In a hierarchical access control scheme, a user can only access data belonging to its class and the class it is authorized to access. In the proposed scheme, a derivation key K_i^d is assigned to each node C_i in the hierarchy. The derivation key is used to compute all the necessary derivation and encryption keys associated with the descendant nodes or ancestor node in the hierarchy using public information. A user assigned to a security class is given the derivation key of that class using which she can derive the encryption key of the same class and derivation keys of all the descendant classes or ancestor class. Derivation keys will be later used to compute the encryption keys of the same class.

Consider a user of node C_i has derivation key K_i^d. Let the user wants to access the data at descendant node C_j such that there is a directed path from node C_i to C_j. The user will first, derive derivation key K_k^d of its immediate successor node C_k in the path towards C_j. The user will compute $K_k^d = Y_{i,k} \oplus H(K_i^d || l_k)$. Similarly, moving forward in the path towards C_j, the user recursively computes C_j^d. Let the user wants to access the data at ancestor node C_p such that there is a directed path from node C_i to C_p. The user will compute $K_p^d = Z_{i,p} \oplus H(K_i^d || l_p)$ and derive derivation key K_p^d of the node C_p. The user with the derivation key K_i^d of any node C_i can compute node encryption key $K_i^e = H(K_i^d || l_i)$. Using the encryption key, the user can now download the node's data, decrypt, and use it. Therefore, a user knowing a node's decryption key can compute the encryption key of any of its descendant nodes in the hierarchy following the above procedure and access their data. □

Definition 2 Soundness: A class user's key space has no key that can be used to compute any of the keys used to decrypt the data associated with an unauthorized node in the hierarchy.

Formally, for each node C_i, the derivation key K_i^d is protected from the coalition of users belongs to node C_j where $C_j \preceq C_i$. Now, we need to discuss additional definitions and the adversarial model. For proof of the proposed scheme, a stronger security notion named key indistinguishability (KI) is considered the security goal based on the standard model that assumes $H()$ as a pseudorandom function. In what follows, the pseudorandom function is defined.

Definition 3 [PRF] To define the pseudorandom function, consider a family of functions F: K×X→R where key $K \in \{0,1\}^\rho$, an input $X \in \{0,1\}^\rho$ and $R \in \{0,1\}^\rho$, where ρ is the security parameter. Assume an adversary algorithm A takes an oracle for a function f: X → R and returns a bit. Function f can either be drawn uniformly at random, and the adversary is given access to the function F, i.e., f ← F, or drawn uniformly at random and the adversary is given access to the function f, i.e., f ← $Rand^{X \leftarrow R}$.

The prf-advantage of adversary A is as follows:

$$Adv_{F,A}^{prf}(\rho) = |Pr[f \leftarrow F : A^f = 1] - pr[f \leftarrow Rand^{X \leftarrow R} : A^f = 1]| \leq \in_{pre}(\rho)$$

where $\in_{pre}(\rho)$ is a negligible function in security parameter ρ that gives any negligible value.

In [2, 20–23] the key indistinguishability property is defined as the inability of an adversary from learning any information about an unauthorized key. In other words, it ensures the inability of the adversary to distinguish a correct key from a random key of the same length. To define key indistinguishability, we consider a static adversary who wants to attack a node w in the hierarchy. Following [2, 22, 24], it is defined as follows.

Definition 4 Definition [IND ST] An HKAS is secure with respect to key indistinguishability if no polynomial-time static adversary A_{IND}^{ST} has a non-negligible advantage in security parameter ρ against a challenger in the next game:

Setup: The challenger executes the key generation algorithm and develops the initial hierarchy. This gives the secret information set S and public information Pub as output.

Challenge: The challenger challenges the adversary A_{IND}^{ST} for challenge node C_w by giving Pub, S', and K_b, where S' contains secret information of every node v such that $w \not\preceq v$, and for K_b, the challenger selects a random bit $b \in \{0,1\}$. If $b = 1$, then $K_1 = K_w$. If $b = 0$, then K_0 is a random string of length ρ.

Break: The adversary A_{IND}^{ST} as his best guess, outputs a bit $b' \in \{0,1\}$ whether he was given K_1 or K_0. If $b' == b$, then the adversary A_{IND}^{ST} wins the game. We define the advantage of the adversary in attacking the scheme as:

$$Adv^{ST}_{A_{IND}}(\rho) = |Pr[b' == b] - 1/2|$$

A scheme is IND ST secure if there is a partially ordered graph G having a polynomial number of nodes where each node w ∈ G:

$$Adv^{ST}_{A_{IND}}(\rho) \leq \in(\rho)$$

where $\in(\rho)$ is the negligible function in security parameter ρ that gives any negligible value.

