On Some Cryptographic Solutions for Access Control in a Tree Hierarchy

Ravinderpal S. Sandhu

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

ABSTRACT

We consider the access control problem in a system where users and information items are classified into security classes organized as a rooted tree, with the most privileged security class at the root. In practise we expect such a tree to be quite broad and shallow. It is also inevitable that new security classes will need to be added as the needs of the organization evolve. We compare some cryptographic techniques which have been proposed in the literature for solution of this problem.

1. INTRODUCTION

Let the *users* and *information items* in a computer or communication system be classified into a rooted tree of ***security classes*** SC_1, SC_2, ..., SC_n. The notation $SC_i > SC_j$ means that SC_i is a predecessor of SC_j in the tree. Similarly $SC_i \geq SC_j$ means that either $SC_i = SC_j$ or $SC_i > SC_j$. If $SC_i \geq SC_j$ we say that SC_i ***covers*** SC_j. Each user is assigned to a security class called his ***clearance***. And each item of information, be it a file or a message, is assigned to a security class called its ***sensitivity***. Our requirement is that users with clearance SC_i can read or create information items with sensitivity SC_j if and only if SC_i covers SC_j. That is the higher up a security class is in the tree the more privileged it is, with the most privileged class at the root.

There are numerous examples of hierarchies where such a classification scheme for access control is useful. For instance, a corporate hierarchy with top management at the root and security classes at successive levels of the tree corresponding to divisions, departments and projects. A manager of a division will have clearance for the security class of that division and thereby the authorization to access information in all departments and projects within the division. Members of a project team on the other hand will be cleared only for that project and will be unable to access information concerning other projects including those within the same department.

We expect that practical tree hierarchies in such applications will have at most a dozen levels or so.

The fan-out at the non-leaf nodes is likely to larger say at most a few dozen. That is the tree is likely to be quite broad and somewhat shallow. It is inevitable that a given hierarchy is likely to change over the course of time, for instance as new departments and projects are created. We consider that any scheme for solving the protection and sharing problem in a hierarchy must be able to accommodate changes in the hierarchy with minimal disruption. This is a very important criterion, perhaps even the most important one, for evaluating a scheme.

Restricting the hierarchy to a rooted tree no doubt excludes many partial orders. Nevertheless a rooted tree is an important and naturally occurring special case whose efficient implementation will certainly have practical benefit. It should of course be possible to accommodate, at least "small", deviations from a tree hierarchy at the cost of some additional effort. That is the mechanism should allow some flexibility while accommodating a tree hierarchy directly and efficiently.

In this paper we consider a number of cryptographic schemes for solving the access control problem in a tree hierarchy. Basic cryptographic terminology and techniques used in these solutions are reviewed in section 2. In section 3 we describe and compare some proposals from the literature for this and related problems [1, 7, 8, 11, 12, 15]. We have made no attempt to cover all published schemes since this is beyond the scope of this paper. The schemes discussed here just happen to be the ones which are personally most familiar. Section 4 concludes the paper.

2. CRYPTOGRAPHIC BACKGROUND

We assume that a conventional cryptosystem such as DES [4, 13] is available with *enciphering and deciphering procedures* E and D respectively. The notation

$$u = E_K(v)$$

means that u is the result of enciphering v using the procedure E with key K. The deciphering procedure with key K is used to recover v, *i.e.*

$$v = D_K(u)$$

Information items can be cryptographically protected by assigning a distinct key K_i to each security class SC_i, to be used for encrypting and decrypting information classified in that class. When an information item x with sensitivity SC_i is to be stored or transmitted in the system it is first encrypted with key K_i to obtain

$$y = E_{K_i}(x)$$

The item is then stored or transmitted as the pair $[name(SC_i),y]$ where $name(SC_i)$ is the name of the security class SC_i. The purpose of appending the name of the security class to the encrypted form of an information item is to indicate how to decrypt the information. Only those users who somehow know K_i will be able to decrypt y to obtain x from

$$x = D_{K_i}(y)$$

Our access control problem will be solved if we can ensure that only those users whose security clearance covers SC_i are able to know K_i.

