Sādhanā (2019)44:235
https://doi.org/10.1007/s12046-019-1216-8

Extended hierarchical key assignment scheme (E-HKAS): how to efficiently enforce explicit policy exceptions in dynamic hierarchies

GAURAV PAREEK* and B R PURUSHOTHAMA

Department of Computer Science and Engineering, National Institute of Technology Goa, Farmagudi, Farmagudi 403401, India
e-mail: gpareek@nitgoa.ac.in; puru@nitgoa.ac.in

MS received 18 February 2019; revised 19 June 2019; accepted 8 September 2019

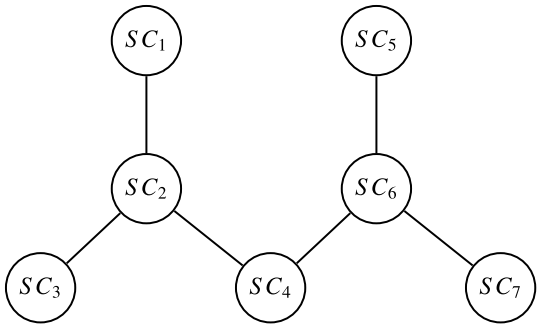
Abstract. In this paper, we focus on practically motivated flexibility requirements for the hierarchical access control model, namely transitive exception and anti-symmetric exception. Additionally, we motivate a new flexibility requirement called "class delegation with descendant(s) safety" in a practical application scenario. We propose our extended hierarchical key assignment scheme (E-HKAS) that satisfies all three aforementioned flexibility requirements in a dynamic hierarchy of security classes. To propose a generic E-HKAS, we model the hierarchical access control policy as a collection of access groups. E-HKAS enforces transitive and anti-symmetric exceptions using an efficient group-based encryption scheme. To enforce class delegation with descendant(s) safety, we propose a novel cryptographic primitive called group proxy re-encryption (GPRE) that supports proxy re-encryption between two access groups. We present an IND-CPA-secure construction of our proposed GPRE scheme and formally prove its security. Performance analysis shows that the proposed E-HKAS enforces explicit transitive and anti-symmetric exceptions more efficiently than the existing approaches in the literature. Computation cost for key derivation is constant and does not depend on the depth of the hierarchy. Also, to enforce class delegation with descendant(s) safety, the proposed E-HKAS requires constant number of computational steps to be executed.

Keywords. Flexible access control; extended hierarchical key assignment; group proxy re-encryption.

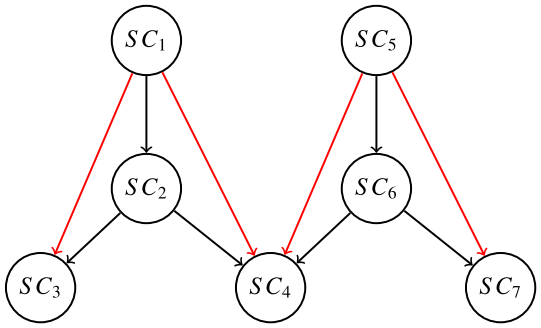
1. Introduction

Cryptographic access control is for a data owner to enforce desired access control policy on encrypted data. A data owner, who can be an individual or a business organization or a government agency, stores varying amounts of data on a shared storage and manages access for a set of users in a controlled manner. A data owner can specify in his policy the authorized and unauthorized user(s) for each of his shared data items. To preserve privacy, data items are encrypted before storing. Cryptographic access control enables the data owner to manage encryption keys of the encrypted data items such that they are accessible to only authorized users. Of all existing access control models, hierarchical access control model is the most widely studied one because of its natural suitability to most organizational structures. In hierarchical access control model, a set of users is divided into disjoint partitions called *security classes*, which in turn are arranged to form a partially ordered set (poset) under the relation "can access" [1]. Figure 1 depicts an example access hierarchy. A security class SC_i for $i \in \{1, 2, 3, 4, 5, 6, 7\}$, can access all data encrypted for itself (*reflexive*). Additionally, SC_1 can access data encrypted for SC_2 but the reverse is not allowed (*anti-symmetric*). SC_1 can access SC_2 and SC_2 can access SC_4 means that SC_1 can access SC_4 (*transitive*). A hierarchical key assignment scheme (HKAS) assigns encryption keys of all security classes in the hierarchy such that the three properties of the partial order "can access" are satisfied. Each security class can access/derive encryption keys of all its descendant security classes using a key derivation procedure. For a dynamic hierarchy, the HKAS has procedures for securely handling dynamic updates in access rights. In particular, if an edge (SC_j, SC_i), such that SC_j can access SC_i, is deleted then SC_j should not be able to access any of the future messages encrypted for SC_i and all its descendants. This is called *forward secrecy*. If a new edge (SC_j, SC_i), so that SC_j can access SC_i, is added then SC_j should be able to access all future messages encrypted for SC_i and its descendants but not the ones sent prior to addition of the edge. This is called *backward secrecy*.

*For correspondence
Published online: 16 November 2019



(a) Hasse diagram of the sample access hierarchy



(b) Sample access hierarchy with descriptive transitive edges (red-colored)

Figure 1. A sample access hierarchy with directed edge (SC_i, SC_j) indicating that SC_i "can access" SC_j. (a) Hasse diagram of the sample access hierarchy. (b) Sample access hierarchy with descriptive transitive edges (red-coloured).

It is noted in numerous works [2–4] that data owners may not always want to enforce anti-symmetry and transitivity restrictions globally. In practical application scenarios, explicit policy exceptions may be desired in the hierarchy by either extending the access rights of a security class or by revoking their designated access to some data items or both. These explicit policy exceptions include *transitive exception* and *anti-symmetric exception* [5, 6]. In transitive exception, SC_i can access SC_j and SC_j can access SC_k but SC_i cannot access SC_k. In anti-symmetric exception, SC_i can access SC_j and SC_j can access SC_i for $SC_i \neq SC_j$. Policy exceptions add flexibility to the existing access control model. Explicitly enforcing these policy exceptions on the traditional hierarchical access control model in selected parts of the hierarchy must not subject the data owner to any substantial computation or storage overhead. In particular, a scheme that enforces explicit transitive and anti-symmetric exceptions with traditional hierarchical access control must satisfy the following performance requirements defined by Atallah *et al* [1] for a traditional HKAS:

1. For a hierarchy of n classes and m edges, public storage requirement is $O(n + m)$.
2. Number of secrets stored by a security class is constant irrespective of the number of classes it can access.
3. Number of symmetric key operations required to derive encryption key(s) of its descendant class(es) is upper bounded by the depth of hierarchy, whereas number of asymmetric key cryptography operations is constant.
4. Dynamic security is ensured by updating encryption keys of classes descendant to the class incident to the edge that is added or removed.

We stress that none of the previous works on hierarchical key assignment with explicit policy exceptions (analysed in section 5.3) satisfy the aforementioned performance requirements. In this paper, we propose a novel HKAS for dynamic hierarchies and term it as extended hierarchical key assignment scheme (E-HKAS). E-HKAS efficiently enforces hierarchical access control in a dynamic hierarchy of security classes with explicit transitive and anti-symmetric exceptions in portions selected by the data owner. Additionally, E-HKAS enforces another practically motivated flexibility requirement named *class delegation with descendant(s) safety*. Class delegation with descendant(s) safety is a novel, practically motivated flexibility requirement introduced in this paper. It allows the data owner to temporarily delegate decryption rights of a security class SC_i to another security class SC_j in such a way that SC_j can access data encrypted directly for SC_i but cannot access data encrypted for any of SC_i's descendant security classes. For example in figure 1, class delegation with descendant(s) safety from SC_6 to SC_2 means that data intended directly for SC_6 is accessible to SC_2 and SC_1. However, data intended for SC_7 is accessible to neither SC_2 nor SC_1. We point out later that class delegation with descendant(s) safety cannot be realized as a special case of transitive or anti-symmetric exception. Also, if the data owner wishes, delegation with descendant(s) safety can be revoked while preserving dynamic security. In the following, we motivate class delegation with descendant(s) safety in a practical application scenario. We also present an overview of the proposed approach to enforce class delegation with descendant(s) safety with transitive and anti-symmetric exceptions.

1.1 *Motivation and problem description*

We extend the scenario of distributed database system considered in [3] to motivate the three explicit policy exceptions, namely transitive exception, anti-symmetric exception and delegation with descendant(s) safety.

1.1a *Transitive exception*: Suppose that there are users in an organization that can access a query processor (QP).

Each user has access to encryption key of its local QP. The QP in turn has access to the encryption key(s), using which the database tables are encrypted. However, accessing the database tables directly by the users is prohibited. Hence, there is no way the users can access the encryption key used for encrypting the database tables. The users issue their queries to QP, which obtains the query result(s) from the tables in encrypted form, decrypts the results and encrypts them again using its own encryption key. The users can now decrypt the results using encryption key of QP, to which they have designated access. Hence, if users are modelled as SC_1, the QP as SC_2 and database tables as SC_3, we require that SC_1 can access SC_2 and SC_2 can access SC_3 but SC_1 cannot access SC_3. Thus, transitive exception is desirable in this scenario.

1.1b *Anti-symmetric exception*: Consider two distinct sites, Site-1 and Site-2, of the same organization as in figure 2. Suppose that a user from Site-2 (modelled as SC_4) wishes to access data that is not present locally but on Site-1 of the distributed storage. The requesting user issues the corresponding query to the local QP, QP2, modelled as SC_5. Since the queried data is not available locally, QP2 forwards the query to the remote QP, QP1, modelled as SC_3. Now, QP1 fetches the desired results from its local database and encrypts them with its own encryption key before sending them to QP2. In order for QP1 to access the query results and make them available to users of Site-2, QP2 must have access to QP1's encryption key. Similarly, if a request for data stored at Site-2 is initiated by users on Site-1, QP1 is required to access QP2's encryption key to serve the request. Since both SC_2 and SC_5 access each other, anti-symmetric exception is a desirable feature in this scenario.

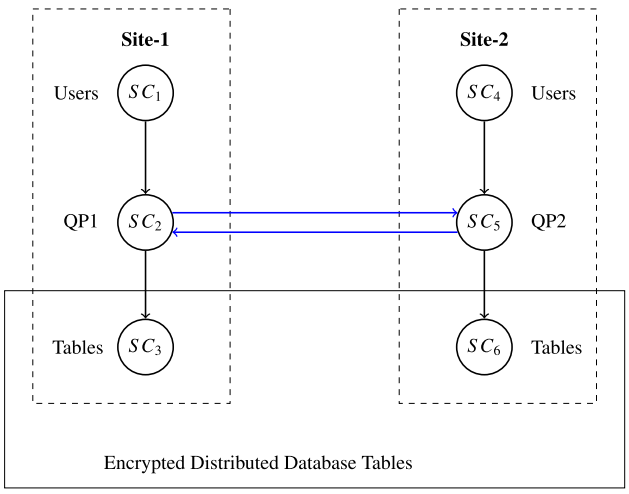


Figure 2. Distributed database system scenario.

1.1c *Class delegation with descendant(s) safety*: In the distributed database system example discussed earlier, suppose that users of Site-2 (SC_4) are allowed to temporarily access database tables encrypted and stored on Site-1. Also, suppose that after the specified duration, decryption rights of users of Site-2 for data stored on Site-1 are revoked. Trivially, this can be achieved by securely transmitting encryption key of QP1 (SC_2) to QP2 (SC_5). Also, to revoke the delegation, the data owner updates the encryption key of QP1. However, if there are N sites, performing secure key transfer every time a delegation request arrives can cause significant performance overhead on the data owner.

