Cryptographic Hierarchical Access Control for Dynamic Structures

Arcangelo Castiglione, Alfredo De Santis, *Member, IEEE*, Barbara Masucci, Francesco Palmieri, Aniello Castiglione, *Member, IEEE*, and Xinyi Huang

***Abstract*— A hierarchical key assignment scheme is a method to assign some private information and encryption keys to a set of classes in a partially ordered hierarchy, in such a way that the private information of a higher class can be used to derive the keys of all classes lower down in the hierarchy. Sometimes, it is necessary to make dynamic updates to the hierarchy, in order to implement an access control policy which evolves with time. All security models for hierarchical key assignment schemes have been designed to cope with static hierarchies and do not consider the issue of performing dynamic updates to the hierarchy. In this paper, we define the concept of hierarchical key assignment schemes supporting dynamic updates, formalizing the relative security model. In particular, we provide the notion of security with respect to key indistinguishability, by considering the dynamic changes to the hierarchy. Moreover, we show how to construct a hierarchical key assignment scheme supporting dynamic updates, by using as a building block a symmetric encryption scheme. The proposed construction is provably secure with respect to key indistinguishability, and provides efficient key derivation and updating procedures, while requiring each user to store only a single private key.**

***Index Terms*— Access control, key assignment, provable security, dynamic structures, adaptive adversary.**

I. Introduction

THE *access control problem* deals with the ability to ensure that only authorized users of a computer system are given access to some sensitive resources. According to their competencies and responsibilities, users are organized in a hierarchy formed by a certain number of disjoint classes, called *security classes*. A hierarchy arises from the fact that some users have more access rights than others. In the real world, there are several examples of hierarchies where access control is required. For example, within a hospital system, doctors can access data concerning their patients such as diagnosis, medication prescriptions, and laboratory tests, whereas researchers can be limited to consult anonymous clinical information for studies. Similar cases abound in other areas, particularly in the government and military ones.

Sometimes, it is necessary to make *dynamic updates* to the hierarchy, in order to implement an access control policy which evolves over time. For example, within a hospital system, whenever a new doctor is hired, it is necessary to assign him to a certain security class. Similarly, whenever a doctor retires, we need to remove him from the corresponding security class. The above situations, concerning single individuals, may be extended to the case where an entire security class needs to be inserted or deleted in the hierarchy. Moreover, the relationships between the classes could change over time. For example, in a complex enterprise security system, an entire class of users with a different security profile may be added as a consequence of the acquisition of a new company or branch, or similarly, the role and mission of an entire company sector may change after a fusion between enterprises, resulting in the need of redefining the structure of the access control hierarchy through the modification of several dependencies among the existing classes.

A *hierarchical key assignment scheme* is a method for assigning an encryption key and some private information to each class in the hierarchy. The encryption key will be used by each class to protect its data by means of a symmetric cryptosystem, whereas, the private information will be used by each class to compute the keys assigned to all classes lower down in the hierarchy. This assignment is carried out by a central authority, the *Trusted Authority (TA)*, which is active only at the distribution phase. Akl and Taylor [1] first proposed an elegant hierarchical key assignment scheme. In their scheme each class is assigned a key that can be used, along with some public parameters generated by the central authority, to compute the key assigned to any class lower down in the hierarchy. Many researchers have subsequently proposed schemes that either have better performances or allow insertion and deletion of classes in the hierarchy (e.g., [2]–[9]).

Atallah et al. [10] first addressed the problem of formalizing security requirements for hierarchical key assignment schemes and proposed two different notions:

A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, and A. Castiglione are with the Department of Computer Science, University of Salerno, Fisciano 84084, Italy (e-mail: arcastiglione@unisa.it; ads@unisa.it; bmasucci@unisa.it; fpalmieri@unisa.it; castiglione@ieee.org).

X. Huang is with the Nanjing University of Information Science & Technology, Nanjing 210000, China, and also with the Fujian Normal University, Fuzhou 350007, China (e-mail: xyhuang81@gmail.com).

Digital Object Identifier 10.1109/TIFS.2016.2581147

security against *key recovery* and with respect to *key indistinguishability*. Informally speaking, the former captures the notion that an adversary should not be able to compute a key to which it should not have access, while in the latter, the adversary should not even be able to distinguish between the real key and a random string of the same length. Different constructions satisfying the above defined notions of security were subsequently proposed in [11]–[15]. Freire et al. [16] proposed new security definitions for hierarchical key assignment schemes. Such definitions, called security against *strong key recovery* and security with respect to *strong key indistinguishability*, provide the adversary with additional compromise capability. Freire et al. showed how the notions of security against key recovery and against strong key recovery *are separated*. On the other hand, in [17] it has been proven that security with respect to strong key indistinguishability is *not stronger* than the one with respect to key indistinguishability, thus establishing the equivalence between such two security notions. A similar result has been shown in the unconditionally secure setting [18]. Other works in this setting can be found in [19]–[21]. Several proposals, e.g., [22]–[29] extended the model proposed in [10] to schemes satisfying additional time-dependent constraints. A more general scenario where the access control is not only hierarchical, but also *shared* between different classes has been recently considered for hierarchical key assignment schemes [30], [31]. Other approaches have been proposed to deal with hierarchical access control in more specific scenarios, as for example in the field of data outsourcing. Damiani et al. [32] introduced the problem of access control enforcement in database outsourced scenarios and proposed a solution based on selective encryption, which exploits hierarchical key derivation methods. Afterwards, in order to support policy updates, di Vimercati et al. [33] proposed a solution based on selective encryption, for the enforcement of access control and the management of its evolution. Finally, in the same line of related work, Blundo et al. [34] proposed a heuristic approach minimizing the number of keys to be stored by the system and distributed to users. Similarly to [32] and [33], this approach is efficient and it is based on a key derivation graph. However, in general, all the formal security models proposed so far consider an operational scenario which is *fixed and immutable*. More precisely, the adversary is not allowed to make any changes to the hierarchy, which is fixed and chosen at the time of the attack. We remark that this fact represents *an important limitation*, since the existing models are not able to characterize several scenarios which may arise in many operating environments.

A. Motivating Examples and Discussion

Advances in wireless communication and electronics have enabled the development of *User-Centric Networks (UCNs)*, in which the highly nomadic lifestyles of end-users, along with the strong entanglement between society and technology, have caused the birth of community networks, where users have very often an active role in dynamically sharing Internet access, as well as in sharing access to resources. Again, consider the emerging Internet of Things (IoT) technology, which provides wearable devices, home appliances and software with the opportunity to share and communicate information over the Internet. Since the shared data may be sensitive, preserving the security of such data is a main concern. More precisely, since IoT enables a constant transfer and sharing of data among things and users, in such a highly dynamic sharing environment access control and authorization are essential to ensure secure communication. We stress that the access to resources in this context presents a dynamic and multi-level behaviour. For example, regular membership could change, due to new users that join the group and active members that leave the group for membership expiration, i.e., "pre-determined" leaves, as well as because of exceptional active member leaves, e.g., in the case of explicit membership revocation, i.e., "unpredictable" leaves. It is easy to observe that ensuring access control in environments where not only users, but also things could be authorized to interact with the system, is a main concern. Moreover, a challenge in IoT is the control of the information collected and shared by mobile devices, which are increasingly small and pervasive, e.g., RFID or micro/nano sensors. In most cases, such devices are wirelessly connected and accessible, thus the challenge is to ensure that the information they collect and store are accessed by only authorized users. For example, the information collected by the body sensors applied to an elderly person should not be accessible by other persons apart from the doctor. Again, a user might be willing to share location information with family and friends to make the information available in aggregated form for improving the public transport, but such a user might not want the information will be used by other third-party service providers. Notice that though single pieces of information, i.e., single measurements, might not represent a noticeable issue for owners of IoT devices, however, since these devices generate data continuously, it is easy to observe that unauthorized access to such wealth of data can cause serious problems and can be used to damage data owners and maybe others. Therefore, it is essential to protect the access to IoT data, as well as to enable the access to data generated by other IoT devices, while preventing the use of data in unauthorized or undesired ways. Finally, in the context of smart cities, IoT hubs act as data aggregators and focal points, where each hub supports not only access to infrastructure data, but also participatory sensing and crowd sourced data in which city employees and citizens contribute directly to the data infrastructure of a city.

As it can be easily noticed from above considerations, it is necessary to extend and improve the existing security models, by providing the adversary with further attack capabilities. That is, the adversary should be given the possibility of performing a polynomial number of arbitrary changes to the hierarchy to reflect the dynamic nature of some operational scenarios. In particular, such changes should emulate as closely as possible all the attacks that can be performed in the real world. As a consequence, to deal with the aforementioned dynamic scenarios, several schemes have been proposed in the literature, e.g., [2], [5], [6], [10], [11], [14], [35]–[40]. However, it is important to emphasize that though each of them has its own characteristics and scenarios of application, all the schemes proposed so far lack of a formal security proof. More precisely, the security of those dynamic schemes is assessed only by means of informal and empirical evaluations. The reason is that all the security models proposed in the state of the art, namely, [10], [16], [25], are not able to manage changing and dynamic situations.

B. Contributions

In this work we consider hierarchical key assignment schemes supporting dynamic updates. We first propose a new security model which extends those that have been defined in the literature. In particular, we extend the notion of security with respect to key indistinguishability provided by Atallah et al., to address the further challenges introduced by the updates to the hierarchy. In this way, we provide the adversary with the ability to emulate all operations that can be performed in a real networking context. We remark that regardless of the particular characteristics of a given scheme, our model turns out to be essential to formally prove the security of such a scheme. Afterwards, we show how to construct a hierarchical key assignment scheme supporting dynamic updates, by using as a building block a symmetric encryption scheme. The proposed construction is provably secure with respect to key indistinguishability, provides efficient key derivation and updating procedures, while requiring each user to store only a single private key.

C. Organization

The paper is organized as follows. In Section II we formalize the concept of hierarchical key assignment scheme supporting dynamic updates, in particular by focusing on the types of updates as well as on the security notions. In Section III we show how to construct a hierarchical key assignment scheme which supports dynamic updates, by using as a building block a symmetric encryption scheme. Finally, in Section IV we conclude the paper.

