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
e-mail: 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 model. \(A\) 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 IMPLEMENTATION

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 \(i\)th subject to the \(j\)th 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 \(O_4\).

![Fig. 1: Access Control Matrix](image)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S_1)</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>(S_2)</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(S_3)</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

0- no access, 1- execute, 2- read, 3- write, 4- delete, 5- own
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 subject’s key, a function f of key K and lock L is used. Mathematically,

\[ a_i = f(K, L) \]  

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 access control matrix is rewritten in its binary form \[ b_i = (b^1_i, b^2_i, \ldots, b^n_i) \] where \( c = L + \log w \) and \( w \) is the maximal value of \( a_i \). The key vectors for each subject are defined as follows:

\[
\begin{align*}
K_{ij} &= (b^1_i, b^2_i, \ldots, b^n_i), \\
K_{j} &= (b^1_j, b^2_j, \ldots, b^n_j).
\end{align*}
\]  

(2)

If \( K'_{ij} \) is the \( j \)th bit in the binary key \( K_{ij} \), then

\[ a_i = (K'_{i1}, K'_{i2}, \ldots, K'_{in}) \]  

(3)

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

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objects</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>010</td>
<td>011</td>
<td>101</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>100</td>
<td>000</td>
<td>001</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>010</td>
<td>001</td>
<td>000</td>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

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 \( S_1 \):**
\[ K_{i1} = 0010, \]
\[ K_{i2} = 1100, \]
\[ K_{i3} = 0110, \]
**Subject \( S_2 \):**
\[ K_{j1} = 1000, \]
\[ K_{j2} = 0001, \]
\[ K_{j3} = 0011, \]
**Subject \( S_3 \):**
\[ K_{i1} = 0000, \]
\[ K_{i2} = 1000, \]
\[ K_{i3} = 0100, \]
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 \( S \) is assigned the following two vectors:

\[
K_{ir} = K_{ir}^1 \cdot K_{ir}^2 \cdot \ldots \cdot K_{ir}^c
\]

for \( i = 1, 2, \ldots, n \) and \( s \leq n \),

where \( x \)th bit of \( K_{ir}^t \) is 0 or 1; 0 for zero bit-string and 1 for non-zero bit-string.

If the bit-string of an access right \( b_j \) contains all zero bits, then \( b_j \) is a zero bit-string, otherwise non-zero bit-string. The key for access right is defined as follows:

\[
K_{ir} = K_{ir}^1 \cdot K_{ir}^2 \cdot \ldots \cdot K_{ir}^c \cdot K_{ir}^{c+1} \cdot \ldots \cdot K_{ir}^n
\]

\[
\ldots \cdot K_{ir}^{c+e} \cdot \ldots \cdot K_{ir}^n
\]

where, \( r \) is the number of '1's in logical key vector \( K_{ir} \), and \( c \) is defined as in section 3. That means \( K_{ir} \) is built from non-zero \( b_j \)'s. For instance, to check any access right \( a_{j,i} \), i.e., the access right of user \( S \), to the file \( O \), at first we examine logical key vector \( K_{ir} \) and find whether the user has access to the file. If the \( j \)th bit of \( K_{ir} \) is 1, then there is an access of the subject \( S \) to \( O \), otherwise i.e., if \( K_{ir}^j \) is bit zero then the subject \( S \) has no access to the object \( O \). Let us see how to initialize key vectors. From binary access control matrix in Fig. 2, we can define the following key vectors. Since \( b_{i1} = 010 \) (non-zero bit-string), \( K_{i1}^t = 1 \) and \( b_{i3} = 000 \) (zero bit-string), \( K_{i3}^t = 0 \) and so on.

\[
K_{i1} = 1110, \quad K_{i1} = 010011101,
K_{i2} = 1011, \quad K_{i3} = 100001011,
K_{i3} = 1100, \quad K_{i3} = 010001.
\]

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 \( K_r \) 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 \( K_r \)’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 `arraykl` 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 an 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 $Is$ in $K_d$) 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 $Subids$ and $Objnames$ (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 $Is$ in $K_d$ ($arraykl$) of the subject and by taking the position of $Is$ of $K_d$, 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 $K_d$ of the subjects. It is efficient because searching for a specific bit in $K_d$ 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 $K_d$ and update $K_d$ 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>
<thead>
<tr>
<th>Subject Number, $i$</th>
<th>Object Number, $j$</th>
<th>Position of $j$ with respect to object list</th>
<th>Checking time, $t_{ack}$ in second</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>52</td>
<td>beginning</td>
<td>0.05</td>
</tr>
<tr>
<td>150</td>
<td>985</td>
<td>middle</td>
<td>0.06</td>
</tr>
<tr>
<td>150</td>
<td>1988</td>
<td>end</td>
<td>0.05</td>
</tr>
<tr>
<td>505</td>
<td>125</td>
<td>beginning</td>
<td>0.18</td>
</tr>
<tr>
<td>550</td>
<td>1025</td>
<td>middle</td>
<td>0.22</td>
</tr>
<tr>
<td>550</td>
<td>1898</td>
<td>end</td>
<td>0.21</td>
</tr>
<tr>
<td>980</td>
<td>122</td>
<td>beginning</td>
<td>0.33</td>
</tr>
<tr>
<td>980</td>
<td>1134</td>
<td>middle</td>
<td>0.38</td>
</tr>
<tr>
<td>980</td>
<td>1987</td>
<td>end</td>
<td>0.34</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