Hence, to circumvent the need for secure transmission of encryption keys for each request, the data owner gives out some public information, using which any semi-trusted entity can transform data encrypted for QP1 (SC_2) in such a way that it can be accessed by QP2 (SC_5). This ciphertext transformation, carried out by any semi-trusted entity, causes no threat to privacy of the data or key material of the two QPs. Moreover, transformation of data encrypted under QP1's encryption key means that encrypted database tables cannot be transformed using the public information (descendant(s) safety). When the delegation has to be revoked, QP1 updates its encryption key to prevent QP2 from accessing any future encryptions. In this way, class delegation with descendant(s) safety does not require the data owner to carry out secure transmission of encryption key(s).

Since entities are modelled as security classes, we state the problem of class delegation with descendant(s) safety as follows. Data intended directly to SC_2 is temporarily accessible to SC_4 and all those classes that have designated access to SC_4. However, SC_4 cannot access the data encrypted for any of SC_2's descendants.

1.2 *Overview of the solution*

The proposed E-HKAS enforces transitive exceptions, anti-symmetric exceptions and class delegation with descendant(s) safety while ensuring that all desirable performance requirements are satisfied. In this section, we present an overview of our proposed approach and informally discuss the cryptographic primitives used for design of the proposed E-HKAS.

We propose to view a hierarchical access control policy $\mathcal{P}$ as a collection of access groups, one per security class. Access group G_i corresponding to a class SC_i contains as its members each security class SC_j such that SC_j can access SC_i, that is

$$G_i = \{SC_i\} \cup \{SC_j : SC_j \text{ can access } SC_i\}. \quad (1)$$

If the partial order "can access" is denoted by "$\preceq$", we have

$$G_i = \{SC_i\} \cup \{SC_j : SC_i \preceq SC_j\}. \quad (2)$$

Suppose ek_i is the encryption key corresponding to class SC_i in the hierarchy. To enforce hierarchical access control policy, the data owner uses group-based encryption scheme [7] for encrypting ek_i using group pubic key of G_i and stores the resulting ciphertext C_i on the public bulletin. The fundamental property of group-based encryption is that all encryptions under a group's public key can be decrypted using individual secret keys of the group's current members. Also, a non-member learns nothing about the message encrypted for the group. Thus, each class $SC_j \in G_i$ can download C_i from the public bulletin, and decrypt it using its individual secret key to obtain ek_i. This requires a single decryption operation.

Hierarchical access control policy $\mathcal{P}$ for hierarchy of figure 1 is the set $\mathcal{P} = \{G_1, G_2, G_3, G_4, G_5, G_6, G_7\}$ where

$$G_1 = \{SC_1\}, G_2 = \{SC_1, SC_2\}, G_3 = \{SC_1, SC_2, SC_3\},$$
$$G_4 = \{SC_1, SC_2, SC_4, SC_5, SC_6\}, G_5 = \{SC_5\},$$
$$G_6 = \{SC_5, SC_6\}, G_7 = \{SC_5, SC_6, SC_7\}.$$

Transitive exception: Suppose that the data owner wishes to enforce explicit transitive exception by denying access of SC_4 to SC_1, that is, despite the fact that $SC_4 \preceq SC_2$ and $SC_2 \preceq SC_1$ the data owner wants to enforce $SC_4 \not\preceq SC_1$. To accomplish this, the data owner removes SC_1 from G_4. As a result, the updated access control policy in terms of access groups is $\mathcal{P}' = \{G_1, G_2, G_3, G'_4, G_5, G_6, G_7\}$ where $G'_4 = \{SC_2, SC_4\}$.

Anti-symmetric exception: To enforce anti-symmetric exception in the original hierarchy of figure 1, such that both $SC_1 \preceq SC_3$ and $SC_3 \preceq SC_1$ hold, the data owner adds a new member SC_3 to the existing access group G_1. The resulting updated access control policy $\mathcal{P}'' = \{G''_1, G_2, G_3, G_4, G_5, G_6, G_7\}$ where $G''_1 = \{SC_1, SC_3\}$.

Class delegation with descendant(s) safety: Consider that a data owner wants to enforce class delegation with descendant(s) safety in the hierarchy of figure 1 so that SC_1 and SC_2 can access for a finite duration of time, data encrypted for SC_6 but not the data intended for any of SC_6's descendants, namely SC_7 in this case. Although SC_4 is a descendant of SC_6, since it is also a descendant of SC_2 in the original hierarchy, it continues to be accessible to SC_2. We extend the notion of group encryption to group proxy re-encryption (GPRE). We apply the proposed GPRE primitive to enforce class delegation with descendant(s) safety with constant computation overhead on the data owner [8]. Proxy re-encryption (PRE) is a cryptographic primitive that enables a delegator A to output a re-encryption key $RK_{A \to B}$ that can be used by any semi-trusted entity, called proxy, to transform ciphertext encrypted for A into one that can be decrypted using B's secret key. This process of transforming the ciphertext is called re-encryption [9]. The proxy does not learn anything about the underlying plaintext or secret key(s) of the communicating parties (A and B). In GPRE, a ciphertext encrypted for group G_i can be re-encrypted using a GPRE key $RK_{i \to j}$ such that the re-encrypted ciphertext can be decrypted by members of another group G_j using their individual secret keys. In the example scenario of figure 1, if the data owner outputs group re-encryption key $RK_{6 \to 2}$, data encrypted for group G_6 can be re-encrypted such that it can be decrypted by members of G_2, i.e., SC_1 and SC_2. However, data encrypted under group public keys of G_7 cannot be decrypted successfully by SC_1 or SC_2. Thus, by employing GPRE, the data owner achieves class delegation with descendant(s) safety by computing a group re-encryption key, which costs him a constant number of computation steps.

In view of the discussion presented earlier, a data owner can enforce all explicit policy exceptions in the dynamic hierarchy either through constant number of group update operations (transitive and anti-symmetric exceptions) or by outputting a single group re-encryption key (delegation with descendant(s) safety). In the following, we list desirable performance requirements of the group-based encryption scheme suitable for extension as GPRE scheme and application in hierarchical access control with explicit policy exceptions.

1. *Constant group public storage*: Access control policy for a hierarchy of n classes is viewed as a collection of n access groups. Thus, O(1) public parameters per access group amount to O(n) overall public storage overhead.
2. *Constant-length secret key of members with overlapped membership*: It is clear from the solution overview that a security class can be a member of multiple access groups. Thus, the underlying group encryption scheme must allow overlapped group membership for group members. However, it must not require the members to store anything apart from a constant-length secret key.
3. *Group update operations with constant re-keying cost*: Underlying group encryption scheme must feature group update operations with constant computation overhead.

The group encryption scheme proposed by Koti and Purushothama [7] satisfies all performance requirements given earlier. We extend their scheme by providing our re-encryption functions and propose a concrete GPRE scheme. We use our concrete GPRE to explicitly enforce all three policy exceptions in a dynamic hierarchy.

2. Related works

In this section, we review important HKAS solutions for traditional hierarchies. Also, we analyse the existing HKAS solutions designed specifically for enforcing explicit policy exceptions in addition to hierarchical access control. Finally, we briefly discuss performance enhancements of

our proposed E-HKAS features with respect to the analysed HKAS, with and without explicit policy exceptions.

Akl and Taylor [10] proposed an early notion of key management for access hierarchies where some users can access more (encrypted) data than others. In particular, keys are assigned such that users higher in the hierarchy can obtain secret keys of the ones lower down in the hierarchy through series of symmetric decryptions and hash computations. Suitable for only static access hierarchies, the scheme proposed by Akl and Taylor [10] requires users higher in the hierarchy to store more number of keys compared with the ones lower in the hierarchy. Later, Sandhu [11] proposed a hierarchical access control solution based on one-way hash functions. However, this scheme also requires classes to store non-constant number of secrets. Likewise, different HKAS solutions [12–15] were proposed aimed at enhancing practical applicability, scalability and security. However, all of them are suitable for static hierarchies and require different numbers of class secrets to be stored by different security classes in the hierarchy. A dynamic hierarchical access control scheme that satisfies the forward and backward secrecy requirements was first proposed by Hui and Chin [16]. However, dynamic procedures of their scheme require updating public parameters corresponding to all the security classes in the hierarchy.

Security notions and concrete performance requirements for a HKAS were formalized by Atallah *et al* [1], who also proposed their key assignment scheme, which satisfies all the requirements defined therein. Their scheme is based on symmetric encryption and hash functions and requires constant-length secrets to be stored by all security classes. The key derivation function requires steps linear in the depth of the hierarchy. Majority of HKAS proposed afterwards are proved to be secure under the formal security notions named "key indistinguishability" (KI) or "key non-recoverability" given by Atallah *et al* [1]. Odelu *et al* [17] proposed a HKAS in public-key cryptography setting that uses linear polynomials and elliptic curve cryptography (ECC). Their scheme is proved to be secure against key recovery (KR) under various practical attack scenarios. Another HKAS by Odelu *et al* [18] employs only symmetric encryption primitives and one-way hash functions. Unlike other previously proposed ECC-based schemes, their scheme is secure against man-in-the-middle attack. Also, it requires constant number of secrets to be stored by each class with linear overall public storage requirement. Chen *et al* [19] designed a HKAS that uses multi-hop PRE schemes. Their scheme is computationally expensive because of the use of multi-linear maps. Each class has to rely on a third-party computation service provider for carrying out $O(d)$ transformations on the ciphertext intended for a class until it becomes ciphertext intended for any superior class. All these transformations involve computation of multi-linear maps. Tang *et al* [20] proposed an elegant scheme involving vectors and linear geometry where each class in the system has public and private keys in the form of key vectors. Encryption key of the descendant classes can be derived using inner product of the vector associated with the private key of the class and that with the public key of its descendants. Their scheme requires $O(n^2)$ public storage overhead for n security classes and $O(d)$ computation cost for key derivation. The HKAS by Pareek and Purushothama [21] is based on transformation of ciphertext, which requires only one transformation instead of $O(d)$. In particular, their scheme requires operations on public transformation keys attached to each edge on a path between the two classes. These operations are computationally more efficient than repeated ciphertext transformation using multi-linear mappings. Chen and Tzeng [22] proposed another HKAS that achieves constant key derivation cost irrespective of the depth of the hierarchy. Their scheme is proved to be secure under the KI model based on the security of Broadcast Encryption and unforgeable one-time signature employed by them. However, it requires $O(n^2)$ public storage. Arguably, in some special cases, dynamic updates in their scheme require updating the whole hierarchy. HKAS solutions reviewed so far are meant for enforcing strict hierarchical access control policies. None of them allows any exception(s) to hierarchical access control policy in any part of the hierarchy.

In many organizations, the access structure is such that the traditional hierarchical model does not suffice. With an objective to allow hierarchical policy exceptions in selected parts of the hierarchy, Yeh, Chow and Newman proposed what is famously known as the YCN scheme [2, 3]. The YCN scheme is a static key assignment scheme, which enhances the scheme by Akl and Taylor [10] to enforce explicit policy exceptions in addition to traditional hierarchical access control. The YCN scheme restructures the whole hierarchy (assign new public parameters to edges and nodes) every time it enforces a new policy exception. Their scheme is for static hierarchies and lacks formal security analysis. Moreover, it has been proved to be insecure by Hwang [23], who provided a counter-evidence to show that even a descendant security class can decrypt ciphertext intended for a given security class. Lin *et al* [4] proposed a secure and flexible HKAS to allow transitive and anti-symmetric exceptions in the traditional hierarchical access control model. However, the key derivation procedure of their scheme requires $O(d)$ modulo exponentiation/multiplication operations to be performed for a hierarchy of depth d. Chang and Chang [5] improved Lin *et al*'s scheme [4] and proposed a tolerant HKAS for dynamic hierarchies with transitive and anti-symmetric exceptions. Their scheme is called tolerant HKAS because it is possible to "securely" update secret key of any given class without having to update the secret(s) of any other class(es). However, their scheme [5] requires $O(n^2)$ public

storage and features inefficient dynamic update operations, which require public parameters corresponding to all ancestors as well as all descendants of the affected security class to be updated by the data owner. In 2015, Chang [6] proposed another flexible HKAS with the same motivation of enforcing explicit transitive and anti-symmetric exceptions in a traditional access hierarchy. The scheme in [6] uses ECC and efficient one-way hash functions to achieve constant key derivation cost. However, storage cost of implementing the scheme is $O(n^2)$ for a hierarchy of n classes. Also, dynamic update(s) requires the data owner to update public keys of all descendant as well as ancestor security classes of the affected security class.

