Malaysian Journal of Computer Science, Vol. 11 No. 1, June 1998, pp. 51-57

IMPLEMENTATION APPROACH OF A DYNAMIC PROTECTION SCHEME WITH BINARY KEY-PAIR

Md. Rafiqul Islam, Harihodin Selamat and Mohd. Noor Md. Sap
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
Jalan Semarak
54100 Kuala Lumpur
Malaysia
Tel: 6-03-2904957
Fax: 6-03-2930933
email: mmcc0004@utmkl.utm.my

ABSTRACT

Describes the implementation approach of a dynamic protection scheme with binary key-pair. The algorithm for checking validation of access request is designed with respect to the implementation approach. Discusses the efficiency of the scheme regarding various searching problems. Brief reviews of binary key method and the binary key-pair method are given. Other implementation methods of access control system that are achieved by employing an access control matrix, have also been reviewed.

Keywords: Access right, dynamic access, key-pair

1.0 INTRODUCTION

Protection is an important issue in a computer system that safeguards the access of multiprogramming environment so that one can use his files, programs, or processes safely and freely share resources with others. The purpose of access control is to limit the operations that a legitimate user of a computer system can perform. Access control is mainly used to prevent information from being destroyed, altered, copied without permission or any other unauthorized usage. In 1972 Graham and Denning [1] developed the abstract protection model for computer systems. The model is based upon protection system defined by a triple *(S, O, A)*, where:
S is a set of subjects (or accessors), the active entities of the model.
O is a set of objects (or resources), the protected entities of the models.
A is an access matrix, with rows and columns corresponding to subjects and objects respectively.

In this paper we describe the implementation approach of a dynamic protection scheme with binary key-pair [2] which is proposed by Islam *et al*. A discussion of various searching problems is also given. For this purpose in next section we shall describe access control matrix and its implementations using various methods. The discussion of access control matrix and its implementation is given in Section 2. The binary key method and the binary key-pair method are reviewed in Section 3 and 4 respectively. Section 5 is devoted to the description of the desired implementation approach. A discussion is given in Section 6. Finally, conclusions appear in Section 7.

2.0 ACCESS MATRIX AND ITS IMPLEMENT-ATION

The protection model of computer system can be represented by an access control matrix [1, 3]. The rows of the matrix represent subjects (users, processes), and the columns of the matrix represent objects (file, disks, or storage segments). Each entry, a_{ij} of the access matrix, represents the access right of the ith subject to the jth object. Here we assume that all the access rights are expressed by numerals. Linear hierarchy of access privileges may be applied here. That means, the right to read implies the right to execute, the right to write implies the right to read and execute and so on. In the following example of Fig. 1, a simple access control matrix is introduced. Here the user S_1 can read object O_1 and write in object O_2 and S_2 can write in O_4.

Objects / *Subjects*	O_1	O_2	O_3	O_4
S_1	*2*	*3*	*5*	*0*
S_2	*4*	*0*	*1*	*3*
S_3	*2*	*1*	*0*	*0*

0- no access, *1*- execute, *2*- read, *3*- write, *4*- delete, *5*- own

Fig. 1: Access Control Matrix

In multiuser system the access matrix will be enormous in size, and most of its entries are likely to be empty [3, 9, 10, 12]. Subsequently, any direct implementation of access matrix is likely to be inefficient. In practice, several methods are used to implement access matrix such as: access control list (ACL) method, capability-based method and key-lock matching method [3, 10, 12-13]. We briefly discuss these methods.

1) *Access control list (ACL) method*: This method corresponds to the column-wise decomposition of the access matrix. Each list is associated with an ACL of pairs (subject, access mode) for all subjects that are permitted to access the object. When a subject requests access to an object, the system searches the access control list of the object to find out whether an entry exists for that subject. If the entry exists, then the system checks whether the required access is allowed, if so, the request is executed, otherwise an error message is raised, and the request is denied. This method is efficient in review of access (examine the access control list of the object) and it is easy to revoke access right (remove the corresponding entry of the subject from the ACL). The inefficiency of this method is the time consumed in searching for all objects that can be accessed by a particular subject (requires examination of each and every ACL list).

2) *Capability-based method*: This method corresponds to the row-wise decomposition of the access matrix. Each subject is associated with a list (called the capability list) of pairs (object, access mode) for all objects that it is permitted to access. In this method, it is easy to review all access that a subject is allowed to perform (by simply examining the capability list of the subject). However, determination of all subjects who can access a particular object is difficult (requires examination of each and every subject's capability list). This method is also inefficient in the revocation of access rights.

