Key management in tree shaped hierarchies

Lanfranco Lopriore

Dipartimento di Ingegneria dell'Informazione, Università di Pisa,
via G. Caruso 16, 56126 Pisa, Italy. E-mail: lanfranco.lopriore@unipi.it

Abstract—We refer to an access control system based on subjects and objects. Subjects are active entities, e.g. processes, while objects are passive entities, e.g. messages exchanged between the nodes of a distributed computing environment. The system is partitioned into security classes organized into a tree shaped hierarchy. A subject assigned to a given class can access the objects in this class and in all the classes that descend from this class in the class hierarchy. To this aim, a key is associated with each class. A mechanism of the protection system, called key derivation, allows a subject that holds the key of a given class to transform this key into the keys of the descendant classes. This mechanism is based on a single, publicly known one-way function. If the class hierarchy is modified, by adding a new class or deleting an existing class, the necessary form of key redistribution is partial, and is limited to the classes in the subtree of the root that is involved in the change.

Keywords: access control; cryptographic key; hierarchy; one-way function; protection.

1 INTRODUCTION

1.1 Tree shaped hierarchies

We refer to a protection system in which active entities, called *subjects*, generates access attempts to passive entities, called *objects* [13], [21]. A subject can be a process, or, in an event driven environment, an activity caused by an event, e.g. a hardware interrupt [17]. The system defines a set of *security classes* organized into a tree shaped hierarchy, whereby each class can have only one parent class, and many children.

Objects are typed. The definition of a given type states the operations that can be applied on the objects of that type. An *access authorization* is a pair $(object, operation)$. Each access authorization is assigned to a class. Let us refer to object B for which two operations are defined, op_1 and op_2. The two access authorizations are (B, op_1) and (B, op_2). If (B, op_1) is in class C_1 and (B, op_2) is in class C_2, then object B belongs to class C_1 for operation op_1, and to class C_2 for operation op_2.

A subject in a given class holds the access authorizations in this class, and in all the descendant classes, reachable in the class hierarchy by repeated steps from parent to child. It follows that a subject in the class that is the root of the hierarchy holds all access authorizations, for instance, and a subject in a leaf class holds only the authorizations in that leaf class. A subject that holds a given access authorization can access the object to

execute the operation specified by that authorization.

Let us first consider the case of objects for which a single operation is defined. In this case, an object belongs to a single class. An example is a distributed computing system consisting of nodes connected to form a network. In this case, the subjects are the nodes, and the objects are the messages exchanged between the nodes. Each node is associated with a class. A single operation is defined for a message, to read its contents. Thus, a message defines a single access authorization, in the class of the node that sends the message. A node that receives a message can read that message only if the class of this node is the same as, or an ancestor of, the class of the message. The inverse is not possible, that is, a node in a given class cannot read the messages sent by the nodes in the ancestor classes.

An example of a distributed system of this type is a sensor network in which the sensor nodes are organized into applications [2], [8]. In each application, a node, called the *application server*, is responsible for collecting data from the other nodes of this application. Applications are categorized, and the servers of the applications in the same category cooperate tightly to transform the collected data into a compact form, suitable for transmission to the base station. In turn, the base station is responsible for the final presentation and delivery of the results of the environmental observations of the entire network. In this organization, the classes are organized into three levels. At level 0, we have the root class featuring a single subject, the base station. At level 1, we have a class for each application category. The application servers in the same category cooperate by message exchanges. Finally, at level 2 we have a class for each application. The sensor nodes in the same class all belong to the same application. Hierarchical access control allows message exchanges across the levels. An application server can receive messages from the sensor nodes of its application, for instance, and the root can receive messages from all the application servers.

If an object type defines more than a single operation, then an object of that type can belong to more than a single class. An example is the document type, for which two operations are defined, to read the document contents, and to replace these contents. In this case, a document D can belong to two different classes, a class C_1 for access authorization $(D, read)$ and a class C_2 for access authorization $(D, write)$. If C_1 is a descendant of C_2, then a subject in C_2 can access D both to read and to write. This is not the case for a subject in class C_1, which can access D to read only.

