Group Key Management: From a Non-hierarchical to a Hierarchical Structure

Sébastien Canard and Amandine Jambert

Orange Labs R&D, 42 rue des Coutures, BP6243, F-14066 Caen Cedex, France
{sebastien.canard,amandine.jambert}@orange-ftgroup.com

Abstract. Since the very beginnings of cryptography many centuries ago, key management has been one of the main challenges in cryptographic research. In case of a group of players wanting to share a common key, many schemes exist in the literature, managing groups where all players are equal or proposing solutions where the group is structured as a hierarchy. This paper presents the first key management scheme suitable for a hierarchy where no central authority is needed and permitting to manage a graph representing the hierarchical group with possibly several roots. This is achieved by using a HMAC and a non-hierarchical group key agreement scheme in an intricate manner and introducing the notion of virtual node.

Keywords: Key Management, Access Control, Hierarchy.

1 Introduction

Key management scheme is one of the fundamental cryptographic primitive after encryption and digital signature. Such scheme allows e.g. two parties to securely exchange information among them. A running direction of research on key management is to generalize two party key agreement schemes to multi party setting, where a group of users try to create cryptographic keys together.

There are currently two main approaches regarding this generalization, depending on the structure of the group. In some cases, all members of the group are considered equals and each of them participates approximately at the same level to the construction of a cryptographic key that is finally shared by all members: this is called "group key management". Many papers exist in the literature in this case and their aim is to make the better generalization of the seminal Diffie-Hellman paper, dealing with authentication or group's dynamicity.

The second approach deals with hierarchy-based access control where members of the group are related one to another by a subordination relation while trying to access some protected documents. In this case, the group is most of the time represented as an oriented graph with no oriented cycle. In this setting, there is one key per group member and the main issue is then to provide a hierarchy of the keys in such a way that it is possible for a group member to derive from her own key all the keys that are lower in the graph. In this case, the dynamicity of the group concerns either the possibility to add or delete nodes in

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 213–225, 2008.

the graph, or the capacity to modify the key of a particular node. Ideally, these modifications imply the modification of a minority of node keys.

1.1 Related Work

The first work on this problem of key management in a hierarchy was by Akl and Taylor in 1983 [1]. Since then a large number of papers have been published [2, 4, 6, 7, 8, 9, 10, 12, 13, 17, 19, 21, 23, 25, 26, 27, 28, 29] and they can be divided into several families.

The first family contains the original paper of Akl and Taylor and its different improvements [1, 7, 10, 13, 17, 21]. These protocols use a Central Authority (CA) to generate keys and related public data. The dynamicity of the graph is not always possible in these proposals, and even in this case, a modification implies the recalculation of the keys of some predecessors. The second family is based on Sibling Intractable Function Family (SIFF) [12, 27]. While these solutions use a CA for generation and dynamism of the graph, their low complexity is quite attractive. The main problem comes from the difficulty to decide if a practical algorithm to generate such function exists or not (even in the literature [2, 22, 28]). The third group of papers uses polynomial interpolation [6, 9, 29] but [6] and [9] do not consider the dynamicity of the group, and the way to update keys in [29] is relatively inefficient. In the last group of papers [2, 4, 8, 19, 26, 28], the keys are randomly generated and the role of the CA is to provide the public link between them. Different possibilities are proposed and the best ones only use low cost operations as hash functions or xor operations. Two other papers [23, 25] use many modular exponentiations and thus induce a high complexity. Note that the solution in [25], even if presented with a CA, can be described without.

Mainly all these proposals use a Central Authority and only consider the case of a rooted graph. It is thus an open problem to describe an efficient graph key management in a multi-rooted oriented graph where (i) no Central Authority is needed and (ii) in which we can manage dynamic graphs.

1.2 Our Contribution

Our main idea in the construction of a graph key management is that we use at the same time two different solutions, depending on the structure of the subgraph we are considering. More precisely, the method to compute the key of a node in the graph depends on the number of fathers this node has. If there is one father, we use a Message Authentication Code (MAC) function on input the key of the father, a counter enumerating (approximately) the number of children of the father and a security constant.

The case where a node has several fathers cannot be treated as the case of one father and we thus adopt a different approach which consists in using group key agreement for a non-hierarchical group (in our case, the group of the fathers). More precisely, we introduce the concept of Refreshable and Replayable Group Key Agreement (R&R-GKA) schemes where the main difference with a traditional GKA scheme is that the internal state information is not truly composed of ephemeral secret information using random data, as it is the case

in existing GKA schemes. Moreover, we require an additional algorithm to replay the creation of the shared key using one private information and some public data, and we finally need a refresh method that permits to renew the shared key with a minimal effect on player's keys.