We observe three common drawbacks of all the HKAS solutions discussed earlier for dynamic hierarchies with explicit exceptions. Firstly, they require information regarding the explicit exceptions at the time of setting up of the system. On the contrary, it may be desired that these exceptions are enforced in an on-demand manner after initial key assignment. Secondly, none of the aforementioned schemes have been analysed from the point-of-view of time required to enforce the explicit policy exceptions at the run-time. Thirdly, none of the aforementioned flexible HKAS solutions satisfies efficiency requirements defined by Atallah *et al* [1] originally for HKAS in the traditional scenario (see section 1.2). With these three issues as motivation, we propose our E-HKAS that enforces explicit policy exceptions in an on-demand manner using procedures that run in constant time while satisfying efficiency requirements desirable for HKAS without policy exceptions. Additionally, we identify new practically motivated flexibility requirement for HKAS, called class delegation with descendant(s) safety (see section 1.1), which is featured by the proposed E-HKAS apart from transitive and anti-symmetric exceptions.

2.1 *Our contributions*

In light of the motivation and performance issues with existing flexible HKAS solutions, we summarize the contributions of our paper as follows.

1. We propose a novel cryptographic primitive called E-HKAS. In addition to hierarchical access control policy, E-HKAS also enforces transitive and anti-symmetric exceptions, in parts of the hierarchy chosen by the data owner, in an on-demand manner. In addition, E-HKAS also enforces a practically motivated novel policy exception for access hierarchies called class delegation with descendant(s) safety.
2. We propose to express the hierarchical access control policy as a collection of access groups and employ GPRE to propose generic E-HKAS.
3. A GPRE scheme is a cryptographic primitive formalized in this paper, which supports PRE between two access groups. We provide a system model and IND-CPA security definitions for GPRE. We extend an efficient group encryption scheme [7] to propose a concrete provably secure construction of the proposed GPRE scheme.
4. Finally, we present concrete construction of the proposed E-HKAS by employing procedures of the proposed GPRE scheme. The proposed concrete E-HKAS features constant key derivation cost, requires constant computational effort to enforce explicit policy exceptions and linear overall public storage overhead. Also, it satisfies KI and forward/backward security requirements.

3. E-HKAS

In this section, we define the system model for E-HKAS. First, we present a brief description of group encryption and HKAS. Then we extend the group encryption primitive to GPRE and present the system model and security notions of GPRE.

3.1 *Preliminaries*

Definition 1 (*Group encryption scheme*): A group encryption scheme for a set of groups $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ such that $\forall i, j \in \{1, 2, \ldots, N\} : G_i \cap G_j \neq \phi$ is defined as a collection of procedures $\Gamma_{\mathcal{G}} = \{\mathsf{SetGroup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{GroupJoin}, \mathsf{GroupLeave}\}$:

1. $\mathsf{SetGroups}\ (1^{\lambda}) \rightarrow \{\mathsf{params}, MSK, MPK, \forall u_i \in G_1 \cup \ldots \cup G_N : sk_i\}$: Takes security parameter λ as input and outputs the system parameters params, master public-secret keys and secret keys sk_i corresponding to each user u_i in the entire population of users.
2. $\mathsf{KeyGen}(\mathsf{params}, A) \rightarrow \{PK_A, SK_A\}$: Takes params and index A as input and outputs public-secret key-pair (PK_A, SK_A) corresponding to the group G_A.
3. $\mathsf{Enc}\ (PK_A, m) \rightarrow C_A$: Encrypts message m intended for G_A under the group's public key PK_A.
4. $\mathsf{Dec}\ (sk_i, C_A) \rightarrow m$: Decrypts the ciphertext C_A of a message m under PK_A using secret key sk_i of a member $u_i \in G_A$.
5. $\mathsf{GroupJoin}(A, i)$: Adds a new user u_i to the group G_A so that it can use $\mathsf{Dec}\ (sk_i, C_A)$ to decrypt all future encryptions for G_A but not the ones sent prior to u_i's joining.
6. $\mathsf{GroupLeave}(A, i)$: Removes an existing member u_i from the group G_A and after u_i is removed, it cannot decrypt any future encryptions for G_A (under PK_A).

Definition 2 (*HKAS*): A HKAS for a collection $S = \{SC_1, SC_2, \ldots, SC_n\}$ of security classes arranged under the partial order $\preceq$ is defined as a collection $\Pi_{(S,\preceq)} = \{\mathsf{Initialize}, \mathsf{Derive}, \mathsf{EdgeDelete}, \mathsf{EdgeNew}\}$ defined as follows.

1. $\mathsf{Initialize}(1^{\lambda}) \rightarrow \{\mathsf{par}, \{\forall SC_i \in S, (pk_i, sk_i)\}\}$: Initializes the system by generating global parameters and key-pairs (pk_i, sk_i) for each security class $SC_i \in S$.

2. Derive(sk_i, SC_j): If $SC_i \preceq SC_j$, then outputs $\perp$. Otherwise, outputs symmetric encryption key ek_j of class SC_j.
3. EdgeDelete (SC_i, SC_j): Deletes an edge (SC_i, SC_j) from the hierarchy where $SC_i \preceq SC_j$ such that SC_i and no SC_l for $SC_l \preceq SC_i$ can access future encryptions of SC_j.
4. EdgeNew (SC_i, SC_j): Adds a new edge (SC_i, SC_j) to the hierarchy where $SC_i \preceq SC_j$ such that SC_i and all SC_l for $SC_l \preceq SC_i$ can access future encryptions of SC_j.

3.2 *Proposed E-HKAS*

Definition 3 (*extended hierarchical key assignment*): If $SC_i \prec SC_j$ denotes SC_i as descendant of SC_j or that $SC_j \neq SC_i$ and SC_j can access SC_i, we define E-HKAS as a collection of algorithms $\Pi^{\star}_{(S,\preceq)} = \Pi_{(S,\preceq)} \cup$ {ExcepTrans, UnExcepTras, ExcepAS, UnExcepAS, Delegate, UnDelegate} defined as follows.

1. ExcepTrans (SC_i, SC_j, SC_k): If $SC_i \prec SC_j \prec SC_k$, for SC_i, SC_j, $SC_k \in S$, updates the hierarchy so that transitive exception $SC_i \not\prec SC_k$ holds despite $SC_i \prec SC_j$ and $SC_j \prec SC_k$. No future encryption intended for SC_i is accessible to SC_k.
2. UnExcepTrans (SC_i, SC_j, SC_k): Revokes a transitive exception tuple $(SC_i \prec SC_j, SC_j \prec SC_k, SC_i \not\prec SC_k)$ such that $SC_i \prec SC_k$ and SC_k can access all future encryptions intended for SC_i but not the ones sent prior to revocation.
3. ExcepAS (SC_i, SC_j): If $SC_i \prec SC_j$ holds, updates the hierarchy such that $SC_j \prec SC_i$ also holds and SC_i can access all future encryptions intended for SC_j.
4. UnExcepAS (SC_i, SC_j): Revokes the anti-symmetric exception tuple $(SC_i \prec SC_j, SC_j \prec SC_i)$ such that SC_i can access no future encryptions intended for SC_j.
5. Delegate(SC_i, SC_j): Delegates decryption rights of a class SC_i to a class SC_j such that all future encryptions intended for SC_i can be accessed by SC_j. However, data encrypted for no SC_k such that $SC_k \prec SC_i$ and $SC_k \not\prec SC_j$ in the original hierarchy can be decrypted by SC_j.
6. UnDelegate(SC_i, SC_j): Consider a class delegation with descendant(s) safety from SC_i to SC_j. This procedure revokes this delegation so that no future encryptions directly intended for SC_i can be decrypted by SC_j.

3.3 *Proposed GPRE scheme*

Definition 4 (*GPRE*): A GPRE is defined as a collection $\Gamma^{\star}_{\mathcal{G}} = \Gamma_{\mathcal{G}} \cup$ {ReKeyGen, ReEnc}.

1. ReKeyGen $(SK_A, SK_B) \rightarrow RK_{A \rightarrow B}$: Takes the secret key SK_A of a delegator group and secret key SK_B of the delegatee group to produce the re-encryption key $RK_{A \rightarrow B}$ for re-encrypting the ciphertext C_A intended for G_A into the one intended for G_B.
2. ReEnc $(RK_{A \rightarrow B}, C_A) \rightarrow C_B$: This procedure takes $RK_{A \rightarrow B}$ and the ciphertext C_A of group G_A as input and transforms it such that it can be decrypted by members of G_B using Dec (sk_i, C_B) where sk_i is the member's individual secret key.

Security of proposed GPRE: We define security of GPRE with respect to a probabilistic polynomial time (PPT) adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, which works in two phases, namely query phase ($\mathcal{A}_0$) and challenge phase ($\mathcal{A}_1$). Throughout the rest of the paper, notation $a \xleftarrow{\$} B$ indicates that a is selected uniformly at random from set B.

In the query phase, $\mathcal{A}_0$ queries the oracles $\mathcal{O}_{k_h}, \mathcal{O}_{k_c}, \mathcal{O}_{rk}, \mathcal{O}_e, \mathcal{O}_r$. The oracle $\mathcal{O}_{k_h}(i) \rightarrow PK_i$ outputs group public key of a group G_i. $\mathcal{O}_{k_c}(i) \rightarrow (PK_i, SK_i)$ outputs group public-secret key-pair of G_i. $\mathcal{O}_{rk}(i,j) \rightarrow RK_{i \rightarrow j}$ outputs re-encryption key from G_i to G_j subject to the condition that either none or both of them should have been queried over $\mathcal{O}_{k_c}$. $\mathcal{O}_e(PK_i, m) \rightarrow C_i$ outputs encryption under G_i's public key of any arbitrary message m. Lastly, $\mathcal{O}_r(RK_{i \rightarrow j}, C_i) \rightarrow C_j$ outputs re-encryption of ciphertext C_i into ciphertext intended for G_j, that is C_j.

In the challenge phase, $\mathcal{A}_1$ outputs two plaintext messages M_0, M_1 of same length and an index A^* of the target group G_{A^*}. The challenger $\mathcal{C}$ challenges $\mathcal{A}$ with a challenge ciphertext $C_{A^*} =$ Enc (PK_{A^*}, M_d) for $d \xleftarrow{\$} \{0,1\}$. $\mathcal{A}$ outputs its best guess d' and wins the game (breaks the security of GPRE scheme) if $d' = d$.