In the straightforward application of this idea a user with security clearance SC_i is given keys for all security classes SC_j covered by SC_i. As observed by Akl and Taylor [1] the disadvantage of this solution is that a large number of keys are held by users with security clearances high up in the hierarchy. Moreover if new security classes are created, by growing branches in the existing tree, keys for these classes must be distributed to all users whose clearance covers these new classes. This is a non-trivial administrative task especially in a distributed environment. It is particularly awkward that keys for new security classes deep down in the tree will need to be distributed to users with high security clearance.

In this paper we consider solutions to the key storage problem based on the idea that a user with security clearance SC_i needs to store only the key K_i. Keys for security classes SC_j covered by SC_i are generated from K_i as needed. In this sense the key K_i for security class SC_i can be viewed as a master key for all security classes covered by SC_i in the tree. It is desirable that the size of K_i be the same for all security classes, or at least be of the same order. Otherwise we can define K_i to include all keys for security classes covered by SC_i. The challenge is to find a method by which it is easy to compute keys for security classes covered by SC_i but it is intractable to compute keys for classes not covered by SC_i.

In the next section we review and compare some cryptographic solutions for the key storage problem based on *one-way functions.* A one-way function is easy to compute but computationally difficult to invert. The use of one-way functions to conceal information was first proposed for storing passwords in a computer system [6, 10, 14, 16] and this has now become a fairly standard practice. Their use for safeguarding cryptographic keys was suggested by Gudes [8] and has been applied in a number of different contexts [1, 2, 3, 7, 9, 11, 12].

To understand the use of one-way functions in generating and safeguarding cryptographic keys it is convenient to first consider the simple case where the security classes are totally ordered, *i.e.* when $SC_1 > SC_2 > ... > SC_n$. Akl and Taylor [1] describe the following solution for this case. The key K_1 for SC_1 is arbitrarily selected. Keys for the other security classes are iteratively generated by

$$K_{i+1} = f(K_i),\ i=1...n-1$$

where f is a publicly known one-way function. A user with security clearance SC_i is given the key K_i. He can then easily compute the key K_j for all security classes covered by SC_i. However, it is computationally infeasible to compute K_j for a security class $SC_j > SC_i$ since this requires inversion of a one-way function.

It is generally accepted that a good cryptosystem can be used to implement a one-way function. For instance the function $f(x)=E_x(x)$ is one candidate. Here x is encrypted using itself as the key. Another possibility, often cited in the literature, is to to encrypt some fixed and publicly known constant c using x as the key, *i.e.* $f(x)=E_x(c)$. In this case computing the inverse of $f(x)$ would amount to computing the key x given that c encrypts as $f(x)$. This is a known plaintext attack from which good cryptosystems are expected to be immune. One-way functions constructed from block encryption algorithms such as DES have the additional property that they operate on fixed length blocks. For the most part we can assume that x fits within one block of b bits. Then $f(x)$ requires no more than b bits.

3. SOME CRYPTOGRAPHIC SOLUTIONS

We now consider three schemes for solving the key storage problem all of which use one-way functions but in very different ways. The one-way functions used are of course assumed to be publicly available. Pragmatic issues regarding the application of these schemes are discussed in context of a broad and shallow tree. We also consider how these schemes will handle addition of a new subtree of security classes.

Section 3.1 describes a solution based on a parameterized *family of one-way functions* [15]. This scheme requires no public information other the definition of this family. It has the significant advantage that changes to the hierarchy are conveniently accommodated. We show that a broad and shallow tree is a particularly good case for this scheme. Section 3.2 consider solutions based on cryptographic sealing [7, 8]. The idea is that keys for the immediate children of SC_i are encrypted using K_i and their encrypted forms are made publicly available as *sealed keys*. Various methods of applying this scheme are discussed. Finally section 3.3 discusses a scheme based on *modular exponentiation* [1, 11, 12] which solves the problem for hierarchies which are arbitrary partial orders. A major drawback however is that addition of new security classes may require recomputation of the keys for a large fraction of the existing classes. It is shown that a broad and shallow tree is a particularly bad case for this scheme.