II. Hierarchical Key Assignment Schemes With Dynamic Updates

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 is 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. [41] 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 [8], the family of the d-dimensional hierarchies [42], 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 graphs, corresponding to partially ordered hierarchies, supporting dynamic updates 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 *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) an 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 returned by $Upd(1^\tau, G, s, k, pub, up, params)$.

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.

A. Types of Updates

In this section we consider different types of updates which can be performed by using the algorithm Upd and we discuss how such updates modify the access rights of the classes in the hierarchy obtained after the update. 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 following we describe each type of updates.

- **Insertion of an edge.** Let u and v be two classes in V such that $(u, v) \notin E$. The insertion of the edge (u, v) in the graph $G' = (V', E')$ requires to update the accessible set of any class which was able to access u in G, in order to include the new access rights. In particular, for any class w such that $u \in A_w^G$, the updated accessible set $A_w^{G'}$ is defined to be $A_w^{G'} = A_w^G \cup A_v^G$. Moreover, the insertion of the edge (u, v) in G' also requires to update the forbidden set of any class which was accessed by v in G, in order to remove all classes which are able to access u. In particular, for any class z such that $z \in A_v^G$, the updated forbidden set $F_z^{G'}$ is defined to be $F_z^{G'} = F_z^G \setminus \{w : u \in A_w^G\}$.
- **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 accessible and forbidden sets of classes in G' 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 requires to check if any class z which was able to access class u in G is still able to access class v in the updated graph G'. More precisely, we have to investigate if there exists another path from z to v avoiding the deleted edge (u, v). If such a path exists, then the accessible set $A_z^{G'}$ is set to be equal to A_z^G. On the other hand, if such a path does not exist, then class v needs to be deleted from A_z^G, and we continue to check whether there exists a path from z and each class w which can be accessed by v, in order to decide whether w has to be deleted from A_z^G [43], [44]. Moreover, the forbidden set of each class w which can be accessed by class v needs to be updated in order to include all classes whose unique path to w has been broken by the deleted edge (u, 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.
- **Key replacement.** Let u be a class in G whose key k_u needs to be replaced, due either to a problem of loss, misuse or after an edge or class deletion in the hierarchy. Such an update does not change the structure of the hierarchy, consequently no accessible or forbidden set needs to be modified.
- **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, we need to choose a new secret information s'_u, which is then distributed to each non-revoked user in class u. This update does not alter the composition of the accessible set $A_u^{G'}$, which is set equal to A_u^G. However, in order to avoid the so called *ex-member problem*, a *key replacement update* for each class $v \in A_u^{G'}$ is needed.

Notice that the first four update types result in a structural modification of the hierarchy, whereas, the last two do not affect its structure. In particular, the last type of update represents a modification of the access control policy.

The *efficiency* of a hierarchical key assignment scheme supporting dynamic updates is evaluated mainly according to the complexity of the updates due to dynamic changes to the hierarchy. In particular, we would like to support dynamic changes by means of only local updates to the public information, without re-distributing private information to the classes affected by such changes. It is important to note that such a re-distribution cannot be avoided in the case of user revocation from a class, which necessarily requires to re-distribute the secret values to the non-revoked users in that class. However, it is desirable that no other private information must be updated.

B. Adversary Model

In the following we discuss the *security* issues for hierarchical key assignment schemes supporting dynamic updates. According to the *security reduction paradigm* introduced by Goldwasser and Micali [45], a scheme is *provably-secure* under a complexity assumption if the existence of an adversary A breaking the scheme is equivalent to the existence of an adversary B breaking the computational assumption [45]. The security notions proposed by Atallah et al. [10] and further extended by Ateniese et al. [25] have been designed to cope with *static hierarchies*. In particular, Atallah et al. proposed two different notions: security against *key recovery* and with respect to *key indistinguishability*. Informally speaking, the former captures the notion that an adversary should not be able to compute a key to which it should not have access, while in the latter, the adversary should not even be able to distinguish between the real key and a random string of the same length.

The above definitions need to be extended in order to address the additional security challenges introduced by the algorithm Upd used for handling dynamic updates to the hierarchy. In the following we consider two different scenarios: in the first one there is a fixed sequence of updates, which is known in advance to the execution of the scheme, whereas, in the second one, the update sequence is not known in advance and can be adaptively chosen. Moreover, for each scenario, we consider two different kinds of adversaries.

1) First Scenario (The Sequence of Updates Is Fixed): Let $\vec{up} = (up_1, \ldots, up_m)$ be a fixed sequence consisting of a polynomial number of updates and let $\vec{G} = (G^0, \ldots, G^m)$ be the corresponding sequence of graphs generated, starting from the initial graph $G^0 = G$, by the updates sequence $\vec{up}$. Moreover, let $\vec{pub} = (pub^0, \ldots, pub^m)$ be the sequence of public information, where each pub^i is the public information associated to the graph G^i, and let $\vec{old\_k} = (old\_k^0, \ldots, old\_k^{m-1})$ be the sequence of *old keys*, where each old_k^i denotes the sequence of keys which have been modified as a consequence of the update up^{i+1}, according to the specification of the Upd algorithm.

The first kind of adversary we consider is a *static adversary* $\texttt{STAT}_{\vec{up},t,u}$ which, given an updates sequence $\vec{up}$, an update index $t \in \{1, \ldots, m\}$ and a class u in the graph G^t, is allowed to access the private information assigned to all classes not allowed to compute the key k_u^t, as well as the sequence $\vec{pub}$ of all public information and the sequence $\vec{old\_k}$ of all old keys which have been modified. The second kind of adversary we consider is an *adaptive adversary* $\texttt{ADAPT}_{\vec{up}}$ which, given an updates sequence $\vec{up}$, is first allowed to access the sequence $\vec{pub}$ of all public information as well as the sequence $\vec{old\_k}$ of all old keys which have been modified. Moreover, the adversary $\texttt{ADAPT}_{\vec{up}}$ is also allowed to access the private information of a polynomial number of corrupted classes in $\vec{G}$ of its choice. Such a choice can be modeled by assuming the existence of a *corrupting oracle*, which provides the adversary with the private information held by the corrupted classes it has chosen. Afterwards, the adversary $\texttt{ADAPT}_{\vec{up}}$ chooses the update index t and the class u in G^t it wants to attack; such a choice is constrained by the fact that u cannot be a descendant of a class which has been previously corrupted by $\texttt{ADAPT}_{\vec{up}}$.

For both types of adversaries, two experiments are considered. In the first one, the adversary is given the key k_u^t, whereas in the second one it is given a random string ρ having the same length as k_u^t. It is the adversary's job to determine whether the received challenge corresponds to k_u^t or to a random string. For both types of adversaries, we require that the adversary will succeed with probability only negligibly different from 1/2.

It is easy to see the advantage of the adversary $\texttt{STAT}_{\vec{up},t,u}$ in distinguishing the key k_u^t from a random string is polynomially related to the advantage of adversary $\texttt{ADAPT}_{\vec{up}}$, since $\texttt{ADAPT}_{\vec{up}}$ can make a polynomial number of different choices for the pair (t, u). Thus, *when the updates sequence is fixed in advance*, security against adaptive adversaries is (polynomially) equivalent to security against static adversaries.

2) Second Scenario (The Sequence of Updates Is Adaptively Chosen): In order to model the adaptive choice of the updates, we assume the existence of an *updating oracle* $\mathcal{U}$. Such an oracle behaves as the *TA* when performing 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 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 the sequence old_k^i of keys which have been modified as a consequence of the update. 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, where old_k^i is a subsequence of k^i. Thus, $\mathcal{U}_{(1^\tau, G^i, s^i, k^i, pub^i)}$ $(up^{i+1}, params^{i+1})$ behaves as $Upd(1^\tau, G^i, s^i, k^i, pub^i, up^{i+1}, params^{i+1})$. Moreover, in order to be ready to reply to the next update query, the oracle updates its state to be $(G^{i+1}, s^{i+1}, k^{i+1}, pub^{i+1})$. In the following, for the sake of simplicity, 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.

In this scenario, we first consider a *static adversary* $\texttt{STAT}_{t,u}$, which is given access to the updating oracle $\mathcal{U}$ for a polynomial number of times and, given an update index t and a class u in G^t, is allowed to access the private information assigned to all classes not allowed to compute the key k_u^t, as well as the sequences $\vec{pub}$ and $\vec{old\_k}$ generated by the responses obtained by $\mathcal{U}$.

The second kind of adversary we consider is an *adaptive adversary* $\texttt{ADAPT}$, which, besides to access the updating oracle $\mathcal{U}$ for a polynomial number of times, can also perform a polynomial number of *class corruption operations*, again in an adaptive order. For any $i \geq 0$, we assume the existence of 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). More precisely, on input a class v in G^i, the corrupting oracle $\mathcal{C}^i_{(s^0, s^1, \ldots, s^i)}(\cdot)$ returns the private information s_v^j, for any $j = 0, \ldots, i$ such that v is in the hierarchy G^j. In the following, for the sake of simplicity, we denote by $\mathcal{C}^i(\cdot)$ the oracle $\mathcal{C}^i_{(s^0, s^1, \ldots, s^i)}(\cdot)$.

Finally, the adversary ADAPT can perform a *class attack operation*, by choosing an update index t and a class u in G^t.

What about the relationships between the above two kinds of adversaries? It is easy to see that, when the sequence of updates is adaptively chosen, the advantage of the adversary $\mathtt{STAT}_{t,u}$ in distinguishing the key k_u^t from a random string cannot be polynomially related to the advantage of the adversary ADAPT, since there is an exponential number of different choices for the updating sequence, which also reflects in an exponential number of possible choices for the pair (t, u). *Thus, when the updates sequence is adaptively chosen, security against adaptive adversaries is not (polynomially) equivalent to security against static adversaries.* Since ADAPT represents the strongest type of adversary, in the scenario where the sequence of updates is not fixed in advance, we will restrict our attention to such a kind of adversary. In particular, we consider an adversary $\mathtt{ADAPT} = (\mathtt{ADAPT}_1, \mathtt{ADAPT}_2)$ running in two stages. 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, 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 updating oracle; 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. 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 in distinguishing the key k_u^t from a random string $\rho \in \{0, 1\}^\tau$. 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 that have been updated in the graph G^t.

