Information Processing Letters 27 (1988) 95–98
North-Holland

29 February 1988

CRYPTOGRAPHIC IMPLEMENTATION OF A TREE HIERARCHY FOR ACCESS CONTROL

Ravinderpal S. SANDHU

Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210, U.S.A.

Communicated by Fred B. Schneider
Received 1 May 1987
Revised 29 May 1987 and 24 July 1987

A cryptographic implementation is proposed for access control in a situation where users and information items are classified into security classes organized as a rooted tree, with the most privileged security class at the root. Each user stores a single key of fixed size corresponding to the user's security class. Keys for security classes in the subtree below the user's security class are generated from this key by iterative application of one-way functions. New security classes can be defined without altering existing keys. The scheme proposed here is based on conventional cryptosystems (as opposed to public key cryptosystems).

Keywords: Cryptography, cryptographic key, one-way function

1. Introduction

We consider a situation where the *users* and *information items* in a computer or communication system are classified into a rooted tree of *security classes* SC_1, $\mathsf{SC}_2, \ldots, \mathsf{SC}_n$. $\mathsf{SC}_i > \mathsf{SC}_j$ signifies that SC_i is a predecessor of SC_j in the tree while $\mathsf{SC}_i \geqslant \mathsf{SC}_j$ also allows for $\mathsf{SC}_i = \mathsf{SC}_j$. If $\mathsf{SC}_i \geqslant \mathsf{SC}_j$, 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*. The 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.

Let a conventional or symmetric cryptosystem such as DES [8] be available with *enciphering* and *deciphering* procedures E and D respectively. That is, $u = E_K(v)$ is the ciphertext for v using key K and $v = D_K(u)$. A distinct key K_i is assigned to each security class SC_i for encrypting and decrypting information classified in that class. An information item x with sensitivity SC_i is stored (or transmitted) in the system as the pair $[E_{K_i}(x), \mathit{name}(\mathsf{SC}_i)]$. The name of the sensitivity class is appended to indicate how to decrypt the information. Only those users who somehow know K_i will be able to perform the decryption. The access control problem is solved if we can ensure that only those users whose clearance covers SC_i will be able to know K_i.

Akl and Taylor [1] describe an elegant solution for the general problem where the hierarchy on security classes is an arbitrary partial order. Each user stores a single key corresponding to his clearance, from which keys for security classes covered by the user's clearance are computed as needed, whereas it is (apparently) intractable to compute keys for security classes not covered by the user's clearance even in collusion with other users. MacKinnon et al. [6,7] present variations of this method which are optimal with respect to a number of criteria. However, these methods all have the significant disadvantage that when a new security class is added, new keys need to be computed for existing security classes which do not cover the new class. This may involve a large fraction of the existing classes and presents a nontrivial administrative task especially in a distributed environment. Moreover, we must either

re-encipher all previously existing information items with these new keys or require users to remember the previous keys. It is particularly awkward that keys for existing security classes which are completely unrelated to the new class will be changed.

In this paper we present a novel cryptographic solution to this access control problem for the important special case of a rooted tree hierarchy. Our solution, although limited to this special case, has the notable property that new security classes can be defined without affecting keys for existing classes. As in the solutions of [1,7], each user holds exactly one key corresponding to his clearance and the size of this key is the same for all users irrespective of their clearance.

2. The solution

Our solution is based on the well-known idea of *one-way functions* which are easy to compute but computationally difficult to invert [9]. Their use for safeguarding cryptographic keys was suggested by Gudes [4] and has been applied in a number of different contexts [1,2,3,5,6,7]. It is generally accepted that a good cryptosystem can be used to implement a one-way function. A commonly used approach is to encrypt some fixed and publicly known constant c using x as the key, i.e., $f(x) = E_x(c)$. Computing the inverse of $f(x)$ then amounts 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.

Now, consider the situation where the security classes are partially ordered in a rooted tree as discussed earlier. We propose to use a family $\mathscr{F}$ of one-way functions $f_p(x)$ where p is a parameter. We assume that $f_p(x)$ is a one-way function for all p. Furthermore, we require that the functions in $\mathscr{F}$ be independent of each other in a sense which we will make precise shortly. The kind of behavior we have in mind is that it should be infeasible to compute $f_q(x)$ given $f_p(x)$. That is, the value of $f_p(x)$ tells us very little about the value of $f_q(x)$. Since f_p is a one-way function, it is of course infeasible to compute $f_q(x)$ by first inverting $f_p(x)$ to obtain x and then computing $f_q(x)$. We additionally require that there be no other tractable method of computing $f_q(x)$ given $f_p(x)$.