Our second trick is used when several fathers have several children in common. In that case, we introduce a virtual node between fathers and children so as to speed up the generation phase by using the "one-father method".

1.3 Organization of the Paper

The paper is organized as follows. After the present introduction, we set up in Section 2 our model for key management in an oriented graph. The cryptographic primitives we will use later are given in Section 3. Section 4 presents our scheme and its security arguments. Finally, we provide a conclusion in Section 5 and the bibliography afterward.

2 Problem and Model

The problem of access control in a hierarchy appears when users get different rights on common resources. As an example, workers in a company use common resources but according to their positions or their departments, they are not allowed to access the same documents. Another example could be on-line newspapers: different subscriptions lead to different rights.

For our part, we study the cryptographic aspect of access control. We represent the hierarchy by a graph and we look at access control as a problem of key management in that graph.

2.1 Notation

We consider an oriented *graph* denoted $G = \{N, A\}$ where $N = \{n_1, n_2, ..., n_l\}$, of cardinality l, is the set of *nodes* (in the following a node is denoted either n_i or simply i) and $A = \{a_1, a_2, ..., a_m\}$, of cardinality m, is the set of *edges*, such that there is no oriented cycles. An edge $a \in A$ corresponds to a couple of nodes (n_i, n_j), representing the fact that there is an edge going from node n_i to node n_j. n_i is called the father and n_j is the child. We denoted by F_i (resp. C_i) the number of fathers (resp. children) of the node n_i. The set of fathers of node n_i is denoted by $\mathcal{F}_i = \{f_i[1], \cdots, f_i[F_i]\}$ and the set of children of a node n_j is denoted $\mathcal{C}_j = \{c_j[1], \cdots, c_j[C_j]\}$.

A *path* $P = \{a_1, \cdots, a_k\}$ of cardinality k is a set of edges where for all $i \in \{1, \cdots, k-1\}$, if $a_i = (n_{i_0}, n_{i_1})$ and $a_{i+1} = (n_{i_2}, n_{i_3})$, then $n_{i_1} = n_{i_2}$: we also talk of the path from the first node to the last one.

If there is a path from node n_i to node n_j, we say that n_i is an *ascendant* of n_j and that n_j is a *descendant* of n_i. We denote by $\mathcal{D}_i$ the set of descendant nodes of node n_i and by $\mathcal{A}_j$ the set of ascendant nodes of node n_j. Note that $\mathcal{F}_i \subset \mathcal{A}_j$ and $\mathcal{C}_i \subset \mathcal{D}_j$.

Each node j represents a subgroup of members that share the same secret cryptographic key, named the node key and denoted k_{n_j} (or simply k_j) related to a public value pv_{n_j} (or simply pv_j). In the following, we consider a subgroup as a unique entity to avoid some authentication problems for which it exists well-known techniques. As we consider oriented graphs, we have a hierarchy between nodes. As a consequence, a node key k_{n_j} should be computable by all members of subgroups/nodes that belong to $\mathcal{A}_j$.

2.2 Actors and Procedures

We present in this section a formal definition of a graph key management scheme for a graph G. A graph key management scheme implies a set $\mathcal{P}$ of l players denoted $\mathcal{P}_1, \cdots, \mathcal{P}_l$. Each player $\mathcal{P}_i$ corresponds to a node i in the graph. In the following, we consider that the graph representation G is known by all players of the system.

Definition 1. *A Graph Key Management scheme (noted GKM) consists in the following algorithms:*