Theorem 2 The proposed scheme is sound.

Proof To prove the theorem, we follow the structure used in [2]. We show that in the proposed scheme, the security against key indistinguishability can be compromised if and only if the pseudorandom function F's security does not hold. Hence, we show how an adversary A^{ST}_{IND} attacking the given scheme will convert into a PRF adversary A_{PRF} attacking F.

In order to prove this, consider a fixed challenge node w in a partially ordered hierarchy G for which adversary A^{ST}_{IND} will be converted into the adversary A_{PRF} with respect to the given assumption.

To prove the statement, we define a series of indistinguishable games G_0, G_1, G_2. We converge to the negligible advantage of the adversary A^{ST}_{IND}'s in the last game. The goal of A^{ST}_{IND} in each game is to guess $(b' \leftarrow \{0,1\})$ the challenged bit b as best of his knowledge. Let $T_0, T_1, ...$ are the events that b' = b in game $G_0, G_1, ...$, respectively. Initially, we check the two special cases, then we give the general description of these cases.

First case: w is the root node in G.

Game G_0: G_0 is the same as the original game specified in definition *IND ST*. The advantage of adversary A^{ST}_{IND} will be written as

$$Adv^{ST}_{A_{IND}}(\rho) = |Pr[b' = b] - 1/2| = |Pr[T_0] - 1/2|$$

Game G_1: It is similar to game G_0 except that the challenger chooses a bit $b \leftarrow \{0,1\}$ at random and compute the derivation key of the challenge node w such that $k^d_w = F_r(l_w)$ if $b = 1$, where F_r is a random function and k^d_w is assigned a random string if $b = 0$. Moreover, public values $Y_{w,u}$ is computed for every immediate descendant u of w, as $Y_{w,u} = k^d_u \oplus H(F_r(l_u))$.

Formally, Lemma *A*.1 (see Appendix) shows that any non-negligible difference in A_{IND}'s behavior between games G_0 and G_1 can be used to construct a PPT (probabilistic polynomial-time) algorithm A_{PRF} that may break the *PRF*'s security with non-negligible advantage. Hence,

$$|Pr[T_1] - Pr[T_0]| \leq \in_{PRF} \quad (1)$$

Where $\in_{PRF}$ is a negligible value and upper bound on the advantage $Adv^{IND}_{PRF}(\rho)$ of any PPT adversary A_{PRF} considering the security of the pseudorandom function F_r. Notice that by our security assumption on the *PRF* family, $\in_{PRF}$ is negligible.

It is seen that the adversary has no information about a bit b in the game. This is because k^d_w is information-theoretically secure from the view of the adversary. The adversary's view is the same regardless of the random value in both cases when $b = 0$ or $b = 1$. Hence,

$$Pr[T_0] = 1/2 \quad (2)$$

Thus, from Equations (1) and (2).

$$Adv^{ST}_{A_{IND}}(\rho) \leq \in_{PRF} \quad (3)$$

It means that the advantage of the adversary A^{ST}_{IND} in *IND ST* game ($G0$) for the correct guess of a bit b will remain the negligible value.

Second case: w is an immediate descendant of the root node in G. **Game** G_0, **Game** G_1: the initial two games are the same as in the first case.

Game G_2: This game is identical to game G_1. The only difference is that in game G_2, the challenger modifies the key generation algorithm so that the secret information is information-theoretically hidden from the adversary's view. So, the derivation key is computed such that $K^d_w = F_r(Rand_r, l_w)$ and public values $Y_{w,u}$ is computed for every successor u of w, as $Y_{w,u} = k^d_u \oplus H(F_r(l_u))$ and $Y_{v,w}$ is computed for every predecessor node v of w, as $Y_{v,w} = Rand_r \oplus H(K^d_v || l_w)$. Now, the challenger chooses a bit $b \leftarrow \{0,1\}$ randomly and compute the derivation key of challenge node w such that $k^d_w = F_r(l_w)$ if $b = 1$ and k^d_w is assigned a random string $Rand_r$ if $b = 0$. Thus, we can conclude from the same argument in Lemma *A*.1 that

$$|Pr[T_2] - Pr[T_1]| \leq \in_{PRF} \quad (4)$$

We can see from the game that the adversary does not know the bit b. It becomes possible because k^d_w is information-theoretically hidden from the view of the adversary. It concludes that the adversarial view will remain the same despite the random value, in either case, $b = 0$ or 1. Hence,

$$Pr[T_2] = 1/2 \quad (5)$$

Thus, from Equations (1), (2), (4) and (5).

$$Adv^{ST}_{A_{IND}}(\rho) \leq 2 \cdot \in_{PRF} \quad (6)$$

It means that the advantage of the adversary A^{ST}_{IND} in *IND ST* game ($G2$) for the correct guess of bit b will remain the same as a negligible value.