We use the standard notation to describe probabilistic algorithm and experiments following [46]. If $A(\cdot, \cdot, \ldots)$ is any probabilistic algorithm then $a \leftarrow A(x, y, \ldots)$ denotes the experiment of running A on inputs $x, y, \ldots$ and letting a be the outcome, the probability being over the coins of A. Similarly, if X is a set, then $x \leftarrow X$ denotes the experiment of selecting an element uniformly from X and assigning x this value. If w is neither an algorithm nor a set, then $x \leftarrow w$ is a simple assignment statement. A function $\epsilon : N \to R$ is *negligible* if for every constant $c > 0$ there exists an integer n_c such that $\epsilon(n) < n^{-c}$ for all $n \geq n_c$.

The next definition formalizes the *key indistinguishability* requirement for hierarchical key assignment schemes supporting dynamic updates.

Definition 2 (IND-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 two experiments:

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

Experiment $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}(1^\tau, G)$
    $(s, k, pub) \leftarrow Gen(1^\tau, G)$
    $(t, u, history) \leftarrow \mathtt{ADAPT}_1^{\mathcal{U}^i(\cdot,\cdot),\mathcal{C}^i(\cdot)}(1^\tau, G, pub)$
    $\rho \leftarrow \{0, 1\}^\tau$
    $d \leftarrow \mathtt{ADAPT}_2^{\mathcal{C}^i(\cdot)}(1^\tau, t, u, history, \rho)$
    return $d$
```

It is required that the class u returned 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 also 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. The advantage of ADAPT is defined as

$$\mathbf{Adv}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN}}(1^\tau, G) = |Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}(1^\tau, G) = 1] - Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}(1^\tau, G) = 1]|$$

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

Notice that if the adversary $\mathtt{ADAPT}_1$ never queries the updating oracle during the first stage of the attack, the above definition reduces to that of security with respect to key indistinguishability against adaptive adversaries for hierarchical key assignment schemes with static hierarchies, referred to as IND-AD in [25]. For such kind of schemes, it has been proven that adaptive adversaries are *polynomialy equivalent* to *static* adversaries, i.e., such that the class to be attacked is chosen in advance to the execution of the scheme. Finally, the weaker definition of security with respect to key recovery against adaptive adversaries for hierarchical key assignment schemes supporting dynamic hierarchies has been provided in [47], where also the relations between all the security notions for such schemes have been clarified.

III. A Construction Based on Symmetric Encryption Schemes

In this section we consider the problem of constructing a hierarchical key assignment scheme supporting dynamic updates using as a building block a symmetric encryption scheme. In particular, we consider the *Two-Level Encryption-Based Construction* (TLEBC) proposed in [25]. Such a construction belongs to the class of *time-bound* hierarchical key assignment schemes, since the key derivation depends not only on the relations between the classes, but also on time constraints. However, since in this paper we are not interested in time-bound schemes, we describe a simplified version of the scheme, without time constraints. In detail, to construct a key assignment scheme supporting dynamic updates, the TLEBC has been properly adapted by providing it with a third algorithm, which is in charge of dynamically updating the access structure. Later on, we prove that the security property of the TLEBC depends on the security property of the underlying encryption scheme. We need to recall the definition of a symmetric encryption scheme and its notions of security.

A. Symmetric Encryption Schemes

We first recall the definition of a symmetric encryption scheme.

Definition 3: A *symmetric encryption scheme* is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of algorithms satisfying the following conditions:

1) The *key-generation algorithm* $\mathcal{K}$ is probabilistic polynomial-time. It takes as input the security parameter 1^τ and produces as output a string key.
2) The *encryption algorithm* $\mathcal{E}$ is probabilistic polynomial-time. It takes as inputs 1^τ, a string key produced by $\mathcal{K}(1^\tau)$, and a message $m \in \{0,1\}^*$, and produces as output the ciphertext y.
3) The *decryption algorithm* $\mathcal{D}$ is deterministic polynomial-time. It takes as inputs 1^τ, a string key produced by $\mathcal{K}(1^\tau)$, and a ciphertext y, and produces as output a message m.

We require that for any string key which can be returned by $\mathcal{K}(1^\tau)$, for any message $m \in \{0,1\}^*$, and for all y that can be returned by $\mathcal{E}(1^\tau, key, m)$, we have that $\mathcal{D}(1^\tau, key, y) = m$.

Now, we define what we mean by a *secure* symmetric encryption scheme. We consider security with respect to *plaintext indistinguishability*, which is an adaption of the notion of *polynomial security* as given in [45]. We imagine an adversary $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$ running in two stages. In advance of the adversary's execution, a random key key is chosen and kept hidden from the adversary. During the first stage, the adversary $\mathtt{A}_1$ outputs a triple $(x_0, x_1, state)$, where x_0 and x_1 are two messages of the same length, and $state$ is some state information which could be useful later. One message between x_0 and x_1 is chosen at random and encrypted to give the challenge ciphertext y. In the second stage, the adversary $\mathtt{A}_2$ is given y and $state$ and has to determine whether y is the encryption of x_0 or x_1. Informally, the encryption scheme is said to be secure with respect to a non-adaptive chosen plaintext attack, denoted by IND-P1-C0 in [48], if every polynomial-time adversary A, which has access to the encryption oracle only during the first stage of the attack and never has access to the decryption oracle, succeeds in determining whether y is the encryption of x_0 or x_1 with probability only negligibly different from $1/2$.

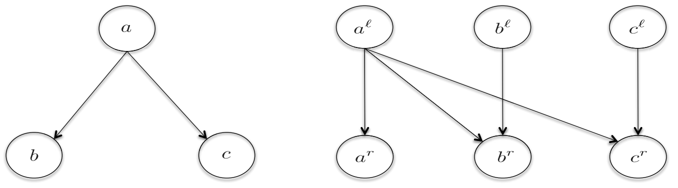

Fig. 1. The graph transformation used in our construction.

Definition 4 (IND-P1-C0): Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and let τ be a security parameter. Let $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$ be an adversary that has access to the encryption oracle only during the first stage of the attack and never has access to the decryption oracle. Consider the following two experiments:

Experiment $\mathbf{Exp}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0-1}}(1^\tau)$	*Experiment* $\mathbf{Exp}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0-0}}(1^\tau)$
$key \leftarrow \mathcal{K}(1^\tau)$	$key \leftarrow \mathcal{K}(1^\tau)$
$(x_0, x_1, state) \leftarrow \mathtt{A}_1^{\mathcal{E}_{key}(\cdot)}(1^\tau)$	$(x_0, x_1, state) \leftarrow \mathtt{A}_1^{\mathcal{E}_{key}(\cdot)}(1^\tau)$
$y \leftarrow \mathcal{E}_{key}(x_1)$	$y \leftarrow \mathcal{E}_{key}(x_0)$
$d \leftarrow \mathtt{A}_2(1^\tau, y, state)$	$d \leftarrow \mathtt{A}_2(1^\tau, y, state)$
return d	**return** d

The advantage of A is defined as

$$\mathbf{Adv}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0}}(1^\tau) = |Pr[\mathbf{Exp}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0-1}}(1^\tau) = 1] - Pr[\mathbf{Exp}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0-0}}(1^\tau) = 1]|$$

The scheme is said to be *secure* in the sense of IND-P1-C0 if the advantage function $\mathbf{Adv}_{\Pi,\mathtt{A}}^{\mathtt{IND-P1-C0}}(1^\tau)$ is negligible, for any adversary A whose time complexity is polynomial in τ.

B. The Two-Level Encryption-Based Construction (TLEBC)

The construction we are going to describe uses a *graph transformation*, starting from the graph $G = (V, E)$. The output of such a transformation is a *two-level graph* $G_{TL} = (V_{TL}, E_{TL})$, where $V_{TL} = V^\ell \cup V^r$ and $V^\ell \cap V^r = \emptyset$, constructed as follows:

- for each class $u \in V$, we place two classes u^ℓ and u^r in V_{TL}, where $u^\ell \in V^\ell$ and $u^r \in V^r$;
- for each class $u \in V$, we place the edge (u^ℓ, u^r) in E_{TL};
- for each pair of classes u and v connected by a path in G, we place the edge (u^ℓ, v^r) in E_{TL}.

Thus, we consider a two-level partially ordered hierarchy, where each level contains the same number of classes and there are no edges between classes at the same level. We remark that this is not a restriction, since any directed graph representing an access control policy can be transformed into a two-level partially ordered hierarchy having the above features, using a technique proposed in [49]. Figure 1 shows an example of the graph transformation described above.

Let Γ be a family of graphs corresponding to partially ordered hierarchies, let $G = (V, E) \in \Gamma$, and let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. The Two-Level Encryption-Based Construction (TLEBC) works as explained in the following.

As for any hierarchical key assignment scheme, the TLEBC is basically composed by two algorithms, namely, the key

Algorithm 1 Key Generation
```
1: procedure $Gen(1^\tau, G)$
2:     Transform the graph $G$ into the two-level graph $G_{TL} = (V_{TL}, E_{TL})$
3:     for all classes $u^\ell \in V^\ell$ do
4:         $s_u \leftarrow \mathcal{K}(1^\tau)$
5:     end for
6:     for all classes $u^r \in V^r$ do
7:         $k_u \leftarrow \{0,1\}^\tau$
8:     end for
9:     Let $s$ and $k$ be the sequences of private information and keys computed above
10:    for any pair of classes $u^\ell \in V^\ell$ and $v^r \in V^r$ such that $(u^\ell, v^r) \in E_{TL}$ do
11:        $p_{(u,v)} \leftarrow \mathcal{E}_{s_u}(k_v)$
12:    end for
13:    Let $pub$ be the sequence of public information computed above
14:    return $(s, k, pub)$
15: end procedure
```

Algorithm 2 Key Derivation
```
1: procedure $Der(1^\tau, G, u, v, s_u, pub)$
2:     Extract the public value $p_{(u,v)}$ from $pub$
3:     $k_v \leftarrow \mathcal{D}_{s_u}(p_{(u,v)})$
4:     return $k_v$
5: end procedure
```

Algorithm 3 The Main Updating Procedure
```
1: procedure $Upd(1^\tau, G, s, k, pub, up, params)$
2: Transform $G$ in the two-level graph $G_{TL} = (V_{TL}, E_{TL})$
3:     if $up$ == Replace($k_v$) then
4:         $replace\_key(1^\tau, G, s, k, pub, k_v, params)$
5:     else if $up$ == Insert_edge($(u, v)$) then
6:         $insert\_edge(1^\tau, G, s, k, pub, (u, v))$
7:     else if $up$ == Delete_edge($(u, v)$) then
8:         $delete\_edge(1^\tau, G, s, k, pub, (u, v), params)$
9:     else if $up$ == Insert_class($v$) then
10:        $insert\_class(1^\tau, G, s, k, pub, v, params)$
11:    else if $up$ == Delete_class($v$) then
12:        $delete\_class(1^\tau, G, s, k, pub, v)$
13:    else if $up$ == Revoke($v, \lambda$) then
14:        $revoke\_user(1^\tau, G, s, k, pub, v, \lambda, params)$
15:    end if
16:    return $(G', s', k', pub')$
17: end procedure
```

Algorithm 4 Key Replacement
```
1: procedure $replace\_key(1^\tau, G, s, k, pub, k_v, params)$
2:     if $params$ is not empty then
3:         parse $params$ as $k_v^{params}$
4:         $k'_v \leftarrow k_v^{params}$
5:     else
6:         $k'_v \leftarrow \{0,1\}^\tau$
7:     end if
8:     $s' \leftarrow s$
9:     $k' \leftarrow k$, with $k'_v$ instead of $k_v$
10:    for all classes $u^\ell \in V^\ell$ such that $(u^\ell, v^r) \in E_{TL}$ do
11:        compute $p'_{(u,v)} \leftarrow \mathcal{E}_{s'_u}(k'_v)$ and replace it in $pub$, obtaining $pub'$
12:    end for
13: end procedure
```

Algorithm 5 Edge Insertion
```
1: procedure $insert\_edge(1^\tau, G, s, k, pub, (u, v))$
2:     $k' \leftarrow k$
3:     $s' \leftarrow s$;
4:     compute $p'_{(u,v)} \leftarrow \mathcal{E}_{s'_u}(k'_v)$ and add it to $pub$, obtaining $pub'$
5: end procedure
```

generation algorithm, denoted as *Gen* and described by means of Algorithm 1, and the key derivation algorithm, denoted as *Der* and outlined through Algorithm 2. More precisely, Algorithm 1 starts by transforming a generic graph belonging to the family of partially ordered hierarchy into the relative two-level representation. Afterwards, such an algorithm is responsible for assigning the secret information and encryption keys to the above defined two-level graph. Instead, for what concerns Algorithm 2, it is simply responsible, given a secret information and a public value, of deriving the encryption key of the class to be accessed.

The third core algorithm of the TLEBC is the one which deals with dynamic updates to the access hierarchy, and from now on it is referred to as *Upd*. For the ease of exposition, this algorithm, outlined by Algorithm 3, is composed of several sub-procedures, each characterizing a type of update. Such procedures can also call each other recursively. In particular, the *Upd* algorithm, given a graph, some parameters, and the type of update, returns a new instance of the graph with the relative secret information and encryption keys updated according to the type of update it performed. We remark that, similarly to what it is done by *Gen*, also *Upd* performs, before each type of update, a transformation on the graph taken as input, in order to obtain a two-level representation of such a graph.

More precisely, as stated before, the update types taken into account are: key replacement, edge insertion, edge deletion, class insertion, class deletion, and user revocation. Again, the procedures responsible for such updates are denoted as *replace_key*, *insert_edge*, *delete_edge*, *insert_class*, *delete_class* and *revoke_user*, respectively, and they are outlined by Algorithm 4, Algorithm 5, Algorithm 6, Algorithm 7, Algorithm 8 and Algorithm 9, respectively.

In detail, Algorithm 4 replaces an old key, referred to as k_v, with a new one, denoted as k'_v. We remark that such an update does not involve any change to the structure of the hierarchy and no accessible or forbidden set needs to be modified. As a consequence, only the public values relative to the replaced

Algorithm 6 Edge Deletion