- *Setup is an algorithm which on input a security parameter τ generates the set of parameters of the system Γ. We now consider that the security parameter τ belongs to Γ.*
- *UserSetup is an algorithm which on input the set of parameters Γ provides each player in $\mathcal{P}$ with a long-lived key pair (sk_i, pk_i). From now on, Γ includes the public keys pk_i of all players.*
- *KeyGeneration is an algorithm which launches a protocol between all players $\mathcal{P}_1, \cdots, \mathcal{P}_l$, each of them taking on input the parameters Γ of the system and the long-lived key pair (sk_i, pk_i). Each player secretly outputs the first instance of the key related to its node, denoted $k_i[0]$. The algorithm outputs the first instance of some related public elements denoted $PE[0]$.*
- *KeyDerivation is an algorithm which on input the parameters Γ, a node j, a player $\mathcal{P}_i$ and an instance ρ provides the player $\mathcal{P}_i$ using the ρ-th instance of her node key $k_i[\rho]$ and the corresponding public elements $PE[\rho]$ with either an error message $\perp$ if $i \notin \mathcal{A}_j$ or the corresponding ρ-th instance of the key of node j, that is $k_j[\rho]$.*
- *KeyRefresh is an algorithm which on input the node j that needs to be refreshed launches a protocol between all players $\mathcal{P}_1, \cdots, \mathcal{P}_l$. Each player takes on input the parameters Γ, the current instance ρ, their corresponding node key $k_i[\rho]$ and the corresponding public elements $PE[\rho]$ and secretly outputs the new instance of the node key, denoted $k_i[\rho + 1]$. The algorithm outputs the new instance of some related public elements denoted $PE[\rho + 1]$.*

Remark 1. The efficiency of the KeyRefresh algorithm is a really important issue and if a particular node needs to be refreshed, this procedure should not (and needs not to) modify all the keys in the graph. The best configuration is when only the keys of the descendant nodes are modified. Note also that, for simplicity reasons, we consider in our model that all the keys change their version during this procedure, even if the new version may be equal to the previous one for some particular nodes.

2.3 Security Properties

A Graph Key Management scheme must have the Key Recovery security property. This corresponds to the fact that any coalition of players can't recover the key of a node which does not belong to their descendants.

The Key Recovery property corresponds to the following Experiment.

Experiment $Exp^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}$:

1. the challenger $\mathcal{C}$ initializes the system and sends the graph to $\mathcal{A}$.
2. $\mathcal{A}$ interacts with the system by generating and refreshing (player) keys, corrupting players and/or keys. At any time of the experiment, it must remain at least one key which is not corrupted. A key is considered as corrupted if at least one of its ascendant is corrupted or if the player corresponding to this node has beforehand been corrupted.
3. $\mathcal{A}$ finally outputs the identifier of the graph key management π, a node i, an instance ρ and an uncorrupted node key k.

We define the success of an adversary $\mathcal{A}$ for this experiment as:

$$Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau) = Pr\left[k = k_i[\pi, \rho]\right].$$

Definition 2 (Key Recovery). *We say that a GKM scheme satisfies the Key Recovery property if* $Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau)$ *is negligible.*

Remark 2. Note that this security model is stronger than the one given in e.g. [2] since this is the adversary who chooses the node he wants to focus on. In [2], a challenger chooses one particular node and the adversary has to output the key of this node. Note also that it is not possible to use a decisional experiment in graph key management where the aim of the adversary is to distinguish a true key from a random one (as it is done for many other key agreement primitives) since it is enough for the adversary to corrupt a descendant node and checks the consistency of the key derivation to win such game.

3 Useful Tools

3.1 The HMAC Functions

A cryptographic message authentication code (MAC) is a cryptographic tool used to authenticate a message and belongs to the family of symmetric cryptography. A *MAC scheme* is composed of a key generation algorithm KeyGen which permits to generate the MAC key denoted K. The code generation algorithm MAC accepts as input the secret key K and an arbitrary-length message m and outputs the message authentication code for message m, under the secret key K: $\Sigma = \mathsf{MAC}(K, m)$. Finally, the code verification algorithm VerMAC takes as input a message m, the secret key K and a message authentication code $\Sigma \in \mathcal{C}$ and outputs 1 if $\Sigma = \mathsf{MAC}(K, m)$ and 0 otherwise.

To be considered as secure, a MAC scheme should resist to existential forgery under chosen-plaintext attacks (EF-CMA), which means that even if an adversary $\mathcal{A}$ has access to an oracle which possesses the secret key and generates MACs for messages chosen by the adversary, $\mathcal{A}$ is unable to guess the MAC for a message it did not query to the oracle.

In our graph key management scheme (see Section 4), the used MAC scheme needs furthermore the pseudorandomness property, which says that an adversary is unable to distinguish the output of a Pseudo-Random Function (PRF) from a true random value. As a consequence, we will use the HMAC construction [20] which has been proved to be a PRF by Bellare [3].

3.2 The Notion of Refreshable and Replayable Group Key Agreement

A Group Key Agreement (GKA) scheme is a mechanism which permits to establish a cryptographic key shared by a group of participants, based on each one's contribution, over a public network. It exists several GKA schemes in the literature, using either an authenticated mode [5] or not [15, 16, 18, 24]. Note that it is possible to transform any unauthenticated protocol to an authenticated one using generic methods [11, 14].

