Supporting dynamic updates in storage clouds with the Akl–Taylor scheme

Arcangelo Castiglione[a], Alfredo De Santis[a], Barbara Masucci[a], Francesco Palmieri[a], Xinyi Huang[b,c], Aniello Castiglione[a,*]

[a] *Department of Computer Science, University of Salerno, Via Giovanni Paolo II, 132, I-84084, Fisciano (SA), Italy*
[b] *Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, Fujian, 350117, China*
[c] *State Key Laboratory of Cryptology, P. O. Box 5159, Beijing, 100878, China*

ARTICLE INFO

Keywords:
Access control
Key assignment
Dynamic updates
Key management
Cloud storage

ABSTRACT

With the wide diffusion of cloud technologies, an ever increasing amount of sensitive data is moved on centralized network-based repository services, providing elastic outsourced storage capacity, available through remote access. This introduces new challenges associated to the security and privacy of outsourced data that has to be dynamically created, shared, updated and removed by a large number of users, characterized by different access rights and views structured according to hierarchical roles.

To address such challenges, and implement secure access control policies in those application domains, several cryptographic solutions have been proposed. In particular, hierarchical key assignment schemes represent an effective solution to deal with cryptographic access control. Starting from the first proposal due to Akl and Taylor in 1983, many hierarchical key assignment schemes have been proposed. However, the highly dynamic nature of cloud-based storage solutions may significantly stress the applicability of such schemes on a wide scale.

In order to overcome such limitations, in this work we provide new results on the Akl–Taylor scheme, by carefully analyzing the problem of supporting dynamic updates, as well as key replacement operations. In doing this, we also perform a rigorous analysis of the Akl–Taylor scheme in the dynamic setting characterizing storage clouds, by considering different key assignment strategies and proving that the corresponding schemes are secure with respect to the notion of key recovery.

1. Introduction

Cloud-based data repositories are emerging network facilities in which a potentially unlimited storage capacity is provided as an on-demand outsourced service over the Internet, by totally hiding platform and implementation details. Nowadays, a huge number of users transparently rely on such outsourced storage solutions, e.g., by using the *Apple Cloud*, *Dropbox* and *Google Drive*, etc. In this way, such users can access their files from anywhere at any time.

* Corresponding author. Fax: +39089969821.
E-mail addresses: arcastiglione@unisa.it (Ar. Castiglione), ads@unisa.it (A. De Santis), bmasucci@unisa.it (B. Masucci), fpalmieri@unisa.it (F. Palmieri), xyhuang81@gmail.com (X. Huang), castiglione@ieee.org, castiglione@acm.org (An. Castiglione).

http://dx.doi.org/10.1016/j.ins.2016.08.093

However, in the presence of sensitive information to be stored, often shared in a controlled manner among several users typically organized into groups whose composition may frequently change according to dynamic access control policies, such data may naturally become an attack target. This scenario is further complicated by the presence of honest-but-curious cloud service providers [17,52,53] or, worse, of malicious providers that aim at disclosing the involved data to make illegal profits.

Therefore, cryptographic access control is an effective way for protecting data privacy, by considering at the same time that such data has to be dynamically created, shared, updated and removed by a number of users not known a priori, which are characterized by different access rights and level of views, often structured according to hierarchical schemes. To address the aforementioned challenges and implement secure access control policies in such complex application domains, several hierarchical key assignment schemes are available, allowing only authorized users to access the data at each specific security level. In particular, the use of cryptographic techniques to enforce access control in hierarchical structures has been first proposed by Akl and Taylor in 1983 [2]. Due to its simplicity and versatility, this scheme has been widely used, for more than thirty years, to implement access control in many scenarios. However, the highly dynamic nature of cloud-based storage solutions may significantly stress the applicability of such a scheme in large-scale infrastructures.

In this work we provide new results on the Akl–Taylor scheme, by carefully analyzing the problem of supporting dynamic updates, such as insertion and deletion of classes and relations between classes in the hierarchy, as well as key replacement operations. In doing this, we also perform a rigorous analysis of the Akl–Taylor scheme in the dynamic setting setting characterizing storage clouds, by considering different key assignment strategies and proving that the corresponding schemes are secure with respect to the notion of key recovery. Those experiences showed us the sustainability of flexible and fine-grained data access control in storage cloud scenarios through hierarchical key assignment schemes. The paper is organized as follows: in Section 2 we review the main works related to our proposal. In Section 3 we describe the architecture and the operating scenario of our solution for supporting dynamic updates in storage clouds. In Section 4 we recall the notions and definitions concerning hierarchical key assignment schemes for dynamic structures. In Section 5 we analyze the hierarchical key assignment scheme proposed by Akl and Taylor. In Section 6 we describe how to support dynamic updates in storage clouds by using the Akl–Taylor scheme, whereas, in Section 7 we analyze the security of the proposed solution. Finally, in Section 8 we conclude the paper.

2. Related work

In 1983 Akl and Taylor [2] proposed the use of cryptographic techniques to enforce access control in hierarchical structures. In particular, they designed a hierarchical key assignment scheme where each class is assigned an encryption key that can be used, along with some public parameters, to compute the key assigned to all classes lower down in the hierarchy. The security of this scheme is related to the infeasibility of extracting r-th roots modulo n, where $r > 1$ is an integer and n is the product of two large unknown primes. However, the analysis made by Akl and Taylor gives only an *intuition* for the security of their scheme.

Subsequently, many researchers have proposed schemes offering different trade-offs in terms of the amount of public and private information and the complexity of key derivation (e.g., [4,6,18,26,32,37,38,40,41,45,46,56]). Many other proposals either support more general access control policies [23,25,39,57] or satisfy additional time-dependent constraints [7,20,27,35,50,51,55,57]. Despite the large number of proposed schemes, many of them lack of a formal security proof and have been shown to be insecure against *collusive attacks* [7,24,48,58,59], whereby two or more classes collude to compute a key to which they are not entitled.

Atallah et al. [3] first proposed two different notions of security for hierarchical key assignment schemes, namely, security against *key recovery* and with respect to *key indistinguishability*. In the former case, an adversary cannot *compute* a key which cannot be derived by the users he has corrupted, whereas, in the latter case, the adversary is not even able to *distinguish* the key from a random string of the same length. Different constructions satisfying the above defined notions of security have been proposed in [29,30]. D'Arco et al. [21,22] analyzed the Akl–Taylor scheme according to the definitions proposed in [3] and showed how to choose the public parameters in order to get instances of the scheme which are secure against key recovery under the RSA assumption. Moreover, they showed how to turn the Akl–Taylor scheme in a construction offering security with respect to key indistinguishability. Ateniese et al. [8] extended the model proposed in [3] to schemes satisfying additional time-dependent constraints and proposed two different constructions offering security with respect to key indistinguishability. Other constructions for time-dependent schemes, offering different trade-offs in terms of amount of public and private information and complexity of key derivation, were shown in [5,9,28].

Recently, Freire et al. [31] proposed new security definitions for hierarchical key assignment schemes, called security against *strong key recovery* and security with respect to *strong key indistinguishability*. Such definitions provide the adversary with additional compromise capability. The equivalence between security with respect to strong key indistinguishability and security with respect to key indistinguishability has been recently proven in [13]. A similar result was previously shown for the *unconditionally secure setting* [11].

In [12,16], a more general scenario has been considered for hierarchical key assignment schemes. In such a scenario, the access control is not only hierarchical, but also *shared* between different classes. In particular, the authors of [16] proposed two constructions for hierarchical and shared key assignment schemes which are secure with respect to key indistinguishability, where the former relies on both *symmetric encryption* and *perfect secret sharing*, whereas, the latter is based on threshold broadcast encryption. Finally, in [15] a model for hierarchical key assignment schemes supporting dynamic updates has

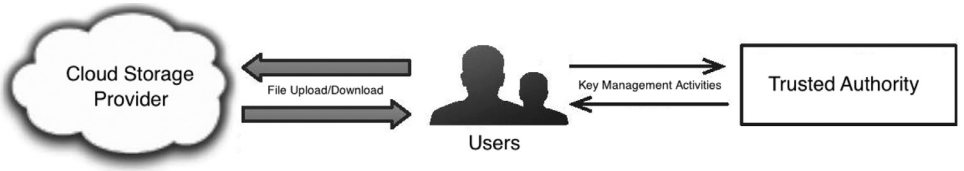

Fig. 1. The operating scenario.

been recently proposed. In particular, the notions of security against key recovery and with respect to key indistinguishability, provided by Atallah et al., have been extended to address the further challenges introduced by the updates to the hierarchy.

Moreover, for what concerns hierarchical access control in clouds, Wang et al. [54] proposed one of the first hierarchical access control solutions for cloud storage, relying on *Ciphertext-Policy Attribute Based Encryption (CP-ABE)* [10] and *Hierarchical Identity Based Encryption (HIBE)* [34] schemes. Subsequently, a distributed model for data storage and access control in clouds was presented in [44]. A new hierarchical key assignment scheme for cloud infrastructures, known as the *CloudHKA* scheme, which follows the *Bell-LaPadula* security model and efficiently deals with user revocation, was proposed in [19]. The solution described in [33] consists of an efficient hierarchical key assignment scheme supporting time-dependent constraints, which is based on a bilinear pairing function and provides access control in cloud environments. Finally, a very recent contribution [49] presents a hierarchical key assignment scheme based on *linear-geometry* as the solution for flexible and fine-grained hierarchical access control in cloud computing.

3. System architecture and operating scenario

The most common way for protecting sensitive data outsourced to third party entities is storing such data in an encrypted form, by keeping the right decryption keys available only to authorized users [47,60,61]. Therefore, such a solution requires the usage of an efficient key management scheme for distributing the keys to legitimate users, often structured in classes according to their privileges and views on the stored data. Clearly, such keys have to reflect the current access rights of a large number of involved users, whose roles, privileges and data views can continuously and dynamically change, by minimizing the need of re-encrypting the related data and performing key re-distribution. Accordingly, the reference architecture describing the operating scenario of our proposal is characterized by three fundamental entities (see Fig. 1):

- The *Cloud Storage Repository*, which is the entity providing outsourced storage facilities. It is in charge of granting the access from outside users to the data, stored by using specific keys (that are not under its control), so that only the users holding the right keys are able to access them. In this work we assume that the cloud storage repository is always online and has unlimited storage capacity.
- The *users* of the storage services, whose data are structured into files. For each file, we consider two types of users. The former is the owner, who decided to outsource his data into the cloud for storing and making them accessible in a controlled way. The latter is the generic file user, interested in the involved data, which, according to the permissions granted by the owner and his position on the access control hierarchy (its security class), needs to access some of the outsourced files. Clearly, data owners encrypt their data files and store them in the cloud for the sharing with other users, who download encrypted data files of their interest from the cloud and then decrypt them, by relying on specific symmetric encryption schemes and keys.
- A *Trusted Authority* (TA), which is responsible for providing the data encryption schemes and managing and distributing the associated keys. Data owners, generic users and the TA are structured in a hierarchy, where the TA acts as the *root of trust* and several hierarchical relations are implemented between users, so that the user belonging to higher classes can also access the data in the lower ones, but not vice versa.

Based on the above operating scenario, we consider some threat assumptions:

- The channels connecting users to cloud storage repository can be public, hence, the data transferred by using these channels might be eavesdropped by outside attackers.
- The cloud storage repository can be “honest-but-curious”, that is, it supports the aforementioned hierarchical scheme, but tries to extract as much secret information as possible. We can also transform this one in a worst-case assumption, in which it is completely untrusted and may collude with malicious/compromised users in harvesting files contents for its own benefit.
- The TA cannot be compromised, even if it can be considered as a *single point of failure*.

The security goal considered in this work is to provide effective hierarchical access control with respect to the most known threats/attacks, also in the presence of frequent access rights revocations, as well as creation/deletion of classes and relations. In the following section we present the theoretical model that describes our contribution in the presented cloud-based scenario.

4. Hierarchical key assignment schemes for dynamic structures

Hierarchical key assignment schemes are widely used to implement secure access control policies in environments where users and resources can be modeled through partially ordered hierarchies. Since hierarchies are a natural way of organizing users according to their positions as parts of an organization, they are widely employed in many different application areas (e.g., database management systems, computer networks, operating systems, military, government communications) and play a fundamental role within any scenario which can be modeled by using *Role-Based Access Control (RBAC)*.