1.2 Multiple keys

We shall refer to an implementation of the access control paradigm, outlined above, based on symmetric keys and conventional cryptography (as opposed to public key cryptography) [22]. In a simple approach, which we call *multiple keys*, a cryptographic key is assigned

to each class, and a subject that holds the key of a given class can exercise all the access authorizations that belong to this class. In this approach, a subject in a given class holds the key of this class and the keys of all the descendant classes.

In the example of a distributed computing system, each message consists of a header and a body. The message header is in plaintext, and contains administrative information items, including the name of the class of the sender node. The message body is enciphered by using the key of this class. A node assigned to a given a class holds the cryptographic keys of this class and of all its descendant classes. When a node receives a message, it reads the name of the class of the message in the message header, and can use the corresponding key to decipher the message only if it holds this key. In the example of the documents, a subject that is aimed at accessing a document to read or to write accompanies the request with a key. The key is correct if it is that of the class of the access authorization corresponding to that document and that operation. If the key is correct, the access is permitted, otherwise the access is denied.

The multiple key approach penalizes the subjects in the most privileged classes, which are required to handle more keys than the subjects in the lower classes [7]. For instance, a subject assigned to the root of the hierarchy is required to handle all keys. Multiple keys tend to become a security hazard. Keys can be lost or stolen, for instance [10], [11], and this may especially be the case when they are delivered to subjects. A related problem is the possibility to add new classes to the class hierarchy, and to eliminate existing classes from the class hierarchy. These modifications of the hierarchy may imply that all the keys, or a subset thereof, are replaced. For instance, in the example of a distributed computing system, a key replacement will take place when a node is evicted from the network, to avoid that this node continues to use the old keys to read the new traffic. The multiple key approach tends to exacerbate the key replacement problem, as several keys must be sent to each node.

1.3 Single key

In an alternative approach, a subject that is assigned to a given security class receives a single key, i.e. the key of this class, independently of the class position in the hierarchy. A mechanism of the protection system, called *key derivation*, allows the subject to generate the keys of the descendant classes. In this approach, key replacement implies the transmission of a single key to each subject.

In the example of a distributed computing system, in a single key environment, a network node assigned to a given class can read the messages from the nodes in this class. Taking advantage of key derivation, the node can generate the keys of the descendant classes, to read the messages from the nodes in these classes. In the example of the documents, a subject in a given class can access the documents corresponding to access

authorizations in its own class. Taking advantage of key derivation, the subject can obtain the keys of the descendant classes, thereby gaining access to the documents corresponding to access authorizations in these classes.

In this paper, we approach the key management problem with reference to a tree shaped hierarchical system. We associate a key with each security class. A subject S assigned to a given class receives a single key, i.e. the key K of this class. A key derivation mechanism allows S to generate the keys of the descendant classes. S is in the position to carry out key derivation autonomously; no intervention of a centralized key management component is necessary. Furthermore, S can distribute key K to another subject S'. Consequently, S' is assigned to the class corresponding to K. Alternatively, S can grant S' a key K' derived from K, and corresponding to a class at a lower hierarchical level. Consequently, S' is assigned to this less privileged class. Efficient support is given to the addition of new classes to the class hierarchy, and to the eviction of existing classes from the hierarchy.

The rest of this paper is organized as follows. Section 2 illustrates the mechanisms for key generation and derivation. Key generation takes advantage of a one-way function, the *key generation function*, and a key, the *master key*, which is chosen at random and is assigned to the root of the class hierarchy. Key derivation allows a subject that possesses the key for a given class to generate the keys for the classes that descend from this class in the hierarchy. Section 3 considers the problems related to the addition of new classes to the class hierarchy, and to the deletion of existing classes from the hierarchy. Section 4 discusses the properties of our system from a number of important viewpoints, which include key forging, key replacement, linear hierarchical access control, and the relation of our work to previous work. Section 5 gives concluding remarks.