3) *Key-lock Matching Method*: The key-lock matching method is a compromise between ACL method and capability-based method. Each list consists of a pair of objects and keys (*O, K*). Each object associates with a list of pairs (*L, R*), the locks and access rights. While a subject sends request to access an object with an access mode *A*, the access control system searches for the specified capability list (*O, K*) and the access-list (*L, R*) for the object. The access is allowed if *L* matches *K* and *A* matches *R*. This method suffers from the difficulty encountered in searching both a list of keys and a list of locks.

In 1984 Wu and Hwang [3] proposed an alternative scheme storing just one key for each subject and one lock for each file. To figure out access rights a_{ij}'s of users to objects, a function *f* of key K_i and lock L_j is used. Mathematically,

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

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*. [11] introduced a method with binary keys. Islam *et al*. [2] proposed a dynamic access control scheme with binary key-pair. Throughout the paper we shall call Chang *et al*.'s method with binary keys as *binary key method* and Islam *et al*.'s dynamic protection scheme with binary key-pair as *binary key-pair method*. A description of the implementation approach of the binary key-pair method is given. The algorithm for checking validation of access request is designed with respect to the implementation approach.

3.0 THE BINARY KEY METHOD

This method is proposed by Chang *et al*. [11] for the implementation of access control matrix in distributed systems. In this scheme, each subject is assigned a binary key, which is derived from the access rights with respect to the objects. The binary key is possessed by the subject, and can be used to derive the access right to the objects. Here each a_{ij} in access control matrix is rewritten in its binary form b_{ij} as $(b_{ij}^1 b_{ij}^2 \cdots b_{ij}^c)$ where $c = \ddot{e}\, 1 + log\ w\ \hat{u}$ and *w* is the maximal value of a_{ij}'s. The key vectors for each subject are defined as follows:

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

If K_{ir}^j is the *j*th bit in the binary key K_{ir}, then

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

By considering the access control matrix in Fig. 1, a binary access control matrix can be found as shown in Fig. 2.

Objects / Subjects	O_1	O_2	O_3	O_4
S_1	*010*	*011*	*101*	*000*
S_2	*100*	*000*	*001*	*011*
S_3	*010*	*001*	*000*	*000*

Fig. 2: The binary access control matrix for Fig. 1

According to equation (2) and from Fig. 2, the key vectors for subjects S_1, S_2 and S_3 are assigned as:

Subject S_1: $K_{11} = 0010$,
$K_{12} = 1100$,
$K_{13} = 0110$,
Subject S_2: $K_{21} = 1000$,
$K_{22} = 0001$,
$K_{23} = 0011$,
Subject S_3: $K_{31} = 0000$,
$K_{32} = 1000$,
$K_{33} = 0100$,

In this method there are *c* key vectors for each subject. It has been easily noticed that the scheme needs to reconstruct the whole system in the case of file deletion and file insertion. On the other hand since the access control matrix usually sparse [3, 9, 12], this method has wastage of storage for zero entries. In order to overcome the above drawbacks, Islam *et al*. [2] proposed the binary key-pair method described below.

4.0 THE BINARY KEY-PAIR METHOD

Here we describe the binary key-pair method with respect to binary access control matrix as in Fig. 2. In this method each subject is assigned two keys. The first key is a logical one and the second key for opening access rights. These keys are derived from access rights with respect to the objects. The keys are possessed by the subject and can be used to derive access right to the objects. From the first key we can know whether a subject has an access to a specific object. Using the bits of logical key we can find the access rights for users to the objects. Each subject S_i is assigned the following two vectors:

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

where *x*th bit of K_{iL}, $K_{iL}^x = 0$ or *1*; *0* for zero bit- string and *1* for non-zero bit-string.

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

$$K_{iR} = K_{iR}^1 K_{iR}^2 \ldots K_{iR}^c K_{iR}^{2c-c+1} \ldots K_{iR}^{2c} \ldots K_{iR}^{rc-c+1} \ldots K_{iR}^{rc} \quad (5)$$

where, *r* is the number of *1*s in logical key vector *KiL*, and *c* is defined as in section 3. 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 S_i to the file O_j at first we examine logical key vector K_{iL} and find whether the user has access to the file. If the *j*th bit of K_{iL} is *1*, then there is an access of the subject S_i to O_j, otherwise *i.e*, if K_{iL}^j bit is zero then the subject S_i has no access to the object O_j. Let us see how to initialize key vectors. From binary access control matrix in Fig. 2, we can define the following key vectors. Since $b_{11} = 010$ (non-zero bit-string), $K_{1L}^1 = 1$ and $b_{14} = 000$ (zero bit-string), $K_{1L}^4 = 0$ and so on.