Consider a set of users divided into a number of disjoint classes, called *security classes*. A security class can represent a person, a department or a user group in an organization. A binary relation $\preceq$ that partially orders the set of classes V is defined in accordance with authority, position or power of each class in V. The poset $(V, \preceq)$ is called a *partially ordered hierarchy*. For any two classes u and v, the notation $u \preceq v$ is used to indicate that the users in v can access u's data. Clearly, since v can access its own data, it holds that $v \preceq v$, for any $v \in V$. The partially ordered hierarchy $(V, \preceq)$ can be represented by the directed graph $G^* = (V, E^*)$, where each class corresponds to a vertex in the graph and there is an edge from class v to class u if and only if $u \preceq v$. We denote by $G = (V, E)$ the *minimal representation* of the graph G^*, namely, the directed acyclic graph corresponding to the *transitive and reflexive reduction* of the graph $G^* = (V, E^*)$. The graph G has the same transitive and reflexive closure of G^*, i.e., there exists a path (of length greater than or equal to zero) from v to u in G if and only if there is the edge (v, u) in E^*. Aho et al. [1] showed that every directed graph has a transitive reduction, which can be computed in polynomial time and is unique for directed acyclic graphs. In the following, we denote by Γ a family of graphs corresponding to partially ordered hierarchies. For example, Γ could be the family of the rooted trees [45], the family of the d-dimensional hierarchies [4], etc.. Let Γ be a family of graphs corresponding to partially ordered hierarchies and let $G = (V, E)$ be a graph in Γ. For any class $v \in V$, let A_v^G be the *accessible set* of v in G, i.e., the set $\{u \in V :$ there is a path from v to u in $G\}$ of classes which can be accessed by v in G. Similarly, let F_v^G be the *forbidden set* of v in G, i.e., the set $\{u \in V :$ there is no path from u to v in $G\}$ of classes which cannot access v in G.

A hierarchical key assignment scheme for a family Γ of partially ordered hierarchies, supporting dynamic updates, has been first introduced in [15] and is defined as follows.

Definition 1. A *hierarchical key assignment scheme* for Γ, supporting *dynamic updates*, is a triple (*Gen*, *Der*, *Upd*) of algorithms satisfying the following conditions:

1. The *information generation algorithm Gen*, executed by the Trusted Authority (TA), is probabilistic polynomial-time. It takes as inputs the security parameter 1^τ and a graph $G = (V, E)$ in Γ, and produces as outputs
   (a) a private information s_u, for any class $u \in V$;
   (b) a key $k_u \in \{0, 1\}^\tau$, for any class $u \in V$;
   (c) a public information *pub*.
   We denote by (s, k, *pub*) the output of the algorithm *Gen* on inputs 1^τ and G, where s and k denote the sequences of private information and of keys, respectively.
2. The *key derivation algorithm Der*, executed by some authorized user, is deterministic polynomial-time. It takes as inputs the security parameter 1^τ, a graph $G = (V, E)$ in Γ, two classes $u \in V$ and $v \in A_u^G$, the private information s_u assigned to class u and the public information *pub*, and produces as output the key $k_v \in \{0, 1\}^\tau$ assigned to class v.
   We require that for each class $u \in V$, each class $v \in A_u^G$, each private information s_u, each key $k_v \in \{0, 1\}^\tau$, each public information *pub* which can be computed by *Gen* on inputs 1^τ and G, it holds that
   $$Der(1^\tau, G, u, v, s_u, pub) = k_v.$$
3. The *update algorithm Upd*, executed by the TA, is probabilistic polynomial-time. It takes as inputs the security parameter 1^τ, a graph $G = (V, E)$ in Γ, the tuple (s, k, *pub*) (generated either by *Gen* or by *Upd* itself), an *update type up*, a sequence of additional parameters *params*, and produces as outputs
   (a) a updated graph $G' = (V', E')$ in Γ;
   (b) a private information s'_u, for any class $u \in V'$;
   (c) a key $k'_u \in \{0, 1\}^\tau$, for any class $u \in V'$;
   (d) a public information *pub*$'$.
   The sequence *params*, if not empty, is used to generate new keys and secret information as a consequence of the update type *up*. We denote by (s', k', *pub*$'$) the sequences of private information, keys, and public information output by *Upd*(1^τ, G, s, k, *pub*, *up*, *params*).

The update types we consider are the following: *insertion of an edge*, *insertion of a class*, *deletion of an edge*, *deletion of a class*, *key replacement*, and *revocation of a user from a class*. Notice that some types of updates can be seen as a sequence of other types of updates. For example, the deletion of a class u can be performed by executing a sequence of edge deletions, one for each edge ingoing u and outgoing from u. On the other hand, the deletion of the edge (u, v) requires a key replacement operation for the class v. Finally, the revocation of a user from a class u requires a sequence of key replacement operations. In the above definition, it is required that the updated graph G' still belongs to the family Γ of partially ordered hierarchies, i.e., only updates which preserve the partial order relation between the classes in the hierarchy are allowed.

4.1. Security

The notions of security against key recovery and with respect to key indistinguishability, first proposed by Atallah et al. [3] for hierarchical key assignment schemes not supporting dynamic updates, have been extended in [15] to address the further security challenges introduced by the algorithm *Upd* used for handling dynamic updates to the hierarchy. More precisely, in order to evaluate the security of a hierarchical key assignment scheme supporting dynamic updates, an *adaptive adversary* ADAPT attacking the scheme has been considered. Such an adversary can make three different types of operations: *performing a dynamic update*, *corrupting a class*, and *attacking a class*.

The first type of operation includes all kinds of updates described before. More precisely, consider an *updating oracle* $\mathcal{U}$, modeling the behavior of the *TA*, which performs the required updates on the hierarchy. At the beginning, the state of the updating oracle is represented by the tuple (G^0, s^0, k^0, pub^0), where (s^0, k^0, pub^0) is the output of algorithm *Gen* on inputs 1^τ and the initial graph G^0. For any $i \geq 0$, the $(i+1)$-th adversary's query to the updating oracle consists of a pair $(up^{i+1}, params^{i+1})$, where up^{i+1} is an update operation on the graph G^i and $params^{i+1}$ is the sequence of parameters associated to the update, which the oracle answers with the updated graph G^{i+1}, the public information pub^{i+1} associated to G^{i+1}, and with a sequence of keys, denoted by old_k^i, which have been modified as a consequence of the update, according to the specification of the algorithm *Upd*. More precisely, the updating oracle $\mathcal{U}_{(1^\tau, G^i, s^i, k^i, pub^i)}(\cdot,\cdot)$, given the query $(up^{i+1}, params^{i+1})$, runs algorithm $Upd(1^\tau, G^i, s^i, k^i, pub^i, up^{i+1}, params^{i+1})$ and returns G^{i+1}, pub^{i+1}, and old_k^i to the adversary. In the following, we denote by $\mathcal{U}^i(\cdot,\cdot)$ the oracle $\mathcal{U}_{(1^\tau, G^i, s^i, k^i, pub^i)}(\cdot,\cdot)$. Due to its adaptive nature, the adversary may require a polynomial number $m = poly(|V|, 1^\tau)$ of dynamic updates, where each update is decided on the basis of the answers obtained from the updating oracle at the previous steps.

The second type of operation is the *class corruption*, which can be performed in an adaptive order and for a polynomial number of classes. For any $i \geq 0$, consider a *corrupting oracle* $\mathcal{C}^i$, which provides the adversary with the private information held by the corrupted classes in the graph G^i. In particular, an adversary's query to the corrupting oracle $\mathcal{C}^i$ consists of a class v in the graph G^i, which the oracle answers with the private information held by class v in all graphs $G^0, G^1, \ldots, G^i$ (if v belongs to them).

Finally, the third type of operation is the *class attack*, where the adversary chooses an update index t and a class u in the hierarchy G^t and is challenged either in computing the key k_u^t or in distinguishing k_u^t from a random string in $\{0,1\}^\tau$, depending on the security requirement.

More precisely, the *adaptive adversary* ADAPT considered in [15] runs in two stages, i.e., $\mathtt{ADAPT} = (\mathtt{ADAPT}_1, \mathtt{ADAPT}_2)$. In advance of the adversary's execution, the algorithm *Gen* is run on inputs 1^τ and G and outputs the tuple (s, k, pub), which is kept hidden from the adversary's view, with the exception of the public information *pub*. During the first stage, the adversary $\mathtt{ADAPT}_1$ is given access to both updating and corrupting oracles for a polynomial number m of times. The responses obtained by the oracles are saved in some state information denoted as *history*. In particular, *history* contains the following information: 1) the sequence of graphs $\vec{G} = (G^0, \ldots, G^m)$; 2) the sequence $\vec{up} = (up^1, \ldots, up^m)$ of updating operations queried to the oracle $\mathcal{U}$; 3) the corresponding sequence of public information $\vec{pub}$; 4) the corresponding sequence $\vec{old\_k}$ of keys which have been modified according to each update; 5) the private information held by all corrupted classes.

After interacting with the updating and corrupting oracles, the adversary chooses an update index t and a class u in G^t, among all the classes in G^t which cannot be accessed by the corrupted classes and such that the key k_u^t cannot be computed by the old keys which have been updated as a consequence of each update. In particular, the chosen class u is such that, for any class v already queried to the corrupting oracle $\mathcal{C}^i(\cdot)$ and any $i = 0, \ldots, m$, v cannot access u in the hierarchy G^i. In the second stage, the adversary $\mathtt{ADAPT}_2$ is given again access to the corrupting oracle and is then challenged either in computing the key k_u^t or in distinguishing k_u^t from a random string in $\{0,1\}^\tau$, depending on the security requirement. In order for the scheme to be secure with respect to *key recovery*, it is required that the adversary will guess the key k_u^t with probability only negligibly different from $1/2^\tau$. Clearly, it is required that the key k_u^t on which the adversary will be challenged is not included in the sequence old_k^{t-1} of keys which have been updated in the graph G^t. The next definition was proposed in [14].

Definition 2. [REC-DYN-AD] Let Γ be a family of graphs corresponding to partially ordered hierarchies, let $G = (V, E) \in \Gamma$ be a graph and let (Gen, Der, Upd) be a hierarchical key assignment scheme for Γ supporting dynamic updates. Let $m = poly(|V|, 1^\tau)$ and let $\mathtt{ADAPT} = (\mathtt{ADAPT}_1, \mathtt{ADAPT}_2)$ be an adaptive adversary that during the first stage of the attack is given access both to the updating oracle $\mathcal{U}^i(\cdot,\cdot)$ and the corrupting oracle $\mathcal{C}^i(\cdot)$, for $i = 1, \ldots, m$, and during the second stage of the attack is given access only to the corrupting oracle. Consider the following experiment:

Experiment $\mathbf{Exp}^{\mathtt{REC-DYN}}_{\mathtt{ADAPT}}(1^\tau, G)$
$(s, k, pub) \leftarrow Gen(1^\tau, G)$
$(t, u, history) \leftarrow \mathtt{ADAPT}_1^{\mathcal{U}^i(\cdot,\cdot), \mathcal{C}^i(\cdot)}(1^\tau, G, pub)$
$k_u^{t,*} \leftarrow \mathtt{ADAPT}_2^{\mathcal{C}^i(\cdot)}(1^\tau, t, u, history)$
return $k_u^{t,*}$

It is required that the class u output by $\mathtt{ADAPT}_1$ is such that v cannot access u in the graph G^i, for any class v already queried to the corrupting oracle $\mathcal{C}^i(\cdot)$. Moreover, it is required that $\mathtt{ADAPT}_2$ never queries the corrupting oracle $\mathcal{C}^i(\cdot)$ on a

class v such that v can access u in the graph G^t. Finally, it is also required that the key k_u^t cannot be computed by any old key in the *history* sequence. The advantage of ADAPT is defined as

$$\mathbf{Adv}_{\mathrm{ADAPT}}^{\mathrm{REC-DYN}}(1^\tau, G) = Pr[k_u^{t,*} = k_u^t].$$

The scheme is said to be *secure in the sense of* REC-DYN-AD if, for each graph $G = (V, E)$ in Γ, the function $\mathbf{Adv}_{\mathrm{ADAPT}}^{\mathrm{REC-DYN}}(1^\tau, G)$ is negligible, for each adaptive adversary ADAPT whose time complexity is polynomial in τ.

