Malaysian Journal of Computer Science, Vol. 10 No. 1, June 1997, pp. 36-41

A DYNAMIC ACCESS CONTROL WITH BINARY KEY-PAIR

Md. Rafiqul Islam, Harihodin Selamat and Mohd. Noor Md. Sap
Faculty of Computer Science and Information Systems
Universiti Teknologi Malaysia
Jalan Semarak
54100 Kuala Lumpur
Malaysia
Tel: 603-2904957
Fax: 603-2930933
email: mmcc0004@utmkl.utm.my

ABSTRACT

Based on the concept of an access control matrix, a new dynamic access control scheme with binary key-pair is proposed, which is different from those based on concept of key-lock pairs. In the proposed scheme, each user is assigned a pair of binary keys, which are derived from the access rights with respect to the files. The derivation of the access rights is simple. The method has a special feature, a file or user can be added to or removed from the system without much effort.

Keywords: Dynamic access control, Key-pair

1.0 INTRODUCTION

Access control is very important in information security systems, because of the increasing complexity of various sorts of information, the large number of users, and the widely used communication networks. The issue of information protection includes secrecy, authenticity and availability. The so-called information privacy is defined as a decision-making of a subject's privilege to access certain information. However, information security is a method or a technique by which the decision of information privacy is executed to protect the legitimate access and to reject the illegitimate one.

In 1972 Graham and Denning [2] developed the abstract protection model for computer systems. The model is based upon protection system defined by a triple (*S, O, A)*, Where:

1. *S* is a set of subjects (or accessors), the active entities of the model.
2. *O* is a set of objects (or resources), the protected entities of the models.
3. *A* is an access matrix, with rows and columns corresponding to subjects and objects respectively. An entry a_{ij} lists the access rights (privileges) of subjects S_i and object O_j .

The access control for a computer system is achieved by employing an access control matrix, as depicted in Fig. 1. Here the user U_1 can Execute file F_3 and Execute/Read file F_4 and U_3 can Execute/Read/Write file F_2. In 1984 Wu and Hwang [3] proposed an alternative Scheme storing just one key for each user and one lock for each file. To figure out access rights a_{ij}s of users to files, a function *f* of Key K_i and Lock L_j is used. Mathematically,

$$a_{ij} = f(K_i, L_j) \quad (1)$$

Files / Users	F_1	F_2	F_3	F_4	F_5	F_6
U_1	0	0	1	2	0	0
U_2	4	0	1	0	0	0
U_3	0	3	0	0	2	0
U_4	0	2	0	2	0	3
U_5	4	0	3	0	0	0

0: No access
1: Execute
2: Execute/Read
3: Execute/Read/Write
4: Execute/Read/Write/Delete

Fig. 1: Access control matrix

Several relevant methods appeared in the literature after Wu and Hwang's work [4-9]. Hwang *et al*. in 1992 proposed a protection method using prime factorization [9]. In 1994 Chang *et al*. [10] introduced a method with binary keys. We are inspired by these two methods and proposed an access control scheme using binary pair-keys for each user. From Hwang *et al*.'s method we exploit only the idea of using non-zero entries. The next section reviews Chang *et al*.'s method which is our main inspiration.

2.0 THE BINARY KEY METHOD

This method is proposed by Chang *et al*. [10] for implementation of access control matrix in distributed systems. In this scheme, each user is assigned a binary key, which is derived from the access rights with respect to the files. The binary key is possessed by the user, and can be used to derive the access right to the files.

Here each a_{ij} in access control matrix is rewritten in its binary form b_{ij} as $(b_{ij}^1 b_{ij}^2 \ldots b_{ij}^c)$ where $c = \ddot{e}\ 1 + log\ w\ \hat{u}$ and w is the maximal value of a_{ij}s. The binary form of an access control matrix with m users and n files is depicted in Fig. 2. The key vectors for each user are defined as follows:

$$\begin{aligned} K_{i1} &= (b_{i1}^1 b_{i2}^1 \ldots b_{in}^1), \\ K_{i2} &= (b_{i1}^2 b_{i2}^2 \ldots b_{in}^2), \\ &\vdots \\ K_{ic} &= (b_{i1}^c b_{i2}^c \ldots b_{in}^c). \end{aligned} \qquad (2)$$

If K_{ir}^j is the jth bit in the binary Key K_{ir}, then

$$a_{ij} = (K_{i1}^j K_{i2}^j \ldots K_{ic}^j) \qquad (3)$$

By considering the access control matrix in Fig. 1, a binary access control matrix can be found as shown in Fig. 3.