```
1: procedure $delete\_edge(1^\tau, G, s, k, pub, (u, v), params)$
2:     $replace\_key(1^\tau, G, s, k, pub, k_v, params)$
3:     Remove the public value $p'_{(u,v)}$ from $pub'$
4: end procedure
```

Algorithm 7 Class Insertion

```
1: procedure $insert\_class(1^\tau, G, s, k, pub, v, params)$
2:     if $params$ is not empty then
3:         parse $params$ as $s_v^{params}$ and $k_v^{params}$
4:         $s'_v \leftarrow s_v^{params}$
5:         $k'_v \leftarrow k_v^{params}$
6:     else
7:         $s'_v \leftarrow \mathcal{K}(1^\tau)$
8:         $k'_v \leftarrow \{0, 1\}^\tau$
9:     end if
10:    add above values into $s$ and $k$, obtaining $s'$ and $k'$
11:    compute $p'_{(v,v)} \leftarrow \mathcal{E}_{s'_v}(k'_v)$
12:    use aforementioned procedure for adding incoming and
       outgoing edges from $v$
13:    add all the public values to $pub$, obtaining $pub'$
14: end procedure
```

Algorithm 8 Class Deletion

```
1: procedure $delete\_class(1^\tau, G, s, k, pub, v)$
2:     Use aforementioned procedure for deleting incoming
       and outgoing edges from $v$
3:     Remove $s_v$ and $k_v$ from $s$ and $k$, obtaining $s'$ and $k'$
4: end procedure
```

Algorithm 9 Revoke User