Notice that if the adversary ADAPT_1 never queries the updating oracle during the first stage of the attack, the above definition reduces to that of security against key recovery in the presence of adaptive adversaries for hierarchical key assignment schemes with fixed hierarchies, referred to as REC-AD in [8]. For such kind of schemes, it has been proven that adaptive adversaries are *polynomially equivalent* to *static* adversaries, i.e., such that the class to be attacked is chosen in advance to the execution of the scheme. The security definition corresponding to such kind of adversaries and schemes is referred to as REC-ST in [8]. Thus, since REC-AD and REC-ST have been shown to be equivalent, in the following we will only consider the latter definition, when analyzing the security of hierarchical key assignment schemes with fixed hierarchies. Again, the notion of security with respect to *key indistinguishability* for *hierarchical key assignment schemes supporting dynamic updates*, referred to as [IND-DYN-AD], has been proposed in [15]. Finally, we stress that the relations among all security notions for *hierarchical key assignment schemes supporting dynamic updates* have been explored and clarified in [14].

5. The Akl–Taylor scheme

In this section we analyze one of the most used hierarchical key assignment schemes, due to Akl and Taylor [2]. In such a scheme each class $u \in V$ holds a single secret value s_u, which coincides with its secret key k_u. An interesting feature of the Akl–Taylor scheme is that it requires a small amount of public information and allows the key derivation to be direct, i.e., each users can derive a key in a single step.

The Akl–Taylor scheme is described in the following. Let Γ be a family of graphs corresponding to partially ordered hierarchies, and let $G = (V, E) \in \Gamma$.

Algorithm $Gen(1^\tau, G)$:

1. Randomly choose two distinct large primes p and q having bitlength τ and compute $n = p \cdot q$;
2. For each class $v \in V$, compute an integer λ_v, such that λ_u divides λ_v if and only if $v \in A_u^G$ (more details are provided immediately after the description of the algorithms);
3. Let *pub* be the sequence of public information computed in the previous step, along with the value n;
4. Randomly choose a global starting value k_0, where $1 < k_0 < n$;
5. For each class $v \in V$, compute the private information s_v and the encryption key k_v as follows:

$$s_v = k_v = k_0^{\lambda_v} \bmod n;$$

6. Let s and k be the sequences of private information and keys, respectively, computed in the previous step;
7. Output (s, k, pub).

Algorithm $Der(1^\tau, G, u, v, s_u, pub)$
Extract the values λ_v and λ_u from *pub* and compute

$$s_u^{\lambda_v/\lambda_u} \bmod n = (k_0^{\lambda_u})^{\lambda_v/\lambda_u} \bmod n = k_v.$$

A crucial issue is how to perform step 2. of Algorithm $Gen(1^\tau, G)$. Akl and Taylor [2] and MacKinnon et al. [40] proposed two different public values assignments such that for each pair of public values λ_u and λ_v, the value λ_u divides λ_v if and only if $v \in A_u^G$.

The Akl–Taylor assignment. For each $v \in V$, choose a distinct prime number p_v and compute the public value λ_v as follows:

$$\lambda_v = \begin{cases} 1 & \text{if } A_v^G = V; \\ \prod_{u \notin A_v^G} p_u & \text{otherwise.} \end{cases}$$

The MacKinnon et al. assignment. Decompose the partially ordered hierarchy $G = (V, E)$ into disjoint chains (a chain is a totally ordered set) and assign each chain a distinct prime number.[1] Then, for each $v \in V$, compute the prime power $n_v = p^i$, where v is the i-th class in the chain whose assigned prime is p. Finally, for each class $v \in V$, compute the public value λ_v as follows:

$$\lambda_v = \begin{cases} 1 & \text{if } A_v^G = V; \\ \mathrm{lcm}_{u \notin A_v^G} n_u & \text{otherwise.} \end{cases}$$

Fig. 2. A partially ordered hierarchy and one of its chain decomposition.

The left hand side of Fig. 2 shows a partially ordered hierarchy, whereas, one of its chain decomposition is shown on the right hand side.

Example of the Akl–Taylor assignment. Consider the hierarchy on the left hand side of Fig. 2, when the primes assigned to the classes are $p_a = 3,\ p_b = 5,\ p_c = 7,\ p_d = 11,\ p_e = 13$. The public values are $\lambda_a = 5,\ \lambda_b = 3 \cdot 7,\ \lambda_c = 3 \cdot 5 \cdot 11, \lambda_d = 3 \cdot 5 \cdot 7,\ \lambda_e = 3 \cdot 5 \cdot 7 \cdot 11$.

Example of the MacKinnon assignment. Consider the chain decomposition shown in Fig. 2, when the prime powers assigned to the classes are $n_a = 3,\ n_b = 5,\ n_c = 3^2,\ n_d = 5^2,\ n_e = 5^3$. The public values are $\lambda_a = 5,\ \lambda_b = 3^2,\ \lambda_c = 3 \cdot 5^2,\ \lambda_d = 3^2 \cdot 5,\ \lambda_e = 3^2 \cdot 5^2$.

Akl and Taylor noticed that in order to construct an Akl–Taylor scheme which is resistant to collusive attacks it is needed that for each $v \in V$ and each $X \subseteq F_v^G$, the $\gcd\{\lambda_u : u \in X\}$ does not divide λ_v. More precisely, they showed the following result:

Lemma 1. [2]. *Let λ and $\lambda_1, \ldots, \lambda_m$ be integers, and let $k \in Z_n$, where $n = p \cdot q$ is the product of two large primes. The power $k^\lambda \bmod n$ can be feasibly computed from the set of powers $\{k^{\lambda_1} \bmod n, \ldots, k^{\lambda_m} \bmod n\}$ if and only if $\gcd\{\lambda_1, \ldots, \lambda_m\}$ divides λ.*

The proof of Lemma 1 relies on the infeasibility of extracting r-th roots modulo n, where $r > 1$ is an integer and n is the product of two large unknown primes.[2] Lemma 1 gives only an *intuition* for the security of the scheme. A rigorous proof of security for the Akl–Taylor scheme was first shown in [22]. In particular, it was proven that the existence of an efficient adversary breaking the security of the scheme against key recovery implies the existence of an adversary which efficiently solves a computational hard problem.

5.1. Complexity assumptions

An *RSA generator* with associated security parameter τ is a randomized algorithm that returns a pair $((n, e), (n, p, q, d))$, where n is the *RSA modulus*, e is the *encryption exponent* and d is the *decryption exponent*, satisfying the following conditions:

- p and q are two distinct large odd primes of τ bits;
- $n = p \cdot q$;
- $e \in Z^*_{\phi(n)}$, where $\phi(n) = (p-1) \cdot (q-1)$;
- $d = e^{-1} \bmod \phi(n)$.

Two strategies to compute the pair $((n, e), (n, p, q, d))$ are used. The former, first chooses the primes p and q, computes n, picks e at random in $Z^*_{\phi(n)}$, and computes d accordingly. Such a strategy yields a *random exponent RSA generator*, denoted $\mathcal{K}^{ran}_{RSA}(1^\tau)$. The latter, fixes the encryption exponent e to be a small odd number, like 3, 17, or $2^{16} + 1$, and then generates the other parameters, accordingly.[3] Given a fixed odd number e, such a strategy yields an *RSA generator for exponent e*, denoted $\mathcal{K}^{fix}_{RSA}(1^\tau, e)$. The RSA generators $\mathcal{K}^{ran}_{RSA}(1^\tau)$ and $\mathcal{K}^{fix}_{RSA}(1^\tau, e)$ are described as follows:

Algorithm $\mathcal{K}^{ran}_{RSA}(1^\tau)$
pick at random two distinct primes p and q of τ bits
$n \leftarrow p \cdot q$
$\phi(n) \leftarrow (p-1) \cdot (q-1)$
$e \leftarrow Z^*_{\phi(n)}$
$d \leftarrow e^{-1} \bmod \phi(n)$
return $((n, e), (n, p, q, d))$

Algorithm $\mathcal{K}^{fix}_{RSA}(1^\tau, e)$
repeat
pick at random two distinct primes p and q of τ bits
until $\gcd(p-1, e) = 1$ and $\gcd(q-1, e) = 1$
$n \leftarrow p \cdot q$
$\phi(n) \leftarrow (p-1) \cdot (q-1)$
$d \leftarrow e^{-1} \bmod \phi(n)$
return $((n, e), (n, p, q, d))$

Let B and G_{RSA} be algorithms where the algorithm G_{RSA} corresponds either to $\mathcal{K}^{ran}_{RSA}(1^\tau)$ or to $\mathcal{K}^{fix}_{RSA}(1^\tau, e)$. Consider the following experiment:

---

[1] The problem of finding a decomposition into a minimal number of chains is solvable in time polynomial in $|V|$ (for example by using Khachiyan's algorithm [36]).

[2] In particular, when $\gcd(r, \phi(n)) = 1$, where $\phi(n)$ is the Euler's totient function, this is the assumption behind the RSA cryptosystem [43]; whereas, if $r = 2$, this assumption is used in the Rabin cryptosystem [42].

[3] There have been some questions raised about the security of this strategy, since it might be possible that roots of small degree are easier to take than roots of a random degree.

Experiment $\mathbf{Exp}_B^{G_{RSA}}$
$((n,e),(n,p,q,d)) \leftarrow G_{RSA}$
$x \leftarrow Z_n^*$
$y \leftarrow x^e \bmod n$
$x' \leftarrow B(n,e,y)$
if $x' = x$ **then return** 1
else return 0

The advantage of B is defined as $\mathbf{Adv}_B^{G_{RSA}} = Pr[\mathbf{Exp}_B^{G_{RSA}} = 1]$.

The RSA generators described above yield the following two assumptions.[4]

Random Exponent RSA Assumption. The function $\mathbf{Adv}_B^{K_{RSA}^{ran}}(1^\tau)$ is negligible, for each probabilistic algorithm B whose time complexity is polynomial in τ.

RSA Assumption for Exponent in a set of odd numbers. Let X be a set of odd numbers. For each $e \in X$, the function $\mathbf{Adv}_B^{K_{RSA}^{fix}}(1^\tau, e)$ is negligible, for each probabilistic algorithm B with time complexity polynomial in τ.

The authors of [22] showed that the Akl–Taylor scheme, according to both Akl–Taylor and MacKinnon et al. assignments, is secure against key recovery, provided that the primes associated to the classes are *properly chosen*. In particular, two different primes choices were considered. The first choice, denoted *fixed primes choice*, yields instances of the Akl-Taylor scheme secure under the RSA assumption for exponent in a set of odd numbers. The second one, denoted *R-random primes choice*, yields instances secure under the random exponent RSA assumption.

- *Fixed primes choice.* Let $PRIMES_\ell = \{p_1, \ldots, p_\ell\}$ be the set of the first ℓ prime numbers greater than 2.
  - Let $u_1, \ldots, u_{|V|}$ be a sorting of V. In the Akl–Taylor assignment, associate a prime $p_j \in PRIMES_{|V|}$ to the class u_j, for each $j = 1, \ldots, |V|$.
  - Let m be the number of disjoint chains in the minimal decomposition of G and let u_j be the first node of the j-th chain. In the MacKinnon et al. assignment, associate a prime $p_j \in PRIMES_m$ to the class u_j, for each $j = 1, \ldots, m$.
- *R -Random primes choice.* Let $\mathtt{R} = \{R_n\}_n$ be a family of sets of integers, where n is an RSA modulus. In the Akl–Taylor assignment (in the MacKinnon et al. assignment, resp.), *choose uniformly at random*, for each class (for each chain in the minimal chain decomposition of the partially ordered hierarchy, resp.), a distinct prime number in R_n, for an appropriately chosen family $\mathtt{R}$.
  In particular, consider the family $\mathtt{R}$ obtained by setting R_n to be equal to the set of integers belonging to the interval $[3, w]$, where $w = 2^{2\tau-2} - 2^\tau$ is a lower bound for the *Euler totient function* $\phi(n)$ (see Lemma 4.3 in [22]). The corresponding primes choice will be denoted in the following as the *random primes choice*.

The next results were proven in [22].

Theorem 1. *Let $G = (V, E)$ be a partially ordered hierarchy. The Akl–Taylor assignment with the fixed primes choice yields a scheme which is secure in the sense of REC−ST under the RSA assumption for exponents in $PRIMES_{|V|}$.*

Theorem 2. *Let $G = (V, E)$ be a partially ordered hierarchy. The Akl–Taylor assignment with the random primes choice yields a scheme secure in the sense of REC−ST under the random exponent RSA assumption.*

Similar results were also proven for the MacKinnon et al. assignment.