$K_{1L} = 1110$, $K_{1R} = 010011101$,
$K_{2L} = 1011$, $K_{2R} = 100001011$,
$K_{3L} = 1100$, $K_{3R} = 010001$.

5.0 IMPLEMENTATION APPROACH OF THE BINARY KEY-PAIR METHOD

Suppose subjects are stored in a list and each subject is associated with a subject number, *SN*. So, if there are *m* subjects, then *SN* ranges from *1* to *m.* Similarly objects are stored in a list and each object is given a object number, *ON* that ranges from *1* to *n* (for *n* objects). Now, with respect to these two lists we shall have to create another list that will hold logical keys and open keys for access rights for all subjects. The keys are stored in a list using the following structure:

```
struct keyvec {
        int arraykl[k];
        int arraykr[s];
        int noofslotinkr;
      }Keyvec;
```

The structure **Keyvec** has two fields of arrays *arraykl* and *arraykr.* The field *noofslotinkr* is required for tracing the number of required slots in the array *arraykr.* Since the length of key vectors K_{ir} for all user is not fixed, so we need to trace the number of slots in the array *arraykr.* We discuss this approach considering Borland C++ implementation. The space for the arrays is dynamically allocated and we write only the used slots of the arrays in the list of key vectors. For each user there is a particular key vector that means specific used structure. As the lengths of key vectors K_{iL}'s are same for all users, so the number of used slots of *arraykl* will be same for each of the users. Hence, the number of required slots of *arraykl* is stored using variable *noofslotinkl* at the beginning of the list and no need to include it in the structure. Binary digits are stored in a*rraykl* using binary shift operator and each slot of the array contains *15* binary digits (without sign bit of integer data type). It can be easily done depending on the decision whether a subject has any access to a object or not. Fifteen binary digits in each slot give us an advantage to group the bits in three (one octal digit) that may be used for checking the number stored in the slot.

Now we discuss how access rights are stored in *arraykr.* If we see carefully, then it can be found that the key vectors are built from the series of digits (numbers for access rights) and these digits may range from 1 to a_{max}, where a_{max} is the maximum of access rights. If a_{max} is less than or equal to 7, then we can store the access rights using base 8 integer (octal number). If it is greater than 7, then hexadecimal number can be used and the value of a_{max} ranges up to 15. Suppose $a_{max} <= 7$ and octal number is used. Thus each slot of *arraykr* contains 5 digits (3 bits for each octal digit) octal number. Octal digits can be stored by shifting *3* bits at a time. If there are *30* bits (*10* octal digits) in K_{iR}, then we need two slots of *arraykr* to store the bits. From the above discussion it can be put down that the number of slots in *arraykl*, $k = \lceil n / 15 \rceil$. But the total number of slots in *arraykr* does not depend on *n* (total number of objects) but

on the number of objects (the number of *1*s in K_{iL}) that can be accessed by a user. Suppose a user has access to *p* objects, then $s = \lceil p/5 \rceil$.

Here we discuss how we can check an access right of a subject to an object. A request to access an object is sent in the form of a triple (*S*, *O*, *R*), where, *S*- subject, *O*- object and *R*- request (access mode). Here, subject is identified by a subject identification number, *Subid*, an object is identified by object name, *Objname* and the request, *R* is sent in numerical form such as *1*, *2*, *3* etc. (as defined above). Thus a triple (*Sub10*, *Obj20*, *3*) means that subject *Sub10* wants a write access to the object *Obj20*. To validate *Subid* and *Objname* we have to search the lists of *Subid*s and *Objname*s (it has been mentioned at the beginning of this section that subjects and objects are stored in lists). If the *Subid* and *Objname* are valid, then we check whether the subject has an access to the object or not (it can be done by checking bits of the respective slots of *arraykl*). We get subject number, *SN* and object number, *ON* from previous searching results and with *SN* and *ON* we can find respective slot of *arraykl* and *arraykr*. If the subject has an access to the object, then we find the access right from the respective slot of *arraykr* (key vector) of the subject. When the requesting access mode is equal to or less than the access right from *arraykr*, then the subject is allowed to access the object, otherwise the access is denied. With respect to the above discussion the algorithm for validation of access request is encoded in access_request_validation algorithm of *Appendix*.