According to equation (2), the key vectors for users U_1, U_2, U_3, U_4 and U_5 are assigned as [Fig. 3]:

User U_1: $K_{11} = 000000$
$K_{12} = 000100$
$K_{13} = 000000$,
User U_2: $K_{21} = 100000$
$K_{22} = 000000$
$K_{23} = 001000$,
User U_3: $K_{31} = 000010$
$K_{32} = 000000$
$K_{33} = 010000$,
User U_4: $K_{41} = 000000$
$K_{42} = 010101$
$K_{43} = 000001$,
User U_5: $K_{51} = 100000$
$K_{52} = 001000$
$K_{53} = 001000$.

In this method there are c Key vectors for each user. It has been easily noticed that the scheme need to reconstruct the whole system in the case of file deletion and file insertion. This is an important point. On the other hand since the access control matrix is usually a sparse [3, 9], this method has wastage of storage for zero entries.

In order to overcome the above weak points, a new dynamic access control method with pair-keys is proposed. Our proposed method is dynamic in the sense that a new file/user can be deleted from, updated on or joined to the system. The strategy of delete/update changes only pair of keys for dedicated users. The details are described in the next section.

Files / *Users*	F_1	F_2	. . .	F_n
U_1	$(b_{11}^1 b_{11}^2 \cdots b_{11}^c)$	$(b_{12}^1 b_{12}^2 \ldots b_{12}^c)$	. . .	$(b_{1n}^1 b_{1n}^2 \cdots b_{1n}^c)$
U_2	$(b_{21}^1 b_{21}^2 \cdots b_{21}^c)$	$(b_{22}^1 b_{22}^2 \cdots b_{22}^c)$	. . .	$(b_{2n}^1 b_{2n}^2 \cdots b_{2n}^c)$
. . .	. . .	. . .	. . . .	. . .
U_m	$(b_{m1}^1 b_{m1}^2 \ldots b_{m1}^c)$	. . .	. . .	$(b_{mn}^1 b_{mn}^2 \ldots b_{mn}^c)$

Fig. 2: Binary access control matrix with m users and n files

Files / Users	F_1	F_2	F_3	F_4	F_5	F_6
U_1	000	000	001	010	000	000
U_2	100	000	001	000	000	000
U_3	000	011	000	000	100	000
U_4	000	010	000	010	000	011
U_5	100	000	011	000	000	000

Fig. 3: The binary access control matrix for Fig. 1

3.0 THE BINARY KEY-PAIR METHOD

Here we describe the binary key-pair method with respect to binary access control matrix as in Fig. 2 as well as in Fig. 3. In the proposed method each user is assigned a pair of keys. The first key is a logical one and the second key for opening access right. These keys are derived from access rights with respect to the files. The keys are possessed by the user and can be used to derive access right to the files. From the first key we can know whether a specific user has access right to a specific file. Using the bit of logical key we can find out the access right for users to files. Each user U_i is assigned the following two vectors:

$$K_{iL} = K_{iL}^1 K_{iL}^2 \ldots K_{iL}^s \quad (4)$$
for $i = 1, 2, \ldots, n$ and $s \leq n$,

where the xth bit of K_{iL} can be defined as follows:

$$K_{iL}^x = \begin{cases} 0 \text{ if } b_{ij} \text{ is zero-bit string} \\ 1 \text{ otherwise.} \end{cases} \quad (5)$$

If the bit string b_{ij} contains all zero bits, then we will say b_{ij} as zero bit string, otherwise non-zero bit string. The key for access right is defined as follows:

$$K_{iR} = K_{iR}^1 K_{iR}^2 \ldots K_{iR}^c K_{iR}^{2c-c+1} \ldots K_{iR}^{2c} \ldots K_{iR}^{rc-c+1} \ldots K_{iR}^{rc} \quad (7)$$

where r is the number of *1*s in logical key vector K_{iL}, and c is defined as in section 2. That means K_{iR} is built from non-zero b_{ij}s. For instance to check any access right a_{ij} i.e., the access right of user U_i to the file F_j, at first we will examine logical key vector K_{iL} and find whether the user has access right to the file. If the jth bit of K_{iL} is *1*, then there is an access right of user U_i to the file F_j, otherwise *i.e*, if K_{iL}^j bit is zero then there is no access of user U_i to the file F_j. Here we will check the access right using *algorithm 1* that is encoded later.