6. Supporting dynamic updates with the Akl–Taylor assignment

In this section we show how to manage changes to the hierarchy with the Akl–Taylor assignment. The idea behind our construction is the following: at the beginning, each class in the graph $G = (V, E)$ is assigned a key with the Akl–Taylor assignment using a global starting value k_0. Whenever an update up has to be performed, yielding to the updated graph $G' = (V', E')$, some keys need to be either created for the first time, in the case of the addition of a new class, or replaced. Let $\mathsf{AssignNewKeys} \subseteq V'$ be the set of classes whose keys need to be either created or replaced and assume the existence of an algorithm *CreateSet* which on input the graph $G = (V, E)$ and the update type up, resulting in the updated graph $G' = (V', E')$, outputs the set AssignNewKeys (we will describe later the details of such an algorithm). Given the set AssignNewKeys, we construct the *historical graph* $G^{hist} = (V^{hist}, E^{hist})$, defined as follows:

- $V^{hist} = V \cup \mathsf{NewClasses}$, where $\mathsf{NewClasses} = \{u^{new} : u \in \mathsf{AssignNewKeys}\}$;
- $E^{hist} = E \cup E^{\mathsf{NewClasses}}$, where $E^{\mathsf{NewClasses}}$ is defined as follows:
  - for each class $u^{new} \in \mathsf{NewClasses}$, if there exists a class $v \in V'$ such that $(u, v) \in E'$ and $v^{new} \notin \mathsf{NewClasses}$, then $(u^{new}, v) \in E^{\mathsf{NewClasses}}$;
  - for each class $u^{new} \in \mathsf{NewClasses}$, if there exists a class $v^{new} \in \mathsf{NewClasses}$ such that $(u, v) \in E'$, then $(u^{new}, v^{new}) \in E^{\mathsf{NewClasses}}$.

For example, let G and G' be the two graphs depicted on the left-hand side and on the right-hand side of Fig. 3, respectively, and let $\mathsf{AssignNewKeys} = \{c, d\}$. The corresponding historical graph G^{hist} is depicted in Fig. 4.

[4] To the best of our knowledge, in the literature there is no analysis of the relationships between the security of these two different strategies.

Fig. 3. Two partially ordered hierarchies.

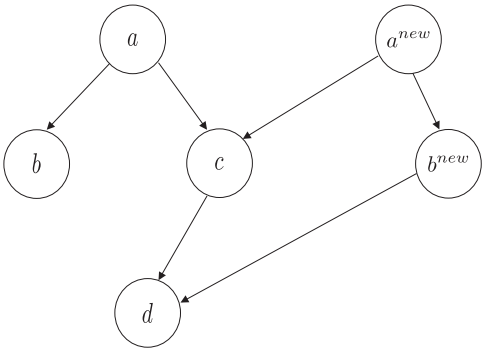

Fig. 4. The historical graph corresponding to the hierarchies in Fig. 3.

It is easy to see that the historical graph $G^{hist} = (V^{hist}, E^{hist})$ has the following features:

- G^{hist} is a partially ordered hierarchy, since the new set $E^{\mathsf{NewClasses}}$ of edges added to E to construct E^{hist} preserves the partial order relation exhibited by E';
- G is a subgraph of G^{hist};
- The set E^{hist} does not contain edges of the form (v, u^{new}), with $v \in V$ and $u^{new} \in \mathsf{NewClasses}$.

As we will show in the following, the historical graph G^{hist} is used to obtain an Akl–Taylor assignment for the updated graph G', starting from an Akl-Taylor assignment for the graph G. In particular, starting from an Akl–Taylor assignment for G, we first set up an Akl-Taylor assignment for G^{hist}, and then use part of such an assignment to obtain an Akl–Taylor assignment for G'. Such an operation is done in such a way to preserve keys which do not need to be changed. This can be accomplished by using a proper choice for the global starting value used in the assignment for G^{hist}, which will be denoted by k_0^{hist}.

We are ready to describe the *Upd* algorithm for the Akl–Taylor assignment. Recall that *Upd* takes as inputs the security parameter 1^τ, the initial graph G, the tuple (s, k, pub) generated by $Gen(1^\tau, G)$ of the Akl–Taylor scheme with the Akl–Taylor assignment, an update type *up*, and a sequence *params* of additional parameters needed for the generation of new keys, as a consequence of the update *up*. In particular, we set *params* to be equal to the global starting value k_0 generated at step 4. of the algorithm $Gen(1^\tau, G)$ in the Akl–Taylor scheme.

Algorithm $Upd(1^\tau, G, s, k, pub, up, params)$
The TA performs the following steps:

1. Parse *pub* to get the sequence of primes assigned by $Gen(1^\tau, G)$ with the Akl–Taylor assignment;
2. Extract the global starting value k_0 from *params*;
3. $\mathsf{AssignNewKeys} \leftarrow CreateSet(G, up)$;
4. Given the set AssignNewKeys, construct the historical graph $G^{hist} = (V^{hist}, E^{hist})$;
5. For each class $v \in \mathsf{AssignNewKeys}$, choose a new prime p_v^{new} not included in the list of primes obtained at step 2.;
6. Let P_{new} be the product of all new primes , i.e.,
$$P_{new} = \prod_{v \in \mathsf{AssignNewKeys}} p_v^{new};$$
7. Set $k_0^{hist} = k_0^{1/P_{new}}$;
8. For each class $v \in V^{hist}$, compute the public value λ_v^{hist} by using the Akl–Taylor assignment for the historical graph G^{hist};
9. For each class $v \in V^{hist}$, let k_v^{hist} be the key obtained by using the Akl–Taylor scheme for the historical graph G^{hist} with the global starting value k_0^{hist};

10. For each class $v \in V'$, if $v \in \mathsf{AssignNewKeys}$, then $\lambda'_v \leftarrow \lambda^{hist}_{v^{new}}$, else $\lambda'_v \leftarrow \lambda^{hist}_v$;
11. For each class $v \in V'$, if $v \in \mathsf{AssignNewKeys}$, then $k'_v \leftarrow k^{hist}_{v^{new}}$, else $k'_v \leftarrow k^{hist}_v$.

The next lemma shows that with the above algorithm all classes in the set $V' \setminus \mathsf{AssignNewKeys}$ maintain the same keys as in the previous assignment. Such a feature follows from the particular form of the value k^{hist}_0, which is set to be equal to $k_0^{1/P_{new}}$, where P_{new} is the product of all new primes assigned to classes in AssignNewKeys.

Lemma 2. *Let $G = (V, E)$ be a partially ordered hierarchy, let up be an update yielding to the updated hierarchy $G' = (V', E')$, let* $\mathsf{AssignNewKeys} \leftarrow \mathit{CreateSet}(G, up)$*, and let $G^{hist} = (V^{hist}, E^{hist})$ be the corresponding historical graph. For any class $v \in V' \setminus$* AssignNewKeys*, it holds that $k'_v = k_v$.*

Proof. Let v be a class in $V' \setminus \mathsf{AssignNewKeys}$. From the *Upd* algorithm we have that

$$k'_v = k^{hist}_v = (k^{hist}_0)^{\lambda^{hist}_v} = (k_0^{1/P_{new}})^{\lambda^{hist}_v}.$$

In the following we prove that, for any $v \in V' \setminus \mathsf{AssignNewKeys}$, the public value λ^{hist}_v is equal to $\lambda_v \cdot P_{new}$. First, notice that λ_v divides λ^{hist}_v, since by construction $G = (V, E)$ is a subgraph of $G^{hist} = (V^{hist}, E^{hist})$. Moreover, the product P_{new} divides λ^{hist}_v, since each factor in P_{new} divides λ^{hist}_v. This is due to the fact that, by construction of G^{hist}, class v does not access any class in the set $\mathsf{NewClasses} = \{u^{new} : u \in \mathsf{AssignNewKeys}\}$ (there are no edges from V to NewClasses in G^{hist}). Therefore, for any $v \in V' \setminus \mathsf{AssignNewKeys}$, it holds that $\lambda^{hist}_v = \lambda_v \cdot P_{new}$ and the lemma follows. □

Consider a class in $u \in \mathsf{AssignNewKeys}$ and the corresponding class $u^{new} \in \mathsf{NewClasses}$. Let k^{hist}_u and $k^{hist}_{u^{new}}$ be the keys assigned to classes u and u^{new} at step 9. of the algorithm *Upd*, respectively. Notice that, since $u \in \mathsf{AssignNewKeys}$, then at step 11. of the algorithm *Upd*, class u is assigned the key $k'_u = k^{hist}_{u^{new}}$, whereas k^{hist}_u can be considered as the *old key* for class u. Which are the relationships between k^{hist}_u and $k^{hist}_{u^{new}}$? Since the set E^{hist} does not contain edges of the form (v, u^{new}), with $v \in V$, it follows the key assigned to each class in V cannot be used to compute the key assigned to u^{new} in G^{hist}. In particular, the key k^{hist}_u cannot be used to compute $k^{hist}_{u^{new}}$. Thus, the use of the historical graph G^{hist} in the algorithm *Upd* allows to realize the so called *forward secrecy*, meaning that a class cannot use its old key to compute keys which will be subsequently assigned to it.

6.1. Construction of the set AssignNewKeys

In order to conclude the description of the algorithm *Upd* for the Akl–Taylor assignment supporting dynamic updates, we need to discuss how to construct the set AssignNewKeys for each update type. We consider the following update types: *insertion of an edge*, *insertion of a class*, *deletion of an edge*, *deletion of a class*, *key replacement*, and *revocation of a user from a class*. Notice that some types of updates can be seen as a sequence of other types of updates. For example, the insertion of a class u can be performed by executing a sequence of edge insertions, one for each edge ingoing u and outgoing from u. Similarly, the deletion of a class u can be performed by executing a sequence of edge deletions, one for each edge ingoing u and outgoing from u. On the other hand, a key replacement for a class u can be performed by first executing the deletion of u followed by the insertion of a new class u'. Finally, the revocation of a user from a class u can be performed by executing a sequence of key replacement operations. In the following we show how to construct the set AssignNewKeys for each type of updates. For each class $v \in V$, we denote by I^G_v (resp. $I^{G'}_v$) the set of classes which can access v in G (resp. G'), i.e., $I^G_v = \{z \in V : v \in A^G_z\}$ (resp. $I^{G'}_v = \{z \in V' : v \in A^{G'}_z\}$).

- **Insertion of an edge.** Let u and v be two classes in V such that $(u, v) \notin E$. Which are the consequences of the insertion of the edge (u, v) in the graph $G' = (V', E')$? It is easy to see that such an update modifies the accessible set A^G_z of any class $z \in I^G_u$, since all classes in A^G_v needs to be included, if they were not already present in A^G_z, due to the existence of a different path from z to v in G. Consequently, the public values λ_z assigned to such classes in G' could contain a smaller number of primes than in G, resulting in the need of a key replacement for such classes. Therefore, the set AssignNewKeys can be constructed by considering the above discussion. Notice that a different, but equivalent, description of AssignNewKeys consists of the set of classes which in G' have access to at least a class in the set $I^{G'}_v \setminus I^G_v$.
- **Insertion of a class.** Let $u \notin V$ be a class to be inserted in the graph G', along with new incoming and outgoing edges. Such an update can be seen as a composition of edge insertions, considering each edge ingoing u and outgoing from u as a separate update. Consequently, the set AssignNewKeys can be determined as explained for the case of edge insertions.
- **Deletion of an edge.** Let u and v be two classes in V such that $(u, v) \in E$. The deletion of the edge (u, v) from the graph G clearly requires a key replacement for class v in G'. Consequently, the keys assigned to all classes in $I^{G'}_v$ needs to be updated, resulting in $\mathsf{AssignNewKeys} = I^{G'}_v$.
- **Deletion of a class.** Let $u \in V$ be a class to be deleted in the graph G, along with its incoming and outgoing edges, thus yielding to the graph G'. This update requires to follow the above described procedure for edge deletion. Moreover, in order to preserve the partial order relation between the classes, such an update also requires to insert a new edge between each predecessor and each successor of the deleted class u. Consequently, the set AssignNewKeys can be determined as explained for the case of edge deletions and edge insertions.