6.0 DISCUSSIONS

In binary key-pair method to search for all the objects that can be accessed by a particular subject, we have to count number of *1*s in K_{iL} *(arraykl)* of the subject and by taking the position of *1*s of $K_{iL,}$, we can find out the objects from the object list. That means we can easily find out the objects that can be accessed by a particular subject. However, in ACL method this kind of search is difficult, because it requires checking of each and every ACL. On the other hand all the subjects who can access particular object can be determined by finding out the object number *j* from object list and by checking *j*th bit of K_{iL} of the subjects. It is efficient because searching for a specific bit in K_{iL} is fast. In capability-based list, searching for all subjects who can access a particular object is inefficient. Looking for key-pair is also efficient. We have implemented the algorithm *1* that is used for validation of access request using Borland C++ programming language with some test data. The experimental results of checking time of access right for a subject list with *1000* users (subjects) and a object list with *2000* files (objects) is depicted in *Table 1*.

From the table, it is noticed that checking time for a user is approximately same for all files. However, the searching time increases with increment of user number. The average searching time, $t_{ack} = 0.053$ sec for $i = 150$, $t_{ack} = 0.203$ sec for $i = 550$ and $t_{ack} = 0.35$ sec for $i = 980$. The above checking times are taken whenever the subject has an access to the object. The binary key-pair method is very fast in verifying access right if the subject has no access (0) to the object. To revoke access right in binary key-pair method we have to reset respective bit of K_{iL} and update K_{iR} of the subject using shift operations.

7.0 CONCLUSIONS

We have described here the implementation approach of a dynamic protection scheme with binary key-pair. We also discussed various searching problems and compared the efficiency of the scheme with other implementation approaches of the access control matrix. The algorithm for checking validation of access request is designed and some data (searching time) are taken by implementing the algorithm using Borland C++ programming language. Short descriptions of the binary key method and the binary key-pair method are also given.

Table 1: Table of checking time of access right
(Total number of users, $m = 1000$ and total number of files, $n = 2000$)

Subject Number, i	***Object Number, j***	***Position of j with respect to object list***	***Checking time, t_{ck} in second***
150	52	beginning	0.05
150	985	middle	0.06
150	1988	end	0.05
505	125	beginning	0.18
550	1025	middle	0.22
550	1898	end	0.21
980	122	beginning	0.33
980	1134	middle	0.38
980	1987	end	0.34

ACKNOWLEDGEMENTS

The authors wish to thank anonymous reviewers for their suggestions to improve the presentation of this paper.

REFERENCES

[1] 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.

[2] M. R. Islam, H. Selamat and M. N. M. Sap, "A dynamic access control with binary key-pair", *Malaysian Journal of Computer Science,* Vol. 10, June 1997, pp. 36-41.

[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", *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", *IEEE 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] T. Wu, "A refined key-lock access control system", *Proc. IEEE 1993 national aerospace and electronics conference*, May 1993, pp. 583-587.

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

[12] R. S. Sandhu and P. Samarati, "Access control: principle and practice", *IEEE communications magazine*, Sept. 1994, pp. 40-48.

[13] L. Gong, *http://ei.cs.vt.edu/~cs5204/protection.basic.htm/*, 1996.

[14] D. E. R. Denning, *Cryptography and data security*; Addison-Wesley, Reading, MA, 1983.

APPENDIX

Access_ request_ validation algorithm.

// Suppose there are *m* subjects and *n* objects. *Subid*s are stored in a list (file) named *sublst*, *Objname*s are stored in a list (file) named *objlst* and the key vectors are stored in a list (file) named *keylst*. To search respective mode through *arraykl* and *arraykr* we use concept of binary search i.e., If *FN* (file number) is less than or equal to *n /2*, then search is made starting from beginning point of the *arraykr*, otherwise from the end of the array. This technique is efficient, because there are many slots of *arraykr* for large *n*. Suppose, we wish to validate the access to object *O* and its number is *j* in *objlst*. To find out the respective access mode from *arraykr* we have to count number of *1*s up to *j* bit of *arraykl.* If $j < n / 2$, then the number of *1*s in *arraykl* is counted starting from first slot up to j bit as well as search is made from the beginning of *arraykr.* Otherwise, counting of number of *1*s in *arraykl* and search for access right is made from the last slot of *arraykl* and *arraykr* respectively. Here % is remainder operator, >> is bit shift operator and := is assignment operator.//

1. **Input**: triple (*S*, *O*, *R*), lists *sublst*, *objlst* and *keylst*.
2. **Output** : permission or denial of access.