In fact, in this paper, we need a GKA with some additional properties that are naturally verified by many of these schemes. We thus introduce the notion of Refreshable and Replayable Group Key Agreement (R&R-GKA) scheme. The main difference with a traditional GKA scheme is that the internal state information is not truly composed of ephemeral secret information using random data. Here, each player has a long-lived key to participate to the protocol but also another "personal" secret key used to create the shared one and replacing the ephemeral secret information. This new secret key can not be considered as "long-lived" since it can be refreshed when necessary. Moreover, a R&R-GKA has the following properties, which are reached most of the time by GKA schemes:

- it is a contributory Group Key Agreement protocol (GKA) [16],
- we require an additional deterministic algorithm which accepts previously fixed inputs and which is (once initialized) replayable by any player using his private information and some public data,
- it should contains a refresh method that permits to renew the shared key with a minimal effect on player's personal secret keys.

Procedures. More formally, we have the following definition, which is derived from [5]. Let $\mathcal{P}$ be the set of potential players for the GKA, that is, $\mathcal{P} = \{\mathcal{P}_1, \cdots, \mathcal{P}_l\}$.

Definition 3. *A R&R-GKA scheme consists in the following algorithms:*

- *Setup is an algorithm which on input τ generates the set of parameters of the system Γ. We now consider that the security parameter τ belongs to Γ.*
- *UserSetup is an algorithm which on input Γ provides each player in $\mathcal{P}$ with a long-lived key pair (sk_i, pk_i). Γ now includes the players public keys pk_i.*

- *KeyGeneration is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$ of players secretly provides each player in $\mathcal{I}$ a first instance of a personal secret key $k_i[\mathcal{I}, 0]$ related to $\mathcal{I}$. This algorithm then launches a protocol between all players in $\mathcal{I}$, each of them taking on input Γ, the long-lived key pair (sk_i, pk_i) and their personal secret key $k_i[\mathcal{I}, 0]$. Each player secretly outputs the first instance of the shared secret key of the set $\mathcal{I}$ denoted $K[\mathcal{I}, 0]$. The algorithm also outputs the first instance of some related public elements denoted $PE[\mathcal{I}, 0]$.*
- *KeyRefresh is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$ and a subset $\mathcal{J} \subset \mathcal{I}$ of players, secretly provides each player in $\mathcal{J}$ with a new instance of her personal secret key related to $\mathcal{I}$ denoted $k_i[\mathcal{I}, \rho + 1]$, if ρ is the current instance. Each player in $\mathcal{I} \setminus \mathcal{J}$ sets $k_i[\mathcal{I}, \rho + 1] = k_i[\mathcal{I}, \rho]$. This algorithm then launches a protocol between all players in $\mathcal{I}$, each of them taking on input the set of parameters Γ, the long-lived key pair (sk_i, pk_i), the two instances of their personal secret key $k_i[\mathcal{I}, \rho]$ and $k_i[\mathcal{I}, \rho + 1]$, $K[\mathcal{I}, \rho]$ and $PE[\mathcal{I}, \rho]$. Each player secretly outputs the new instance of the shared secret key of the set $\mathcal{I}$ denoted $K[\mathcal{I}, \rho + 1]$. The algorithm also outputs the new instance of the public elements denoted $PE[\mathcal{I}, \rho + 1]$.*
- *KeyRetrieve is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$, a player $\mathcal{P}_i \in \mathcal{I}$ and an instance ρ, provides the player $\mathcal{P}_i$ taking on input the parameters Γ, the ρ-th instance of her personal secret key $k_i[\mathcal{I}, \rho]$ and the corresponding public elements $PE[\mathcal{I}, \rho]$ with the corresponding ρ-th instance of the common secret for $\mathcal{I}$, that is $K[\mathcal{I}, \rho]$.*

Note that the UserSetup procedure is done only once whereas the KeyGeneration one can be done several times, possibly in a concurrent manner.

Security property. It is commonly believed that the best security property for group key agreement is the Key Independence one (also known as Authenticated Key Exchange (AKE) property), where the aim of the adversary is to distinguish a true shared key from a random value. In this paper, we need to be sure that an adversary can not learn a non-corrupted instance of the personal secret key of a player. Since the adversary has access to the KeyRetrieve method, this is obviously related to the Key Recovery property, which says that the adversary can not compute a non-corrupted instance of the shared key.