Consider the following security game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

Experiment Exp $^{\mathsf{CPA}}_{\Gamma^{\star}_{\mathcal{G}}, \mathcal{A}}(1^{\lambda})$

1. params $\leftarrow$ Setup (1^{λ}).
2. *Query-phase:*
   $(\{PK_A\}, \{(PK_C, SK_C)\}, \{RK_{A \rightarrow B}\}, A^*, M_0, M_1, st)$
   $\leftarrow \mathcal{A}_0^{\mathcal{O}_{k_h}(.), \mathcal{O}_{k_c}(.), \mathcal{O}_{rk}(.,.), \mathcal{O}_e(.,.), \mathcal{O}_r(.,.)}$ (params), for G_A, G_B, G_C, $G_{A^*} \in \mathcal{G}$ and st, the state information.
3. Select a bit $d \xleftarrow{\$} \{0,1\}$.
4. $C_{A^*} \leftarrow$ Enc (PK_{A^*}, M_d).
5. *Guess phase:*
   $d' \leftarrow \mathcal{A}_1^{\mathcal{O}_{k_h}(.), \mathcal{O}_{k_c}(.), \mathcal{O}_{rk}(.,.), \mathcal{O}_e(.,.), \mathcal{O}_r(.,.)}$ (params, C_{A^*}, st).
6. Return d'.

Advantage of $\mathcal{A}$ in the game defined earlier is given as

$$Adv^{\mathsf{IND-GPRE-CPA}}_{\Gamma^{\star}_{\mathcal{G}}, \mathcal{A}}(1^{\lambda}) = \left| \begin{array}{c} \Pr[\mathbf{Exp}^{\mathrm{CPA}}_{\Gamma^{\star}_{\mathcal{G}}, \mathcal{A}}(1^{\lambda}) = 1 | d = 0] \\ \\ - \Pr[\mathbf{Exp}^{\mathrm{CPA}}_{\Gamma^{\star}_{\mathcal{G}}, \mathcal{A}}(1^{\lambda}) = 1 | d = 1] \end{array} \right|.$$

Alternatively

$$Adv^{\mathsf{IND-GPRE-CPA}}_{\Gamma^{\star}_{\mathcal{G}},\mathcal{A}}(1^{\lambda}) = |2\Pr[d' = d] - 1|.$$

The proposed GPRE scheme $\Gamma^{\star}_{\mathcal{G}}$ is said to be secure under IND-GPRE-CPA security if $Adv^{\mathsf{IND-GPRE-CPA}}_{\Gamma^{\star}_{\mathcal{G}},\mathcal{A}}(1^{\lambda}) \leq negl(\lambda)$.

4. Our concrete construction of GPRE scheme

In this section, we present our concrete GPRE scheme. For this, we extend the group encryption scheme by Koti and Purushothama [7]. Also, we prove security of our proposed GPRE under IND-GPRE-CPA.

4.1 *Nishat et al's group encryption scheme [7]*

Considering that each group G_A consists of n_A number of members, Nishat *et al*'s group encryption can be re-written as follows.

1. SetGroups: Outputs (params, MSK, MPK, $\forall u_i : sk_i$):
   - Computes the tuple params $= (p, \mathbb{G}_1, \mathbb{G}_2, g, h)$ where p is a sufficiently large prime, $\mathbb{G}_1, \mathbb{G}_2$ are the cyclic groups of order p and g, h are distinct generators of $\mathbb{G}_1$.
   - Computes $MSK = \{a_1, a_2\}$ where $(a_1, a_2) \xleftarrow{\$} (\mathbb{Z}^*_p)^2$.
   - Computes $MPK = \{g^{a_1}, g^{a_2}\}$.
   - For each u_i, computes $sk_i = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ as follows:
     - Select a random $r_i \xleftarrow{\$} \mathbb{Z}^*_p$.
     - Compute $s_1 = h^{r_i}$, $s_2 = g^{r_i}$, $s_3 = {s_1}^{{a_2}^{-1}} = h^{r_i a_2^{-1}}$, $s_4 = g^{a_1 r_i {a_2}^{-1}}$, $s_5 = h^{{a_2}^{-1}}$, $s_6 = g.h^{r_i}$ and $s_7 = r_i$.
2. KeyGen: Outputs group public key $PK_A = (PK_{A_1}, PK_{A_2})$:
   - For each group G_A, computes $PK_A = (PK_{A_1}, PK_{A_2})$:
     - Choose a group secret key $SK_A = K_A \xleftarrow{\$} \mathbb{Z}^*_p$ such that $K_A < r_i,\ \forall\ i \in \{1, \ldots, n_A\}$ and choose $a \xleftarrow{\$} \mathbb{Z}^*_p$.
     - Compute and publish $\psi_A = (a \times r_1 \times \cdots \times r_{n_A}) - K_A$.
     - Compute $PK_A = \{PK_{A_1} = g^{a_1 K_A}, PK_{A_2} = g^{a_2 K_A}\}$.
3. Enc: Outputs ciphertext $C_A = (c_1, c_2, c_3)$ for group G_A:
   - Choose $t \xleftarrow{\$} \mathbb{Z}_p$ and output the ciphertext $C_A = (c_1, c_2, c_3)$ of message M intended for the group G_A where $c_1 = e(g, PK_{A_1})^t M, c_2 = (h.PK_{A_1})^t$ and $c_3 = (PK_{A_2})^t$.
4. Dec: Decrypts C_A using secret key sk_i of a user $u_i \in G_A$:
   - Compute $K_A = r_i - \psi_A \bmod r_i$.
   - Compute $\alpha = s_1 (s_2)^{K_A}$ and $\beta = \dfrac{s_3 (s_4)^{{K_A}^{-1}}}{(s_5)^{{K_A}^{-1}}}$.
   - Compute $\tau = \dfrac{e(s_6, c_2).e(\beta, c_3)}{e(\alpha, c_2)}$ and $M = \dfrac{c_1}{\tau}$.

   Correctness
   - If $u_i \in G_A$ then
     $r_i - \psi_A \bmod r_i$
     $= r_i - (a \times r_1 \times \cdots \times r_i \times \cdots \times r_{n_A}) - K_A \bmod r_i$
     $= K_A$

     $$\alpha = h^{r_i} g^{r_i K_A}$$
     $$\beta = h^{a_2^{-1}(r_i - K_A^{-1})} g^{a_1 a_2^{-1} r_i K_A^{-1}}$$
     $$\tau = \frac{e(s_6, c_2).e(\beta, c_3)}{e(\alpha, c_2)} = e(g, PK_{A_1})^t.$$

     Finally, $\dfrac{c_1}{\tau} = \dfrac{Me(g, PK_{A_1})^t}{e(g, PK_{A_1})^t} = M.$
   - Otherwise, if $u_i \notin G_A$ then
     $r_i - \psi_A \bmod r_i$
     $= r_i - (a \times \prod_{j \neq i} r_j) - K_A \bmod r_i$
     $\neq K_A$.
     Hence, decryption is not correct if $u_i \notin G_A$.
5. GroupJoin: Consider that a new user $u_{(n_A+1)}$ joins G_A:
   - Choose $(K'_A, a') \xleftarrow{\$} (\mathbb{Z}^*_p)^2$ such that $K'_A < r_i,\ \forall\ i \in \{1, \ldots, (n_A + 1)\}$.
   - Compute $PK'_A = (PK'_{A_1} = g^{a_1 K'_A}, PK'_{A_2} = g^{a_2 K'_A})$.
   - Publish $\psi'_a = a' \times r_1 \times r_2 \times \cdots \times r_{(n_A+1)} - K'_A$.

   All future encryptions for G_A are encrypted under updated group public key PK'_A.
   Backward secrecy
   Since $u_{n_A+1} \in G_A$,
   $r_{n_A+1} - \psi'_A \bmod r_{n_A+1}$
   $= r_{n_A+1} - (a' \times r_1 \times r_2 \times \cdots \times r_{(n_A+1)} - K'_A) \bmod r_{n_A+1}$
   $= K'_A$.
   Hence, u_{n_A+1} can obtain the correct updated secret K'_A. Therefore, it can decrypt all future encryptions under PK'_A.
6. GroupLeave: Consider that user u_{n_A} leaves G_A:
   - Choose $(K'_A, a') \xleftarrow{\$} (\mathbb{Z}^*_p)^2$ such that $K'_A < r_i,\ \forall\ i \in \{1, \ldots, (n_A - 1)\}$.
   - Compute $PK'_A = (PK'_{A_1} = g^{a_1 K'_A}, PK'_{A_2} = g^{a_2 K'_A})$.
   - Publish $\psi'_a = a' \times r_1 \times r_2 \times \cdots \times r_{(n_A-1)} - K'_A$.

   All future encryptions for G_A are encrypted under updated group public key PK'_A.
   Forward secrecy
   Since $u_{n_A} \notin G_A$,
   $r_{n_A} - \psi'_A \bmod r_{n_A}$
   $= r_{n_A} - (a' \times \prod_{j \neq n_A} r_j) - K'_A \bmod r_{n_A}$
   $\neq K'_A$.

Hence, u_{n_A} cannot obtain the correct updated secret K'_A. Therefore, it cannot decrypt the future encryptions under PK'_A.

4.2 *Our re-encryption function*

To design our re-encryption function, we consider that component h of the *MPK* initialized in the *SetGroup* procedure is defined as $h = g^x$ for some master-secret component $x \in \mathbb{Z}_p^*$. Hence, for our re-encryption function, $MSK = MSK \cup \{x\}$.

1. ReKeyGen: For a pair of groups G_A and G_B, outputs re-encryption key $RK_{A\to B} = (rk^{(1)}_{A\to B}, rk^{(2)}_{A\to B}, rk^{(1)}_{A\to B})$ where

$$rk^{(1)}_{A\to B} = K_A^{-1}K_B$$
$$rk^{(2)}_{A\to B} = (x + a_1K_A)^{-1}(x + a_1K_B)$$
$$rk^{(3)}_{A\to B} = g^{(a_2K_A)^{-1}a_1(K_B-K_A)}.$$

2. ReEnc: Outputs re-encrypted ciphertext $C_B = (\overline{c_1}, \overline{c_2}, \overline{c_3})$ where

$$\begin{aligned}\overline{c_1} &= c_1 e(c_3, rk^{(3)}_{A\to B})\\ &= e(g,g)^{a_1K_At} Me(g^{a_2K_At}, g^{(a_2K_A)^{-1}a_1(K_B-K_A)})\\ &= Me(g, g^{a_1K_B})^t = Me(g, PK_{B_1})^t\\ \overline{c_2} &= (c_2)^{rk^{(2)}_{A\to B}}\\ &= (g^x g^{a_1K_A})^{(x+a_1K_A)^{-1}(x+a_1K_B)} = (hg^{a_1K_B})^t = (hPK_{B_1})^t\\ \overline{c_3} &= (c_3)^{rk^{(1)}_{A\to B}} = (g^{a_2tK_B})^{K_A^{-1}K_B} = (g)^{a_2K_Bt} = (PK_{B_2})^t.\end{aligned}$$

C_B obtained earlier is of the same form as the output of Enc (PK_B, M). Hence, it can be decrypted using Dec (sk_i, C_B) where u_i is a legitimate member of group G_B.

4.3 *Security of the proposed GPRE scheme*

We establish IND-GPRE-CPA security of the proposed GPRE scheme in the following theorem.

Theorem 1. *The proposed GPRE scheme is secure under IND-GPRE-CPA of Definition 4 given that V-Decisional Diffie–Hellman (V-DDH) assumption [7] is intractable, that is, for* $g \in \mathbb{G}_1, x, y, z, \in \mathbb{Z}_p$ *and* $Q \xleftarrow{\$} \mathbb{G}_1$, *the distributions* $(g, g^x, g^{xy}, g^{xz}, g^{yz})$ and $(g, g^x, g^{xy}, g^{xz}, Q)$ *are computationally indistinguishable for a PPT algorithm.*

Proof. Let $\mathcal{A}$ be a PPT adversary that has non-negligible advantage ϵ in attacking the proposed GPRE scheme in the sense of definition IND-GPRE-CPA. We construct another PPT adversary $\mathcal{B}$ using $\mathcal{A}$ to break the V-DDH assumption also with non-negligible probability.

