Lightweight Symmetric Encryption Algorithm for Secure Database

Hanan A. Al-Souly, Abeer S. Al-Sheddi, Heba A. Kurdi
Computer Science Department, Computer and Information Sciences College
Imam Muhammad Ibn Saud Islamic University
Riyadh, Saudi Arabia

Abstract—**Virtually all of today's organizations store their data in huge databases to retrieve, manipulate and share them in an efficient way. Due to the popularity of databases for storing important and critical data, they are becoming subject to an overwhelming range of threats, such as unauthorized access. Such a threat can result in severe financial or privacy problems, as well as other corruptions. To tackle possible threats, numerous security mechanisms have emerged to protect data housed in databases. Among the most successful database security mechanisms is database encryption. This has the potential to secure the data at rest by converting the data into a form that cannot be easily understood by unauthorized persons. Many encryption algorithms have been proposed, such as Transposition-Substitution-Folding-Shifting encryption algorithm (TSFS), Data Encryption Standard (DES), and Advanced Encryption Standard (AES) algorithms. Each algorithm has advantages and disadvantages, leaving room for optimization in different ways. This paper proposes enhancing the TSFS algorithm by extending its data set to special characters, as well as correcting its substitution and shifting steps to avoid the errors occurring during the decryption process. Experimental results demonstrate the superiority of the proposed algorithm, as it has outperformed the well-established benchmark algorithms, DES and AES, in terms of query execution time and database added size.**

Keywords—Encryption; Security; Protection; Transposition; Substitution; Folding; Shifting

I. INTRODUCTION

The tremendous development of technology and data storage leads organizations to depend on database systems. Organizations store huge amounts of data in secured databases in order to retrieve them in a fast and secure way. Some of the stored data is considered sensitive and has to be protected.

In the presence of security threats, database security is becoming one of the most urgent challenges because much damage to data can happen if it suffers from attacks and unauthorized access. With databases in complex, multi-tiered applications, attackers may reach the information inside the database. Damage and misuse of sensitive data that is stored in a database does not only affect a single user; but possibly an entire organization [1]. We can categorize the attackers into three types: intruder, insider, and administrator. Intruders are external people who infiltrate a database server to steal or tamper with data. Insiders are authorized users in a database system, who conduct some malicious works. Administrators can be database administrators (DBA) or system administrators (SA), and both have absolute rights to database systems. However, if they are malicious, the security of the database may be damaged [2]. Insider and administrator attackers have gathered more attention in recent years because they can access a database without any effort, and they use important data in a wrong way. Database encryption has the potential to secure data at rest by providing data encryption, especially for sensitive data, avoiding the risks such as misuse of the data [1]. In order to achieve a high level of security, the complexity of encryption algorithms should be increased with minimal damage to database efficiency, ensuring performance is not affected.

There are many research studies in the database security field. Some of them have efficient implementations. Also, many encryption algorithms have been proposed, some of which have appealing features but still need further development, one such algorithm is the Transposition, Substitution, Folding and Shifting TSFS algorithm, known as the TSFS algorithm [1]. The TSFS algorithm provides a high degree of security, using a number of features. However, it supports only numbers and alphabetic characters that are not enough to protect different types of sensitive data. Another deficit of the TSFS algorithm is during the substitution and shifting processes where some errors occur during the decryption process.

This paper provides a secure and efficient encryption method that encrypts only sensitive data without using special hardware. It enhances TSFS algorithm by extending its data set to special characters, and corrects substitution and shifting processes, by providing more than one modulo factor and four 16-arrays respectively in order to avoid the error that occurs during the decryption steps. Moreover, this paper draws a comparison between the enhanced TSFS algorithm (ETSFS) and two other famous encryption algorithms, namely Data Encryption Standard (DES) and Advanced Encryption Standard (AES) algorithms, and evaluates their performance in terms of query execution time and database added size.

The remaining parts of this paper are organized as follows: section 2 reviews existing work on database encryption techniques. Section 3 introduces the ETSFS algorithm and explains its procedure, while section 4 introduces the implementation of the ETSFS algorithm and suggested structure. Section 5 presents a comparative study between the algorithms, evaluates performance, reports results and discusses them. Finally, section 6 concludes with a summary of contributions and makes suggestions for future research work.