Example 1: Initialization of key vectors.

From binary access control matrix in Fig. 3, we can define the following key vectors. Since $b_{11} = 000$ (zero bit string), $K_{1L}^1 = 0$ and $b_{13} = 001$ (non-zero bit string), $K_{1L}^3 = 1$.

$K_{1L} = 0011$, $K_{1R} = 001010$,
$K_{2L} = 101$, $K_{2R} = 100001$,
$K_{3L} = 01001$, $K_{3R} = 011100$,
$K_{4L} = 010101$, $K_{4R} = 010010011$,
$K_{5L} = 101$, $K_{5R} = 100011$,

Algorithm 1. Verification of access right.
Input: i, j, K_{iL}, K_{iR}.
Output: a_{ij}.
Steps:

1. Input i, j, K_{iL}, K_{iR};
2. **If** $K_{iL}^j = 1$ then
   begin
   If $j > 1$ then
   p = number of *1*s up to jth bit of K_{iL};
   else
   $p = 1$;
   $a_{ij} = K_{iR}^{pc-c+1} K_{iR}^{pc-c+2} \ldots K_{iR}^{pc}$;
   end;
   else
   a_{ij} = Zeros;
3. Output a_{ij}.

Example 2: *Verification of access right*.

Case I: For instance we want to verify the access right, a_{46}. Here $i = 4, j = 6$, $K_{4L}^6 = 1$, $p = 3$;
so we can write $a_{46} = K_{4R}^7 K_{4R}^8 K_{4R}^9 = 011$, which is correct.

Case II: Check access right, a_{44}. Here $i = 4, j = 4$, $K_{4L}^4 = 1$, $p = 2$; $a_{44} = K_{4R}^4 K_{4R}^5 K_{4R}^6 = 010$,
which is correct.

Case III: Check access right, a_{21}. Here $i = 2$, $j = 1, K^1_{2L} = 1, p = 1; a_{21} = K^1_{2R} K^2_{2R} K^4_{2R} = 100$, which is correct.

4.0 IMPLEMENTATION OF DYNAMIC ACCESS CONTROL

In this section, we devise algorithms to implement the dynamic access control, such as access right changing and file updating (deletion and addition of a file). We will discuss the case of user updating by example, because it can be performed simply by reconstructing or deleting the relevant pair of keys.

Algorithm 2: Access right changing.
/* Let access right a_{ij} be changed by bit string $l_{ij} = l^1_{ij} l^2_{ij} \ldots l^c_{ij}$. */
Input: $j, K_{iL}, K_{iR}, l_{ij}$.
Output: K_{iL}, K_{iR}.
Steps:

1. Input $j, K_{iL}, K_{iR}, l_{ij}$.
2. Compute p = number of *1*s up to *j*th bit of Key vector K_{iL};
   r = number of *1*s in K_{iL};
3. **If** l_{ij} is a non-zero bit string then
   begin
   If $K^j_{iL} = 0$ **then**
   begin
   $p = p + 1$;
   $r = r + 1$;
   Reset bit K^j_{iL} of K_{iL};
   end;
   $K_{iR} = K^1_{iR} K^2_{iR} \ldots K^c_{iR} \ldots K^{(p-1)c}_{iR} l^1_{ij} l^2_{ij} \ldots l^c_{ij} K^{(p+1)c-c+1}_{iR} \ldots K^{rc}_{iR}$;
   end;
   else
   begin
   $K_{iR} = K^1_{iR} K^2_{iR} \ldots K^c_{iR} \ldots K^{(p-1)c}_{iR} K^{(p+1)c-c+1}_{iR} \ldots K^{(r-1)c}_{iR}$
   Reset bit K^j_{iL} of K_{iL};
   end;
4. Output K_{iL}, K_{iR};

Example 3: Changing access right.

Case I: Suppose $a_{43} = 000$ will be changed into $l_{ij} = 100$. In this case $j = 3, K_{4L} = 10101$, $K_{4R} = 010010011$, $K^3_{4L} = 0$, $p = p + 1 = 2$, $r = r + 1 = 4$; then
$K_{4R} = 010100010011$ (updated), and $K_{4L} = 011101$ (by resetting).