On V-DDH input $(g, g^a, g^{as}, g^{ak}, T) \in \mathbb{G}_1^6$, $\mathcal{B}$ sets up the GPRE world. If $T = g^{ks}$, $\mathcal{B}$ outputs $\mu = 0$; otherwise, $\mathcal{B}$ outputs $\mu = 1$. The interaction between $\mathcal{A}$ and $\mathcal{B}$ is described as follows.

Set-up phase: $\mathcal{B}$ randomly selects $w, x, t \xleftarrow{\$} \mathbb{Z}_p$ and sets $g_1 = g^t$, $g_2 = g^w$, $g_3 = g_1^w$ and $h = g_1^x$.

Query phase I: Adversary $\mathcal{A}$ issues queries to $\mathcal{B}$ as follows.

- $\mathcal{O}_{k_h}$ *queries*: For each queried group G_A, $\mathcal{B}$ responds with the group public key $PK_A = (PK_{A_1} = g_1^{aK_A}, PK_{A_2} = g_3^{aK_A})$ and stores aK_A as the group G_A's secret key. $\mathcal{B}$ makes the entry $(c_A = 0, PK_A, SK_A)$ to K^{list}.
- $\mathcal{O}_{k_c}$ *queries*: For a corrupted group G_A, $\mathcal{B}$ outputs $SK_A = aK_A$ and $PK_A = (PK_{A_1} = g^{aK_A}, g_2^{aK_A})$. $\mathcal{B}$ gives both PK_A and SK_A to $\mathcal{A}$ and stores $(c_A = 1, PK_A, SK_A)$ to K^{list}.
- $\mathcal{O}_{rk}$ *queries*: For pair G_A, G_B, $\mathcal{B}$ outputs $RK_{A\to B}$ as follows.
  - If $(c_A = 1 \wedge c_B = 1) \vee (c_A = 0 \wedge c_B = 0)$, output:

$$(aK_B)(aK_A)^{-1} = K_BK_A^{-1} = RK^{(1)}_{A\to B} \tag{3}$$

$$(x + aK_A)^{-1}(x + aK_B) = RK^{(2)}_{A\to B} \tag{4}$$

$$(g_2)^{(aK_A)^{-1}(aK_B-aK_A)} = g^{(waK_A)^{-1}a(K_B-K_A)} = RK^{(3)}_{A\to B}. \tag{5}$$

    It is trivial to verify that re-encryption key components computed using information in $\mathcal{B}$'s view are indeed valid.
  - Otherwise, output $\perp$.
- $\mathcal{O}_r$ *queries*: Since all re-encryption keys generated in $\mathcal{O}_{rk}$ phase are valid, $\mathcal{B}$ replies to $\mathcal{O}_r$ queries with the output of ReEnc $(rk_{A\to B}, M)$ for any query plaintext message M.

□

Challenge phase: When $\mathcal{A}$ wishes to end query phase I, it selects two equal-length messages M_0, M_1 and sends them to $\mathcal{B}$. $\mathcal{B}$ chooses a random bit $d \xleftarrow{\$} \{0,1\}$ and generates the challenge ciphertext $C_{A^\star}$ as follows:

$$\begin{aligned}c_1^\star &= M_d e(g,T)\\ &= e(g, g^{K_{A^\star}s})M_d = e(g, g_1^{aK_{A^\star}})^{s/at}M_d = e(g, PK_{A_1^\star})^r M_d\\ c_2^\star &= (T)^{(1+\frac{x}{aK_{A^\star}})}\\ &= (g^{sK_{A^\star}})^{(1+\frac{x}{aK_{A^\star}})} = (g^{sK_{A^*}})(g^{sx/a}) = (g_1^{sK_{A^*}/t})(g_1^{sx/at})\\ &= (g_1^x)^{s/at}(g_1^{aK_{A^*}})^{s/at} = (hg_1^{aK_{A^\star}})^{s/at} = (hPK_{A_1^\star})^r\\ c_3^\star &= (T)^w\\ &= (g^{sK_{A^\star}})^w = (g_1^{aK_{A^\star}w})^{s/at} = (PK_{A_2^\star})^r.\end{aligned}$$

$C_{A^\star} = (c_1^\star, c_2^\star, c_3^\star)$ is given to $\mathcal{A}$.

Query phase II: $\mathcal{A}$ continues to issue all oracle queries as in query phase I and subject to the same constraints.

Guess: $\mathcal{A}$ submits a guess d' for d.

This completes description of the game. If $d' = d$, $\mathcal{B}$ will output $\mu' = 0$, indicating that the given tuple is a valid V-DDH tuple. Otherwise, $\mathcal{B}$ outputs $\mu' = 1$, indicating that the given tuple is a random one.

Suppose there exists a PPT algorithm that can solve V-DDH with advantage ϵ. The total advantage of $\mathcal{B}$ is given by $\frac{1}{2}Pr[\mu' = \mu|\mu = 1] + \frac{1}{2}Pr[\mu' = \mu|\mu = 0] - \frac{1}{2} = \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$. However, we know from [7] that V-DDH is intractable. Hence, we conclude that there does not exist an adversary that can break the scheme with non-negligible probability.

Table 1. Generic construction of E-HKAS using GPRE.

E-HKAS	GPRE procedures
	- SetGroups $(1^\lambda) \rightarrow \text{params}, MSK, MPK, \{sk_i\}$ $\forall SC_A \in S$ do:
Initialize	- KeyGen $(\text{params}, A) \rightarrow \{PK_A, SK_A\}$ - Enc$(PK_A, ek_A) \rightarrow C_A$
Derive	Each $SC_i \in G_A$ does the following: - Dec $(sk_i, C_A) \rightarrow ek_A$
EdgeDelete	To delete an edge (SC_A, SC_l), $SC_A \preceq SC_l$: - Execute: GroupLeave(A, l) - For each $SC_i \prec SC_A$: GroupLeave(i, l)
EdgeNew	To add a new edge (SC_A, SC_d), $SC_A \preceq SC_d$: - Execute: GroupJoin(A, d) - For each $SC_i \prec SC_A$: GroupJoin(i, l)
ExcepTrans	To enforce transitive exception in $SC_i \prec SC_j \prec SC_k$, do: - GroupLeave($i$, k)
UnExcepTrans	To revoke transitive exception in $\{SC_i, SC_j, SC_k\}$, do: - GroupJoin(i, k)
ExcepAS	To enforce anti-symmetric exception in $SC_i \prec SC_j$ so that $SC_j \prec SC_i$, do: - GroupJoin(j, i)
UnExcepAS	To enforce anti-symmetric exception in $\{SC_i, SC_j\}$ so that $SC_j \not\prec SC_i$ do: - GroupLeave(j, i)
Delegate	If SC_A wishes to delegate with descendant(s) safety to SC_B, do: - ReKeyGen $(SK_A, SK_B) \rightarrow RK_{A\rightarrow B}$ - ReEnc $(RK_{A\rightarrow B}, C_A) \rightarrow C_B$ - Dec $(sk_i, C_B) \rightarrow ek_A$, where $SC_i \in G_B$
UnDelegate	If G_A wishes to revoke delegation to G_B: - Update ek_A to ek'_A and SK_A to SK'_A - Publish PK'_A and $C'_A \leftarrow$ Enc(PK'_A, ek'_A)

5. Concrete E-HKAS using the proposed GPRE scheme

As discussed in section 1.2, enforcing a hierarchical access control policy using group encryption leads to efficient key derivation and dynamic update operations. GPRE, a novel primitive extended from group encryption, efficiently enforces all explicit policy exceptions in the hierarchy. Table 1 summarizes a generic construction of the proposed E-HKAS using procedures of our proposed GPRE scheme. In this section, we first present our concrete construction of the proposed E-HKAS by employing procedures of the proposed GPRE scheme. Later, we discuss how the proposed E-HKAS enforces explicit policy exceptions in some parts (specified by the data owner) of hierarchy while preserving the hierarchical structure in rest of the hierarchy. Finally, we analyse performance of the proposed E-HKAS and compare its efficiency to those of existing HKAS solutions both with and without explicit policy exceptions.

5.1 *Concrete construction*

In this section, we use procedures of the concrete construction of the proposed GPRE scheme $\Gamma^{\star}_{\mathcal{G}}$ to construct our proposed E-HKAS $\Pi^{\star}_{(S,\preceq)}$.

5.1a *Initialize*$(1^\lambda) \rightarrow$ par, $\{\forall SC_i \in S, (pk_i, sk_i)\}$: For a set of security classes S and the given hierarchical structure, this procedure is executed as follows:

Step-1 Execute SetGroups (1^λ) of $\Gamma^{\star}_{\mathcal{G}}$, where λ is the security parameter input to the procedure Initialize, to output

- **(a)** {params, MSK, MPK} as described in procedure SetGroups of section 4.1.
- **(b)** $|S|$ distinct constant-length secret keys SK_1, SK_2, …, $SK_{|S|}$ as described in procedure SetGroups of section 4.1 and $\forall SC_i \in S$, initialize class secret key with the group member's secret key, i.e., $sk_i = SK_i$.

Step-2 For $i \in \{1, 2, \ldots, |S|\}$, do the following:

- **(a)** Form an access group G_i corresponding to SC_i where $G_i = \{SC_i\} \cup \{SC_j, \text{such that} SC_i \preceq SC_j\}$.
- **(b)** For access group G_i, execute KeyGen (params, i) to output group public key PK_i and group public parameter ψ_i as described in procedure KeyGen of section 4.1.

Step-3 For $i \in \{1, 2, \ldots, |S|\}$, secretly compute $|S|$ distinct class encryption keys ek_i by uniform and random selection from $\mathbb{G}_2$, i.e., $ek_i \xleftarrow{\$} \mathbb{G}_2$.

Step-4 Execute Enc (PK_i, ek_i) to output C_i, ciphertext corresponding to ek_i and under group public key PK_i as described in procedure Enc of section 4.1.

Finally, output $\mathsf{par} = \{\mathsf{params}\} \cup \{MPK\}$ and $\forall SC_i \in S$: (pk_i, sk_i) where $pk_i = \{PK_i, \psi_i, C_i\}$ and $sk_i = SK_i$. Here, par is stored on the global public storage, collection pk_i is stored as the public bulletin entry corresponding to SC_i and sk_i is delivered to SC_i via a secure communication channel.

5.1b *Derive*$(sk_i, SC_j) \rightarrow ek_j$: Any user in class SC_i can derive the valid class encryption key ek_j of SC_j if and only if $SC_j \preceq SC_i$ as follows:

Step-1 Download $pk_j = \{PK_j, \psi_j, C_j\}$ from the public bulletin and extract $C_j = \mathsf{Enc}(PK_j, ek_j)$ from it.

Step-2 Execute Dec (sk_i, C_j) to obtain underlying plaintext corresponding to C_j, that is, ek_j.

Correctness

Because $SC_j \preceq SC_i$, the relation $SC_i \in G_j$ also holds. Therefore, Dec (sk_i, C_j) translates to decryption of ciphertext intended for group G_j using secret key of one of its members. Hence, by correctness property of the proposed GPRE scheme

$$\begin{aligned}\mathsf{Dec}(sk_i, C_j) &= \mathsf{Dec}(sk_i, \mathsf{Enc}(PK_j, ek_j)) \\ &= ek_j.\end{aligned}$$

In other words, correctness of Derive follows from correctness of Dec of group encryption discussed in section 4.1. Whereas if $SC_j \not\preceq SC_i$, which means $SC_i \notin G_j$, IND-GPRE-CPA security proved in Theorem 1 means that output of the procedure Dec (sk_i, C_j) is independent of class encryption key ek_j encrypted under group public key PK_j of G_j.