3.1. A FAMILY OF ONE-WAY FUNCTIONS

As mentioned earlier, a well known method for constructing a one-way function is to encrypt some fixed and publicly known constant c using x as the key, *i.e.* $f(x)=E_x(c)$. We generalize this to obtain a family of one-way functions by replacing the constant by a parameter p, that is $f_p(x)=E_x(p)$. Now computing the inverse of $f_p(x)$ amounts to computing the key x given that p encrypts as $f(x)$. So this is a known plaintext attack which is infeasible for secure cryptosystems. Hence $f_p(x)$ is a one-way function for every p. We say the collection of functions $f_p(x)$ is a parameterized family of one-way functions.

Given such a publicly known family of one-way functions, the keys for the security classes are generated as follows.

1. For the security class at the root assign an arbitrary key.
2. If SC_j is an immediate child of SC_i in the tree let $K_j=f_{name(SC_j)}(K_i)=E_{K_i}(name(SC_j))$.

A user with security clearance SC_i is given the key K_i. Since the family of one-way functions is publicly known and the names of the security classes are public, he can easily compute the key K_j for all security classes SC_j covered by SC_i. However it is computationally infeasible to compute K_j for a security class $SC_j > SC_i$ since this amounts to the inversion of one or more one-way functions.

Finally it should be computationally infeasible to compute K_j from K_i for SC_j incomparable with SC_i. To see what this entails consider the simple case where SC_i and SC_j are immediate children of SC_k. Then

$$K_i=E_{K_k}(name(SC_i))$$
$$K_j=E_{K_k}(name(SC_j))$$

By the assumed security of the cryptosystem it is infeasible to compute K_j from K_i by solving the known plaintext problem of the former equation to derive K_k and then using the latter equation to compute K_j. For a strong cryptosystem we believe it can be safely assumed that there will also be no other tractable method of computing K_j from K_i in this situation. Moreover even if we know the keys for a large number of siblings it will be infeasible to compute the keys for a sibling outside the known set. That is collusion among the siblings is infeasible. Similar considerations apply to incomparable classes which are not siblings.

This scheme accommodates hierarchical names for the security classes quite readily. With hierarchical names the immediate children of a security class have distinct names but children of different security classes may have the same name. When generating keys for the security classes it is necessary that the immediate children of SC_i get distinct keys. So the name used in our family of one way functions $E_K(name(SC))$ need only be the last field in the unique pathname of SC in the tree hierarchy. Thus the well known benefits of hierarchical names are available in this scheme.

To estimate the computational overhead of this scheme consider a tree hierarchy with 11 levels. A user with clearance for the root will need up to 10 applications of the encryption algorithm to derive a key for a security class at a leaf. If the same cryptosystem is also used for enciphering and deciphering information items this represents an overhead of 10*b bits for each message or file, where b is the block size of the cryptosystem. For DES b=8 (assuming there is a parity bit on each byte) so the overhead is effectively 80 bytes per information item. For a large file with say 8000 bytes this overhead is a negligible 0.1%. For a very small 80 byte file on the other hand the overhead is 100%. In practice we could also cache the recently derived keys so this overhead gets amortized over a number of files and messages rather than being incurred for each information item. In any case these numbers suggest that the overhead is quite tolerable and would hardly be the limiting factor in application of this scheme.

3.2. SEALED KEYS

Gudes [8] discusses a solution for the rooted tree case in which for each security class SC_i we have a publicly available *sealed key*. Sealed keys are generated as follows.

1. Select an arbitrary and distinct key K_i for each SC_i.
2. The sealed key for SC_i is $E_{K_k}(K_i)$ where SC_k is the parent of SC_i in the hierarchy. (The root has no sealed key.)

Thus knowledge of a key for a security class permits decryption of the sealed keys for the children of that class. By repeating this procedure a user who has clearance for SC_i can decrypt the sealed keys for every SC_j such that $SC_i > SC_j$. The computational overhead in computing these keys is exactly the same as in the scheme of section 3.1. Addition of a new security class as a leaf in the existing hierarchy is easily accommodated by computing and making available its sealed key.