Case II: Assume $a_{35} = 100$ will be changed into $l_{ij} = 010$. Here j = 5, $K_{3L} = 01001$, $K_{3R} = 011100$, $K^5_{3L} = 1$, l_{ij} is a non-zero bit string, $p = 2, r = 2$; $K_{3R} = 011010$ (updated).

Case III: If $a_{23} = 001$ by $l_{ij} = 000, j = 3, K_{2L} = 101, K_{2R} = 100001, p = 2$; then
$K_{2R} = 100$, and $K_{2L} = 100$ (by resetting).

Algorithm 3. File Updating.

Section 1: File Addition.

/* Let file F_q is added and the access right of user U_i is denoted as $l_{iq} = l^1_{iq} l^2_{iq} \ldots l^c_{iq}$. */
Input: $q, l_{iq}, K_{iL}, K_{iR}$.
Output: K_{iL}, K_{iR}.
Steps:

1. Input $q, l_{iq}, K_{iL}, K_{iR}$.
2. **If** l_{iq} is zero bit string then
   K_{iL} and K_{iR} remain unchanged;
   else
   begin
   Update K_{iL} by putting *1* in K^q_{iL} position;
   $K_{iR} = K^1_{iR} K^2_{iR} \ldots K^{rc}_{iR} l^1_{iq} l^2_{iq} \ldots l^c_{iq}$;
   end;
3. Output K_{iL}, K_{iR}.

Section 2: File Deletion.
/* Let file F_j is deleted */
Input: j, K_{iL}, K_{iR}.
Output: K_{iL}, K_{iR}.
1. Input j, K_{iL}, K_{iR};
2. **If** $K^j_{iL} = 0$ then
   K_{iL} and K_{iR} remain unchanged;
   else
   begin
   Compute p = number of *1*s up to *j*th bit of K_{iL};
   $K_{iR} = K^1_{iR} K^2_{iR} \ldots K^{(p-1)c}_{iR} K^{(p+1)c-c+1}_{iR} \ldots K^{(r-1)c}_{iR}$;
   Reset bit K^j_{iL} of K_{iL};
   end;
3. Output K_{iL}, K_{iR}.

Example 4: *Addition and deletion of files.*

Here we consider the system in Fig. 3 and let F_7 added to the system that U_2 can write, U_5 can delete. So there will be $a_{27} = 011$ and $a_{57} = 100$. Here it is required to update first K_{iL}, then K_{iR} that is enough.
$K_{2L} = 1010001$ (putting *1* in K^j_{iL} position),
$K_{2R} = 100001011$ (putting bit string);
$K_{5L} = 101001, K_{5R} = 100011100$.
If file F_2 is deleted that U_3 can write, $a_{32} = 011$ and U_4 can read, $a_{42} = 010$. Now we have to update first K_{iR}, then K_{iL}.
Therefore we get, $K_{3R} = 100$ (Shift left),

K_{3L} = *000001* (by resetting bit K_{3L}^{j}); K_{4R} = *0111*, K_{4L} = *000101*.

Example 5: Addition and deletion of users.

Let us reconsider the system in Fig. 3. If U_6 is added to the system, who will read file F_3 and write in file F_5, then we just construct two key vectors K_{6L} and K_{6R} for user U_6.

Suppose U_4 is deleted who can read F_2, F_4 and write in F_6. Here we shall just delete the two key vectors for U_4.

5.0 STORAGE REQUIREMENT

We know that the access control matrix is usually sparse. So here we shall consider non-zero-rate *r*, which is defined as the ratio of non-zero entries in the access control matrix. The storage: For
K_{iL}*s* = *mn* bits (*n* bits for each user, which is maximum).
For K_{iR}*s* = *mnrc* bits.
Then the required storage = (*mnrc* + *mn*) bits.

Example 6: Storage calculation.

Let there are *m = 2000* users and *n = 1000 files*, *c = 3*, *r = 0.1*, then the storage requirement for the system, *mnrc + mn = 2000 ´ 1000 ´ 3 ´ 0.1 + 2000 ´ 1000 = 26,00000* bits. But in case of Chang *et al*.'s method [10], *mnc = 2000 ´ 1000 ´ 3 = 60,00000* bits.
Suppose *r = 0.5*, so *mnrc + mn = 2000 ´ 1000 ´ 3 ´ 0.5 + 2000 ´ 1000 = 30,00000* bits. But in Chang *et al.'s* method same as above,
mnc = 60,00000 bits. That means if the non-zero rate is *50%* , the system in our method takes less storage.
If *c = 5*, then *mnrc + mn = 2000 ´ 1000 ´ 5 ´ 0.1 + 2000 ´ 1000 = 30,00000* bits. But in Chang *et al*.'s method, *mnc = 20000 ´ 1000 ´ 5 = 100,00000* bits.