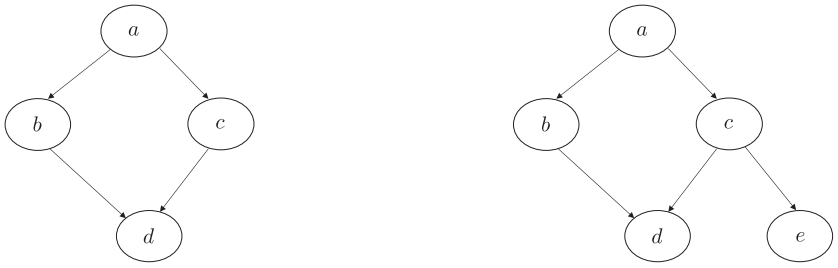

Fig. 5. Two partially ordered hierarchies.

- **Key replacement.** Let u be a class in G whose key k_u needs to be replaced, due either to a problem of loss of misuse. Such an update can be seen as a sequence of two updates, that is, the deletion of the class u followed by the insertion of a class u'. Consequently, the set AssignNewKeys can be determined as explained for the case of class deletions and class insertions.
- **User revocation.** Let u be a class in G, containing a certain number of users which share the same access rights. Whenever a user in u needs to be revoked, in order to avoid the so called *ex-member problem*, a *key replacement update* for each class $v \in A_u^{G'}$ is needed. Consequently, the set AssignNewKeys can be determined as explained for the case of key replacements.

Example 1 (Insertion of an edge)**.** Consider the hierarchy on the left-hand side of Fig. 3 and let p_a, p_b, p_c, and p_d be the primes associated to the classes a, b, c, and d, respectively. The public values associated to the classes are as follows:

$$\lambda_a = 1, \quad \lambda_b = p_a \cdot p_c \cdot p_d, \quad \lambda_c = p_a \cdot p_b, \quad \lambda_d = p_a \cdot p_b \cdot p_c,$$

whereas, the private keys associated to the classes are as follows:

$$k_a = k_0, \quad k_b = (k_0)^{p_a \cdot p_c \cdot p_d}, \quad k_c = (k_0)^{p_a \cdot p_b}, \quad k_d = (k_0)^{p_a \cdot p_b \cdot p_c}.$$

Suppose the edge (b, d) has been added to the hierarchy, as shown on the right-hand side of Fig. 3. Then, AssignNewKeys $= \{a, b\}$. Let p_a^{new} and p_b^{new} be the new primes assigned to the classes in AssignNewKeys. In order to reflect the addition of the edge (b, d), consider the *historical graph* depicted in Fig. 4. The TA updates the public values as follows:

$$\lambda_a^{hist} = p_{a^{new}} \cdot p_{b^{new}}, \qquad \lambda_b^{hist} = p_a \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{b^{new}} \qquad \lambda_c^{hist} = p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{b^{new}},$$
$$\lambda_d^{hist} = p_a \cdot p_b \cdot p_c \cdot p_{a^{new}} \cdot p_{b^{new}}, \quad \lambda_{a^{new}}^{hist} = p_a \cdot p_b, \qquad \lambda_{b^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_{a^{new}}.$$

The updated value of the global starting value is $k_0^{hist} = k_0^{\frac{1}{p_{a^{new}} \cdot p_{b^{new}}}}$ and the private keys associated to the classes of the hierarchy in Fig. 4 are as follows:

$$k_a^{hist} = (k_0^{hist})^{p_{a^{new}} \cdot p_{b^{new}}} = k_0 \qquad k_b^{hist} = (k_0^{hist})^{p_a \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{b^{new}}} = (k_0)^{p_a \cdot p_c \cdot p_d},$$
$$k_c^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{b^{new}}} = (k_0)^{p_a \cdot p_b}, \quad k_d^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_{a^{new}} \cdot p_{b^{new}}} = (k_0)^{p_a \cdot p_b \cdot p_c},$$
$$k_{a^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b} = (k_0)^{\frac{p_a \cdot p_b}{p_{a^{new}} \cdot p_{b^{new}}}}, \qquad k_{b^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_{a^{new}}} = (k_0)^{\frac{p_a \cdot p_b \cdot p_c}{p_{b^{new}}}}.$$

From the above assignment, it is easy to see that classes c and d maintain their old keys.

Example 2 (Insertion of a class)**.** Consider the hierarchy on the left-hand side of Fig. 5 and let p_a, p_b, p_c, and p_d be the primes associated to the classes a, b, c, and d, respectively. The public values associated to the classes are as follows:

$$\lambda_a = 1, \quad \lambda_b = p_a \cdot p_c, \quad \lambda_c = p_a \cdot p_b, \quad \lambda_d = p_a \cdot p_b \cdot p_c,$$

whereas, the private keys associated to the classes are as follows:

$$k_a = k_0, \quad k_b = (k_0)^{p_a \cdot p_c}, \quad k_c = (k_0)^{p_a \cdot p_b}, \quad k_d = (k_0)^{p_a \cdot p_b \cdot p_c}.$$

Suppose the class e has been added to the hierarchy, as shown on the right-hand side of Fig. 5. Then, AssignNewKeys $= \{a, c, e\}$. Let $p_{a^{new}}$, $p_{c^{new}}$, and $p_{e^{new}}$ be the new prime numbers assigned to the classes in AssignNewKeys. In order to reflect the addition of the class e, consider the *historical graph* depicted in Fig. 6.

The TA updates the public values as follows:

$$\lambda_a^{hist} = p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}, \qquad \lambda_b^{hist} = p_a \cdot p_c \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}, \qquad \lambda_c^{hist} = p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}},$$
$$\lambda_d^{hist} = p_a \cdot p_b \cdot p_c \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}, \quad \lambda_{a^{new}}^{hist} = p_a \cdot p_c, \qquad \lambda_{c^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_{a^{new}},$$
$$\lambda_{e^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{c^{new}}.$$

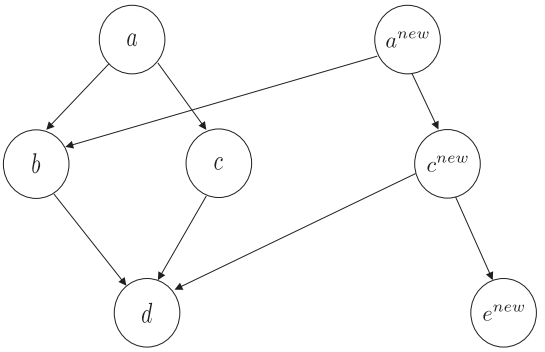

Fig. 6. The historical graph corresponding to the hierarchies in Fig. 5.

Fig. 7. Two partially ordered hierarchies.

The updated value of the global starting value is $k_0^{hist} = k_0^{\frac{1}{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}}}$ and the private keys associated to the classes of the hierarchy in Fig. 6 are as follows:

$$k_a^{hist} = (k_0^{hist})^{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}} = k_0, \qquad k_b^{hist} = (k_0^{hist})^{p_a \cdot p_c \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_c},$$
$$k_c^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_b}, \qquad k_d^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_b \cdot p_c},$$
$$k_{a^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_c} = k_0^{\frac{p_a \cdot p_c}{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{e^{new}}}}, \qquad k_{c^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_{a^{new}}} = k_0^{\frac{p_a \cdot p_b \cdot p_c}{p_{c^{new}} \cdot p_{e^{new}}}},$$
$$k_{e^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{c^{new}}} = (k_0)^{\frac{p_a \cdot p_b \cdot p_c \cdot p_d}{p_{e^{new}}}}.$$

From the above assignment, it is easy to see that classes b and d maintain their old keys.

Example 3 (Deletion of an edge)**.** Consider the hierarchy on the left-hand side of Fig. 7 and let p_a, p_b, p_c, p_d, and p_e be the primes associated to the classes a, b, c, d, and e, respectively. The public values associated to the classes are as follows:

$$\lambda_a = 1, \quad \lambda_b = p_a \cdot p_c \cdot p_e, \quad \lambda_c = p_a \cdot p_b, \quad \lambda_d = p_a \cdot p_b \cdot p_c \cdot p_e, \quad \lambda_e = p_a \cdot p_b \cdot p_c \cdot p_d$$

whereas, the private keys associated to the classes are as follows:

$$k_a = k_0, \quad k_b = (k_0)^{p_a \cdot p_c \cdot p_e}, \quad k_c = (k_0)^{p_a \cdot p_b}, \quad k_d = (k_0)^{p_a \cdot p_b \cdot p_c \cdot p_e}, \quad k_e = (k_0)^{p_a \cdot p_b \cdot p_c \cdot p_d}.$$

Suppose the edge (b, d) has been deleted by the hierarchy as shown on the right-hand side of Fig. 7. Then, AssignNewKeys $= \{a, c, d\}$. Let $p_{a^{new}}$, $p_{c^{new}}$, and $p_{d^{new}}$ be the new prime numbers assigned to the classes in AssignNewKeys. In order to reflect the deletion of the edge (b, d), consider the *historical graph* depicted in Fig. 8.

The TA updates the public values as follows:

$$\lambda_a^{hist} = p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}, \qquad \lambda_b^{hist} = p_a \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}},$$
$$\lambda_c^{hist} = p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}, \qquad \lambda_d^{hist} = p_a \cdot p_b \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}},$$
$$\lambda_e^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}, \qquad \lambda_{a^{new}}^{hist} = p_a \cdot p_c,$$
$$\lambda_{c^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}}, \qquad \lambda_{d^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}}.$$

The updated value of the global starting value is $k_0^{hist} = k_0^{\frac{1}{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}}}$ and the private keys associated to the classes of the hierarchy in Fig. 8 are as follows:

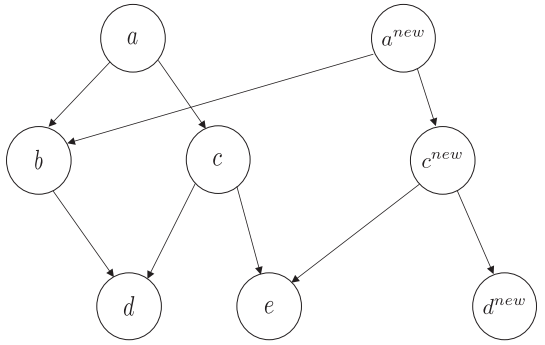

Fig. 8. The historical graph corresponding to the hierarchies in Fig. 7.

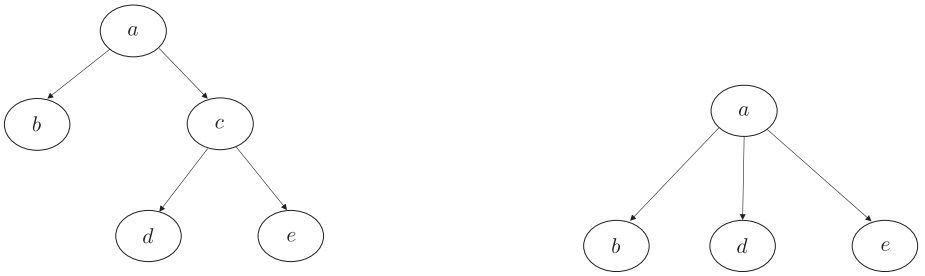

Fig. 9. Two partially ordered hierarchies.

$$k_a^{hist} = (k_0^{hist})^{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}} = k_0, \qquad k_b^{hist} = (k_0^{hist})^{p_a \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}} = k_0^{p_a \cdot p_c \cdot p_e},$$

$$k_c^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}} = k_0^{p_a \cdot p_b}, \qquad k_d^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}} = k_0^{p_a \cdot p_b \cdot p_c \cdot p_e},$$

$$k_e^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}} = k_0^{p_a \cdot p_b \cdot p_c \cdot p_d}, \qquad k_{a^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_c} = k_0^{\frac{p_a \cdot p_c}{p_{a^{new}} \cdot p_{c^{new}} \cdot p_{d^{new}}}},$$

$$k_{c^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}}} = k_0^{\frac{p_a \cdot p_b \cdot p_c \cdot p_d}{p_{c^{new}} \cdot p_{d^{new}}}}, \qquad k_{d^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{c^{new}}} = k_0^{\frac{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e}{p_{d^{new}}}}$$

From the above assignment, it is easy to see that classes b and d maintain their old key.