2 KEY MANAGEMENT

2.1 Key generation

Function f is *one-way* if given x it is easy to determine $f(x)$, and given $f(x)$ it is computationally unfeasible to determine x [3], [15]. A one-way function can be implemented by using a block cipher adapted to guarantee that the resulting function is not invertible [19]. In a common approach, a publicly known constant c is encrypted using x as the key, i.e. $f(x) = E_x(c)$ [20]. In a given hierarchical system, we take advantage of a one-way function, the key generation function, which is publicly known. We shall denote this function by g.

The n classes that form the hierarchy are numbered from 0 to $n-1$. Class number 0 denotes the root; the other class numbers are assigned from left to right at increasing distance from the root. Let C_r be the name of the class whose number is r. Figure 1

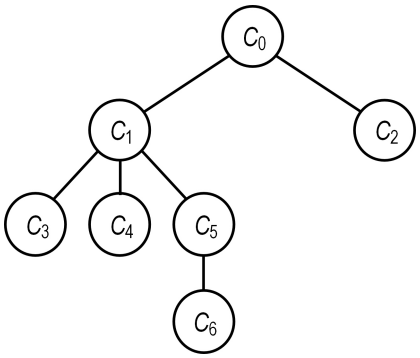

Figure 1: An example of a class hierarchy. The classes are numbered from left to right at increasing distance from the root.

shows an example of a class hierarchy named in this way. A key is assigned to each class. Let K_r denote the key of class C_r. A master key, chosen at random, is reserved for the root, C_0. This key is denoted by K_{master}. Thus we have $K_0 = K_{master}$. All the other keys are derived from K_{master} by iterated applications of key generation function g, obtained by considering the class numbers.

In detail, the key of a class that is at distance r from the root is obtained starting from K_{master} by applying function g iteratively r times. We define the *map* M_s of node C_s as the set formed by quantity s and the numbers of the classes that descend from C_s, i.e. the nodes reachable from C_s by repeated steps from parent to child. For instance, in Figure 1, we have $M_1 = \{1, 3, 4, 5, 6\}$, and $M_5 = \{5, 6\}$. Let class C_r be the parent of class C_s. By H_s we denote the *mapped complement* of key K_r *according to* the map M_s of C_s, which is obtained as follows. We divide key K_r into m *subkeys* $k_{r,0}, k_{r,1}, \ldots, k_{r,m-1}$, where $m \geq n$. We consider the subkeys $k_{r,i}$ of K_r for which the corresponding number i is included in map M_s. Each of these subkeys is bitwise complemented (thus, for instance, if M_s includes class number 1, then we complement subkey $k_{r,1}$). Key K_s is obtained by applying key generation function g to H_s. Thus we have $K_s = g(H_s)$.

With reference to the tree structure of Figure 1, Figure 2 shows the evaluation of keys K_1 and K_5 starting from key K_0. In this example, $m = 8$, and consequently, K_0 is partitioned into 8 subkeys, $k_{0,0}$ to $k_{0,7}$. Let us denote the bit configuration of the generic subkey by $xx \ldots x$, and the corresponding bitwise complement by $\bar{x}\bar{x} \ldots \bar{x}$. We have $K_0 = K_{master}$. In the generation of K_1, submap M_1 is $\{1, 3, 4, 5, 6\}$ and, consequently, subkeys $k_{0,1}$, $k_{0,3}$, $k_{0,4}$, $k_{0,5}$ and $k_{0,6}$ of key K_0 are bitwise complemented. The result H_1 is used as a parameter of key generation function g to obtain K_1, i.e. $K_1 = g(H_1)$. In the generation of K_5, submap M_5 is $\{5, 6\}$. Consequently, subkeys $k_{1,5}$ and $k_{1,6}$ are bitwise complemented to obtain H_5. Then we have $K_5 = g(H_5)$.