```
1: procedure $revoke\_user(1^\tau, G, s, k, pub, v, \lambda, params)$
2:     if $params$ is not empty then
3:         parse $params$ as $s_v^{params}$
4:         $s'_v \leftarrow s_v^{params}$
5:     else
6:         $s'_v \leftarrow \mathcal{K}(1^\tau)$
7:     end if
8:     Distribute $s'_v$ to each non-revoked user in the class $v$
9:     $s' \leftarrow s$, with $s'_v$ instead of $s_v$
10:    for all classes $u^r \in V^r$ such that $(v^\ell, u^r) \in E_{TL}$ do
11:        $replace\_key(1^\tau, G, s, k, pub, k_u, NULL)$
12:    end for
13: end procedure
```

key need to be re-computed, by using the existing secret information. Algorithm 5 inserts an edge, denoted as (u, v), between two already existing classes, denoted as u and v, respectively. Such an update simply involves the encryption of the key k_u by using the secret information s_v. Algorithm 6 deletes an edge, called (u, v), by first replacing the encryption key k_v and then deleting the public value $p'_{(u,v)}$ from the public parameters of the updated graph, referred to as pub'. Algorithm 7 inserts a new class v in the considered graph. In particular, such an algorithm, given the secret information and the encryption key k_v, computes all the public values involving class v. We remark that Algorithm 7 makes use of the $insert\_edge$ procedure, for creating the public values relative to the edges incoming in and outgoing from v. Algorithm 8 removes a class v from the considered graph. More precisely, such an algorithm makes use of the $delete\_edge$ procedure for deleting edges incoming in and outgoing from v, then it removes the secret information s_v and encryption key k_v. Finally, Algorithm 9 revokes a user λ from a given class v. In detail, such an algorithm distributes a new secret information, referred to as s'_v to each non-revoked user in the class v; then, in order to avoid the so called ex-member problem, Algorithm 9 calls the $replace\_key$ procedure for each class in the accessible set of class v in the updated graph G'.

Notice that some updates do not require the sequence of additional parameters $params$, whereas, for other updates, such a sequence might be not empty. In particular, the *insertion of an edge* and the *deletion of a class* do not require additional parameters, since they do not involve the choice of fresh private information or keys. We remark that no update, with the exception of user revocation, requires re-distribution of the secret information to classes. Thus, in the TLEBC, dynamic changes to the hierarchy can be accomplished by means of only local updates to the public information only.

1) Analysis of the Scheme: In the following we show that the security property of the TLEBC depends on the security property of the underlying encryption scheme. We prove that if the encryption scheme $\Pi = (\mathcal{K}, \mathcal{D}, \mathcal{E})$ is secure in the sense of `IND-P1-C0`, then the TLEBC is secure in the sense of `IND-DYN-AD`, respectively.

We first give an informal description of the ideas on which the proof is based. The proof uses two well known techniques: *black-box reductions* [45] and *hybrid arguments* [50]. In general, a black-box reduction is used to show that, given a protocol constructed from a cryptographic primitive, if the protocol can be broken somehow, then also the underlying primitive can be broken. On the other hand, the hybrid argument technique is used to argue that two *probability ensembles*, i.e., two sequences of probability distributions defined over the same probability space, are *computationally indistinguishable*. In this type of proof, one defines a sequence, constituted by a *polynomial number* (in the security parameter) of probability ensembles, also called the *hybrids*, where the extreme hybrids correspond to the two ensembles to be shown indistinguishable. Since the total number of hybrids is polynomial, a non-negligible gap between the extreme hybrids translates into a non-negligible gap between a pair of adjacent hybrids.

In our security proof the probability ensembles are given by the view of an adaptive adversary `ADAPT` which, after making a polynomial number of updating and corrupting queries, attacks a class v^r in the two-level hierarchy obtained after the t-th update on the initial graph $G = (V, E)$. In particular, such a view contains the public information pub^i associated to the graph G^i, for $i = 1, \dots, t$, the private information held by the corrupted classes along with a final value, which corresponds to the key k_h assigned to the chosen class v^r

after the t-th update. The two extreme hybrids we consider are characterized in one case by the adversary's view when the public values are generated according to the TLEBC, thus containing encryptions of the key k_h, while in the other case by the adversary's view when *part* of the public values is generated by encrypting a randomly chosen value ρ having the same length as k_h. More precisely, the public values which are modified in the last hybrid correspond to those associated to the edges, say $(v_1^\ell, v^r), (v_2^\ell, v^r), \ldots, (v_m^\ell, v^r)$, entering class v^r in the two-level hierarchy obtained after the t-th update. Thus, in the last hybrid, the public information is completely independent on the last input of the adversary, i.e., k_h, since the values associated to all edges entering class v^r are computed as encryptions of a randomly chosen value ρ. We define a sequence of $m+1$ hybrids, where each pair of adjacent ones, say the j-th and the $(j+1)$-th hybrid, differ only in the public value associated to a certain edge entering v^r, say (v_{j+1}^ℓ, v^r), which is equal to the encryption of the key k_h in the latter one and to the encryption of a random value ρ, having the same length as k_h, in the former one. For each pair of adjacent hybrids we show, by means of a black-box reduction, that the corresponding views are computationally indistinguishable by the adversary ADAPT, otherwise we could construct an adversary $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$ which breaks the security of the symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{D}, \mathcal{E})$ in the sense of IND-P1-C0.

The algorithm $\mathtt{A}_1$, on input 1^τ, makes queries to its encryption oracle $\mathcal{E}_{key}(\cdot)$ and outputs a triple $(x_0, x_1, state)$, where $x_0, x_1 \in \{0,1\}^\tau$ and $state$ is some state information. One message between x_0 and x_1 is chosen at random and encrypted to give the *challenge* ciphertext y. Then, algorithm $\mathtt{A}_2$ is given y and $state$ and has to determine whether y is the encryption of x_0 or x_1. More precisely, the algorithm A, in order to exploit ADAPT's ability in distinguishing between the j-th and the $(j+1)$-th hybrid, has first to prepare the inputs for it. Such an information, with the exception of the value associated to the $(j+1)$-th edge entering class v^r, can be easily constructed by $\mathtt{A}_1$ following the same lines as the *Gen* and *Upd* algorithms in the TLEBC, with an important difference: whenever $\mathtt{A}_1$ has to construct the public information associated to the first j edges entering class v^r, it computes the encryptions, with the appropriate private keys, of the random value x_0. On the other hand, the $(j+1)$-th edge entering class v^r will be assigned the value of the challenge ciphertext y, whereas, all subsequent edges entering class v^r will be encryptions of the value x_1, which plays the role of the key k_h assigned to v^r.

It is important to notice that, due to the dynamic behavior of ADAPT, adversary A has no control on the sequence of updating queries asked by ADAPT, thus, it cannot decide in advance to which class its encryption oracle $\mathcal{E}_{key}(\cdot)$ will be associated. Therefore, A makes its guess at the beginning, by randomly choosing a class whose private information will be implicitly set equal to the unknown *key*, but it might fail. Indeed, if the chosen class does not correspond to the $(j+1)$-th one having an edge entering class v^r, then A cannot continue its simulation and needs to restart itself. On the other hand, if the simulation goes well, A outputs the same output as ADAPT: if ADAPT states that its view corresponds to that in the j-th experiment, then the adversary A can be sure that the received challenge y comes from the encryption of the message x_1, otherwise by the encryption of the message x_0. Clearly, the success probability of $\mathtt{A}_1$ is equal to $1/q$, where q denotes the number of choices which can be made by $\mathtt{A}_1$, meaning that, in the worst case, $\mathtt{A}_1$ will need to restart itself q times. Thus, the next result holds.

Theorem 5: If the encryption scheme $\Pi = (\mathcal{K}, \mathcal{D}, \mathcal{E})$ is secure in the sense of IND-P1-C0, then the TLEBC is secure in the sense of IND-DYN-AD.

Proof: Let Γ be a family of graphs corresponding to partially ordered hierarchies. Assume by contradiction that the TLEBC is not secure in the sense of IND-DYN-AD. Thus, there exists a graph $G = (V, E)$ in Γ and an adaptive adversary $\mathtt{ADAPT} = (\mathtt{ADAPT}_1, \mathtt{ADAPT}_2)$ which is able to distinguish between experiments $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}(1^\tau, G)$ and $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}(1^\tau, G)$ with non-negligibile advantage. Recall that the only difference between $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}$ and $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}$ is the last input of ADAPT, which corresponds to a real key assigned by the TLEBC in the former experiment and to a random value chosen in $\{0,1\}^\tau$ in the latter. Thus, while in $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}$ the public information is related to the last input of ADAPT, in $\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}$ it is completely independent on such a value.

Without loss of generality, let $V = \{v_1, \ldots, v_n\}$ and let $k_1, \ldots, k_n$ be the keys assigned to the classes in V^r by algorithm *Gen* of the TLEBC. For any $i \geq 1$, denote by k_{n+i} the i-th key which either has been created using the *insert_class* procedure (see **Algorithm 7**) or has been modified using the *replace_key* procedure (see **Algorithm 4**). We restrict our interest to the two above procedures, since those corresponding to the other types of updates either do not require to choose new keys (see **Algorithm 5** and **Algorithm 8**) or invoke the *replace_key* procedure (see **Algorithm 6** and **Algorithm 9**). Moreover, let $s_1, \ldots, s_n$ be the private information assigned to the classes in V^ℓ by algorithm *Gen* of the TLEBC and, for any $i \geq 1$, denote by s_{n+i} the i-th private information which either has been created using the *insert_class* procedure (see **Algorithm 7**) or has been modified using the *revoke_user* procedure (see **Algorithm 9**). We restrict our interest to the two above procedures, since those corresponding to the other types of updates do not require to choose new private information. For example, consider the following sequence of updates on the two-level graph in Figure 1, yielding to the graph depicted in Figure 2: insert class d and edge (d, c), replace key k_c, insert class e and edge (e, c), replace key k'_c. According to the above defined enumeration, the corresponding sequence of keys is $k_1 = k_a$, $k_2 = k_b$, $k_3 = k_c$, $k_4 = k_d$, $k_5 = k'_c$, $k_6 = k_e$, and $k_7 = k''_c$. On the other hand, the corresponding sequence of private information is $s_1 = s_a$, $s_2 = s_b$, $s_3 = s_c$, $s_4 = s_d$, and $s_5 = s_e$.

Let $q(n, 1^\tau)$ be the running-time of ADAPT, where q is a bivariate polynomial. For any $i = 1, \ldots, q(n, 1^\tau)$, let $\mathtt{S}_i$ be an adversary which behaves as $\mathtt{ADAPT}_1$ until the choice of the key to be attacked. If the chosen key is equal to k_i, then $\mathtt{S}_i$ continues to follow $\mathtt{ADAPT}_2$, otherwise it outputs 0. The advantage of ADAPT can be written as shown in Eq. 1.

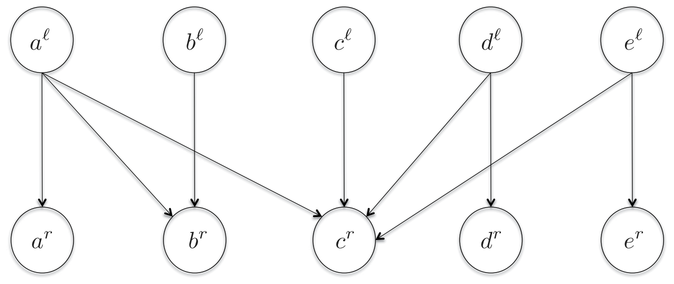

Fig. 2. Two-level hierarchy obtained after a list of updates.

Since $\mathbf{Adv}^{\mathtt{IND-DYN}}_{\mathtt{ADAPT}}(1^\tau, G)$ is non-negligible, then there exists at least an index h, where $1 \le h \le q(n, 1^\tau)$, such that $\mathbf{Adv}^{\mathtt{IND-DYN}}_{\mathtt{S}_h}(1^\tau, G)$ is non-negligible. Thus, there exists an adversary $\mathtt{S}_h$ which distinguishes between $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}(1^\tau, G)$ with non-negligible advantage. We distinguish the two following cases:

- **Case 1:** $h \ge n+1$. This case corresponds to the scenario where the key k_h chosen by the adversary either has been created using the *insert_class* procedure (see **Algorithm 7**) or has been modified using the *replace_key* procedure (see **Algorithm 4**).
- **Case 2:** $1 \le h \le n$. 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.

a) 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 v in the graph G^t. Thus, attacking the key k_h corresponds to attack the class v^r in the two-level hierarchy $G_{TL} = (V_{TL}, E_{TL})$ obtained after the t-th update. Assume that there are m classes which are able to access class v^r in G_{TL}, without loss of generality, let $v_1^\ell, \ldots, v_m^\ell$ be such classes. We construct a sequence of $m+1$ experiments

$$\mathbf{Exp}_{h,0}, \mathbf{Exp}_{h,1}, \ldots, \mathbf{Exp}_{h,m},$$

all defined over the same probability space. In each experiment we modify the way the view of adversary $\mathtt{S}_h$ is computed, while maintaining the view's distributions indistinguishable among any two consecutive experiments. For any $j = 1, \ldots, m$, experiment $\mathbf{Exp}_{h,j}$ is defined as follows:

```
Experiment $\mathbf{Exp}_{h,j}(1^\tau, G)$
    $(s, k, pub) \leftarrow Gen(1^\tau, G)$
    $(t, v, history) \leftarrow \mathtt{S}_h^{\mathcal{U}^{i,j}(\cdot,\cdot),\mathcal{C}^i(\cdot)}(1^\tau, G, pub)$
    $d \leftarrow \mathtt{S}_h^{\mathcal{C}^i(\cdot)}(1^\tau, t, v, history, k_h)$
    return $d$
```

In experiment $\mathbf{Exp}_{h,j}$ we first use the algorithm Gen of the TLEBC to assign private information and keys to the classes, as well as public information to the edges of the two-level hierarchy. Then, in the first stage of the attack, the updating oracle queried by adversary $\mathtt{S}_h$ uses an algorithm Upd^j, which is a modification of the algorithm Upd used in the TLEBC. We remark that for this reason the updating oracle is denoted by $\mathcal{U}^{i,j}(\cdot,\cdot)$ in the experiment. The algorithm Upd^j differs from Upd for the way it computes the public information associated to the first j edges, say $(v_1^\ell, v^r), (v_2^\ell, v^r), \ldots, (v_j^\ell, v^r)$, entering class v^r in the two-level hierarchy. In particular, the public values associated to such edges are computed as encryptions of a value ρ randomly chosen in $\{0,1\}^\tau$, instead of the encryption of the key k_h assigned to v^r, whereas, the public values associated to subsequent edges entering v^r are not modified. Notice that experiment $\mathbf{Exp}_{h,0}$ is the same as $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}$. Indeed, the public information is related to the last input of $\mathtt{S}_h$, since the values associated to all edges entering class v^r are computed as encryptions of k_h. On the other hand, experiment $\mathbf{Exp}_{h,m}$ is the same as $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}$. In fact, the public information is completely independent on the last input of $\mathtt{S}_h$, since the values associated to all edges entering class v^r are computed as encryptions of a random value ρ chosen in $\{0,1\}^\tau$. Figure 3 shows two adjacent experiments in the sequence $\mathbf{Exp}_{7,0}, \mathbf{Exp}_{7,1}, \ldots, \mathbf{Exp}_{7,4}$ of five experiments obtained when attacking the key k_7 in Figure 2.

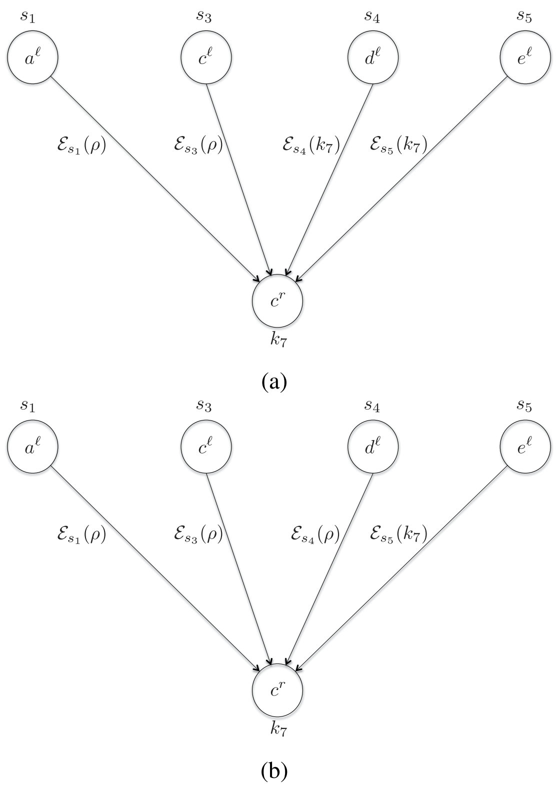

Fig. 3. Two adjacent experiments. (a) $\mathbf{Exp}_{7,2}$. (b) $\mathbf{Exp}_{7,3}$.

b) Indistinguishability of any pair of adjacent experiments: In the following we show that, for any $j = 0, \ldots, m-1$, experiments $\mathbf{Exp}_{h,j}$ and $\mathbf{Exp}_{h,j+1}$ cannot be distinguished with non-negligible advantage.

Assume by contradiction that there exists a polynomial-time distinguisher $\mathtt{B}_j$ which is able to distinguish between the adversary $\mathtt{S}_h$'s views in experiments $\mathbf{Exp}_{h,j}$ and $\mathbf{Exp}_{h,j+1}$ with non-negligible advantage. Notice that the views of the distinguisher $\mathtt{B}_j$ in such two experiments differ only for the public value associated to the edge (v_{j+1}^ℓ, v^r), which is equal to the encryption of the real key k_h in the latter experiment and to the encryption of a random value having the same length as the real key in the former experiment. We show how to construct a polynomial-time adversary $\mathtt{A} = (\mathtt{A}_1, \mathtt{A}_2)$,

$$\begin{aligned}\mathbf{Adv}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN}}(1^\tau, G) &= |Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}(1^\tau, G) = 1] - Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}(1^\tau, G) = 1]| \\ &\leq \sum_{i=1}^{q(n,1^\tau)} |\ Pr[\mathtt{ADAPT}_1 \text{ chooses } k_i] \cdot Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-1}}(1^\tau, G) = 1 | \mathtt{ADAPT}_1 \text{ chooses } k_i] - \\ &\qquad + Pr[\mathtt{ADAPT}_1 \text{ chooses } k_i] \cdot Pr[\mathbf{Exp}_{\mathtt{ADAPT}}^{\mathtt{IND-DYN-0}}(1^\tau, G) = 1 | \mathtt{ADAPT}_1 \text{ chooses } k_i]\ | \\ &= \sum_{i=1}^{q(n,1^\tau)} Pr[\mathtt{ADAPT}_1 \text{ chooses } k_i] \cdot |\ Pr[\mathbf{Exp}_{\mathtt{S}_i}^{\mathtt{IND-DYN-1}}(1^\tau, G) = 1] - Pr[\mathbf{Exp}_{\mathtt{S}_i}^{\mathtt{IND-DYN-0}}(1^\tau, G) = 1]\ | \\ &= \sum_{i=1}^{q(n,1^\tau)} Pr[\mathtt{ADAPT}_1 \text{ chooses } k_i] \cdot \mathbf{Adv}_{\mathtt{S}_i}^{\mathtt{IND-DYN}}(1^\tau, G). \qquad (1)\end{aligned}$$

using adversary $\mathtt{B}_j$, which breaks the security of the encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ in the sense of IND-P1-C0.

Description of Algorithm $\mathtt{A}_1^{\mathcal{E}_{key}(\cdot)}(1^\tau)$.

1) First, $\mathtt{A}_1$ randomly chooses an index $1 \leq \gamma \leq q(n, 1^\tau)$, such that the private information s_γ will be implicitly set equal to the unknown *key* for its encryption oracle $\mathcal{E}_{key}(\cdot)$.
2) Afterwards, $\mathtt{A}_1$ simulates algorithm $Gen(1^\tau, G)$ as follows:
   a) If $1 \leq \gamma \leq n$, it constructs the private information s_β for all $\beta \in \{1, \dots, n\} \setminus \{\gamma\}$, as well as all encryption keys $k_1, \dots, k_n$. On the other hand, if $\gamma \geq n+1$, it constructs all private information $s_1, \dots, s_n$ as well as all encryption keys $k_1, \dots, k_n$.
   b) Then, it constructs the sequence *pub* of public values associated to all edges in the two-level hierarchy obtained from G. In particular, if $1 \leq \gamma \leq n$, $\mathtt{A}_1$ makes use of its encryption oracle $\mathcal{E}_{key}(\cdot)$ for all edges outgoing class v_γ^ℓ.
3) Moreover, $\mathtt{A}_1$ chooses at random two messages $x_0, x_1 \in \{0,1\}^\tau$: in particular, x_1 will play the role of the key k_h, assigned to class v^r, in the next steps.
4) Then, $\mathtt{A}_1$ calls adversary $\mathtt{B}_j$ on inputs 1^τ, G and *pub*. In the first stage of the attack, adversary $\mathtt{B}_j$ can make a polynomial number of *updating* and *corrupting queries*. In particular, the t-th updating query of $\mathtt{B}_j$ consists of a pair $(up^{t+1}, params^{t+1})$, where up^{t+1} is an update operation on the graph G^t and $params^{t+1}$ is a sequence of parameters associated to the update. On the other hand, the t-th corrupting query consists of a class c in the graph G^t.
   a) Each *updating query* can be answered by adversary $\mathtt{A}_1$ by means of a simulation of a modified version of the algorithm Upd, which corresponds either to Upd^j or to Upd^{j+1}. We recall that the algorithm Upd^j differs from Upd only for the way it computes the public information associated to the first j edges entering class v^r in the two-level hierarchy. More precisely, the algorithm used by $\mathtt{A}_1$ to answer updating queries, and in particular, to compute the public information associated to each edge (v_i^ℓ, v^r) works as follows:
      i) For each $i = 1, \dots, j$, $\mathtt{A}_1$ computes $p'_{(v_i,v)}$ as the encryption, with the current private information assigned to class v_i^ℓ, of the random value x_0.
      ii) If v_{j+1}^ℓ does not correspond to the γ-th class whose private information has been either inserted or modified, where γ is the index chosen at step 1, then $\mathtt{A}_1$ restarts from the beginning; otherwise, it leaves to $\mathtt{A}_2$ the task of associating the challenge y to the edge (v_{j+1}^ℓ, v^r).
      iii) For each $i = j+2, \dots, m$, $\mathtt{A}_2$ computes $p'_{(v_i,v)}$ as the encryption, with the current private information assigned to class v_i^ℓ, of the value x_1.

      On the other hand, the public information associated to each other edge is computed as follows:
      i) For each edge (v_{j+1}^ℓ, z^r), where $z \neq v$, if v_{j+1}^ℓ does not correspond to the γ-th class whose private information has been either inserted or modified, where γ is the index chosen at step 1, then $\mathtt{A}_1$ restarts from the beginning; otherwise, it uses its encryption oracle $\mathcal{E}_{key}(\cdot)$ to compute $p'_{(v_{j+1},z)}$ as the encryption, with the unknown *key*, of the current key assigned to class z^r.
      ii) For each edge (w^ℓ, z^r), where $w^\ell \neq v_{j+1}^\ell$ and $z \neq v$, $\mathtt{A}_1$ computes the public value $p'_{(w,z)}$ as the encryption, with the current private information assigned to class w^ℓ, of the current key assigned to class z^r.
   b) The t-th *corrupting query*, corresponding to a class c in G^t can be answered by adversary $\mathtt{A}_1$ with the sequence of private information held by the corrupted class c in all graphs up to G^t, if c belongs to them. We remark that $\mathtt{A}_1$ is able to respond to such a query since in step 1. it has generated the sequence of all private information $s_1, \dots, s_n$, with the exception of s_γ, corresponding to the unknown *key*, if $1 \leq \gamma \leq n$; moreover, all subsequent private information $s_{n+1}, \dots, s_{q(1^\tau,n)}$ has been generated by $\mathtt{A}_1$ when answering updating queries involving class insertions or user revocations.
   c) Upon finishing its updating and corrupting queries, adversary $\mathtt{B}_j$ outputs the triple $(v^r, t, history^-)$, where $history^-$ contains the following information:

- The initial graph G along with all updated graphs $G^1, \ldots, G^t$.
- The sequence of updating operations $up^1, \ldots, up^t$ queried by $\mathtt{B}_j$.
- The corresponding sequences of public information obtained by the interaction with $\mathtt{A}_1$ as a response for the updating queries, with the exception of the public value associated to the edge (v^{ℓ}_{j+1}, v^r).
- The corresponding sequences of keys $old\_k^0, \ldots, old\_k^{t-1}$, which have been modified according to each update. Notice that such sequences have also been obtained by the interaction with $\mathtt{A}_1$, which has generated $k_1, \ldots, k_n$ in step 2.(a), and all subsequent keys in response to *insert_class* and *replace_key* queries.
- The private information held by all corrupted classes, obtained by the interaction with $\mathtt{A}_1$ as a response for the corrupting queries.