From the above example it is cleared that our method takes less storage than that of Chang *et al*.'s method.

6.0 THE ADVANTAGES OF THE METHOD

The proposed method has the following advantages:

1) Initialization of key vectors for each user is simple.
2) We propose a simpler procedure of access right checking.
3) Access right changing is easy.
4) Updating users is very simple.
5) Updating files is also easy and dynamic. While the binary key method needs to reconstruct the whole system.
6) Compacted space is used for storing key-pair for each user and the required storage is less than that of binary key method.

7.0 CONCLUSION

In this method we devise algorithms for access right checking and implementation of dynamic access control, such as access right changing and updating files. One good feature of our system is that insertion or deletion of any file can be successfully implemented without reconstructing all key-vectors. The storage requirement is also less than that of Chang *et al*.'s method.

REFERENCES

[1] D. E. R. Denning, *Cryptography and Data Security*. Addison-Wesley, Reading, MA, 1983.

[2] G. S. Graham and P. J. Denning, "Protection-Principle and Practice*"*, *Proc. Spring Joint Computer Conf*., Vol. 40, AFIPS Press, Montvale, NJ, 1972, pp. 417-429.

[3] M. L. Wu and T. Y. Hwang, "Access Control with Single-Key-Lock". *IEEE Transaction on Software Engg*., Vol. SE-10, No. 2, 1984, pp. 185-191.

[4] C. C. Chang, "On the design of a key-lock-pair mechanism in information protection systems *"*. *BIT*, Vol. 26, 1986, pp. 410-417.

[5] C. C. Chang, "An Information Protection Scheme Based upon Number Theory". *The Computer Journal*, Vol. 30, No. 3, 1987, pp. 249-253.

[6] C. K. Chang and T. M. Jiang, "A Binary Single-Key-Lock System for Access Control". *IEE Transaction on Computers*, Vol. 38, No. 10, 1989, pp. 1462-1466.

[7] C. S. Laih, L. Harn and J. Y. Lee, "On the design of a single-key-lock mechanism based on Newton's interpolating polynomial". *IEEE Transaction on Software Engineering*, Vol. 15, No. 9, 1989, pp. 1135-1137

[8] J. K. Jan, C. C. Chang and S. J. Wang, "A dynamic Key-Lock-Pair Access Control Scheme". *Computers & Security*, Vol. 10, 1991, pp. 129-139.

[9] J. J. Hwang, B. M. Shao and P. C. Wang, "A New Access Control Method Using Prime Factorization". *Computer Journal*, Vol. 35, No. 1, 1992, pp. 16-20.

[10] C. C. Chang, J. J. Shen and T. C. Wu, "Access control with binary keys". *Computers & Security*, Vol. 13, 1994, pp. 681-686.

BIOGRAPHY

Md. Rafiqul Islam obtained his Master of Science in Engineering (Computers) from Azerbaijan Polytechnic Institute in 1987. He is an Assistant Professor of Computer Science and Engineering Discipline of Khulna University, Khulna of Bangladesh. Currently, he is on study leave and doing Ph.D. at the Faculty of Computer Science and Information Systems of the Universiti Teknologi Malaysia. His research areas include design and analysis of algorithms, Database security and Cryptography. He has published a number of papers related to these areas. He is an associate member of Bangladesh Computer Society.

Harihodin Selamat holds an M.Sc from Cranfield University, UK and a Ph.D. from the University of Bradford, UK both in computer science. Currently he is an Associate Professor in the Faculty of Computer Science and Information Systems at the Universiti Teknologi Malaysia. His research area includes Database security, Database design and Software engineering.

Mohd Noor Md. Sap is an Associate Professor in the Faculty of Computer Science and Information Systems at Universiti Teknologi Malaysia. He holds degrees in computer science: a B.Sc.(Hon) from the National University of Malaysia, an M.Sc from Cranfield University, UK, and a Ph.D. from the University of Strathclyde, UK. He is currently carrying out research in Database security, Case-based reasoning and Information retrieval.