5.1c *EdgeDelete* (SC_A, SC_l): To delete an edge (SC_A, SC_l) where $SC_A \preceq SC_l$ (which means $SC_l \in G_A$), do the following.

Step-1 For each $SC_i \in \{SC_A\} \cup \{SC_j : SC_j \prec SC_A\}$ (that is, SC_A and all its descendants), do the following:

- **(a)** Execute GroupLeave(i, l) as described in section 4.1 to remove SC_l from G_i and output updated group public parameters PK_i' and ψ_i' to preserve forward secrecy as described in GroupLeave.
- **(b)** Select an updated class encryption key $ek_i' \xleftarrow{\$} \mathbb{G}_2$ for SC_i and compute the corresponding ciphertext $C_i' \leftarrow \mathsf{Enc}(PK_i', ek_i')$ as described in section 4.1.
- **(c)** Update the public bulletin entry pk_i corresponding to SC_i as $pk_i' = \{PK_i', \psi_i', C_i'\}$.

5.1d *EdgeNew* (SC_A, SC_l): To insert a new edge (SC_A, SC_l) where $SC_A \preceq SC_l$ (which means $SC_l \in G_A$), do the following.

Step-1 For each $SC_i \in SC_A \cup \{SC_j : SC_j \prec SC_A\}$ (that is, SC_A and all its descendants), do the following:

- **(a)** Execute GroupJoin(i, l) as described in section 4.1 to add SC_l to group G_i and output updated group public parameters PK_i' and ψ_i' to preserve backward secrecy as described in GroupJoin.
- **(b)** Select an updated class encryption key $ek_i' \xleftarrow{\$} \mathbb{G}_2$ for SC_i and compute the corresponding ciphertext $C_i' \leftarrow \mathsf{Enc}(PK_i', ek_i')$ as described in section 4.1.
- **(c)** Update the public bulletin entry pk_i corresponding to SC_i as $pk_i' = \{PK_i', \psi_i', C_i'\}$.

5.1e *ExcepTrans* (SC_i, SC_j, SC_k): Before executing this procedure, $SC_i \prec SC_j$, $SC_j \prec SC_k$ and $SC_i \prec SC_k$ means that $SC_j \in G_i$, $SC_k \in G_j$ and $SC_k \in G_i$. To enforce explicit transitive exception such that $SC_i \prec SC_j$, $SC_j \prec SC_k$ but $SC_i \not\prec SC_k$, do the following:

Step-1 Execute GroupLeave(i, k) as described in section 4.1 to remove SC_k from group G_i and output updated group public parameters PK_i' and ψ_i' to preserve forward secrecy as described in GroupLeave.

Step-2 Select an updated class encryption key $ek_i' \xleftarrow{\$} \mathbb{G}_2$ for SC_i and compute the corresponding ciphertext $C_i' \leftarrow \mathsf{Enc}(PK_i', ek_i')$ as described in section 4.1.

Step-3 Update the public bulletin entry pk_i corresponding to SC_i as $pk_i' = \{PK_i', \psi_i', C_i'\}$.

Security class SC_k continues to remain member of access group G_j and SC_j remains member of G_i. Hence the relations $SC_j \prec SC_k$ and $SC_i \prec SC_j$ continues to hold.

5.1f *UnExcepTrans* (SC_i, SC_j, SC_k): With security classes SC_i, SC_j, SC_k where $SC_i \prec SC_j$, $SC_j \prec SC_k$ but $SC_i \not\prec SC_k$ as input, this procedure revokes the transitive exception, so that $SC_i \prec SC_j$ $SC_j \prec SC_k$ and $SC_i \prec SC_k$, using the following steps:

Step-1 Execute GroupJoin(i, k) as described in section 4.1 to add SC_k in group G_i and output updated group public parameters PK_i' and ψ_i' to preserve backward secrecy as described in GroupLeave.

Step-2 Select an updated class encryption key $ek_i' \xleftarrow{\$} \mathbb{G}_2$ for SC_i and compute the corresponding ciphertext $C_i' \leftarrow \mathsf{Enc}(PK_i', ek_i')$ as described in section 4.1.

Step-3 Update the public bulletin entry pk_i corresponding to SC_i as $pk_i' = \{PK_i', \psi_i', C_i'\}$.

After these steps, SC_k becomes a member of G_i. Hence, the relations $SC_i \prec SC_k$ holds.

5.1g *ExcepAS* (SC_i, SC_j): If $SC_i \prec SC_j$, it implies that $SC_j \in G_i$. However, due to anti-symmetric requirement of the hierarchy, $SC_j \not\prec SC_i$ or $SC_i \notin G_j$. To enforce explicit anti-symmetric exception so that $SC_i \prec SC_j$ and $SC_j \prec SC_i$ for $SC_i \neq SC_j$, do:

Step-1 Execute GroupJoin(j, i) as described in section 4.1 to add SC_i in group G_j and output updated group public parameters PK_j' and ψ_j' to preserve backward secrecy as described in GroupLeave.

Step-2 Select an updated class encryption key $ek_j' \xleftarrow{\$} \mathbb{G}_2$ for SC_j and compute the corresponding ciphertext $C_j' \leftarrow \mathsf{Enc}(PK_j', ek_j')$ as described in section 4.1.

Step-3 Update the public bulletin entry pk_j corresponding to SC_j as $pk_j' = \{PK_j', \psi_j', C_j'\}$.

After these steps, SC_i can successfully decrypt encryptions for the group G_j (or SC_j). Therefore, $SC_j \prec SC_i$ holds.

5.1h *UnExcepAS* (SC_i, SC_j): If both $SC_i \prec SC_j$ and $SC_j \prec SC_i$ hold (anti-symmetric exception), it implies that $SC_i \in G_j$ and $SC_j \in G_i$. To revoke the anti-symmetric exception, so that $SC_j \not\prec SC_i$ in the updated hierarchy, do the following:

Step-1 Execute GroupLeave(j, i) as described in section 4.1 to remove SC_i from group G_j and output updated group public parameters PK_j' and ψ_j' to preserve forward secrecy as described in GroupLeave.

Step-2 Select an updated class encryption key $ek_j' \xleftarrow{\$} \mathbb{G}_2$ for SC_j and compute the corresponding ciphertext $C_j' \leftarrow \mathsf{Enc}(PK_j', ek_j')$ as described in section 4.1.

Step-3 Update the public bulletin entry pk_j corresponding to SC_j as $pk_j' = \{PK_j', \psi_j', C_j'\}$.

After these steps, SC_i cannot decrypt encryptions for the group G_j (or SC_j). Therefore, $SC_j \not\prec SC_i$ holds.

5.1i *Delegate* (SC_i, SC_j): To delegate decryption rights of SC_i to SC_j subject to the condition that despite being able to access SC_i, SC_j cannot access any SC_l where $SC_l \prec SC_i$ (SC_i's descendant(s)), this procedure works as follows.

Step-1 Execute ReKeyGen (sk_i, sk_j), as described in section 4.2, with class secret keys of SC_i and SC_j as input to output re-encryption key $RK_{i \to j}$ and store it on the public bulletin.

Step-2 To delegate from SC_i to SC_j with descendant(s) safety:

- **(a)** Extract C_i from pk_i stored on the public bulletin.
- **(b)** Execute ReEnc $(RK_{i \to j}, C_i)$, described in section 4.2, to output the re-encrypted ciphertext C_j.

Step-3 A user in class SC_j executes Dec(sk_j, C_j), described in section 4.1, to obtain the underlying plaintext corresponding to C_j, which is ek_i.

As a result of the steps given earlier, class encryption key ek_i of SC_i is accessible to users in SC_j (class delegation). However, encryption key(s) of none of the descendants of SC_i can be accessed by users in SC_j (descendant(s) safety).

5.1j *UnDelegate* (SC_i, SC_j): To revoke class delegation with descendant(s) safety enforced using Delegate, this procedure executes the following steps:

Step-1 Select a new group secret key SK_i' for access group G_i (or security class SC_i) through random selection from $\mathbb{Z}_p^*$, that is, $SK_i' \xleftarrow{\$} \mathbb{Z}_p^*$. Transmit SK_i' securely to SC_i, which updates the class secret key $sk_i' = SK_i'$.

Step-2 Compute the corresponding updated group public key PK_i' and group public parameter ψ_i' for group G_i using procedure KeyGen as described in section 4.1.

Step-3 Select an updated class encryption key $ek_i' \xleftarrow{\$} \mathbb{G}_2$ for SC_i and compute the corresponding ciphertext $C_i' \leftarrow \mathsf{Enc}(PK_i', ek_i')$ as described in section 4.1.

Step-4 Update the class public parameters $pk_i' = \{PK_i', \psi_i', C_i'\}$ corresponding to SC_i and store the updated value pk_i' on the public bulletin.

At the end of this procedure, the re-encryption/delegation key $RK_{i \to j}$ gets invalidated, since the public key pk_i of the delegator class SC_i is updated without any updates to the re-encryption key. As a result, anything encrypted under pk_i' cannot be re-encrypted using the same old $RK_{i \to j}$.

5.2 *Discussion*

We consider an example and discuss system operation (key assignment and key derivation with flexibility) of the proposed E-HKAS. Through this example, we show how hierarchical structure is preserved using the proposed strategy. Also, we informally discuss various security aspects of the proposed E-HKAS.

5.2a *Key assignment and key derivation*: For analysing correctness of key derivation, we show with an example that in the proposed E-HKAS, a security class SC_i can

access an encryption key ek_j if and only if $SC_j \preceq SC_i$. For this, we show that the following three requirements ($\mathbf{R_1}$, $\mathbf{R_2}$ and $\mathbf{R_3}$) are satisfied.

$\mathbf{R_1}$: Each $SC_i \in S$ can access the encryption key ek_i
$\mathbf{R_2}$: If $SC_j \prec SC_i$ then SC_i can access ek_j and
$\mathbf{R_3}$: If $SC_j \not\preceq SC_i$, then SC_i cannot access ek_j.

Consider an access hierarchy with $S = \{SC_1, SC_2, SC_3, SC_4\}$ whose Hasse diagram (POSET representation) and graph representation with descriptive edges are given in figure 3(a) and (b), respectively. Each directional edge (SC_i, SC_j) in figure 3(b), represented by an arrow from SC_i to SC_j, implies that $SC_j \prec SC_i$ (or SC_i can access SC_j), whereas, in figure 3(a), an edge (SC_i, SC_j) implies that SC_j can access SC_i and all its descendants. The proposed E-HKAS models the access control policy $\mathcal{P}$ in the hierarchy of figure 3 as a collection $\mathcal{P} = \{G_1, G_2, G_3, G_4\}$, where each G_i for $i \in \{1, 2, 3, 4\}$ is formed using the relation in Eq. (2), that is

$$G_1 = \{SC_1\}, G_2 = \{SC_1, SC_2\}, G_3 = \{SC_1, SC_2, SC_3\},$$
$$G_4 = \{SC_1, SC_2, SC_4\}.$$

The group public information $\{pk_1, pk_2, pk_3, pk_4\}$ corresponding to each group is stored on the public bulletin as shown in figure 3(c). Each class SC_i can access a message encrypted under PK_j if $SC_i \in G_j$.

Firstly, it can be seen that $\forall i \in \{1, 2, 3, 4\}$, SC_i is a member of G_i. Therefore, by correctness requirement of GPRE analysed in section 4.1, $\mathsf{Dec}(sk_i, \mathsf{Enc}(PK_i, ek_i)) = ek_i$. Hence, SC_i can access ek_i, which means requirement $\mathbf{R_1}$ is satisfied.