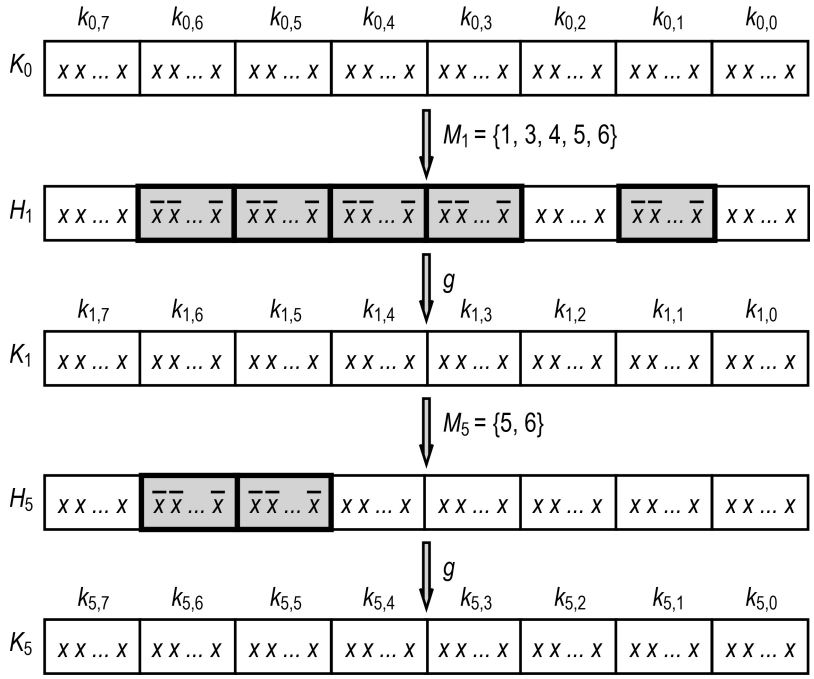

Figure 2: Generation of keys K_1 and K_5 starting from key K_0. H_1 is the mapped complement of K_0 according to map M_1 of class C_1. H_5 is the mapped complement of key K_1 according to map M_5 of class C_5.

2.2 Key derivation

A subject that possesses key K_r of class C_r can take advantage of this key to access the objects in this class. As seen in Section 1, in a tree shaped hierarchical system we require that a subject assigned to a given class can also access the objects in all the descendants of this class. In our system, an effect of this type is obtained by key derivation.

Let S be a subject assigned to class C_r, which holds the key K_r of this class. S is in the position to derive the key K_s of class C_s that is a direct child of C_r. Key derivation is similar to key generation. Let M_s be the map of class C_s, and let H_s be the mapped complement of key K_r according to M_s. We have $K_s = g(H_s)$.

In the hierarchical system of Figure 1, a subject S that holds the key K_1 of class C_1 can derive the key K_5 of class C_5, a direct child of C_1. We have $M_5 = \{5, 6\}$. The mapped complement H_5 of key K_1 according to map M_5 is the result of a bitwise complement of subkeys $k_{1,5}$ and $k_{1,6}$. Finally, $K_5 = g(H_5)$.

Let us now consider a class C_t that is a descendant, but not a direct child, of C_r. Subject S that holds key K_r can derive key K_t of class C_t by multiple actions of key derivation, traversing the class hierarchy from C_r to C_t to derive the keys of the intermediate classes. In the example of Figure 1, let us suppose that subject S holds key K_1 of class C_1 and is aimed at obtaining key K_6 of class C_6. First, S will derive key K_5 of class C_5, which is a direct child of C_1. Then, S will derive key K_6 of class C_6, which is a direct child of C_5.

2.3 Key weakening

A subject S that possesses the key K_r of a given class C_r can grant a copy of this key to another subject S'. Consequently, S' is authorized to access the objects in C_r and in all the descendants of C_r. Alternatively, S can grant S' the key K_s of a class C_s that descends from C_r. As a result, S' is authorized to access the objects in C_s and in all the descendants of C_s. In fact, K_s embodies a weakened version of K_r, which grants access to a limited portion of the objects accessible by using K_r. S can carry out the key weakening process autonomously, by taking advantage of key derivation, as has been illustrated in Section 2.2. In detail, if C_s is a direct child of C_r, subject S will simply carry out a single action of key derivation, to generate K_s starting from K_r. If K_s is a descendant but not a direct child of K_r, more key derivation steps will be required, which also involve all the intermediate classes in the path from C_r to C_s.