5) Finally, $\mathtt{A}_1$ outputs the triple $(x_0, x_1, state)$, where $state$ contains the triple $(v^r, t, history^-)$ returned by $\mathtt{B}_j$.

Notice that, in step 4.(a), algorithm $\mathtt{A}_1$ needs to restart itself from the beginning in case $j+1 \neq \gamma$, where γ is the index chosen at step 1. This happens because $\mathtt{A}_1$ has no control on the sequence of updating queries asked by adversary $\mathtt{B}_j$, that is, it cannot decide in advance to which class the encryption oracle $\mathcal{E}_{key}(\cdot)$ will be associated. Therefore, $\mathtt{A}_1$ makes its guess at the beginning, by randomly choosing an index $\gamma \in \{1, \ldots, q(1^\tau, n)\}$, but it might fail. Clearly, the success probability of $\mathtt{A}_1$ is equal to $1/q(1^\tau, n)$, meaning that, in the worst case, $\mathtt{A}_1$ will need to restart itself $q(1^\tau, n)$ times.

Description of Algorithm $\mathtt{A}_2(1^\tau, y, state)$.

1) First, $\mathtt{A}_2$ parses $state$ in order to obtain the sequence $history^-$.
2) Then, $\mathtt{A}_2$ adds the missing value y in the sequence $history^-$, associating it to the edge (v^{ℓ}_{j+1}, v^r). The updated sequence of public information is denoted by $history$.
3) Afterwards, $\mathtt{A}_2$ calls adversary $\mathtt{B}_j$ on inputs 1^τ, t, v^r, $history$, and x_1.
4) Finally, $\mathtt{A}_2$ outputs the same output as $\mathtt{B}_j$.

Notice that, if y corresponds to the encryption of x_1, then the random variable associated with the adversary's view is exactly the same as the one associated with the adversary view in experiment $\mathbf{Exp}_{h,j+1}$, whereas, if y corresponds to the encryption of x_0, it has the same distribution as the one associated with the adversary's view in experiment $\mathbf{Exp}_{h,j}$. Thus, if adversary $\mathtt{B}_j$ is able to distinguish between such two views with non-negligible advantage, it follows that adversary $\mathtt{A}$ is able to break the security of the encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ in the sense of IND-P1-C0. Contradiction.

Hence, for any $j = 0, \ldots, m-1$, experiments $\mathbf{Exp}_{h,j}$ and $\mathbf{Exp}_{h,j+1}$ cannot be distinguished with non-negligible advantage. It follows that experiments $\mathbf{Exp}_{h,0}$ and $\mathbf{Exp}_{h,m}$ cannot be distinguished with non-negligible advantage, for any $h = n+1, \ldots, q(n, 1^\tau)$. Therefore, no adversary $\mathtt{S}_h$, for $h = n+1, \ldots, q(n, 1^\tau)$, distinguishes between $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}(1^\tau, G)$ with non-negligible advantage.

c) Analysis of case 2: As done for **Case 1**, we can show that no adversary $\mathtt{S}_h$, where $1 \le h \le n$, distinguishes between $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}(1^\tau, G)$ with non-negligible advantage. The proof is similar to that for **Case 1** and uses a sequence of $m+1$ slightly different experiments

$$\widehat{\mathbf{Exp}}_{h,0}, \widehat{\mathbf{Exp}}_{h,1}, \ldots, \widehat{\mathbf{Exp}}_{h,m},$$

where m denotes the number of edges entering class v^r. More precisely, let $\mu < m$ be the number of edges entering class v^r before the first updating operation; for each $j = 1, \ldots, \mu$, experiment $\widehat{\mathbf{Exp}}_{h,j}$ uses an algorithm Gen^j, which is a modified version of the algorithm Gen used in the TLEBC. Such an algorithm differs from Gen for the way it computes the public information associated to the first j edges entering class v^r in the two-level hierarchy. On the other hand, for each $\mu + 1 \le j \le m$, experiment $\widehat{\mathbf{Exp}}_{h,j}$ first uses the algorithm Gen^μ to compute the public information associated to the first μ edges entering v^r, then uses the algorithm Upd^j, described when analyzing **Case 1**, in order to compute the public information associated to edges from $\mu + 1$ to j. As done before, we can show that experiments $\widehat{\mathbf{Exp}}_{h,0}$ and $\widehat{\mathbf{Exp}}_{h,m}$ cannot be distinguished with non-negligible advantage, for any $h = 1, \ldots, n$. Therefore, no adversary $\mathtt{S}_h$, for $h = 1, \ldots, n$, distinguishes between $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}(1^\tau, G)$ with non-negligible advantage.

To conclude, we have proven that no adversary $\mathtt{S}_h$, for $h = 1, \ldots, q(n, 1^\tau)$, distinguishes between $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{S}_h}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{S}_h}(1^\tau, G)$ with non-negligible advantage. Therefore, no dynamic adaptive adversary ADAPT has non-negligible advantage in distinguishing between experiments $\mathbf{Exp}^{\mathtt{IND-DYN-1}}_{\mathtt{ADAPT}}(1^\tau, G)$ and $\mathbf{Exp}^{\mathtt{IND-DYN-0}}_{\mathtt{ADAPT}}(1^\tau, G)$. Thus, the TLEBC is secure in the sense of IND-DYN-AD. ■

2) Performance of the Scheme: In this section we analyze the performance of the TLEBC according to several parameters, such as space requirements for public and private information, efficiency of key derivation and updating procedures, and security. In detail, regarding the space requirements, the TLEBC needs a public value for each edge in the graph G_{TL} used in the construction. More precisely, the number of public values is $|E_{TL}| = O(|E^*|)$, where $G^* = (V, E^*)$ is the directed graph that can be obtained from $G = (V, E)$ by adding to E all self-loops and edges which are implied by the property of the transitive closure. On the other hand, each user belonging to a certain class has to store a single secret value. Moreover, users are required to perform a single decryption in order to derive a key.

Regarding the efficiency of key derivation, we need to specify which kind of encryption to use in our construction. In order to obtain a hierarchical key assignment scheme that is secure with respect to IND-DYN-AD, we can use the IND-P1-C0 secure symmetric encryption scheme, called the *XOR construction*, proposed in [51]. This construction makes use of a function family $\mathcal{F}$. Assuming that $\mathcal{F}$ is a pseudo-random function family (which can be constructed from any one-way function [52]), the XOR construction has been shown to be secure in the sense of IND-P1-C0 (see [48], [51]). The most efficient constructions for pseudorandom function families were proposed by Naor and Reingold [53]. In their proposals, the cost of evaluating a pseudo-random function is comparable to two modular exponentiations. In practice, for what concerns realistic applications, an efficient implementation of the resulting hierarchical key assignment scheme could be obtained by using the HMAC [54] to realize the function family $\mathcal{F}$. We emphasize that the HMAC can be instantiated through secure hash functions, and many of them can be implemented efficiently directly on hardware, as for example the SHA-3 Cryptographic Hash Standard, called Keccak, recently released by NIST [55].

The TLEBC construction supports dynamic changes to the hierarchy by means of local updates to the public information, without requiring the re-distribution of private information to classes affected by such changes (except for user revocation, where such a re-distribution is needed). This also happens in the Extended Construction in [10], in the TLEBC in [25] and in the DEBC in [14]. In the following we compare our scheme with respect to the above cited ones. In particular, we clarify that our TLEBC construction is the same as the one proposed in [25] when considering a single period of time, thus no comparison between them is needed. As for the Extended Construction in [10], our scheme is better, since it benefits from the two-level feature of the hierarchy, after the graph transformation performed by the *Gen* algorithm. Indeed, such a feature allows to reduce the number of local updates to the public information following dynamic updates. For example, whenever an edge (u, v) has to be deleted from the hierarchy, only a key replacement for the key k_v, which requires the computation of a single encryption later added to the public information, is needed in our scheme. On the other hand, the Extended Construction in [10] requires a pseudorandom function evaluation and an encryption operation for each class lower down class v. Moreover, whenever a key replacement operation for a class v is needed, the extended scheme in [10] also requires, besides local updates to the public information, the update of the secret information s_v, whereas, this does not happen in our construction. For the same reasons already outlined before, compared to the DEBC in [14], our construction requires a smaller number of local updates to the public information after a dynamic update.

IV. Conclusions

In this work we have considered hierarchical key assignment schemes supporting dynamic updates, such as insertions and deletions of classes and relations between classes, as well as key replacements and user revocations. We have extended existing security notion for hierarchical key assignment schemes, namely, security with respect to *key indistinguishability*, by providing the adversary with further attack abilities. Then, we have shown how to construct a hierarchical key assignment scheme supporting dynamic updates by using as a building block a symmetric encryption scheme. It is important to emphasize that this is the first available scheme crafted for non-static environments, where the adversary is allowed to dynamically update the hierarchy. The proposed construction is provably secure with respect to key indistinguishability and requires a single computational assumption. Moreover, it provides efficient key derivation and updating procedures, while requiring each user to store only a single private key. For its simplicity, effectiveness and robustness the proposed scheme may result in a fundamental practice for hierarchical access control applications in dynamic scenarios.