4 Our Key Management Scheme

In this section, we present our solution of key management in an oriented graph structure. We first give an overview and then detail all procedures. Note that it is possible to use our method either in a centralized or in a distributed mode. In the first case, the roots generate all the keys and finally distribute them to all nodes. In the latter case, all nodes participate in the generation of the keys. In both cases, it is possible to construct the keys of the hierarchy while all players are not necessarily connected all the time.

4.1 Overview of Our Solution

One of our main ideas in the construction of a graph key management is that we use at the same time two different solutions, depending on the structure of the subgraph we are considering. More precisely, the method to compute the key of a child depends on the number of fathers this child has. We thus describe the two possible methods.

The case of one father. In this case, we use a simple HMAC function. Let s_i be a counter specific to the node i. This counter represents the number of times the node i has computed a new key using his own. s_i is related to the number of children, the number of refresh and potentially the dynamicity of the graph (see Section 4.5) and is maintained by the node. For each new child, the node i computes the HMAC function using its key k_i and the message corresponding to the concatenation of the counter s_i and a random constant number $C \in \{0,1\}^\tau$ specific to the graph. After that, this counter is incremented for the next child.

The case of several fathers. Here, we adopt a different approach which consists in using a non-hierarchical group key agreement (GKA) scheme. Let us consider a node i having several fathers $f_i[1], \ldots, f_i[F_i]$, where F_i is the number of fathers. Each father will be a player in the GKA scheme. By construction, each node is consequently related to a node key which will be used as a personal secret key in the GKA scheme.

As this shared value should be first computed interactively but also needs to be recalculated non-interactively, it should be possible for a father to use his node key and some public values to compute off-line the key of his child: we consequently need a R&R-GKA scheme such as described in the previous section.

Virtual nodes. The problem with the above technique is firstly that the known group key agreement schemes are deterministic and secondly that it implies many computations for all actors. Our second trick is used when two or more fathers have several children in common. In that case, we introduce a virtual node between fathers and children so as to speed up the generation phase.

This virtual node v is inserted between the fathers (f_1, f_2 and f_3) and the children (c_1 and c_2), as shown in Figure 1.

This new node is also related to a cryptographic key denoted k_v, computed from the keys of the fathers using the above method based on group key agreement schemes. Next, from this virtual key k_v it is possible to compute the keys for all children using the "one father" method, since this virtual node becomes the unique virtual father of several children.

4.2 Detail Procedures

We are now able to describe in details the different algorithms and protocols of our Graph Key Management scheme. Let τ be the security parameter, M be a secure MAC such as defined in the previous section and GKA be a secure

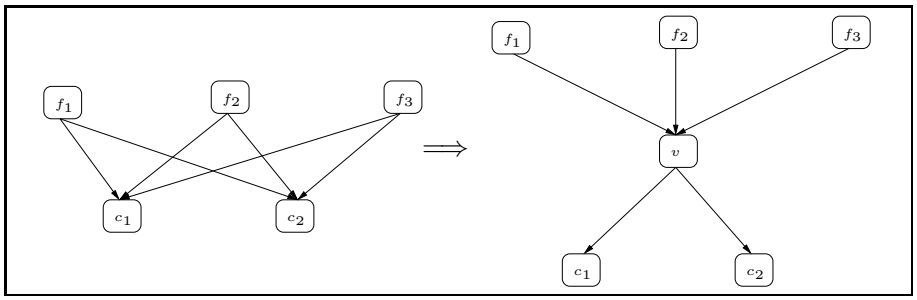

Fig. 1. The case of several fathers having several children

R&R-GKA scheme such as defined previously. Let $G = \{N, A\}$ be a graph where $N = \{1, 2, \cdots, l\}$ and $A = \{a_1, a_2, \cdots, a_m\}$. In the following, a node is tag as *keyed* when its key has been computed.