Suppose SC_1 wishes to access ek_3. Since $SC_1 \in G_3$, by the correctness requirement of the proposed GPRE scheme, $\mathsf{Dec}(sk_1, \mathsf{Enc}(PK_3, ek_3)) = ek_3$ holds. Hence $\mathbf{R_2}$ is satisfied.

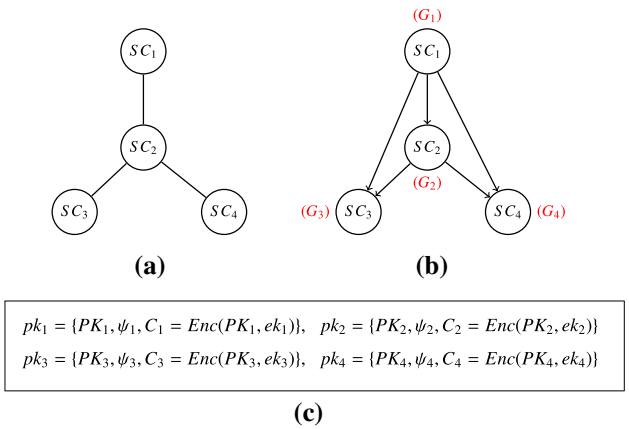


Figure 3. (a) Hasse diagram of the initial access hierarchy, (b) initial hierarchy with descriptive edges and (c) public bulletin after initial key assignment.

Finally, consider that SC_3 wishes to access ek_2. Since $SC_3 \notin G_2$, by correctness requirement of the proposed GPRE scheme, $\mathsf{Dec}(sk_3, \mathsf{Enc}(PK_2, ek_2)) = \perp$. This is because a non-member cannot decrypt the encryptions intended for an access group. Hence requirement $\mathbf{R_3}$ is satisfied.

Since $\mathbf{R_1}$, $\mathbf{R_2}$ and $\mathbf{R_3}$ hold true, the proposed E-HKAS satisfies the correctness requirement and preserves the hierarchical structure of the access control model.

5.2b *Security of class encryption key*: Atallah *et al* [1] proposed two early security notions, called "security against KR" and "KI" for hierarchical key assignment. The notion of KI, which is considered more practical than KR, was further refined by Freire *et al* [24] into strong key indistniguishability (SKI). However later, Castiglione *et al* [25] formally established that the notion of SKI is not stronger than KI. We informally discuss how the proposed E-HKAS satisfies KI if the underlying GPRE scheme is IND-GPRE-CPA secure.

KI captures inability of a probabilistic polynomial time (PPT) adversary to distinguish encryption key ek_{i^*} of a given target security class SC_{i^*} (decided by the adversary) from a random bit-string ρ such that $|ek_{i^*}| = |\rho|$. As part of the state and corruption information to be queried prior to deciding SC_{i^*}, the adversary can access all public information and class secret keys of all descendant security classes of SC_{i^*}.

By construction, encryption key ek_i corresponding to any class SC_i is encrypted for G_i using Enc of the proposed GPRE scheme. Also, corrupting any security class SC_i that is a descendant of SC_i is equivalent to corrupting non-members of the group G_i. Hence, distinguishing ek_i from a random bit-string of same length is reduced to distinguishing the output of $\mathsf{Enc}(PK_i, ek_i)$. However, due to IND-GPRE-CPA security of the proposed GPRE scheme established formally in Theorem 1, distinguishing encryptions corresponding to the two messages m_0, m_1 of same length is computationally infeasible for a PPT adversary. If $m_0 = ek_{i^*}$ and $m_1 = \rho$, IND-GPRE-CPA security directly implies KI security of the proposed E-HKAS. Therefore, the proposed E-HKAS is secure under KI given that the underlying GPRE scheme is secure under IND-GPRE-CPA security notion given in Definition 4. Since IND-GPRE-CPA security of the proposed GPRE scheme $\Gamma^{\star}_{\mathcal{G}}$ has been formally proved in Theorem 1, we can say that the proposed E-HKAS $\Pi^{\star}_{(S,\preceq)}$ is secure under KI.

5.2c *Security of dynamic updates*: As we have seen in the concrete construction of the proposed E-HKAS given in section 5.1, enforcing dynamic updates $\mathsf{EdgeNew}$ and $\mathsf{EdgeDelete}$ involves execution(s) of $\mathsf{GroupJoin}$ and $\mathsf{GroupLeave}$, respectively. It has been shown in section 4.1 that the procedures $\mathsf{GroupJoin}$ and $\mathsf{GroupLeave}$ of the proposed GPRE scheme satisfy backward and forward secrecy requirements, respectively.

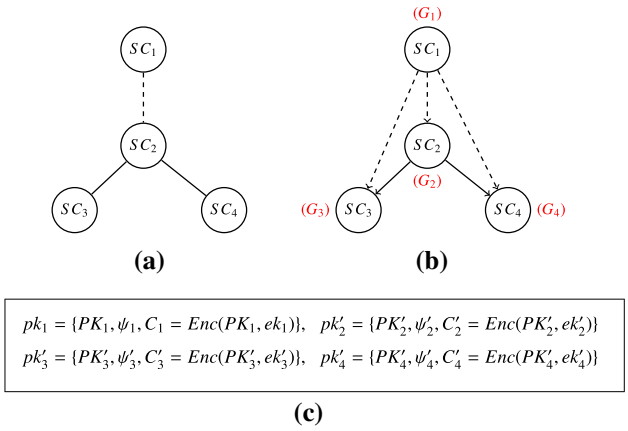


Figure 4. (a) Hasse diagram of hierarchy with edge (SC_2, SC_1) deleted, (b) corresponding updated hierarchy with directed edges $(SC_1, SC_2), (SC_1, SC_3)$ and (SC_1, SC_4) deleted and (c) public bulletin after deleting (SC_2, SC_1).

As an example, consider that in the hierarchy of figure 4, edge (SC_2, SC_1) where $SC_2 \prec SC_1$ has to be deleted. In the updated hierarchy, SC_1 should not have designated access to encryption keys corresponding to SC_2, SC_3 and SC_4. Hence, class SC_1, which was initially a simultaneous member of G_2, G_3 and G_4, has to be "securely" removed from these three groups. For this, the data owner individually updates the public parameters corresponding to these groups to (PK'_2, ψ'_2), (PK'_3, ψ'_3) and (PK'_4, ψ'_4) as described in procedure GroupLeave of section 4.1. Data owners also choose new unique and random encryption keys ek'_2, ek'_3, ek'_4 and encrypt them under PK'_2, PK'_3 and PK'_4 to output C'_2, C'_3, C'_4, respectively, using procedure Enc described in section 4.1. Now, data owners assign updated public parameters $pk'_i = \{PK'_i, \psi'_i, C'_i\}$ for each SC_i such that $i \in \{2, 3, 4\}$. Due to forward secrecy property of procedure GroupLeave, SC_1 cannot correctly decrypt any C'_i to obtain the underlying updated encryption key ek'_i for $i \in \{2, 3, 4\}$. An example of adding back the edge (SC_2, SC_1) into the hierarchy has the same procedure except that it executes GroupJoin instead of GroupLeave of the proposed GPRE scheme. Backward secrecy of GroupJoin implies that SC_1 is unable to access encryptions intended for $\{SC_2, SC_3, SC_4\}$ prior to invoking the procedure EdgeNew (SC_2, SC_1).

5.3 *Performance analysis and comparison*

In this section, we discuss applicability of the proposed E-HKAS through both theoretical and practical analyses. We compare performance of various procedures of the proposed E-HKAS to those of traditional HKAS solutions without explicit policy exceptions in table 2. The comparison shows that while the proposed E-HKAS features a constant-time key derivation procedure, it also satisfies all other desirable efficiency requirements defined for traditional HKAS. Thus, the proposed E-HKAS requires constant number of public-key operations for key derivation, and linear overall public storage with each class having to store a secret key of constant length. We stress that none of the schemes compared in table 2 enforces any explicit policy exception.

Table 3 presents comparison of existing flexible HKASs [4–6] with ours with respect to general characteristics. We omit the comparison of the scheme of Yeh *et al* [2] as it has been proved to be insecure by Hwang [23]. We observe that there is trade-off between cost of key derivation and cost of enforcing explicit policy exceptions. Ours is the only one

Table 2. Comparison of the proposed E-HKAS with existing traditional HKAS solutions.

Scheme	# Private keys per class	# Public components	Key derivation cost	Supports dynamics?†	Explicit exceptions?
Atallah *et al* [1]	1	$n_e + \mu n$	$d(t_{Dec-DES} + t_{PRG})$	✔	✗
Santis *et al* [26]	1	$(n_e + 2n)\mu$	$(d+2)t_{Dec-DES}$	✔	✗
Freire *et al* [24]	1	2	$(d+1)t_{PRG}$	✗	✗
Chen *et al* [19]	1	$n_e + 2n$	$dt_{mp} + t_m + t_e$	✔	✗
Tang *et al* [20]	4	$n^2 + 1$	$2t_{PRG} + (4d+2)t_m + (2d+1)t_a$	✔	✗
Pareek and Purushothama [21]	1	$4n + n_e$	$(d+2)t_e + t_m$	✔	✗
Chen and Tzeng [22]	1	$5n + n^2$	$2t_m + t_e + 2t_b$	✔	✗
E-HKAS	7	$7n$	$3t_m + 2t_b$	✔	✔

d represents maximum of all the distances between two classes (depth),

n denotes the number of classes and n_e the number of edges,

μ denotes the encrypted block size, output of the DES algorithm,

$t_{Dec-DES}$ is the computation cost required to decrypt a DES encrypted block of message,

t_{PRG} is the computation cost required to compute a Pseudo-Random Number Generator,

$t_a, t_m, t_e, t_b, t_{mp}$: time-costs for one modulo addition, multiplication, exponentiation, bilinear pairing and multilinear pairing, respectively,

† does not require updating the whole hierarchy.

Table 3. General comparison of proposed E-HKAS with the existing HKAS supporting policy exceptions.

Characteristics	Lin *et al* [4]	Chang and Chang [5]	Chang [6]	E-HKAS
Constant key derivation cost	✗	✔	✔	✔
Linear public storage	✔	✗	✗	✔
Constant ExcepTrans cost	✔	✗	✗	✔
Constant ExcepAS cost	✔	✗	✗	✔
Support for class delegation	✗	✗	✗	✔
Localized effect of updates	✔	✗	✗	✔
Secure under assumption	RSA	C. R. Hash	C. R. Hash	V-DDH

C. R. Hash stands for collision-resistant hash function.

V-DDH stands for the V-Decisional Diffie–Hellman assumption.

Table 4. Comparison of computation efficiency.

Scheme	Derivation cost	Edge delete/add	Anti-symmetric exception	Transitive exception	Class delegation
Lin *et al* [4]	$dt_e + t_m$	$O(n)(t_e + t_m)$	$O(n_a)(t_e + t_m)$	$O(n_a)(t_e + t_m)$	NA
Chang and Chang [5]	$t_H + t_X$	$O(n_d^2)(t_H + t_X)$	$O(n_d)(t_H + t_X)$	$O(n_d)(t_H + t_X)$	NA
Chang [6]	$3t_H + t_m$	$O(n_d^2)(t_m + 2t_H)$	$O(n_d)(t_m + 2t_H)$	$O(n_d)(t_m + 2t_H)$	NA
E-HKAS	$3t_b + t_e + 3t_m$	$n_d(t_b + 4t_e + 4t_m)$	$t_b + 4t_e + 4t_m$	$t_b + 4t_e + 4t_m$	$t_b + 3t_e + 8t_m$

n_d and n_a are the number of descendants and ancestors of the class incident to the deleted/added edge, respectively;

t_p, t_e, t_m, t_X and t_H are the time-costs of a bilinear pairing, modular exponentiation, multiplication, XOR operation and a hash function, respectively.

that features constant key derivation cost and constant cost for enforcing explicit policy exceptions. The main reason for this is constant group update cost of the underlying group encryption scheme.