General case: The challenge node w is any node in the hierarchy except the root node or any of its successors. Hence, we begin the series of games G_0, G_1 considering the root nodes. We take each of the ancestors of w in the order specified below and propose a game imitating the structure of game G_1 (for other root nodes if any) or imitating the structure of game

G_2. The ancestors of w are processed according to the rules mentioned below until w itself is reached.

To process a node, all its ancestor nodes have to be processed. If v is another root node in G, then node v is processed by running the game G_v following the approach described in the first case. If v is not a root node in G, then follow the approach described in the second case.

Game G_i ($\mathbf{1} \leq \mathbf{i} \geq \mathbf{h}$): Assuming there are h number of nodes to process including a challenge node w. It can be seen from game G_i that,

$$|Pr[T_i] - Pr[T_{i-1}]| \leq \in_{PRF} \tag{7}$$

Adding inequalities for $i = 1..h$, we get

$$|Pr[T_h] - Pr[T_0]| \leq h \cdot \in_{PRF} \tag{8}$$

To conclude this proof, we obtain from the last game G_h that the adversary in the game has no knowledge regarding the bit b. This is because k_w^d here is information-theoretically hidden from the adversary's view, and thus it is indistinguishable from a random string. Therefore, the correct guessing probability for b in the last game G_h by the adversary is,

$$Pr[T_h] = 1/2 \tag{9}$$

From (8) and (9), we can conclude that

$$Pr[T0] \leq 1/2 + h \cdot \in_{PRF} \tag{10}$$

This implies that in-game G_0, the correct guessing probability is not more than 1/2. It concludes this proof and shows that the proposed scheme is KI secure. □

4. Comparison

In this section, we compare our proposed scheme with the other similar existing flexible HKAS schemes [4–7] in the literature. Table 1 compares the proposed scheme with similar existing schemes. For comparison, the table considers the following attributes: system public storage requirement, secret storage per user per subscription (or per node), decryption key derivation time, consideration of anti-symmetric exception, and transitive exception, support of class delegation, and addressing dynamic operations. In the table, n and r are the number of nodes and the edges, respectively.

The hierarchical access control constructions store public values, which are used for key derivation purposes by the users in the hierarchy. As shown in the table, Yeh *et al* [4] scheme follows Akl & Taylor scheme [1] and stores two public parameters each of size $n \times 3072$ bits at each node. Therefore, the total public store will be $2n^2 \times 3072$. In Lin *et al* [6] scheme, only one parameter for each node is stored of 3072 bits each. Y F Chang [5] scheme stores one parameter of size $n \times 256$ bits and two additional parameters of size 256 bits for each node. In Pareek *et al* [7] scheme, seven public parameters for each node of size 3072 bits each are stored. While in the proposed scheme, one public label is stored for each node and each edge of size 128 bits each.

For symmetric-key cryptography, the proposed scheme assumes a 128-bit security level. The other schemes [4, 6, 7] in the table are based on asymmetric key cryptography, and thus the considered key size is approx 3027 bits. Y F Chang [5] scheme is based on elliptic Curve Cryptography, so the considered key size is 256 bits. Whereas the proposed scheme is based on symmetric-key cryptography and the considered key size is 128 bits.

For deriving the key of the target node having a d number of nodes between source and the target node, Yeh *et al* [4] and Lin *et al* [6] schemes perform d times modular exponentiation operation. In Y F Chang [5] scheme, multiplication operations, hash operations, modulo multiplications, and addition operations are performed. Whereas, in the Pareek *et al* [7] scheme, bilinear pairings, multiplications, and modular exponentiation operations are performed. The proposed scheme uses d (i.e., the length of the path considered) efficient hash operations and XOR operations to derive a node's key.

All of the schemes [4–7] compared in the table describe both anti-symmetric and transitive exceptions but the proposed scheme considered only the anti-symmetric exception and not the transitive exception. The reason for not considering the transitive exception is that it usually disallows general hierarchical considerations in a system.