- $\mathsf{Setup}(1^\tau)$: this algorithm consists first in choosing at random a value $C \in \{0,1\}^\tau$ and second in executing the $GKA.\mathsf{Setup}(1^\tau)$ procedure which outputs $GKA.\Gamma$. The output of this algorithm is then $\Gamma = (C, GKA.\Gamma)$. This procedure also modifies the graph to insert virtual nodes, such as explained above and described in Figure 1. We denote $\tilde{G} = \{\tilde{N}, \tilde{A}\}$ the new graph with, by convention, $\tilde{N} = \{1, 2, \cdots, \tilde{l}\}$ and $\tilde{A} = \{\tilde{a}_1, \cdots, \tilde{a}_m\}$. Note that taken on input the initial graph G, the new graph $\tilde{G}$ is unique.
- $\mathsf{UserSetup}(\Gamma)$: it consists in executing the $GKA.\mathsf{UserSetup}(GKA.\Gamma)$ procedure which provides each player with a long-lived key-pair (sk_i, pk_i). All public keys are included in $GKA.\Gamma$ and thus in Γ.
- $\mathsf{KeyGeneration}()$: for each node $i \in \tilde{N}$, there are several cases.
  - i has no father in the graph: the node key $k_i[0]$ is chosen at random in $\{0,1\}^\tau$. There is no corresponding public value in this case.
  - i has one father $f \in \tilde{N}$ in the graph: let s_f be the number of current keyed children of f. Then

$$k_i[0] = M.\mathsf{MAC}(k_f[0], C \| s_f + 1).$$

    The new number of keyed children $s_f + 1$ for node f concatenated with the focused node i corresponds to the related public information $pk_i = s_f + 1 \| i$ in this case.
  - i has F_i fathers $(f_1, \cdots, f_{F_i})$: they execute the $GKA.\mathsf{KeyGeneration}$ procedure on input $\mathcal{I} = \{f_1, \cdots, f_{F_i}\}$ where each $f \in \mathcal{I}$ is "given" their node key $k_f[0]$ as a personal secret key $k_f[\mathcal{I}, 0]$. During the execution of this algorithm, the protocol between all players in $\mathcal{I}$ is launched. The key $k_i[0]$ of the node i is then the output of this protocol, that is $K[\mathcal{I}, 0]$. The related public element is then $pk_i = PE[\mathcal{I}, 0] \| i$ where $PE[\mathcal{I}, 0]$ is outputted by the $GKA.\mathsf{KeyGeneration}$ algorithm.

  At the end, at each node corresponds a key $k_i[0]$ and the algorithm outputs the first instance of the public element $PE[0]$ corresponding to the set of all public information $pk_i[0]$ outputted in the second and third cases.

– KeyDerivation(Γ, j, i, ρ): we need first to choose the best path between the nodes i and j. The choice of the smallest one (using standard graph shortest path finder algorithms) is not necessarily the best one. Obviously, in terms of computational efficiency, the case of one father (computation of a MAC) is more efficient than the case of several fathers (execution of a GKA protocol). Consequently, we should choose the path where the number of requests to the GKA is the smallest one. This path can be found either by exhaustive search or using a shortest path finder for weighted graphs. We now consider that this algorithm exists (note that it can be executed only once at the creation of the graph).

  Then, for each node v in the path between the node i and the descendant node j, this algorithm works as follows
  - if v has one father f: let $s_v \| v$ be the part of the public element $PE[\rho]$ corresponding to the focused node v. Then, computes

$$k_v[\rho] = M.\mathsf{MAC}(k_f[\rho], C \| s_v).$$

  - if v has F_v fathers $(f_1, \cdots, f_{F_v})$: let $pk_v = PE[\mathcal{V}, \rho] \| v$ be the part of the public element $PE[\rho]$ corresponding to the node v with $\mathcal{V} = \{f_1, \cdots, f_{F_i}\}$. Let $f \in \mathcal{V}$ be the father for which the key is known. This key can come from either the input of the algorithm or by derivation using one of the two methods. Then, executes the GKA.KeyRetrieve procedure on input $\mathcal{V}$, f and ρ. f takes as inputs $GKA.\Gamma$, $k_f[\mathcal{V}, \rho] = k_f[\rho]$, $PE[\mathcal{V}, \rho]$ and obtains the corresponding instance of the key $k_v[\rho]$.

– KeyRefresh(j): since we use virtual nodes, the targeted node has necessarily one father f (either a "true" father or a virtual node). We necessarily have $k_f[\rho + 1] = k_f[\rho]$ if ρ is the current instance. Thus, if we denote by s_f the number of times this father has computed a key for one of his children (either during the KeyGeneration procedure or a previous KeyRefresh one), then computes

$$k_j[\rho + 1] = M.\mathsf{MAC}(k_f[\rho + 1], C \| s_f + 1).$$

  The corresponding public information becomes $pk_j = s_f + 1 \| j$.

  Now that the key of the targeted node has been refreshed, the case of his own child has to be studied. There are then two cases for the new targeted node i, depending on the number of fathers the node has. If there is only one, we can do again what we have done for the first refresh.