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 $K_j = f_{name(\mathsf{SC}_j)}(K_i)$.

That is, for each child of a security class in the tree we use a different one-way function. The choice of the one-way function is based on the name of the child. A user with security clearance SC_i is given the key K_i. Since the family $\mathscr{F}$ 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 $\mathsf{SC}_j > \mathsf{SC}_i$ since this amounts to the inversion of one or more one-way functions. Finally, it should be computationally infeasible to compute K_j for SC_j incomparable with SC_i. It is this last requirement which motivates the independence condition on the family of one-way functions. A formal statement of the independence condition is beyond the scope of this paper. Nevertheless, we will try to make the requirement precise in the context of a particular family of one-way functions.

We generalize the well-known one-way function $f(x) = E_x(c)$ to obtain a family of one-way functions by replacing the constant c by the parameter of the family, i.e., $f_p(x) = E_x(p)$. Now, inverting $f_p(x)$ amounts to computing the key x given that p encrypts as $f_p(x)$. So, this is a known plaintext attack which is infeasible for a secure cryptosystem. Hence, $f_p(x)$ is a family of one-way functions. If the parameter p is chosen to be the name of the security class, then the keys for the immediate children $\mathsf{SC}_{j_1}, \mathsf{SC}_{j_2}, \ldots, \mathsf{SC}_{j_k}$ of SC_i are generated from K_i as follows:

$$K_{j_1} = E_{K_i}\big(name(\mathsf{SC}_{j_1})\big),$$
$$K_{j_2} = E_{K_i}\big(name(\mathsf{SC}_{j_2})\big),$$
$$\vdots$$
$$K_{j_k} = E_{K_i}\big(name(\mathsf{SC}_{j_k})\big).$$

To ensure that the sizes of all the keys are the same we assume here that the names of the security classes fit within the block size of the cryptosystem. It must be infeasible to compute any of $K_{j_2},\ldots,K_{j_k}$ from K_{j_1}. By the assumed security of the cryptosystem we know that it is infeasible to do so by computing K_i from the fact that $name(\mathsf{SC}_{j_1})$ is encrypted as K_{j_1}. Our independence requirement is that all other methods of computing K_{j_2} from K_{j_1} also be infeasible without some additional information.

To prevent collusion among the siblings we also need the additional property that even with knowledge of $K_{j_2},\ldots,K_{j_k}$ it is infeasible to compute K_{j_1}. Once again, the most direct method of computation amounts to a known plaintext attack to find K_i given $k-1$ pairs of values

$$\left(name\left(\mathsf{SC}_{j_2}\right),\ K_{j_2}\right),\ldots,\left(name\left(\mathsf{SC}_{j_k}\right),\ K_{j_k}\right).$$

This is infeasible for a good cryptosystem. And, again, our independence requirement is that all other methods of computing K_{j_1} from $K_{j_2},\ldots,K_{j_k}$ also be infeasible without some additional information.

The notion of independence of the family of one-way functions has been illustrated for security classes which are siblings. The notion generalizes to arbitrary security classes which are unrelated. By definition, these security classes have a common ancestor in the rooted tree from which their keys are derived by iterative application of one-way functions. Straightforward attacks by inverting these functions to obtain the key for this ancestor are ruled out by the assumed security of the cryptosystem. The independence assumption rules out feasibility of other methods of attack either by an individual user of in collusion with other users.

We believe that this independence requirement will be satisfied if the family of one-way functions $f_p(x)=E_x(p)$ is based on a secure cryptosystem. One has to be a little careful here about some extreme-case situations. It is theoretically conceivable that siblings of all possible names exist. If we know the encrypted form of all but one of these names, we can determine the encrypted form of the remaining name since it must be different from all the others. Situations such as this create difficulties for the formal statement, let alone the proof, of the independence requirement. Such extreme cases are no real threat. For instance, with a block size of 64 bits there are $2^{64}\approx 10^{19}$ possible (plaintext, ciphertext) pairs for each key. Knowledge of all but one of these pairs for a given key requires a formidable amount of storage and would happen in our context anyway only if there were an astronomical number of siblings in the tree.