A major issue in applying this idea is how to store the sealed keys. The sealed keys can be distributed publicly, which requires n*b bits with n security classes and a cryptosystem with block size b. In a distributed system the sealed keys may need to be replicated at multiple sites. When new security classes are created it will be necessary to update this information at all sites. At the cost of some storage overhead we can store the sealed keys for all ancestors of SC_i in every file or message of sensitivity SC_i. This approach has the advantage that the sealed keys needed to derive K_i are immediately available. The only additional information needed is one of the keys for the ancestors of SC_i. The latter solution is particularly usable in context of a shallow broad tree, with new security classes being added ever so often.

3.3. MODULAR EXPONENTIATION

Akl and Taylor [1] describe a mathematically elegant solution for the general case where the hierarchy on security classes is an arbitrary partial ordering. In their method a publicly known integer t_i is assigned to each security class SC_i with the following properties.

1. t_i divides $t_j \Leftrightarrow SC_i \geq SC_j$
2. gcd $\{t_j | SC_j \not\geq SC_i\}$ does not divide t_i

A random secret key K_0 and a secret pair of large primes p and q are selected whose product M is made available publicly. The security class SC_i is assigned the key

$$K_i = K_0^{t_i} \bmod \mathrm{M}$$

Keys for security classes covered by SC_i are easily computed as follows.

$$K_j = K_0^{t_j} \bmod \mathrm{M} = K_i^{t_j/t_i} \bmod \mathrm{M}$$

If $SC_i \not\geq SC_j$ by the first property t_j/t_i is not an integer and the computation of K_j from K_i is considered intractable provided M cannot be factored. This requires that p and q have approximately a hundred decimal digits each. The second property prevents collusion among users in breaking the scheme. Akl and Taylor propose the following method for assigning t_i's.

1. Assign a distinct prime p_i to each SC_i.
2. Let $t_i = \Pi \ \{p_j | SC_i \not\geq SC_j\}$

These t_i's grow very rapidly with increasing number of security classes, making it somewhat questionable whether the method is practical. For n security classes the value of these t_i's are $O((n\log_2 n)^n)$ which requires $O(n\log_2 n)$. The time requirement for computing a derived key is $O(n\log_2 n)$ multiplications [1].

Since the t_i's are public we require a total of $O(n^2\log_2 n)$ bits of public information. With n=100 this is ≈70K bits, which is sizable but not intolerable. If n=1000 we require a total of ≈10M bits. An alternative to making all t_i's publicly available is to append t_i to each encrypted information item of sensitivity SC_i. Each file or message then carried with it the information needed to decrypt it. With a hundred security classes this represents an overhead of ≈700 bits per information item.

A broad and shallow tree is a particularly bad case for the scheme of Akl and Taylor since such a tree will have a large number of leaves, each of which is incomparable with all other security classes. The largest t_i's will occur at the leaves and there will be many of them. The biggest drawback of the Akl and Taylor scheme is that if a new security class is added to the tree the t_i's and K_i's for existing security classes which do not cover the new class will need to be recomputed. If a new leaf is added to a broad and shallow tree this will involve a large fraction of the existing security classes. For instance in a tree with divisions, departments and projects at successive levels the introduction of a new project will require recomputation of the keys for all other projects. It is particularly unfortunate that it will also require recomputation of the keys for all departments and divisions to which the new project does not belong. To avoid recomputation of the keys when a new leaf is added, we might attempt to initially set up a sufficient number of security classes for a large number of divisions, departments and projects and allocate these to new security classes as they get created.