Table 4 compares time-costs of various operations involved in the HKAS with explicit policy exceptions. Table 5 compares our E-HKAS with the existing HKAS with policy exceptions in terms of storage efficiency. We stress that ours is the only one that satisfies all performance requirements given by Atallah *et al* [1] – constant-length secret keys, linear public storage overhead and constant number of asymmetric key operations. Proposed E-HKAS requires $O(n)$ public storage, constant-length secret key with a key derivation procedure that requires constant number of public-key cryptography operations to be performed by class users. However, the scheme by Lin *et al* [4] requires the class users to perform $O(d)$ asymmetric-key cryptography operations for a hierarchy of depth d. To enforce one explicit transitive and anti-symmetric exception in an on-demand manner, their scheme requires public parameters corresponding to all ancestors classes of the affected class to be updated by the data owner. The schemes by Chang and Chang [5] and Chang [6] feature more efficient key derivation procedures than that of Lin *et al*. However, it requires public parameters to be stored for each relation, which is tightly bounded by $O(n^2)$ where n is the number of classes. Storing $O(n^2)$ public parameters results in impractical computation burden for ensuring secure dynamic update(s) in the hierarchy. Enforcing an explicit anti-symmetric or transitive exception requires updating public parameters corresponding to each ancestor and each descendant of the affected security class. On the other hand, the proposed E-HKAS enforces transitive and anti-symmetric exceptions in an on-demand manner using procedures that require constant time irrespective of the number of ancestors/descendants of the affected class. Also, the proposed E-HKAS is the only one that explicitly enforces class delegation with descendant(s) safety.

Table 5. Comparison of storage efficiency.

Scheme	# Class secrets	Overall public storage
Lin *et al* [4]	O(1)	$m\lvert\mathbb{Z}_p\rvert$
Chang and Chang [5]	O(1)	$O(n^2)\lvert\mathbb{G}_1\rvert$
Chang [6]	O(1)	$O(n^2)\lvert\mathbb{Z}_p\rvert + 4n\lvert\mathbb{G}_1\rvert$
E-HKAS	O(1)	$n(5\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert + \lvert\mathbb{Z}_p\rvert)$

n and m are the number of classes and number of edges in the hierarchy, respectively.

$\lvert\mathbb{G}_1\rvert, \lvert\mathbb{G}_2\rvert$ and $\lvert\mathbb{Z}_p\rvert$ are the size of components from $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{Z}_p$, respectively.

We implemented the proposed E-HKAS using Pairing-based Cryptography (PBC) library [1] on a system supporting 4 GB RAM, Intel Core i3 processor, having 64-bit Ubuntu 16.04 LTS. The data owner and all security classes have the same system configuration. The results of implementation are shown in figure 5. Initial key assignment in the

[1] https://crypto.stanford.edu/pbc/.

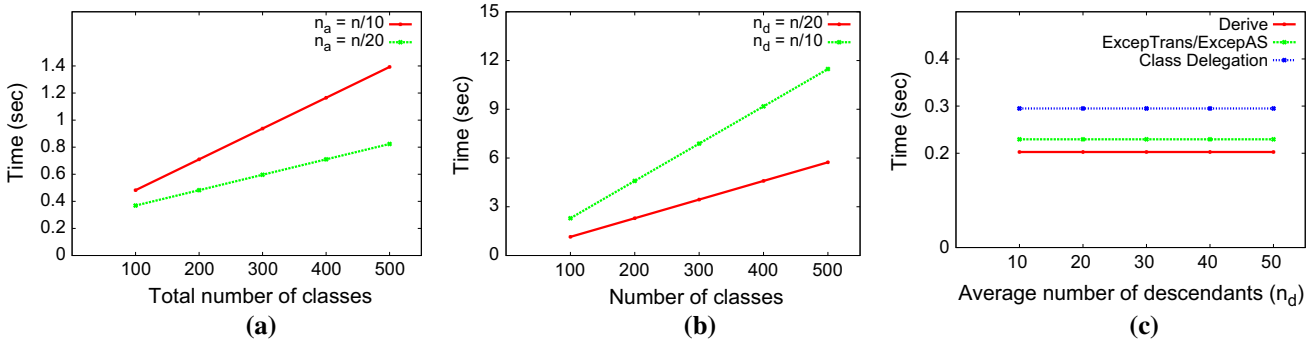


Figure 5. (a) Time-cost of initialization with number of classes and average number of ancestors (n_a) of each class, (b) time-cost of dynamic updates (edge delete/addition) with number of classes and average number of descendants (n_d) of each class and (c) time-costs of key derivation and enforcing policy exceptions with average number of descendants (n_d) of each class.

hierarchy requires formation of access groups and generation of public-secret key-pairs for each security class. Formation of groups requires computational efforts linear in the number of group members n_a (ancestors of a given class). Variation of time-cost of one execution of the procedure Initialize with respect to the number of classes with different average numbers of ancestors is shown in figure 5(a). Clearly, time-cost when $n_a = n/10$ is higher compared with the cost when $n_a = n/20$. Variation of time-cost for carrying out an EdgeDelete or EdgeNew operation with respect to the number of descendants is depicted in figure 5(b). Time required for dynamic updates when $n_d = n/10$ and $n_d = n/20$ are both in acceptable limits. Finally, variation of time-costs for key-derivation and enforcing all three explicit policy exceptions (anti-symmetric exception, transitive exception and class delegation with descendant(s) safety) is presented in figure 5(c). The costs of carrying out dynamic updates and explicit policy exceptions remain constant with respect to n as well as n_d. Also, all these three costs are within the acceptable limits. Therefore, implementation results reflect practical applicability of the proposed E-HKAS for both individual users in security classes as well as the data owner.

6. Conclusions

Flexibility of an access control model with practical safety is an important requirement that affects its applicability in practical scenarios. In this paper, we have proposed E-HKAS that efficiently enforces explicit transitive and anti-symmetric exceptions in a dynamic hierarchy of security classes with constant computation overhead per exception on the data owner. Additionally, E-HKAS enforces class delegation with descendant(s) safety, which is our new practically motivated flexibility requirement. To realize E-HKAS, we have proposed a primitive called GPRE and presented a generic construction of E-HKAS using GPRE. Also, we have proposed a concrete IND-CPA-secure GPRE scheme and proved its security. Performance analyses suggest that the proposed E-HKAS is the only one that efficiently enforces all policy exceptions in a dynamic hierarchy while satisfying all desirable performance requirements. In particular, the costs of enforcing policy exceptions are constant with respect to the number of descendants or ancestors of a security class. The proposed E-HKAS is proved to be secure under V-DDH assumption in standard model whose hardness can be reduced to that of DDH (Decisional Diffie–Hellman) assumption. E-HKAS does not assume anything apart from the standard model.

Acknowledgements

This work was supported by the Ministry of Human Resource Development, Government of India.

References

[1] Atallah M J, Frikken K B and Blanton M 2005 Dynamic and efficient key management for access hierarchies. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pp. 190–202

[2] Yeh J H, Chow R and Newman R 1998 A key assignment for enforcing access control policy exceptions. In: *Proceedings of the International Symposium on Internet Technology*, pp. 54–59

[3] Yeh J H, Chow R and Newman R 2003 Key assignment for enforcing access control policy exceptions in distributed systems. *Inf. Sci.* 152: 63–88

[4] Lin I C, Hwang M S and Chang C C 2003 A new key assignment scheme for enforcing complicated access control policies in hierarchy. *Future Gener. Comput. Syst.* 19: 457–462

[5] Chang Y F and Chang C C 2007 Tolerant key assignment for enforcing complicated access control policies in a hierarchy. *Fundam. Inform.* 76: 13–23

[6] Chang Y F 2015 A flexible hierarchical access control mechanism enforcing extension policies. *Secur. Commun. Netw.* 8: 189–201

[7] Koti N and Purushothama B R 2016 Group-oriented encryption for dynamic groups with constant rekeying cost. *Secur. Commun. Netw.* 9: 4120–4137

[8] Pareek G and Purushothama B R 2019 Flexible cryptographic access control through proxy re-encryption between groups. In: *Proceedings of the 20th International Conference on Distributed Computing and Networking*, pp. 507–507

[9] Qin Z, Xiong H, Wu S and Batamuliza J 2016 A survey of proxy re-encryption for secure data sharing in cloud computing. *IEEE Trans. Serv. Comput.* 9: 1–18

[10] Akl S G and Taylor P D 1983 Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS'83)* 1: 239–248

[11] Sandhu R S 1998 Cryptographic implementation of a tree hierarchy for access control. *Inf. Process. Lett.* 27: 95–98

[12] Zhang Q and Wang Y 2004 A centralized key management scheme for hierarchical access control. In: *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'04)*, pp. 2067–2071

[13] Ferrara A L and Masucci B 2003 An information-theoretic approach to the access control problem. In: *Proceedings of the Italian Conference on Theoretical Computer Science*, pp. 342–354

[14] Chang C C, Lin I C, Tsai H M and Wang H H 2004 A key assignment scheme for controlling access in partially ordered user hierarchies. In: *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA'04)*, pp. 376–379

[15] Das M L, Saxena A, Gulati V P and Phatak D B 2005 Hierarchical key management scheme using polynomial interpolation. *ACM SIGOPS Oper. Syst. Rev.* 39: 40–47

[16] Hui M T and Chin C C 1995 A cryptographic implementation for dynamic access control in a user hierarchy. *Comput. Secur.* 14: 159–166

[17] Odelu V, Das A K and Goswami A 2013 A novel linear polynomial-based dynamic key management scheme for hierarchical access control. *Int. J. Trust Manag. Comput. Commun.* 1: 156–174

[18] Odelu V, Das A K and Goswami A 2013 An effective and secure key-management scheme for hierarchical access control in e-medicine system. *J. Med. Syst.* 37: 1–18

[19] Chen Y R, Chu C K, Tzeng W G and Zhou J 2013 Cloudhka: a cryptographic approach for hierarchical access control in cloud computing. In: *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'13)*, pp. 37–52

[20] Tang S, Li X, Huang X, Xiang Y and Xu L 2016 Achieving simple, secure and efficient hierarchical access control in cloud computing. *IEEE Trans. Comput.* 65: 2325–2331

[21] Pareek G and Purushothama B R 2017 On efficient access control mechanisms in hierarchy using unidirectional and transitive proxy re-encryption schemes. In: *Proceedings of the 14th International Conference on Security and Cryptography (SECRYPT'17)*, pp. 519–524

[22] Chen Y R and Tzeng W G 2017 Hierarchical key assignment with dynamic read–write privilege enforcement and extended KI-security. In: *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'17)*, pp. 165–183

[23] Hwang M S 2000 Cryptanalysis of YCN key assignment scheme in a hierarchy. *Inf. Process. Lett.* 73: 97–101

[24] Freire E S V, Paterson K G and Poettering B 2013 Simple, efficient and strongly KI-secure hierarchical key assignment schemes. In: *Proceedings of the Cryptographers Track at the RSA Conference (CT-RSA)*, pp. 101–114

[25] Castiglione A, Santis A D and Masucci B 2015 Key indistinguishability versus strong key indistinguishability for hierarchical key assignment schemes. *IEEE Trans. Depend. Secure Comput.* 13: 451–460

[26] Santis A D, Ferrara A L and Masucci B 2011 Efficient provably-secure hierarchical key assignment schemes. *Theor. Comput. Sci.* 412: 5684–5699