Example 4 (Deletion of a class)**.** Consider the hierarchy on the left-hand side of Fig. 9 and let p_a, p_b, p_c, p_d, and p_e be the primes associated to the classes a, b, c, d, and e, respectively. The public values associated to the classes are as follows:

$$\lambda_a = 1, \quad \lambda_b = p_a \cdot p_c \cdot p_d \cdot p_e, \quad \lambda_c = p_a \cdot p_b, \quad \lambda_d = p_a \cdot p_b \cdot p_c \cdot p_e, \quad \lambda_e = p_a \cdot p_b \cdot p_c \cdot p_d$$

whereas, the private keys associated to the classes are as follows:

$$k_a = k_0, \quad k_b = (k_0)^{p_a \cdot p_c \cdot p_d \cdot p_e}, \quad k_c = (k_0)^{p_a \cdot p_b}, \quad k_d = (k_0)^{p_a \cdot p_b \cdot p_c \cdot p_e}, \quad k_e = (k_0)^{p_a \cdot p_b \cdot p_c \cdot p_d}.$$

Suppose the class c has been deleted by the hierarchy, as shown on the right-hand side of Fig. 9. Then, AssignNewKeys $= \{a, d, e\}$. Let $p_{a^{new}}$, $p_{d^{new}}$ and $p_{e^{new}}$ be the new primes assigned to classes in AssignNewKeys. In order to reflect the deletion of the class c, consider the *historical graph* depicted in Fig. 10.

The TA updates the public values as follows:

$$\lambda_a^{hist} = p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}, \qquad \lambda_b^{hist} = p_a \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}},$$

$$\lambda_c^{hist} = p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}, \qquad \lambda_d^{hist} = p_a \cdot p_b \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}},$$

$$\lambda_e^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}, \qquad \lambda_{a^{new}}^{hist} = p_a \cdot p_c \cdot p_d \cdot p_e,$$

$$\lambda_{d^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{e^{new}}, \qquad \lambda_{e^{new}}^{hist} = p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}}.$$

The updated value of the global starting value is $k_0^{hist} = k_0^{\frac{1}{p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}}}$ and the private keys associated to the classes of the hierarchy in Fig. 10 are as follows:

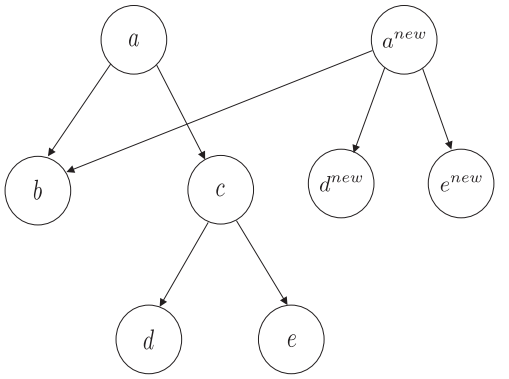

Fig. 10. The historical graph corresponding to the hierarchies in Fig. 9.

$$k_a^{hist} = (k_0^{hist})^{p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}} = k_0, \qquad k_b^{hist} = (k_0^{hist})^{p_a \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_c \cdot p_d \cdot p_e},$$

$$k_c^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_b}, \qquad k_d^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_b \cdot p_c \cdot p_e},$$

$$k_e^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}} = k_0^{p_a \cdot p_b \cdot p_c \cdot p_d}, \qquad k_{a^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_c \cdot p_d \cdot p_e} = k_0^{\frac{p_a \cdot p_c \cdot p_d \cdot p_e}{p_{a^{new}} \cdot p_{d^{new}} \cdot p_{e^{new}}}},$$

$$k_{d^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{e^{new}}} = k_0^{\frac{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e}{p_{d^{new}}}}, \qquad k_{e^{new}}^{hist} = (k_0^{hist})^{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e \cdot p_{a^{new}} \cdot p_{d^{new}}} = (k_0)^{\frac{p_a \cdot p_b \cdot p_c \cdot p_d \cdot p_e}{p_{e^{new}}}}.$$

From the above assignment, it is easy to see that class b maintains its old key.

7. Proving the security of the Akl–Taylor scheme supporting dynamic updates

In this section we show that the Akl–Taylor assignment supporting dynamic updates is secure against key recovery, provided that *also the new primes* associated to the classes, as a consequence of each update, are properly chosen.

Let $(up^1, \ldots, up^m)$ be a sequence of m updates performed on the graph $G = (V, E)$ by using the algorithm *Upd*. This yields to the sequence of graphs $(G^0, G^1, \ldots, G^m)$, where $G^0 = G$ and, for each $j = 1, \ldots, m$, $G^j = (V^j, E^j)$ is the graph resulting from the j-th update. Let $\mathsf{AssignNewKeys}^0 = V$ and, for each $j = 1, \ldots, m$, let

$$\mathsf{AssignNewKeys}^j \leftarrow \mathit{CreateSet}(G^{j-1}, up^j).$$

Moreover, for each $j = 0, \ldots, m$, let

$$\alpha_j = \sum_{i=0}^{j} |\mathsf{AssignNewKeys}^i|.$$

Let $G^{hist_0} = G$ and for each $j = 1, \ldots, m$, consider the *j-th historical graph* $G^{hist_j} = (V^{hist_j}, E^{hist_j})$, defined as follows:

- $V^{hist_j} = V^{hist_{j-1}} \cup \mathsf{NewClasses}^j$, where $\mathsf{NewClasses}^j = \{u^{new_j} : u \in \mathsf{AssignNewKeys}^j\}$;
- $E^{hist_j} = E^{hist_{j-1}} \cup E^{\mathsf{NewClasses}^j}$, where $E^{\mathsf{NewClasses}^j}$ is defined as follows:
  - for each class $u^{new_j} \in \mathsf{NewClasses}^j$, if there exists a class $v \in V^j$ such that $(u, v) \in E^j$ and $v^{new_j} \notin \mathsf{NewClasses}^j$, then $(u^{new_j}, v) \in E^{\mathsf{NewClasses}^j}$;
  - for each class $u^{new_j} \in \mathsf{NewClasses}^j$, if there exists a class $v^{new_j} \in \mathsf{NewClasses}^j$ such that $(u, v) \in E^j$, then $(u^{new_j}, v^{new_j}) \in E^{\mathsf{NewClasses}^j}$.

It is easy to see that the j-th historical graph G^{hist_j}, which is defined in terms of both the $(j-1)$-th historical graph $G^{hist_{j-1}}$ and the set $\mathsf{AssignNewKeys}^j$, carries information on the sequence $(up^1, \ldots, up^j)$ of updates on G. Notice that if we set $j = 1$, we obtain the construction of the historical graph for the single update described in the previous section. For example, let G be the initial graph on the left hand side of Fig. 11, and assume the TA performs the sequence of updates (*insert edge* (b, d), *insert class* e), yielding to the other two graphs depicted in Fig. 11. The corresponding graph G^{hist_2} is depicted in Fig. 12.

Let G^{hist_m} be the m-th historical graph obtained by the sequence $(up^1, \ldots, up^m)$ of m updates on the graph G by using the algorithm *Upd*. It is easy to see that total number of primes required to perform the sequence of m updates is equal to

$$\alpha_m = \sum_{j=0}^{m} |\mathsf{AssignNewKeys}^j| = \mathcal{O}(|V| \cdot m).$$

Indeed, each class in the m-th historical graph G^{hist_m} is assigned a distinct prime. Such primes are then used for the creation of the new public values.

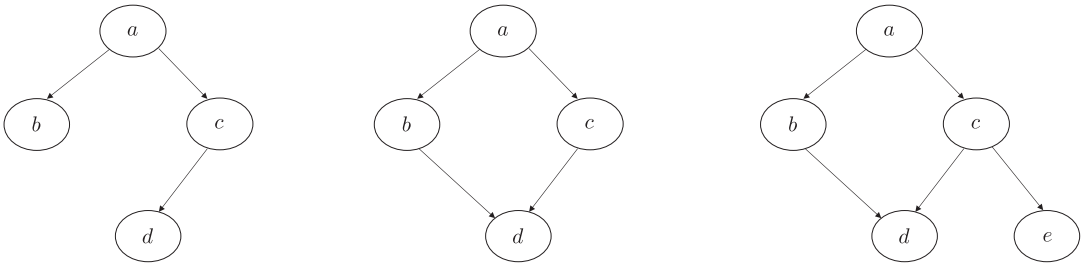

Fig. 11. A sequence of graphs.

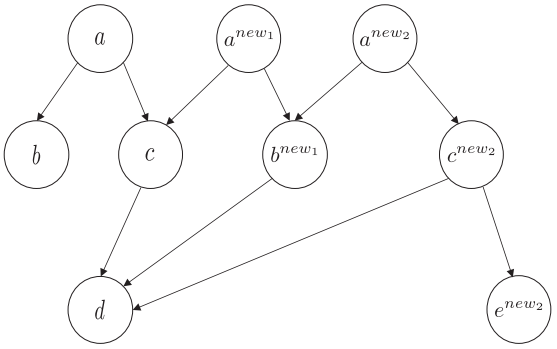

Fig. 12. The graph G^{hist_2} corresponding to the three hierarchies in Fig. 11.

In the following we show that the use of the m-th historical graph G^{hist_m} allows to simplify the attacking scenario, since the roles of *old keys* and that of *private information of corrupted classes* can be unified. Indeed, consider the experiment $\mathbf{Exp}^{\mathrm{REC-DYN}}_{\mathrm{ADAPT}}(1^\tau, G)$ of Definition 2. Let ADAPT be an adaptive adversary which chooses to attack the class u in the graph G^t. Let $1 \leq j \leq t$ be an index such that class u belongs to the set $\mathsf{AssignNewKeys}^j$, i.e., the corresponding key k_u^j has been created as a consequence of the j-th update. Consider the j-th historical graph G^{hist_j}, which is a subgraph of G^{hist_m}: there exists a class u^{new_j} which has been inserted in G^{hist_j}, but was not present in $G^{hist_{j-1}}$, such that no classes in $G^{hist_{j-1}}$ have edges ending in u^{new_j}. Thus, all classes in $G^{hist_{j-1}}$ can be seen as *corrupted classes* for u^{new_j}. In particular, notice that also class $u^{new_{j-1}}$, if present in $G^{hist_{j-1}}$, is considered to be corrupted for u^{new_j}. On the other hand, the key assigned to class u in G^{j-1} belongs to the sequence of *old keys* which have been modified as a consequence of an update. Clearly, when considering the m-th historical graph G^{hist_m}, the set of corrupted classes for u^{new_j} previously considered has to be augmented in order to contain all other classes which have been inserted as a consequence of subsequent updates and which are not allowed to compute the key held by class u in the graph G^t. To summarize, the use of the m-th historical graph G^{hist_m} allows to simplify the attacking scenario, since the roles of *old keys* and that of *private information of corrupted classes* can be unified.

For example, consider the sequence of graphs depicted in Fig. 11, and assume the adaptive adversary ADAPT chooses to attack the class b in the last graph. Notice that the key for such a class has been modified after the first update, i.e., the insertion of the edge (b, d) in the initial graph G, and has not been changed after the second update, i.e., the insertion of class e in G_1, yielding to the last graph G_2. In terms of the graph G^{hist_2} depicted in Fig. 12, this corresponds to considering the class b^{new_1}, for which all classes in the initial graph G can be seen as *corrupted*. In particular, notice that also class b in G^{hist_2} is considered to be corrupted for b^{new_1}. Moreover, also the classes c^{new_2} and e^{new_2}, inserted during the construction of G^{hist_2} from G^{hist_1}, have to be considered corrupted for b^{new_1}.

When performing a sequence of updates on the graph G by using the algorithm *Upd*, the global starting value used in each assignment depends on the global starting value k_0 used by the Akl–Taylor scheme, as well as on all the new primes introduced by each update. For each $j = 1, \ldots, m$, let

$$P_{new_j} = \prod_{v \in \mathsf{AssignNewKeys}^j} p_{v^{new_j}}.$$

Moreover, for each $j = 1, \ldots, m$, let $\beta_j = \prod_{i=1}^{j} P_{new_i}$ and let $k_0^{hist_j}$ be the global starting value used by the algorithm *Upd* for the assignment on the graph G^j. The following lemma holds.