Table 1. Comparison of the proposed scheme with similar existing schemes

Schemes⇒ Functional ⇓	Yeh *et al* [4]	Lin *et al* [6]	Y F Chang [5]	Pareek *et al* [7]	Proposed Scheme
Public storage: (bits)	$2n^2$ (3072 bits)	n (3072 bits)	$(n^2 + 2n)$ (256 bits)	$7n$ (3072 bits)	$3n$ (128 bits)
Private storage per user subscription	2×3072 bits	2×3072 bits	2×256 bits	7×3027 bits	128 bits
Key derivation time	High	High	Medium	Medium	Low
Anti-symmetric exception	Yes	Yes	Yes	Yes	Yes
Transitive exception	Yes	Yes	Yes	Yes	No
Class delegation(s)	No	No	No	Yes	Yes
Dynamic operation	No	No	Yes	Yes	Yes

The proposed scheme and the Pareek *et al* [7] scheme also address the class delegation problem. Yeh *et al* [4] and Lin *et al* [6] schemes failed to address the dynamic operations. Therefore from table 1, it is evident that the proposed scheme requires less public and private storage as compared to the other existing schemes. It has also addressed the dynamic operations and the class delegation problem.

5. Conclusions and future work

The rare but more generic situations in a practical organizational hierarchy will be implemented using exceptions. This work proposed the symmetric key-based hierarchical access control scheme efficiently addressing the anti-symmetric exception and class delegation problem compared with the existing asymmetric key-based schemes. This will be the first such scheme addressing exceptions using symmetric key cryptography to the best of our knowledge. The proposed scheme requires low system public storage, secret storage per user per subscription, and key derivation time compared to the similar existing schemes. The dynamic operations are addressed and handled locally in the hierarchy. The security of the scheme is formally analyzed against the stronger notion of security named key indistinguishability.

As future work, the proposed scheme's performance will be experimentally evaluated and compared with the other similar existing schemes. Also, the anti-transitive exception in a hierarchy using symmetric keys needs to be further reviewed.

Appendix I. APPENDIX

Lemma 1 $|Pr[G_1] - Pr[G_0]| \leq \in_{PRF}$

Proof Let's assume there exists an adversary A_{IND} that can distinguish between game G_0 and game G_1. We now demonstrate how to construct an algorithm A_F, using A_{IND} as a black-box, can distinguish between truly random and pseudorandom functions.

Algorithm A_F runs the PRF game described in Definition 3.3. Thus, it has given access to an oracle function g(), either a truly random function or a pseudorandom function. Algorithm A_F randomly chooses between one of the game G_0 and game G_1 to simulates the environment of A_{IND}. Here, if A_F is interacting with a pseudorandom function g(), then the simulation becomes the same as game G_0; otherwise, it is the same as game G_1. First, the access hierarchy for the acyclic graph is set up, where key K_W is computed via oracle g as follows: $K_w^d = g(l_w)$, and for every successor u of w, $Y_{w,u} = K_w^d \oplus g(l_u)$.

It is equivalent to game G_0 when oracle function g() computes as a pseudorandom function, and equivalent to game G_1 when oracle function g() computes as a true random function.

After serving A_{IND} with the resulting public information, A_F can readily reply to any Corrupt query that A_{IND} may issue since A_F knows all the secret keys except K_w^d. On receiving the challenge query from A_{IND}, A_F picks a random bit b $\in \{0,1\}$. If b = 0, then A_F computes the actual key k_w^d associated with node W; otherwise, if b = 1, A_F assigns a random key of the same length to k_w^d.

Finally, A_{IND}, as his best guess outputs bit b' $\in \{0,1\}$ whether he was given the actual key k_w^d or a random key. If b = b', then A_F gives an output 1, assuming it is a pseudorandom function; otherwise, A_F gives an output 0, assuming it is a truly random function. Now we have

$$
\begin{aligned}
\in_{PRF} &\geq Adv_{A_F}^{PRF}(\rho) \\
&= |Pr[A_F \text{ outputs } 1|\text{g is a PRF}] \\
&\quad - Pr[A_F \text{ outputs } 1|\text{g is a random function}]| \\
&= |Pr[\text{b=b'}|\text{g is a PRF}] \\
&\quad - Pr[\text{b=b'}|\text{g is a random function}]| \\
&= |Pr[A_{IND} \text{ guesses b' correctly}|\text{g is a PRF}] \\
&\quad - Pr[A_{IND} \text{ guesses b' correctly} \\
&\qquad |\text{g is a random function}]| \\
&= |Pr[T_0] - Pr[T_1]|.
\end{aligned}
$$