In the example of Figure 1, subject S that holds the key K_1 of class C_1 can transfer this key to subject S'. Consequently, S' gains access to all the objects that belong to class C_1 as well as to the descendants of C_1, namely C_3 to C_6. To grant a more limited access, subject S generates key K_6 of class C_6, for instance, and transfers this key to S'. Consequently, S' gains access to the objects in a single class, C_6. The weakening process from key K_1 to key K_6 requires two key derivation steps, the first to transform K_1 into K_5, and the second to transform K_5 into K_6.

3 CLASS MANAGEMENT

The general class numbering rule, introduced in Section 2.1 (from left to right at increasing distance from the root) is inessential, and in fact, our key generation algorithm considers class numbers, but is independent of the numbering rule. An important consequence is that the addition of new classes, and the elimination of existing classes, never imply a form of class renumbering, even if the resulting assignment of numbers to classes is not consistent with the general rule.

In our system, the addition of a class implies a key redistribution, which is only partial and is limited to a single subtree of the root, i.e. the subtree of the new class. Figure 3a shows the addition of class C_7 as a leaf of the hierarchy. The subtree of the root that contains C_7 is formed by two nodes, C_2 and C_7. The new class modifies map M_2 of class C_2, which was $\{2\}$ and becomes $\{2, 7\}$. As seen in Section 2.1, the map determines the key, and consequently, key K_2 changes. Afterwards, map $M_7 = \{7\}$ of the new class C_7 is used to evaluate key K_7.

As a further example, Figure 3b shows the addition of an internal class (a class that is not a leaf). The new class, C_7, is positioned at distance 2 from the root. In this case, map M_1 of class C_1 was $\{1, 3, 4, 5, 6\}$ and becomes $\{1, 3, 4, 5, 6, 7\}$. It follows that key K_1