Lemma 3. *Let $G = (V, E)$ be a partially ordered hierarchy and let m be the number of updates performed by the TA by using the Akl–Taylor scheme with the algorithm Upd. For each $j = 1, \ldots, m$, it holds that*

$$k_0^{hist_j} = k_0^{1/\beta_j}.$$

7.1. Assigning primes with the fixed primes choice

In the following we consider the fixed primes choice for the Akl–Taylor assignment supporting dynamic updates. Given a sequence of m updates $(up^1, \ldots, up^m)$ on a graph G, for any $j = 1, \ldots, m$, the assignment of primes to the classes in the set $\mathsf{AssignNewKeys}^j$ is done as follows:

- Let $v_1, \ldots, v_{|\mathsf{AssignNewKeys}^j|}$ be a topological sorting of the classes in $\mathsf{AssignNewKeys}^j$;
- For each $i = 1, \ldots, |\mathsf{AssignNewKeys}^j|$, associate prime $p_{\alpha_{j-1}+i} \in PRIMES_{\alpha_j} \setminus PRIMES_{\alpha_{j-1}}$ to class v_i.

Now we are ready to prove our main result.

Theorem 3. *The Akl–Taylor assignment with the fixed primes choice yields to an Akl–Taylor scheme supporting m updates which is secure in the sense of REC−DYN−AD under the RSA assumption for exponents in* $PRIMES_{\alpha_m}$.

Proof. Assume by contradiction that the Akl–Taylor assignment supporting m dynamic updates, when all the primes are chosen according to the fixed primes choice, is not secure in the sense of REC−DYN−AD. Thus, there exist a graph $G \in \Gamma$ and a polynomial-time adaptive adversary $\mathtt{ADAPT} = (\mathtt{ADAPT}_1, \mathtt{ADAPT}_2)$ whose advantage $\mathbf{Adv}^{\mathrm{REC-DYN}}_{\mathtt{ADAPT}}(1^\tau, G)$ is non-negligible. We show how to turn ADAPT into a polynomial-time adversary B which breaks the RSA assumption for exponents in $PRIMES_{\alpha_m}$.

For each $j = 1, \ldots, \alpha_m$, denote by p_j and k_j the j-th prime in the set $PRIMES_{\alpha_m}$ and the j-th key which either has been created or has been modified by the Akl–Taylor assignment supporting m dynamic updates, respectively. Moreover, for each $j = 1, \ldots, \alpha_m$, let $\mathtt{S}_j$ be an adversary which behaves as $\mathtt{ADAPT}_1$ until it outputs the triple $(u, t, history)$ (recall that this means that the adversary $\mathtt{ADAPT}_1$ has chosen to attack the class u in the graph G^t). If key k_u^t associated to the class u in G^t is equal to k_j, then $\mathtt{S}_j$ continues to follow $\mathtt{ADAPT}_2$, otherwise it outputs the string 0^τ. The advantage of ADAPT can be written as

$$\mathbf{Adv}^{\mathrm{REC-DYN}}_{\mathtt{ADAPT}}(1^\tau, G) \leq \sum_{j=1}^{\alpha_m} Pr[\mathtt{ADAPT}_1 \text{ chooses } k_j \text{ as the key to be computed}] \cdot \mathbf{Adv}^{\mathrm{REC-DYN}}_{\mathtt{S}_j}(1^\tau, G).$$

Since $\mathbf{Adv}^{\mathrm{REC-DYN}}_{\mathtt{ADAPT}}(1^\tau, G)$ is non-negligible, then there exists at least an index h, where $1 \leq h \leq \alpha_m$ such that $\mathbf{Adv}^{\mathrm{REC-DYN}}_{\mathtt{S}_h}(1^\tau, G)$ is non-negligible. In particular, in the first stage of the attack, $\mathtt{S}_h$ outputs the triple $(u, t, history)$, whereas, in the second stage, $\mathtt{S}_h$ outputs the key k_h associated to the class u in the graph G^t. Let $e \in PRIMES_{\alpha_m}$ and assume, w.l.o.g., that $e = p_h$. We construct a polynomial-time adversary B that, on input a triple (n, e, y), where n is chosen according to the *RSA generator* $\mathcal{K}^{fix}_{RSA}(1^\tau, e)$, uses the adversary $\mathtt{S}_h$ to compute, with non-negligible advantage, a value $x \in Z_n^*$ such that $y = x^e \bmod n$. □

We distinguish the two following cases:

- **Case 1:** $h \geq |V| + 1$. This case corresponds to the scenario where the key k_h chosen by the adversary has been created as a consequence of an update.
- **Case 2:** $1 \leq h \leq |V|$. This case corresponds to the scenario where the key k_h chosen by the adversary has been assigned to some class in the initial graph G.

Analysis of case 1. Assume that the key k_h chosen by the adversary either has been created or has been modified by the t-th update operation, which has assigned such a key to a certain class u in the graph G^t. Therefore, $u \in \mathsf{AssignNewKeys}^t$ and e is the new prime associated to class u in the t-th historical graph G^{hist_t}. Let $u_1, \ldots, u_{|V|}$ be a topological sorting of the classes in V. The adversary B, on inputs (n, e, y) first computes the inputs for $\mathtt{S}_h$ as follows:

1. For each $j = 1, \ldots, |V|$, assign a prime $p_j \in PRIMES_{|V|}$ to the class u_j;
2. For each $j = 1, \ldots, |V|$, compute the integer $\lambda_{u_j} = \prod_{w \notin A^G_{u_j}} p_w$ ($\lambda_{u_j} = 1$ if $A^G_{u_j} = V$);
3. Set pub to be the sequence of integers computed in the previous step, along with the value n.

Then, adversary B calls $\mathtt{S}_h$ on inputs 1^τ, G, and pub. In the first stage of the attack, adversary $\mathtt{S}_h$ can make a polynomial number of *updating* and *corrupting queries*. The responses to such queries will be saved into the *history* variable.

- Each *updating query* will be answered by adversary B through a simulation of the algorithm *Upd* and will produce the updated public information along with the sequence of keys which need to be replaced. Let $(up^1, \ldots up^m)$ be the sequence of updates requested by $\mathtt{S}_h$. We will show how adversary B constructs pub^j and old_k^{j-1}, for any $j = 1, \ldots, m$.
  Construction of the sequence pub^j. In order to answer the j-th updating query, adversary B needs to assign some new prime numbers to the set $\mathsf{AssignNewKeys}^j$. This is done by using the fixed primes choice. The public information pub^j, containing the value λ_v^j for each class v in G^j, can be easily computed by adversary B by using the algorithm *Upd*.
  Construction of the sequence old_k^{j-1}. The sequence old_k^{j-1} will contain the keys which have been modified as a sequence of the update up^j, i.e., the key $k_v^{j-1} = (k_0^{hist_{j-1}})^{\lambda_v^{j-1}}$, for each class $v \in \mathsf{AssignNewKeys}^j$. In the following we show that, if adversary B knows the value $\beta_t = \prod_{i=1}^t P_{new_i}$, then it can easily compute the above keys. Indeed, setting the global starting value k_0 to be equal to $y^{\beta_t/e}$, where (n, e, y) is the input for B and using Lemma 3, for any $j = 1, \ldots, t-1$, it follows that

$$k_0^{hist_{j-1}} = k_0^{\frac{1}{\beta_{j-1}}} = y^{\frac{\beta_t}{e \cdot \beta_{j-1}}}.$$

Notice that adversary B can efficiently perform the above computation, since the product $e \cdot \beta_{j-1}$ divides β_t. This follows from the fact that β_{j-1} is contained in β_t and e is the prime associated to the class $u \in \mathsf{AssignNewKeys}^t$. The value $k_0^{hist_{j-1}}$ is then used to compute the key k_v^{j-1}, for each class $v \in \mathsf{AssignNewKeys}^j$.
Thus, if adversary B knows the value β_t, it can compute all keys in the sequence old_k^{j-1}. Notice that, since the fixed primes choice is used, then $\beta_t = \prod_{\ell=1}^{\alpha_t} p_\ell$, i.e., it is equal to the product of the first α_t primes. Thus, the value β_t can be easily constructed if B knows the value α_t. Since such a value is not known in advance, B makes its guess by choosing uniformly at random a value between 1 and α_m, but it might fail. Indeed, if the chosen value does not correspond to α_t, then B cannot continue its simulation and needs to restart itself. We conclude that B succeeds in constructing the sequence old_k^{j-1} with probability $1/\alpha_m$.

- Let v be a *corrupting query* made by $\mathtt{S}_h$ to the corrupting oracle $\mathcal{C}^i$, where $1 \leq i \leq t$. As before, if adversary B knows the value β_t, then it can compute the answer to such a query, consisting of the key k_v^j held by class v in the graph G^j, for any $j = 1, \ldots, i$ such that v cannot access u in G^j. Indeed, setting setting the global starting value k_0 to be equal to $y^{\beta_t/e}$, and using Lemma 3, it follows that $k_v^j = ((k_0^{hist_j}))^{\lambda_v^j} = y^{\frac{\beta_t}{e \cdot \beta_j}}$. Notice that such a key can be computed since the product $e \cdot \beta_j$ divides β_t. Using the same arguments discussed above, we conclude that B succeeds in answering corrupting queries with probability $1/\alpha_m$.
- Upon finishing its updating and corrupting queries, in the first stage of the attack, adversary $\mathtt{S}_h$ outputs the triple (u, t, *history*), where *history* contains all the responses obtained by the interaction with B.

In the second stage of the attack, adversary $\mathtt{S}_h$, on inputs the triple (u, t, *history*), outputs the key k_h assigned to the class u in the graph G^t. Since the global starting value $k_0^{hist_t}$ is equal to $y^{1/e}$, it follows that the key k_h computed by adversary $\mathtt{S}_h$ is equal to

$$k_h = (k_0^{hist_t})^{\lambda_u^t} = y^{\lambda_u^t/e}.$$

Notice that $gcd(\lambda_u^t, e) = 1$. Indeed, since e and λ_u^t are the prime number and the public value assigned to class u in the graph G^t, respectively, it follows that e does not divide λ_u^t. Therefore, adversary B can use the *Extended Euclidean Algorithm* to compute integers a and b such that

$$e \cdot a + \lambda_u^t \cdot b = 1.$$

Then, B computes

$$y^a \cdot k_h^b \bmod n = x^{e \cdot a} \bmod n \cdot x^{\lambda_u^t \cdot b} \bmod n = x^{e \cdot a + \lambda_u^t \cdot b} \bmod n = x.$$

Notice that adversary $\mathtt{S}_h$'s view in the above simulation is *identically distributed* as the one obtained in a real execution of the scheme with the fixed primes choice, and all the computations needed to construct such a view can be performed in polynomial time. Finally, it is easy to see that $\mathbf{Adv}_B^{\mathcal{K}^{fix}_{RSA}}(1^\tau, e) = 1/\alpha_m \cdot \mathbf{Adv}_{\mathtt{S}_h}^{\mathtt{REC-DYN}}(1^\tau, G)$. Since $\mathbf{Adv}_{\mathtt{S}_h}^{\mathtt{REC-DYN}}(1^\tau, G)$ is non-negligible, the theorem holds.

Analysis of case 2. The proof is similar to the one of the previous case.

Following the lines of Theorems 2 and 1, we can also prove that the Akl–Taylor assignment with the random primes choice yields to a dynamic Akl–Taylor scheme which is secure in the sense of REC-DYN-AD under the random exponent RSA assumption.

Theorem 4. *The Akl–Taylor assignment with the random primes choice yields to a dynamic Akl–Taylor scheme which is secure in the sense of REC-DYN-AD under the random exponent RSA assumption.*

Similar results can be also proven for the MacKinnon et al. assignment.

8. Conclusions

Cloud storage repositories provide extremely convenient options for large scale outsourcing of shareable data, but, despite the potential benefits introduced by the cloud paradigm, the security of the data stored in the cloud still constitutes a serious obstacle to its wide deployment, mainly in scenarios where sensitive information is involved. Aiming at addressing the security and privacy issues in outsourced data storage, we present an efficient solution, based on the Akl–Taylor scheme, for flexible and fine-grained access control, supporting dynamic updates in cloud environments. Again, we specifically show how to manage changes to the hierarchy, in order to effectively perform updates, which are very common in storage clouds. We believe that the proposed scheme can serve as an efficient solution in such fully dynamic environments, since it requires a small amount of public information and allows key derivation to be directly performed in a single step. Finally, we demonstrate by means of a formal proof, that the proposed scheme is secure against key recovery, provided that also the new primes associated to the classes, as a consequence of each update, are properly chosen. More precisely, our construction is the first one proved secure with respect to the notion of REC-DYN-AD.