MacKinnon et al [11, 12] describe improvements to the Akl and Taylor scheme which generate smaller t_i's and show that these methods are optimal in generating the smallest values of the t_i's. The major improvement is that instead of assigning a distinct prime to each security class, they first decompose the hierarchy into disjoint chains (linear orderings) and assign a distinct prime only to each chain. The classes in each chain are assigned increasing powers of the prime assigned to the chain. Dilworth [5] has shown that the minimal number of chains needed to a decompose a hierarchy in this manner is equal to the maximum size of an anti-chain (a set of elements which are mutually incomparable). In a tree this equals the number of leaves which in a broad and shallow tree is going to be large. So in our context the optimizations of MacKinnon et al do not appear to have much impact. Moreover changes in the hierarchy are not a part of their criteria so these methods continue to require reassignment of keys whenever the hierarchy is changed [12].

4. CONCLUSION

We have focused on cryptographic techniques for access control in the special case of a tree hierarchy.

We expect these trees are likely to be quite broad and shallow in practise. We also consider it inevitable that new subtrees will need to be added as the requirements of the organization change.

We have reviewed and compared three techniques which use one-way functions in different ways for solving this problem. The technique based on parameterized one-way functions [15] requires no public information, other than the one-way functions. Changes in the hierarchy are conveniently accommodated. The other two techniques require quite substantial amount of public information in addition to the one-way functions, although this is smaller and increases linearly for sealed keys. The technique based on sealing key records [8] can accommodate changes in the hierarchy conveniently. The technique based on exponentiation [1] cannot handle changes in the hierarchy dynamically. There are other published techniques [2, 3, 7, 9] which we have not been able to cover due to lack of time and their omission is in no way intended as a negative comment.

REFERENCES

[1] Akl, S. G. and Taylor, P. D., "Cryptographic Solution to a Problem of Access Control in a Hierarchy", *ACM Transactions on Computer Systems 1*, 3 (Aug. 1983), 239-248.

[2] Denning, D. E. and Schneider, F. B., "Master Keys for Group Sharing", *Information Processing Letters 12*, 1 (Feb. 1981), 23-25.

[3] Denning, D. E., Meijer, H., and Schneider, F. B., "More on Master Keys for Group Sharing", *Information Processing Letters 13*, 3 (Dec. 1981), 125-126.

[4] Denning, D. E., *Cryptography and Data Security,* (Addison-Wesley, 1982).

[5] Dilworth, R. P., "A Decomposition Theorem for Partially Ordered Sets", *Ann. Math. Ser. 2 51* (1950), 161-166.

[6] Evans, A., Kantrowitz, W. and Weiss, E., "A User Authentication System not Requiring Secrecy in the Computer", *CACM 17*, 8 (Aug. 1974), 437-442.

[7] Gifford, D. K., "Cryptographic Sealing for Information Security and Authentication", *CACM 25*, 4 (April 1982), 274-286.

[8] Gudes, E., "The Design of a Cryptography Based Secure File System", *IEEE Trans. Softw. Engg. SE-6*, 5 (Sept. 1980), 411-420.

[9] Ingemarsson, I. and Wong, C. K., "A User Authentication Scheme for Shared Data Based on Trap-Door One-Way Functions", *Information Processing Letters 12*, 2 (Apr. 1981), 63-67.

[10] Lamport, L., "Password Authentication with Insecure Communication", *CACM 24*, 11 (Nov. 1981), 770-772.

[11] MacKinnon, S. and Akl, S. G., "New Key Generation Algorithms for Multilevel Security", *IEEE Symposium on Security and Privacy* (April 1983), 72-78.

[12] MacKinnon, S. J., Taylor, P. D., Meijer, H., and Akl, S. G., "An Optimal Algorithm for Assigning Cryptographic Keys to Control Access in a Hierarchy", *IEEE Transactions on Computers C-34*, 9 (Sept. 1985), 797-802.

[13] National Bureau of Standards, *Data Encryption Standard,* (FIPS Publication 46, NBS, 1977).

[14] Purdy, G. B., "A High Security Log-In Procedure", *CACM 17*, 8 (Aug. 1974), 442-445.

[15] Sandhu, R. S., *Cryptographic Implementation of a Tree Hierarchy for Access Control,* (manuscript submitted to *Information Processing Letters*, April 1987).

[16] Wilkes, M. V., *Time-Sharing Computer Systems,* (Elsevier/MacDonald, 1972).