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

REFERENCES


APPENDIX

Access_request_validation_algorithm.
// Suppose there are m subjects and n objects. Subids are stored in a list (file) named sublst, Objnames 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 F N (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 1s up to j bit of arraykl. If j < n/2, then the number of 1s 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 1s 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 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 or not end of file objlst do
Begin
Read data from file objlst and take Objname;
If Objname = O then f2 := 1;

```plaintext
\begin{align*}
\text{fno} & := \text{ON;} \\
\text{End;} \\
\text{End;}
\end{align*} \\
\textbf{If} f2 = 0 \textbf{then} O \text{ is not valid object name and exit;

step 4:} \textbf{If} f2 =1 \textbf{then} \\
// If S and O are valid then Start finding respect \\
// access mode that is stored in \textit{arraykr}; \\
\textbf{Begin} \\
\text{While} i < \text{sno} \textbf{do} \\
\text{Begin} \\
\text{Read data from file \textit{keylst}} \\
// \textit{read respective keyvector of the subject} //; \\
i := i +1; \\
\text{End;} \\
\text{End;}
\textbf{If} f2 = 0 \textbf{then} O \text{ is not valid object name and exit;}

Step 5: \textbf{Else} \\
\textbf{Begin} \\
//Concept of binary search is used. // \\
\textbf{Begin} \\
\text{for} i :=0 \text{to} j \textbf{do} \\
\text{Begin} \\
\text{temp} := \textit{Keyvec.arraykl[ i ]}; \\
\textbf{While} \textit{temp} \neq 0 \textbf{do} \\
\text{Begin} \\
lbit := \textit{temp} \% 2; \\
\textbf{If} lbit = 1 \textbf{then} nbit := nbit + 1; \\
\textit{temp} := \textit{temp} / 2; \\
\text{End;} \\
\text{End;} \\
temp := lkey >> shd; \\
\textbf{While} \textit{temp} \neq 0 \textbf{do} \\
\text{Begin} \\
lbit := \textit{temp} \% 2; \\
\textbf{If} lbit = 1 \textbf{then} nbit := nbit + 1; \\
\textit{temp} := \textit{temp} / 2; \\
\text{End;} \\
\text{End;} \\
\text{End;}
\textbf{End;}
\end{align*}

\begin{align*}
\text{temp} & := \textit{Keyvec.arraykr[ i ]}; \\
\text{rkey} & := \textit{Keyvec.arraykr[ i ]}; \\
\textit{temp} & := \textit{Keyvec.arraykr[ i ]}; \\
\text{shd} & := (i +1) \times 5 - \text{nbit}; \\
\textbf{End} \hspace{1cm} // \text{end of loop if sno} \leq \text{mid} //
\end{align*}

\begin{align*}
\text{Else} \\
// \text{If sno} > \text{mid} // \\
\textbf{Begin} \\
i :=0; \\
\text{temp} := lkey; \\
\textbf{While} i < \text{shd} \textbf{do} \\
\text{Begin} \\
lbit := \textit{temp} \% 2; \\
\textbf{If} lbit = 1 \textbf{then} nbit := nbit + 1; \\
\textit{temp} := \textit{temp} / 2; \\
\text{End;} \\
\text{End;} \\
\textbf{for} i := j +1 \text{ to \textit{nofslotinkl} - 1} \textbf{do} \\
\text{Begin} \\
\text{temp} := \textit{Keyvec.arraykl[ i ]}; \\
\textbf{While} \textit{temp} \neq 0 \textbf{do} \\
\text{Begin} \\
lbit := \textit{temp} \% 2; \\
\textbf{If} lbit = 1 \textbf{then} nbit := nbit + 1; \\
\textit{temp} := \textit{temp} / 2; \\
\text{End;} \\
\text{End;} \\
i := \textit{Keyvec.nofslotinkr} - 1; \\
\text{temp} := \textit{Keyvec.arraykr[ i ]}; \\
\textbf{While} \textit{temp} \neq 0 \textbf{do} \\
\text{Begin} \\
lbit := \textit{temp} / 8; \\
\textit{ndig} := \textit{ndig} + 1; \\
\text{End;} \\
\textbf{If} nbit < \text{ndig} \textbf{Begin} \\
\textit{shd} := \textit{nbit} - 1; \\
\text{rkey} := \textit{Keyvec.arraykt[ i ]}; \\
\text{End;} \\
\textbf{Else} // \text{if nbit} > \text{ndig} // \\
\textbf{Begin} \\
i := i - 1; \\
nbit := nbit - \text{ndig}; \\
\end{align*}
```
Implementation Approach of a Dynamic Protection Scheme with Binary Key-Pair

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 then 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.