In theory, it is possible that two security classes which are not siblings get assigned the same key. Consider two security classes with names X and Y whose parents in the tree have keys K_1 and K_2 respectively. Our proposal is to assign these security classes the keys $E_{K_1}(X)$ and $E_{K_2}(Y)$ respectively. Now, for given X, K_1, and K_2 it is trivial to construct a Y such that these two keys are identical. We simply compute $Y=D_{K_2}(E_{K_1}(X))$. However, the likelihood of such a Y being chosen as the name of a security class is exceedingly small.

Finally, we note that there is a problem in using DES to implement our proposal. DES has the peculiar quirk that it uses a 56-bit key to encrypt a 64-bit block of plaintext yielding 64 bits of ciphertext. Since we wish to use the ciphertext as a key it must somehow be reduced to 56 bits. This introduces an inevitable element of degeneracy and the accompanying risk that two 64-bit ciphertexts which are different will be reduced to the same 56-bit key.

3. Some pragmatic issues

We now consider some pragmatic issues regarding the application of our scheme with $f_p(x)=E_x(p)$. To keep the size of keys for all security classes fixed, the names of the security classes must fit within the block size of the cryptosystem, a typical block size being 64 bits. This is an awkward restriction, especially if we contemplate having hundreds of security classes. We might attempt to accommodate longer names by somehow reducing them to 64 bits. But this method has potential security flaws. If two siblings SC_{j_1} and SC_{j_2} have names which are reduced to the same 64-bit quantity, then the keys for both security

classes will be identical. In case the names allowed are considerably bigger than 64 bits, there will be significant degeneracy here.

Fortunately, our proposal can accommodate hierarchical names for the security classes quite readily. That is, immediate children of a security class have distinct names limited to 64 bits, but children of different security classes may have the same name. The unique name of a security class SC consists of the names of all security classes in the path from the root to SC in order, separated by some special character, say "/". 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 $E_{K_i}(name(SC))$ need only be the last field in the unique pathname of SC. If another security class SC_j has a child with the same name as SC, there is no conflict since the key for the child will now be $E_{K_j}(name(SC))$. The name appended to the encrypted form of each information item to identify the sensitivity of the item must be the complete pathname.

One problem with our solution is that when a user with a high security clearance SC_i needs to generate the key for a security class SC_j which is deep down in the tree, an intermediate key must be generated for each security class in the path from SC_i to SC_j. To estimate the computational overhead incurred due to this, consider a tree with ten levels and a cryptosystem with a block size of 8 bytes. A user cleared for the root may need nine applications of the enciphering procedure to derive a key for the security classes at the leaves. If the same cryptosystem is also used for the encryption and decryption of files this represents a possible overhead of $9 \times 8 = 72$ bytes. If the file is big, say 72 000 bytes, the overhead is a negligible 0.1%. On the other hand, for a very small file of say 72 bytes the overhead is 100%. In practice we doubt that trees of greater than ten levels will be used, so these numbers are indicative of the worst case. Moreover, with a cache for derived keys, when the user cleared for the root decrypts several files classified at the leaves, the overhead will be amortized over these files rather than being incurred for each file.

Acknowledgment

We would like to express our appreciation for the referees whose comments clarified several technical details in my mind and in the paper.

References

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

[2] D.E. Denning, H. Meijer and F.B. Schneider, More on master keys for group sharing, *Inform. Process. Lett.* **13** (3) (1981) 125–126.

[3] D.E. Denning and F.B. Schneider, Master keys for group sharing, *Inform. Process. Lett.* **12** (1) (1981) 23–25.

[4] E. Gudes, The design of a cryptography based secure file system, *IEEE Trans. Software Engrg.* **SE-6** (5) (1980) 411–420.

[5] I. Ingemarsson and C.K. Wong, A user authentication scheme for shared data based on trap-door one-way functions, *Inform. Process. Lett.* **12** (2) (1981) 63–67.

[6] S.J. MacKinnon and S.G. Akl, New key generation algorithms for multilevel security, in: *Proc. IEEE Symp. on Security and Privacy* (IEEE Press, 1983) 72–78.

[7] S.J. MacKinnon, P.D. Taylor, H. Meijer and S.G. Akl, An optimal algorithm for assigning cryptographic keys to control access in a hierarchy, *IEEE Trans. Comput.* **C-34** (9) (1985) 797–802.

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

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