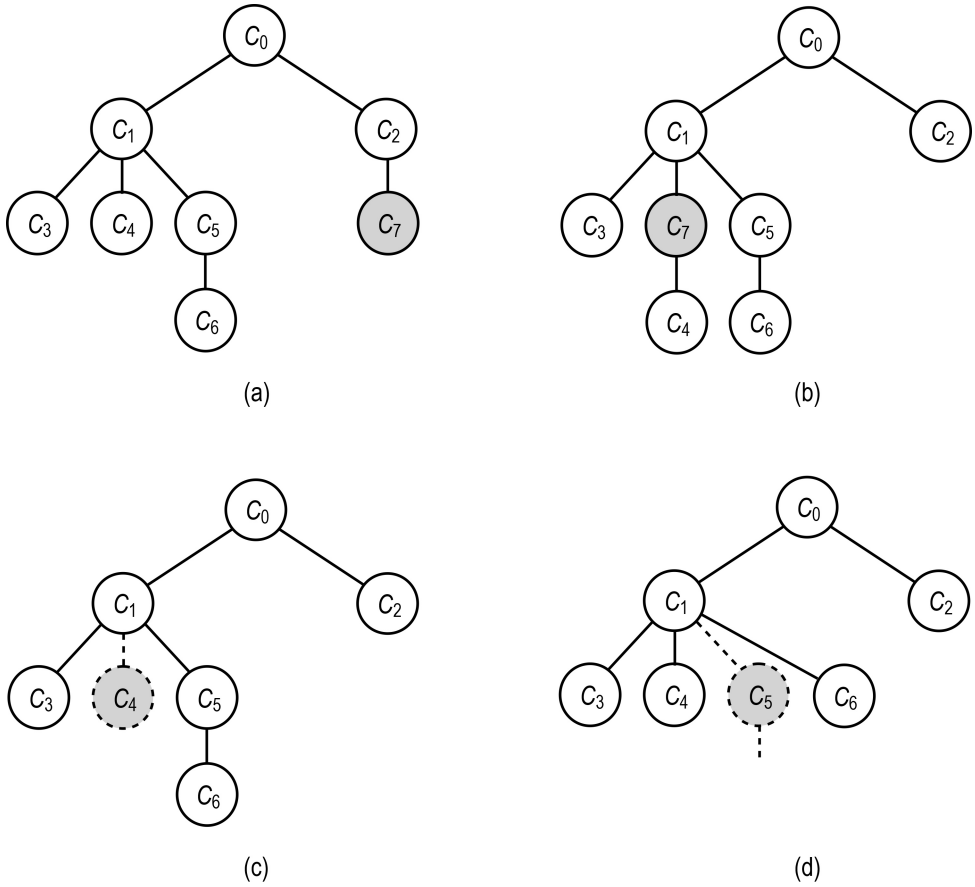

Figure 3: Modifications of the class hierarchy shown in Figure 1: (a) addition of a leaf class; (b) addition of an internal class; (c) elimination of a leaf class; and (d) elimination of an internal class.

changes. Consequently, the keys of all the descendants of C_1 should be recalculated, as K_1 determines their values.

Similarly, the elimination of an existing class that is part of a given subtree of the root implies the distribution of new keys to all the classes in this subtree. Figure 3c shows the elimination of a leaf class, C_4. Map M_1 of class C_1 was {1, 3, 4, 5, 6} and becomes {1, 3, 5, 6}. It follows that key K_1 changes, as is the case for the keys of all the descendants of C_1, whose values are determined by K_1.

Finally, Figure 3d shows the elimination of an internal class, C_5. In this case, too, map M_1 of class C_1 changes: it was {1, 3, 4, 5, 6} and becomes {1, 3, 4, 6}. This implies the replacement of the key K_1 of C_1, and of the keys of all the descendants of C_1.

4 DISCUSSION

4.1 Key forging

Let us suppose that a subject S attempts to forge the key of the root of a given class

hierarchy from scratch. S does not know the value of the master key K_{master} of this hierarchy, and consequently, it uses a value chosen at random. If master key values are large and sparse, the probability of a match with the correct value is vanishingly low. Similar considerations can be made for the keys derived from K_{master}, and relevant to less privileged classes.

Let us now hypothesize that subject S possesses a valid key K for a given class C, and is aimed at amplifying K to transform it into the key K' of another class C' that is an ancestor of C in the class hierarchy. This means that K' precedes K in the iterative key generation process illustrated in Section 2.1. However, key generation function g, used in this process, is one-way, and cannot be inverted. Consequently, if this function is based on a secure cryptosystem, it is impossible to use value K to obtain a previous value K'.

4.2 Key replacement

A node which is evicted from a computer network should not be allowed to decipher future messages. It follows that a key redistribution is necessary. Similarly, a node that is added to the network should not be allowed to decipher old messages, sent before its arrival, if it has recorded these messages. These two requirements are called *forward secrecy* and *backward secrecy*, respectively [4], [5], [9].

Furthermore, a node that is degraded from a given class C_r to a descendant class C_s should be prevented from decrypting the new traffic of the classes between C_r and C_s. With reference to the class hierarchy of Figure 1, consider a node in class C_1 that is degraded to class C_5. In a situation of this type, we have to replace keys K_1, K_3 and K_4. Conversely, a node that is promoted from class C_s to an ancestor class C_r should be prevented from decrypting the old traffic of the classes between C_r and C_s, to which it had no access before promotion, if it has recorded this traffic. In the example of Figure 1, if a node in class C_5 is promoted to class C_1, we have to replace keys K_1, K_3 and K_4. These two requirements are called *downward secrecy* and *upward secrecy*, respectively [16], [18]

In our system, as seen in Section 2.1, key generation is an iterative process that starts from master key K_{master}. If we change K_{master}, the old keys are invalidated; the new keys should be recalculated and distributed to all the classes. Key distribution is a simple process that starts at the root and proceeds in repeated steps from parent to child. The root uses K_{master} to derive the keys of its child classes, and then it distributes each key to the corresponding class. The key derivation and distribution process is iterated in each class, until the leaves of the class hierarchy are reached.