References

[1] A.V. Aho, M.R. Garey, J.D. Ullman, The transitive reduction of a directed graph, SIAM J. Comput. 1 (2) (1972) 131–137.
[2] S.G. Akl, P.D. Taylor, Cryptographic solution to a problem of access control in a hierarchy, ACM Trans. Comput. Syst. 1 (3) (1983) 239–248.
[3] M.J. Atallah, M. Blanton, N. Fazio, K.B. Frikken, Dynamic and efficient key management for access hierarchies, ACM Trans. Inf. Syst. Secur. 12 (3) (2009).
[4] M.J. Atallah, M. Blanton, K.B. Frikken, Key Management for non-tree access hierarchies, in: D.F. Ferraiolo, I. Ray (Eds.), SACMAT 2006,11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, California, USA, June 7–9, 2006, Proceedings, ACM, 2006, pp. 11–18.
[5] M.J. Atallah, M. Blanton, K.B. Frikken, Incorporating temporal capabilities in existing key management schemes, in: Computer Security – ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24–26, 2007, Proceedings, 2007, pp. 515–530, doi:10.1007/978-3-540-74835-9_34.
[6] M.J. Atallah, K.B. Frikken, M. Blanton, Dynamic and efficient key management for access hierarchies, in: V. Atluri, C. Meadows, A. Juels (Eds.), Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7–11, 2005, ACM, 2005, pp. 190–202.
[7] G. Ateniese, A. De Santis, A.L. Ferrara, B. Masucci, Provably-secure time-bound hierarchical key assignment schemes, in: ACM CCS 2006, 2006, pp. 288–297.
[8] G. Ateniese, A. De Santis, A.L. Ferrara, B. Masucci, Provably-Secure time-Bound hierarchical key assignment schemes, J. Cryptology 25 (2) (2012) 1–15.
[9] E. Bertino, N. Shang, S.S.W. Jr., An efficient time-bound hierarchical key management scheme for secure broadcasting, IEEE Trans. Dependable Sec. Comput. 5 (2) (2008) 65–70.
[10] J. Bethencourt, A. Sahai, B. Waters, Ciphertext-policy attribute-based encryption, in: Security and Privacy, 2007. SP'07. IEEE Symposium on, IEEE, 2007, pp. 321–334.
[11] M. Cafaro, R. Civino, B. Masucci, On the equivalence of two security notions for hierarchical key assignment schemes in the unconditional setting, IEEE Trans. Dependable Sec. Comput. 12 (4) (2015) 485–490.
[12] A. Castiglione, A. De Santis, B. Masucci, Hierarchical and Shared Key Assignment, in: 17th International Conference on Network-Based Information Systems, NBIS 2014, IEEE, 2014, pp. 263–270.
[13] A. Castiglione, A. De Santis, B. Masucci, Key indistinguishability versus strong key indistinguishability for hierarchical key assignment schemes, IEEE Trans. Dependable Sec. Comput. 13 (4) (2016) 451–460.
[14] A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, A. Castiglione, On the relations between security notions in hierarchical key assignment schemes for dynamic structures, in: Proceedings of the 21st Australasian Conference on Information Security and Privacy - ACISP 2016, Part II, Melbourne, Australia, Lecture Notes in Computer Science, Springer Verlag., vol. 9723, 2016, pp. 1–18.
[15] A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, A. Castiglione, X. Huang, Cryptographic hierarchical access control for dynamic structures, IEEE Trans. Inf. Forensics Secur. 11 (10) (2016b) 2349–2364.
[16] A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, A. Castiglione, J. Li, X. Huang, Hierarchical and shared access control, IEEE Trans. Inf. Forensics Secur. 11 (4) (2016a) 850–865.
[17] Q. Chai, G. Gong, Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers, in: Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10–15, 2012, IEEE, 2012, pp. 917–922.
[18] T. Chen, Y. Chung, Hierarchical access control based on chinese remainder theorem and symmetric algorithm, Comput. Secur. 21 (6) (2002) 565–570.
[19] Y.-R. Chen, C.-K. Chu, W.-G. Tzeng, J. Zhou, Cloudhka: a cryptographic approach for hierarchical access control in cloud computing, in: Applied Cryptography and Network Security, Springer, 2013, pp. 37–52.
[20] H. Chien, Efficient time-Bound hierarchical key assignment scheme, IEEE Trans. Knowl. Data Eng. 16 (10) (2004) 1301–1304.
[21] P. D'Arco, A. De Santis, A.L. Ferrara, B. Masucci, Security and Tradeoffs of the Akl-Taylor scheme and its variants, in: MFCS 2009, LNCS, vol. 5734, 2009, pp. 247–257.
[22] P. D'Arco, A. De Santis, A.L. Ferrara, B. Masucci, Variations on a theme by Akl and Taylor: security and tradeoffs, Theoretical Comp. Sci. 411 (2010) 213–227.
[23] A. De Santis, A.L. Ferrara, B. Masucci, Cryptographic key assignment schemes for any access control policy, Inf. Process. Lett. 92 (4) (2004) 199–205.
[24] A. De Santis, A.L. Ferrara, B. Masucci, Enforcing the security of a time-Bound hierarchical key assignment scheme, Inf. Sci. 176 (12) (2006) 1684–1694.
[25] A. De Santis, A.L. Ferrara, B. Masucci, Unconditionally secure key assignment schemes, Discrete Appl. Math. 154 (2) (2006) 234–252.
[26] A. De Santis, A.L. Ferrara, B. Masucci, Efficient provably-secure hierarchical key assignment schemes, in: MFCS 2007, LNCS, 4708, 2007, pp. 371–382.
[27] A. De Santis, A.L. Ferrara, B. Masucci, New constructions for provably-secure time-bound hierarchical key assignment schemes, in: SACMAT 2007, 12th ACM Symposium on Access Control Models and Technologies, Sophia Antipolis, France, June 20–22, 2007, Proceedings, 2007, pp. 133–138.
[28] A. De Santis, A.L. Ferrara, B. Masucci, New constructions for provably-Secure time-Bound hierarchical key assignment schemes, Theor. Comp. Sci. 407 (1–3) (2008) 213–230.
[29] A. De Santis, A.L. Ferrara, B. Masucci, Efficient provably-Secure hierarchical key assignment schemes, Theor. Comp. Sci. 412 (41) (2011) 5684–5699.
[30] E.S.V. Freire, K.G. Paterson, Provably secure key assignment schemes from factoring, in: Information Security and Privacy – 16th Australasian Conference, ACISP 2011, Melbourne, Australia, July 11–13, 2011. Proceedings, 2011, pp. 292–309, doi:10.1007/978-3-642-22497-3_19.
[31] E.S.V. Freire, K.G. Paterson, B. Poettering, Simple, efficient and strongly KI-secure hierarchical key assignment schemes, in: E. Dawson (Ed.), Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings, Lecture Notes in Computer Science, vol. 7779, Springer, 2013, pp. 101–114.
[32] L. Harn, H. Lin, A cryptographic key generation scheme for multilevel data security, Comput. Secur. 9 (6) (1990) 539–546.
[33] B.-Z. He, C.-M. Chen, T.-Y. Wu, H.-M. Sun, An efficient solution for hierarchical access control problem in cloud environment, Math. Prob. Eng. 2014 (2014).
[34] J. Horwitz, B. Lynn, Towards hierarchical identity-based encryption, in: Advances in Cryptology-EUROCRYPT 2002, Springer, 2002, pp. 466–481.
[35] H. Huang, C. Chang, A new cryptographic key assignment scheme with time-Constraint access control in a hierarchy, Comput. Stand. Interfaces 26 (3) (2004) 159–166.
[36] L.G. Khachiyan, A polynomial algorithm in linear programming, Doklady Akademii Nauk SSSR 244 (1979) 1093–1096.
[37] H. Liaw, S. Wang, C. Lei, A dynamic cryptographic key assignment scheme in a tree structure, Comput. Math. Appl. 25 (6) (1993) 109–114.
[38] C.-H. Lin, Dynamic key management schemes for access control in a hierarchy, Comput. Commun. 20 (15) (1997) 1381–1385.
[39] I.-C. Lin, M.-S. Hwang, C.-C. Chang, A new key assignment scheme for enforcing complicated access control policies in hierarchy, Future Gener. Comput. Syst. 19 (4) (2003) 457–462.
[40] S.J. MacKinnon, P.D. Taylor, H. Meijer, S.G. Akl, An optimal algorithm for assigning cryptographic keys to control access in a hierarchy, IEEE Trans. Comput. 34 (9) (1985) 797–802.
[41] H. Min-Shiang, A cryptographic key assignment scheme in a hierarchy for access control, Math. Comput. Model. 26 (2) (1997) 27–31.
[42] M.O. Rabin, Digitalized Signatures and Public-key Functions as Intractable as Factorization, Technical Report, Massachusetts Institute of Technology (Cambridge, MA US), Laboratory for Computer Science, Cambridge, MA, USA, 1979.
[43] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-Key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.
[44] S. Ruj, A. Nayak, I. Stojmenovic, DACC: distributed access control in clouds, in: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, IEEE, 2011, pp. 91–98.
[45] R.S. Sandhu, Cryptographic implementation of a tree hierarchy for access control, Inf. Process. Lett. 27 (2) (1988) 95–98.
[46] V.R.L. Shen, T. Chen, A novel key management scheme based on discrete logarithms and polynomial interpolations, Comput. Secur. 21 (2) (2002) 164–171.

[47] M. Sookhak, A. Gani, M.K. Khan, R. Buyya, Dynamic remote data auditing for securing big data storage in cloud computing, Inf. Sci. (2015), doi:10.1016/j.ins.2015.09.004. Article in Press, http://www.sciencedirect.com/science/article/pii/S0020025515006581.
[48] Q. Tang, C.J. Mitchell, Comments on a cryptographic key assignment scheme, Comput. Stand. Interfaces 27 (3) (2005) 323–326.
[49] S. Tang, X. Li, X. Huang, Y. Xiang, L. Xu, Achieving simple, secure and efficient hierarchical access control in cloud computing, IEEE Trans. Comput. 65 (7) (2016) 2325–2331.
[50] W. Tzeng, A time-bound cryptographic key assignment scheme for access control in a hierarchy, IEEE Trans. Knowl. Data Eng. 14 (1) (2002) 182–188.
[51] W. Tzeng, A secure system for data access based on anonymous authentication and time-dependent hierarchical keys, in: F. Lin, D. Lee, B.P. Lin, S. Shieh, S. Jajodia (Eds.), Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, Taipei, Taiwan, March 21–24, 2006, ACM, 2006, pp. 223–230.
[52] C. Wang, N. Cao, J. Li, K. Ren, W. Lou, Secure ranked keyword search over encrypted cloud data, in: 2010 International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, June 21–25, 2010, IEEE Computer Society, 2010, pp. 253–262.
[53] C. Wang, K. Ren, J. Wang, Secure and practical outsourcing of linear programming in cloud computing, in: INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10–15 April 2011, Shanghai, China, IEEE, 2011, pp. 820–828.
[54] G. Wang, Q. Liu, J. Wu, Hierarchical attribute-based encryption for fine-grained access control in cloud storage services, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, ACM, 2010, pp. 735–737.
[55] S. Wang, C. Laih, Merging: an efficient solution for a time-Bound hierarchical key assignment scheme, IEEE Trans. Dependable Sec. Comput. 3 (1) (2006) 91–100.
[56] T. Wu, C. Chang, Cryptographic key assignment scheme for hierarchical access control, Comput. Syst. Sci. Eng. 16 (1) (2001) 25–28.
[57] J. Yeh, R. Chow, R. Newman, A key assignment for enforcing access control policy exceptions, in: Proc. of the International Symposium on Internet Technology, 1998, pp. 54–59.
[58] X. Yi, Security of Chien's efficient time-Bound hierarchical key assignment scheme, IEEE Trans. Knowl. Data Eng. 17 (9) (2005) 1298–1299.
[59] X. Yi, Y. Ye, Security of Tzeng's time-Bound key assignment scheme for access control in a hierarchy, IEEE Trans. Knowl. Data Eng. 15 (4) (2003) 1054–1055.
[60] J. Zhang, Q. Dong, Efficient ID-based public auditing for the outsourced data in cloud storage, Inf. Sci. 343–344 (2016) 1–14, doi:10.1016/j.ins.2015.12.043.
[61] Y. Zhang, X. Chen, J. Li, D.S. Wong, H. Li, I. You, Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing, Inf. Sci. (2016), doi:10.1016/j.ins.2016.04.015. Article in Press, http://www.sciencedirect.com/science/article/pii/S002002551630250X.