References

[1] S. G. Akl and P. D. Taylor, “Cryptographic solution to a problem of access control in a hierarchy,” *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 239–248, 1983.

[2] M. J. Atallah, K. B. Frikken, and M. Blanton, “Dynamic and efficient key management for access hierarchies,” in *Proc. 12th ACM Conf. Comput. Commun. Security, CCS*, Alexandria, VA, USA, Nov. 2005, pp. 190–202.

[3] L. Harn and H.-Y. Lin, “A cryptographic key generation scheme for multilevel data security,” *Comput. Security*, vol. 9, no. 6, pp. 539–546, Oct. 1990.

[4] H. Min-Shiang, “A cryptographic key assignment scheme in a hierarchy for access control,” *Math. Comput. Model.*, vol. 26, no. 2, pp. 27–31, Jul. 1997.

[5] H. T. Liaw, S. J. Wang, and C. L. Lei, “A dynamic cryptographic key assignment scheme in a tree structure,” *Comput. Math. Appl.*, vol. 25, no. 6, pp. 109–114, 1993.

[6] C.-H. Lin, “Dynamic key management schemes for access control in a hierarchy,” *Comput. Commun.*, vol. 20, no. 15, pp. 1381–1385, 1997.

[7] S. J. Mackinnon, P. D. Taylor, H. Meijer, and S. G. Akl, “An optimal algorithm for assigning cryptographic keys to control access in a hierarchy,” *IEEE Trans. Comput.*, vol. 34, no. 9, pp. 797–802, Sep. 1985.

[8] R. S. Sandhu, “Cryptographic implementation of a tree hierarchy for access control,” *Inf. Process. Lett.*, vol. 27, no. 2, pp. 95–98, 1988.

[9] Y.-F. Chang, “A flexible hierarchical access control mechanism enforcing extension policies,” *Security Commun. Netw.*, vol. 8, no. 2, pp. 189–201, 2015.

[10] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken, “Dynamic and efficient key management for access hierarchies,” *ACM Trans. Inf. Syst. Security*, vol. 12, no. 3, 2009, Art. no. 18.

[11] A. De Santis, A. L. Ferrara, and B. Masucci, “Efficient provably-secure hierarchical key assignment schemes,” in *Proc. 32nd Int. Symp., Math. Found. Comput. Sci. MFCS*, vol. 4708. Ceský Krumlov, Czech Republic, 2007, pp. 371–382.

[12] P. D’Arco, A. De Santis, A. L. Ferrara, and B. Masucci, “Security and tradeoffs of the Akl–Taylor scheme and its variants,” in *Proc. 34th Int. Symp. Math. Found. Comput. Sci. MFCS*, vol. 5734. Novy Smokovec, High Tatras, Slovakia, 2009, pp. 247–257.

[13] P. D’Arco, A. De Santis, A. L. Ferrara, and B. Masucci, “Variations on a theme by Akl and Taylor: Security and tradeoffs,” *Theoretical Comput. Sci.*, vol. 411, no. 1, pp. 213–227, 2010.

[14] A. De Santis, A. L. Ferrara, and B. Masucci, “Efficient provably-secure hierarchical key assignment schemes,” *Theoretical Comput. Sci.*, vol. 412, no. 41, pp. 5684–5699, 2011.

[15] E. S. V. Freire and K. G. Paterson, “Provably secure key assignment schemes from factoring,” in *Proc. 16th Australasian Conf. Inf. Security Privacy (ACISP)*, vol. 6812. Melbourne, Australia, Jul. 2011, pp. 292–309.

[16] E. S. V. Freire, K. G. Paterson, and B. Poettering, “Simple, efficient and strongly KI-secure hierarchical key assignment schemes,” in *Proc. 13th Int. Conf. Topics Cryptol. (RSA)*, vol. 7779. San Francisco, CA, USA, 2013, pp. 101–114.

[17] A. Castiglione, A. De Santis, and B. Masucci, “Key indistinguishability vs. strong key indistinguishability for hierarchical key assignment schemes,” *IEEE Trans. Dependable Secure Comput.*, in press, doi: http://dx.doi.org/10.1002/sec.1463

[18] M. Cafaro, R. Civino, and B. Masucci, “On the equivalence of two security notions for hierarchical key assignment schemes in the unconditional setting,” *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 485–490, Jul./Aug. 2015.

[19] A. De Santis, A. L. Ferrara, and B. Masucci, “Unconditionally secure key assignment schemes,” *Discrete Appl. Math.*, vol. 154, no. 2, pp. 234–252, 2006.

[20] A. L. Ferrara and B. Masucci, "An information-theoretic approach to the access control problem," in *Proc. 8th Italian Conf., Theoretical Comput. Sci. ICTCS*, vol. 2841. Bertinoro, Italy, 2003, pp. 342–354.

[21] A. De Santis, A. L. Ferrara, and B. Masucci, "A new key assignment scheme for access control in a complete tree hierarchy," in *Coding and Cryptography* (Lecture Notes in Computer Science), vol. 3969, Ø. Ytrehus, Ed. Bergen, Norway: Springer, 2005, pp. 202–217.

[22] G. Ateniese, A. De Santis, A. L. Ferrara, and B. Masucci, "Provably-secure time-bound hierarchical key assignment schemes," in *Proc. 13th ACM Conf. Comput. Commun. Security CCS*, Alexandria, VA, USA, 2006, pp. 288–297.

[23] M. J. Atallah, M. Blanton, and K. B. Frikken, "Incorporating temporal capabilities in existing key management schemes," in *Proc. 12th Eur. Symp. Res. Comput. Security*, vol. 4734. Dresden, Germany, 2007, pp. 515–530.

[24] A. De Santis, A. L. Ferrara, and B. Masucci, "New constructions for provably-secure time-bound hierarchical key assignment schemes," *Theoretical Comput. Sci.*, vol. 407, nos. 1–3, pp. 213–230, 2008.

[25] G. Ateniese, A. De Santis, A. L. Ferrara, and B. Masucci, "Provably-secure time-bound hierarchical key assignment schemes," *J. Cryptol.*, vol. 25, no. 2, pp. 243–270, 2012.

[26] J.-S. Pan, T.-Y. Wu, C.-M. Chen, and E. K. Wang, "An efficient solution for time-bound hierarchical key assignment scheme," in *Genetic and Evolutionary Computing*. Switzerland: Springer, 2015, pp. 3–9.

[27] C.-C. Lee, C.-T. Li, S.-T. Chiu, and S.-D. Chen, "Time-bound key-aggregate encryption for cloud storage," *Security Commun. Netw.*, 2016.

[28] A. De Santis, A. L. Ferrara, and B. Masucci, "New constructions for provably-secure time-bound hierarchical key assignment schemes," in *Proc. 12th ACM Symp. Access Control Models Technol. (SACMAT)*, Sophia Antipolis, France, 2007, pp. 133–138.

[29] A. De Santis, A. L. Ferrara, and B. Masucci, "Enforcing the security of a time-bound hierarchical key assignment scheme," *Inf. Sci.*, vol. 176, no. 12, pp. 1684–1694, 2006.

[30] A. Castiglione, A. De Santis, and B. Masucci, "Hierarchical and shared key assignment," in *Proc. IEEE 7th Int. Conf. Netw.-Based Inf. Syst. (NBIS)*, Sep. 2014, pp. 263–270.

[31] A. Castiglione *et al.*, "Hierarchical and shared access control," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 850–865, Apr. 2016.

[32] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Selective data encryption in outsourced dynamic environments," *Electron. Notes Theoretical Comput. Sci.*, vol. 168, pp. 127–142, Feb. 2007.

[33] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *Proc. 33rd Int. Conf. Very Large Data Bases*, Vienna, Austria, 2007, pp. 123–134.

[34] C. Blundo *et al.*, "Managing key hierarchies for access control enforcement: Heuristic approaches," *Comput. Security*, vol. 29, no. 5, pp. 533–547, 2010.

[35] J.-W. Lo, M.-S. Hwang, and C.-H. Liu, "An efficient key assignment scheme for access control in a large leaf class hierarchy," *Inf. Sci.*, vol. 181, no. 4, pp. 917–925, 2011.

[36] C.-H. Lin, "Hierarchical key assignment without public-key cryptography," *Comput. Security*, vol. 20, no. 7, pp. 612–619, 2001.

[37] V. R. L. Shen and T.-S. Chen, "A novel key management scheme based on discrete logarithms and polynomial interpolations," *Comput. Security*, vol. 21, no. 2, pp. 164–171, 2002.

[38] C. Yang and C. Li, "Access control in a hierarchy using one-way hash functions," *Comput. Security*, vol. 23, no. 8, pp. 659–664, 2004.

[39] T.-S. Chen and J.-Y. Huang, "A novel key management scheme for dynamic access control in a user hierarchy," *Appl. Math. Comput.*, vol. 162, no. 1, pp. 339–351, 2005.

[40] A. V. D. M. Kayem, S. G. Akl, and P. Martin, "On replacing cryptographic keys in hierarchical key management systems," *J. Comput. Security*, vol. 16, no. 3, pp. 289–309, 2008.

[41] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 131–137, 1972.

[42] M. J. Atallah, M. Blanton, and K. B. Frikken, "Key management for non-tree access hierarchies," in *Proc. 11th ACM Symp. Access Control Models Technol. (SACMAT)*, Lake Tahoe, CA, USA, 2006, pp. 11–18.

[43] J. A. L. Poutré and J. van Leeuwen, "Maintenance of transitive closures and transitive reductions of graphs," in *Graph-Theoretic Concepts in Computer Science* (Lecture Notes in Computer Science), vol. 314, H. Göttler and H. J. Schneider, Eds. Bad Staffelstein, Germany: Springer, 1987, pp. 106–120.

[44] G. F. Italiano, "Finding paths and deleting edges in directed acyclic graphs," *Inf. Process. Lett.*, vol. 28, no. 1, pp. 5–11, 1988.

[45] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[46] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.

[47] A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, and A. Castiglione, "On the relations between security notions in hierarchical key assignment schemes for dynamic structures," in *Proc. 21st Australasian Conf. Inf. Security Privacy (ACISP)*, vol. 9723. Melbourne, Australia, 2016, pp. 1–18.

[48] J. Katz and M. Yung, "Characterization of security notions for probabilistic private-key encryption," *J. Cryptol.*, vol. 19, no. 1, pp. 67–95, 2006.

[49] A. De Santis, A. L. Ferrara, and B. Masucci, "Cryptographic key assignment schemes for any access control policy," *Inf. Process. Lett.*, vol. 92, no. 4, pp. 199–205, 2004.

[50] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, no. 4, pp. 850–864, 1984.

[51] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *Proc. 38th IEEE Symp. Found. Comput. Sci.*, Oct. 1997, pp. 394–403.

[52] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[53] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004.

[54] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proc. 16th CRYPTO*, 1996, pp. 1–15.

[55] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak," in *"Proc. Eurocrypt 2013-32nd Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, vol. 7881. Athens, Greece, 2013, pp. 313–314.