When a key is received by a given class, it is distributed to each subject in this class. The transmission cost of an action of this type is limited to a single key for each subject. The subject that receives the new key of its own class will be in the position to evaluate the new keys of the descendant classes autonomously, taking advantage of the key derivation

mechanism. As seen in Section 1.2, this in sharp contrast with the multiple key approach, whereby key redistribution means to give each subject the new key of its class and of all the descendants of its class.

4.3 Relation to previous work

Hierarchical access control systems can be classified into three categories, directed acyclic graphs, trees, and linear hierarchies [13]. In a directed acyclic graph hierarchy, each security class can have many parents (direct ancestors) and many children (direct descendants). In a tree shaped hierarchy, each class can have a single parent and many children. Finally, in a linear hierarchy, each class can have a single parent and a single child. In this work, we have considered tree shaped hierarchies.

Hassen *at al.* [12] proposed an effective classification of the existing key management schemes. They distinguish the *independent-keys* and the *dependent-keys* schemes. In an independent-keys scheme, a key is generated at random for each security class. Each subject that is a member of a given class holds the key of this class and the keys of all the descendant classes. In contrast, in a dependent-keys scheme, each subject holds a single key, which is combined with public functions and/or parameters to derive the other keys. These schemes usually take advantage of complex theoretical cryptographic notions, e.g. the fundamental properties of prime numbers [1], [6], [7], [11].

Our proposal is in the dependent-keys category, and is based on the well known concept of a one-way function [14], [20], [23]. We use a single, publicly known one-way function for the entire hierarchy. We take advantage of class names, and the parent-child relationships between the classes, to derive the keys of all the classes in the class hierarchy starting from a single, random key assigned to the root. Noticeably, in key derivation, each class needs to be aware of the composition of only a fraction of the class hierarchy, i.e. its own descendants. All keys have the same size, so the amount of storage that is necessary in each class is fixed (in contrast, in [1], public parameters can be very large for a large number of classes).

A related observation is relevant to linear hierarchies, in which each class can have a single parent and a single child [12], [18]. In a simple implementation of an access control system of this type, all keys are related by using a one-way function g so that key K_0 of the highest class C_0 is chosen at random, and key K_i of class C_i is given by $g(K_{i-1})$. Each subject receives a single key, and uses the one-way function to derive the keys of the descendant classes. Insertion or extraction of a new class in the class hierarchy induces a complete redistribution of all keys, as is necessary for backward secrecy in the case of an insertion, and for forward secrecy in the case of an extraction.

Of course, a linear hierarchical system is a special case of a tree shaped system. It follows that our proposal for tree shaped hierarchies can be used for linear hierarchies as

well. As seen in Section 2.2, in each key derivation step, we take advantage of the map of a given class to derive the key of this class from the key of its parent class. The map is aimed at differentiating the derived keys of the children of the same class. In fact, the map of each given child is unique in the hierarchy (i.e. intrinsically different from those of the other children), as it is determined by the name of this child and the names of its descendants. Thus, the transformation of the key of the parent class into the keys of the child classes produces different keys for different children, as it uses different maps. On the other hand, in a linear hierarchical system, each node has a single child, and this makes the map mechanism unnecessary.

5 CONCLUDING REMARKS

We have considered a protection system paradigm based on subjects, objects and access authorizations. The system defines security classes organized into a tree shaped hierarchy. An access authorization includes the name of an object, and an optional specification of one of the operations defined by the object type. A subject in a given class can access the objects in this class and in all the classes that descend from this class, hierarchically. To this aim, a key is associated with each class. A key derivation mechanism allows a subject that holds the key of a given class to generate the keys of the descendant classes. The subject is in the position to carry out key derivation autonomously. No intervention of a centralized key management component is necessary. We have obtained the following results:

- A single, publicly known one-way function is sufficient to generate all keys.
- The memory requirements for storage of a key is fixed, and is independent of the number of classes in the class hierarchy.
- If the class hierarchy is changed, by adding a new class or deleting an existing class, the necessary form of key redistribution is partial, and is limited to the classes in the subtree of the root that is involved in the change.
- If master key values are large and sparse, it is virtually impossible for a malevolent subject to forge valid keys. Similarly, if the one-way property of the key generation function is based on a secure cryptosystem, it is impossible to promote the valid key of a given class into the key of an ancestor class.