  If there are several fathers $(f_1, \cdots, f_{F_i})$, let $pk_i = PE[\mathcal{I}, \rho] \| i$ be the part of the public element $PE[\rho]$ corresponding to the focused node i and where $\mathcal{I} = \{f_1, \cdots, f_{F_i}\}$. Let $\mathcal{F} \subset \mathcal{I}$ be the set of fathers for which the key has been previously refreshed. Then, executes the GKA.KeyRefresh procedure on input $\mathcal{I}$ and $\mathcal{F}$, where by assumption, each element $f \in \mathcal{F}$ has already received a new instance of its node key $k_f[\mathcal{I}, \rho + 1] = k_f[\rho + 1]$. This key is next used as a personal secret key in the GKA.KeyRefresh procedure. Again, for each $v \in \mathcal{I} \setminus \mathcal{F}$, we have $k_v[\rho + 1] = k_v[\rho]$. During the execution of this algorithm, the protocol between all players in $\mathcal{I}$ is launched. The new

instance of the key $k_i[\rho+1]$ of the node i is then the output of this protocol, that is $K[\mathcal{I}, \rho+1]$. The related public information is $pk_i = PE[\mathcal{I}, \rho+1]\|i$ where $PE[\mathcal{I}, \rho+1]$ is outputted by the $GKA.\mathsf{KeyGeneration}$ algorithm. The new instance of the public elements is then the set of all pk_i.

4.3 Security Considerations

Theorem 1. *Our key management scheme verifies the key recovery property under the existential unforgeability and the pseudorandomness of the HMAC and the key recovery property of the Group Key Management scheme.*

Proof. The idea of the proof is to play two different games with a possible adversary against the key recovery of our graph key management scheme. In the first game, we design a machine winning the EF-CMA experiment of the MAC scheme and in the second game, our machine tries to win the key recovery experiment for the R&R-GKA scheme. We thus flip a coin and play one of the two games. In case of success, we end and otherwise, we flip another coin. We are sure to succeed with probability 1/2. Due to space limitation, the complete proof is not given here.

Remark 3. During the KeyGeneration procedure, it is possible for a player to cheat by not giving the right key to one of her descendant. In order to detect such fraud, it is possible to add a key confirmation procedure where each node publishes a label related to its secret node key.

4.4 Efficiency Considerations

It is possible to instantiate our generic construction with the Group Key Agreement scheme presented in [16], where the solution is based on the use of a tree. While this solution is not the most efficient one regarding the complexity point of view, it fits very well our needs.

During the key generation phase of our construction, the case where there is only one father is very efficient since only needing a HMAC operation. When there are several fathers, we first insert the virtual node and then compute the corresponding node key using e.g. [16]. For each virtual node v, each father computes $\log_2(F_v) + 1$ modular exponentiations in a group of prime order. We should add the so-called blinded keys [16], which corresponds to $\log_2(F_v) - 1$ modular exponentiations in a group of prime order for the whole group.

4.5 The Dynamic Case

In case of a dynamic graph, we need to add two new procedures.

AddNode. The key of the j-th child of a node is computed from his father's one thanks to a specific counter s_f. In the static case, this counter is set to the number of current children plus the number of refreshes. In the dynamic case, this counter is also incremented when a new node is added to this father.

DeleteNode. We need first to modify the graph. The targeted node is deleted and the different links between ascendants and descendants are created. Note that if a path already exists from an ascendant to a descendant, there is no need to create a new one. In the second step, we refresh the keys in the sub-graph with all fathers of the deleted node as root(s). This step uses techniques described in either the KeyGeneration procedure or the KeyRefresh one and may use the dynamicity of the R&R-GKA scheme (deletion of a group member) if necessary.

5 Conclusion

This paper describes the first key management scheme suitable for multi-rooted oriented graphs with no oriented cycle without needing the presence of a central authority. In the most general setting, this scheme can be used for access control in a group with a hierarchical structure. Our construction mainly takes advantage of the use of a group key agreement designed for a non-hierarchical structure. We finally use virtual nodes in our graph so as to speed up the key generation phase.

Acknowledgments. We are grateful to Marc Girault and Jacques Traoré for their suggestions, and to anonymous referees for their valuable comments.