Step 1: Enter the triple (*S*, *O*, *R*);

Step 2: Take *S* from the triple and *f1:= 0*;
// This is the step for checking validation of S //
While *f1 ¹ 1* or not end of file *sublst* **do**
Begin
Read data from file *sublst* and take *Subid*;
If *Subid = S* **then** *f1 := 1*;
sno := SN;
End;
If *f1 = 0* **then** *S* is not a valid id and exit;

Step 3: **If** *f1 = 1* **then**
// If *S* is valid then check validation of *O* //
Begin
f2 := 0;
While *f2 ¹ 1* or not end of file *objlst* **do**
Begin
Read data from file *objlst* and take *Objname*;
If *Objname = O* **then** *f2 := 1*;

```
            fno := ON;
            End;
             End;
 If f2 = 0 then O is not valid object name and exit;

step 4:  If f2 =1 then
 // If S and O are valid then Start finding respect
          Begin
  // access mode that is stored in arraykr //
          i:=0;
          While i < sno do
           Begin
           Read data from file keylst
/ /read respective keyvector of the subject // ;
           i := i +1;
           End;
           End;
           j := (fno -1)/15;
// Effort to find out respective bit from arraykl. //
           lkey := Keyvec.arraykl[ j ];
            If n < (j + 1) ´ 15
            Begin
            ndig :=0;
            If j = 0 then ndig :=n;
            Else ndig := n - j *15;
            End;
            Else ndig :=15;
            If j =0 then shd :=ndig -fno;
            Else
             Begin
            shd :=sno - 15 *j;
            shd := ndig -shd;
            End;
             lmod :=(lkey>>shd) % 2;
If  lmod =0 then the subject has no access to the object and
exit;

Step 5:  Else
           Begin
           // if lmod =1 //
             mid :=n / 2;
             nbit:=0;
             If fno <= mid then
       //Concept of binary search is used. //
              Begin
              for i :=0 to j do
               Begin
               temp := Keyvec.arraykl[ i ];
               While temp ¹ 0 do
               Begin
               lbit := temp % 2;
                If lbit = 1 then nbit := nbit + 1;
              temp := temp / 2;
              End;
              End;
              temp : = lkey >> shd;
              While temp ¹ 0 do
                Begin
                  lbit := temp %2;
                If lbit = 1 then nbit := nbit + 1;
                 temp = temp /2;
                 End;
                i := 0;
                i := (nbit -1) / 5;
               rkey := Keyvec.arraykr[ i ];
               temp :=  rkey;
              ndig := 0;
              shd := 0;
              While temp ¹ 0 do
               Begin
               temp := temp / 8;
               ndig :=  ndig + 1;
               End;
               shd := (i + 1) *5 - nbit;
               End
// end of loop if sno <= mid //
            Else
        // If sno > mid //
           Begin
            i :=0;
            temp := lkey;
            While i <=  shd do
             Begin
            lbit := temp % 2;
           If  lbit = 1 then nbit := nbit + 1;
           temp := temp / 2;
           End;
          for i :=  j + 1 to nofslotinkl - 1 do
           Begin
           temp := Keyvec.arraykl[ i ];
           While temp ¹ 0 do
           Begin
            lbit := tem % 2;
           If lbit = 1 then nbit := nbit + 1;
            temp := temp / 2;
             End;
              End;         // end of for loop //
            i := Keyvec.noofslotinkr - 1;
           temp := Keyvec.arraykr[ i ];
           ndig :=0;
          While temp ¹ 0 do
           Begin
            temp := temp / 8;
            ndig := ndig + 1;
           End;
          If nbit <= ndig
          Begin
          shd := nbit - 1;
         rkey :=Keyvec.arraykt[ i ];
         End;
        Else   // if nbit > ndig //
        Begin
        i := i - 1;
        nbit := nbit - ndig;
```

```
    ns := 0;
   ns := (nbit -1) / 5;
    i := i - ns;
  shd := nbit -ns *5;
 shd := shd -1;
       rkey := Keyvec.arraykr [ i ];
      End;
      End;
 // end of loop else i.e., if sno > mid //
    mod := (rkey >> shd *3) % 8;
     End;
 // end of loop else i.e., if lmod = 1 //
```

Step 6: **If** *R* <= *mod* **tehn** access to object *O* is allowed;
Else access is denied;

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, Bangladesh. Currently, he is on study leave and doing Ph.D at Faculty of Computer Science and Information Systems, in 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 MSc from Cranfield University, UK and a Ph.D from the University of Bradford, UK both in computer science. Currently he is an Associate Professor at the Faculty of Computer Science and Information Systems in Universiti Teknologi Malaysia. His research areas include Database security, Database design and Software engineering.

Mohd Noor Md. Sap is an Associate Professor at the Faculty of Computer Science and Information Systems in Universiti Teknologi Malaysia. a B.Sc. (Hons) from the National University of Malaysia, an MSc 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.