We have considered two examples of applications, typed objects, whereby an object can belong to more than a single class, up to the limit of a class for each operation defined by the object type, and a distributed computing system, in which the subjects are the network nodes and the objects are the message exchanged between the nodes. More work

is necessary to comply with the communication requirements of a proficient network, e.g. a tree shaped wireless sensor network in which each node needs to communicate with its siblings as well as with its ancestors .

REFERENCES

[1] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[3] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: a survey. Technical report, Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australia, 1995.

[4] O. Cheikhrouhou, A. Koubâa, G. Dini, and M. Abid. RiSeG: a ring based secure group communication protocol for resource-constrained wireless sensor networks. *Personal and Ubiquitous Computing*, 15(8):783–797, December 2011.

[5] X. Chen, K. Makki, K. Yen, and N. Pissinou. Sensor network security: a survey. *IEEE Communications Surveys & Tutorials*, 11(2):52–73, second quarter 2009.

[6] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, pages 1–14, Venice, Italy, July 2006. IEEE.

[7] A. De Santis, A. L. Ferrara, and B. Masucci. Cryptographic key assignment schemes for any access control policy. *Information Processing Letters*, 92(4):199–205, 2004.

[8] G. Dini and L. Lopriore. Distributed storage protection in wireless sensor networks. *Journal of Systems Architecture*, 61(5–6):256–266, May–June 2015.

[9] G. Dini and L. Lopriore. Key propagation in wireless sensor networks. *Computers & Electrical Engineering*, 41:426–433, January 2015.

[10] E. Gudes. The design of a cryptography based secure file system. *IEEE Transactions on Software Engineering*, SE-6(5):411–420, 1980.

[11] L. Harn and H.-Y. Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, 1990.

[12] H. R. Hassen, H. Bettahar, A. Bouadbdallah, and Y. Challal. An efficient key management scheme for content access control for linear hierarchies. *Computer Networks*, 56(8):2107–2118, 2012.

[13] H. R. Hassen, A. Bouabdallah, H. Bettahar, and Y. Challal. Key management for content access control in a hierarchy. *Computer Networks*, 51(11):3197–3219, 2007.

[14] Z.-H. He and Y.-S. Li. Dynamic key management in a user hierarchy. In *Proceedings of the 2nd International Conference on Anti-counterfeiting, Security and Identification*, pages 298–300, Guiyang, China, August 2008. IEEE.

[15] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[16] M. A. Moulavi and H. Parvar. Agent based bandwidth reduction for key management in hierarchical group communication. In *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware*, pages 1–5, Bangalore, India, January 2007. IEEE.

[17] T. Newby, D. A. Grove, A. P. Murray, C. A. Owen, J. McCarthy, and C. J. North. Annex: a middleware for constructing high-assurance software systems. In *Proceedings of the 13th Australasian Information Security Conference*, pages 25–34, Sydney, Australia, January 2015. ACS.

[18] V. Odelu, A. K. Das, and A. Goswami. LHSC: an effective dynamic key management scheme for linear hierarchical access control. In *Proceedings of the Fifth International Conference on Communication Systems and Networks*, pages 1–9, Bangalore, India, January 2013. IEEE.

[19] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: a synthetic approach. In *Proceedings of the 13th Annual International Cryptology Conference*, pages 368–378, Santa Barbara, California, USA, August 1993. Springer.

[20] R. S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.

[21] L. Seitz, J.-M. Pierson, and L. Brunie. Key management for encrypted data storage in distributed systems. In *Proceedings of the Second IEEE International Security in Storage Workshop*, pages 20–30, Washington, DC, USA, October 2003. IEEE.

[22] M. Stamp. *Information Security: Principles and Practice.* John Wiley & Sons, Hoboken, NJ, USA, second edition, 2011.

[23] C. Yang and C. Li. Access control in a hierarchy using one-way hash functions. *Computers & Security*, 23(8):659–664, 2004.