References

1. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. In: ACM (ed.) ACM Trans. Comput. Syst. (TOCS 1983), vol. 1, pp. 239–248 (1983)
2. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: ACM CCS 2005, pp. 190–202 (2005)
3. Bellare, M.: New proofs for nmac and hmac. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
4. Birget, J., Zou, X., Noubir, G., Ramamurthy, B.: Hierarchy-based access control in distributed environments. In: IEEE International Conference on Communications, vol. 1, pp. 229–233 (2001)
5. Bresson, E., Chevassut, O., Pointcheval, D.: Provably secure authenticated group diffie-hellman key exchange. ACM Trans. Inf. Syst. Secur. 10(3), 10 (2007)
6. Chang, C.-C., Lin, I.-C., Tsai, H.-M., Wang, H.-H.: A key assignment scheme for controlling access in partially ordered user hierarchies. In: AINA 2004, p. 376. IEEE Computer Society, Los Alamitos (2004)
7. Chick, G.C., Tavares, S.E.: Flexible access control with master keys. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 316–322. Springer, Heidelberg (1990)
8. Chou, J.-S., Lin, C.-H., Lee, T.-Y.: A novel hierarchical key management scheme based on quadratic residues. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, pp. 858–865. Springer, Heidelberg (2004)
9. Das, M.L., Saxena, A., Gulati, V.P., Phatak, D.B.: Hierarchical key management scheme using polynomial interpolation. SIGOPS 39(1), 40–47 (2005)
10. De Santis, A., Ferrara, A.L., Masucci, B.: Cryptographic key assignment schemes for any access control policy. Inf. Process. Lett. 92(4), 199–205 (2004)

11. Desmedt, Y., Lange, T., Burmester, M.: Scalable authenticated tree based group key exchange for ad-hoc groups. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 104–118. Springer, Heidelberg (2007)
12. Hardjono, T., Zheng, Y., Seberry, J.: New solutions to the problem of access control in a hierarchy. Technical Report Preprint 93-2 (1993)
13. He, M., Fan, P., Kaderali, F., Yuan, D.: Access key distribution scheme for level-based hierarchy. In: PDCAT 2003, pp. 942–945 (2003)
14. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. J. Cryptol. 20(1), 85–113 (2007)
15. Kim, Y., Perrig, A., Tsudik, G.: Group key agreement efficient in communication. IEEE Trans. Comput. 53(7), 905–921 (2004)
16. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Trans. Inf. Syst. Secur. 7(1), 60–96 (2004)
17. Kuo, F.H., Shen, V.R.L., Chen, T.S., Lai, F.: Cryptographic key assignment scheme for dynamic access control in a user hierarchy. In: IEE Proceedings Computers and Digital Techniques, vol. 146, pp. 235–240 (1999)
18. Lee, S., Kim, Y., Kim, K., Ryu, D.-H.: An efficient tree-based group key agreement using bilinear map. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846. Springer, Heidelberg (2003)
19. Lin, C.-H.: Hierarchical key assignment without public-key cryptography. Computers & Security 20, 612–619 (2001)
20. Krawczyk, H., Bellare, M., Canetti, R.: Hmac: Keyed-hashing for message authentication. In: RFC 2104 (1997)
21. MacKinnon, S.J., Taylor, P.D., Meijer, H., Akl, S.G.: An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. IEEE Trans. Comput. 34(9), 797–802 (1985)
22. Ragab Hassen, H., Bouabdallah, A., Bettahar, H., Challal, Y.: Key management for content access control in a hierarchy. Comput. Netw. 51(11), 3197–3219 (2007)
23. Ray, I., Narasimhamurthi, N.u.: A cryptographic solution to implement access control in a hierarchy and more. In: SACMAT 2002, pp. 65–73. ACM, New York (2002)
24. Steiner, M., Tsudik, G., Waidner, M.: Key agreement in dynamic peer groups. IEEE Transactions on Parallel and Distributed Systems 11(8), 769–780 (2000)
25. Wu, J., Wei, R.: An access control scheme for partially ordered set hierarchy with provable security (2004/295) (2004), `http://eprint.iacr.org/`
26. Zhang, Q., Wang, Y.: A centralized key management scheme for hierarchical access control. In: IEEE GLOBECOM 2004, vol. 4, pp. 2067–2071 (2004)
27. Zheng, Y., Hardjono, T., Pieprzyk, J.: Sibling intractable function families and their applications. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 124–138. Springer, Heidelberg (1993)
28. Zhong, S.: A practical key management scheme for access control in a user hierarchy. Computers & Security 21, 750–759 (2002)
29. Zou, X., Karandikar, Y., Bertino, E.: A dynamic key management solution to access hierarchy. Int. J. Netw. Manag. 17(6), 437–450 (2007)