It concludes the statement. □

References

[1] Selim G A and Peter D T 1983 Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.* 1: 239–248

[2] Mikhail J A, Marina B, Nelly F and Keith B F 2009 Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.* 12:18:1–18:43

[3] Jason C and Hemanth K 2006 Delegation in role-based access control. In: *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, pp 174–191

[4] Jyh-haw Y, Randy C and Richard N 2003 Key assignment for enforcing access control policy exceptions in distributed systems. *Inf. Sci.* 152: 63–88

[5] Ya-Fen C 2015 A flexible hierarchical access control mechanism enforcing extension policies.*Secur. Commun. Netw.* 8: 189–201

[6] Iuon-Chang L, Min-Shiang H and Chin-Chen C 2003 A new key assignment scheme for enforcing complicated access control policies in hierarchy. *Future Gener. Comput. Syst.* 19: 457–462

[7] Gaurav P and Purushothama B R 2019 Extended hierarchical key assignment scheme (e-hkas): how to efficiently enforce explicit policy exceptions in dynamic hierarchies. *Sādhanā* 44: 235

[8] Mikhail J A, Keith B F and Marina B 2005 Dynamic and efficient key management for access hierarchies. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA*, pp 190–202

[9] Arcangelo C, de Santis A and Barbara M 2016 Key indistinguishability versus strong key indistinguishability for hierarchical key assignment schemes.*IEEE Trans. Dependable Secur. Comput.* 13: 451–460

[10] Jyh-haw Y 2009 Enforcing non-hierarchical access policies by hierarchical key assignment schemes. *Inf. Process. Lett.* 110: 46–49

[11] Ping-Kun H, Mu-Ting L and Iuon-Chang L 2018 Flexible hierarchical key assignment scheme with time-based assured deletion for cloud storage. In: *International Computer Symposium*, pp 599–607, Springer

[12] Tsung-Chih H, Tzer-Long C, Tzer-Shyong C and Yu-Fang C 2019 Elliptic curve cryptosystems-based date-constrained hierarchical key management scheme in internet of things. *Sensors Mater.* 31: 355–364

[13] Neal K 1987 Elliptic curve cryptosystems. *Mathe. Comput.* 48: 203–209

[14] Victor S M 1985 Use of elliptic curves in cryptography. In: *Conference on the theory and application of cryptographic techniques*, pp 417–426, Springer.

[15] Ezedin B and Ravi S S 2000 Framework for role-based delegation models. In: *16th Annual Computer Security Applications Conference (ACSAC 2000), 11–15 December 2000, New Orleans, Louisiana, USA*, page 168

[16] Jason C and Hemanth K 2008 Delegation in role-based access control.*Int. J. Inf. Secur.* 7: 123–136

[17] Indrajit R, Dieudonne M, Indrakshi R and Keesook J H 2013 A model for trust-based access control and delegation in mobile clouds. In: *Data and Applications Security and Privacy XXVII-27th Annual IFIP WG 11.3 Conference, DBSec 2013, Newark, NJ, USA, July 15-17,2013. Proceedings*, pp 242–257

[18] Anna Lisa F, Federica P and Chiara R 2021 Verifiable hierarchical key assignment schemes. In: *IFIP Annual Conference on Data and Applications Security and Privacy*, pp 357–376, Springer.

[19] Naveen Kumar and Anish Mathuria 2019 Comprehensive evaluation of key management hierarchies for outsourced data. *Cybersecurity*, 2: 8

[20] Eduarda S V F, Kenneth G P and Bertram Poettering 2013 Simple, efficient and strongly ki-secure hierarchical key assignment schemes. In: *Cryptographers' Track at the RSA Conference*, pp 101–114, Springer.

[21] Paolo D A, De Santis A, Anna Lisa Ferrara and Barbara Masucci 2010 Variations on a theme by akl and taylor: Security and tradeoffs. *Theor. Comput. Sci.*, 411: 213–227

[22] Giuseppe A, de Santis A, Anna L F and Barbara M 2012 Provably-secure time-bound hierarchical key assignment schemes. *J. Cryptol.* 25: 243–270

[23] Vikas R V, Naveen K and Shafiqul A 2021 Classifying time-bound hierarchical key assignment schemes. In: *Advances in Computer, Communication and Computational Sciences*, pp 111–119, Springer

[24] Naveen K, Shailesh T, Zhigao Z, Krishn K M and Arun K S 2018 An efficient and provably secure time-limited key management scheme for outsourced data. *Concurr. Comput. Practice Exp.*, p 30