II. Related Work

Due to the important role that encryption techniques play in securing database systems, numerous algorithms have emerged with different techniques and performance. Bouganim and Pucheral proposed a smart card solution to protect data privacy; the owners of databases can access the data using a client terminal that is supported by smart card devices [3]. This proposed solution is considered as a secure and an effective solution, but it is complicated and expensive [1]. Database encryption greatly affects database performance because each time a query runs, a large amount of data must be decrypted. Therefore, [4] suggests that encrypting sensitive data only can provide the needed security without affecting the performance.

In [1] [5] [6] encryption algorithms were proposed depending on encryption of sensitive data only. Kaur et al. proposed a technique to encrypt numeric data only using a fixed data field type and length [5]. However, this algorithm does not support encryption of character data. Agrawal et al [6] also, proposed an encryption scheme for numeric data with an important feature that allows queries or any comparison operations to be applied directly on encrypted data sets without decrypting them. The scheme uses indexes of database over encrypted tables, but it is only applied to numeric data, additionally, it has not investigated key management. In some application, where the data is backed up frequently, we need to control the access to data and support multilevel access. So, Hwang and Yangb proposed a multilevel database encryption system with subkeys, which can encrypt/decrypt the whole table, column or row. Also, this system can encrypt each row with different subkeys according to a security class of the data element. This system is based on the Chinese Remainder Theorem [7].

The DES algorithm is one of the famous encryption algorithms that uses a symmetric-key to change 64-bit of a plain text into 64-bit of a cipher text, using 56-bit of the key and 16 rounds. [8]. It is, now, considered as insecure for many applications; this is mainly due to the size of key, which is too small [9]. The work in [10] presents the AES algorithm as a replacement for the DES algorithm as a standard for data encryption. It is a symmetric-key algorithm that takes 128-bit for the plain text and 128, 192, or 256-bit for the key, the length of the key specifies the number of rounds in the algorithm.

Finally, Manivannan and Sujarani [1] proposed efficient database encryption techniques using the TSFS algorithm, which is a symmetric-key algorithm. Its main features include using transposition and substitution ciphers techniques that are important in modern symmetric algorithms as they have diffusion and confusion.

Also, it encrypts only the sensitive data, so, it limits the added time for encryption and decryption operations. The algorithm utilizes three keys and expands them into twelve sub-keys using the key expansion technique to provide effective security for the database. In order to improve the security, this algorithm uses twelve rounds and two different keys in each round.

However, TSFS algorithm applies only to alphanumeric characters; it does not accept special characters or symbols. More details about the TSFS algorithm is provided in [11], which builds a system that generates different numbers of secret keys based on the TSFS algorithm along with other algorithms to ensure a high security level of encrypted data.

III. Propsed Algorithm (ETSFS)

The main objective of this paper is to enhance the TSFS algorithm [1] and accordingly to provide a high security to the databases whilst limiting the added time cost for encryption and decryption by encrypting sensitive data only. The ETSFS algorithm can encrypt the data that consists of alphabetic characters from A to Z, all numbers and the following symbols: (*, -, ., /, :, @ and _). The ETSFS algorithm is a symmetric encryption algorithm, meaning each transformation or process must be invertible and have inverse operation that can cancel its effect. The key also must be used in inverse order.

ETSFS algorithm uses four techniques of transformations, which are transposition, substitution, folding and shifting. Fig. 1 presents the encryption algorithm, where the decryption algorithm reverses the encryption algorithm. The following sections describe the four techniques and contain the algorithms in pseudo-code format to be easy to understand:

A. Transposition

Transposition transformation changes the location of the data matrix elements by using diagonal transposition that reads the data matrix in the route of zigzag diagonal starting from the upper left corner after getting the data and pads it with *s if it is less than 16 digits [1]. Fig. 2 shows the transposition process when the entered data was: 6923@domain.Sa, where Fig. 3 shows the transposition algorithm in encryption side, then, Fig. 4 shows the transposition algorithm in decryption side.

```
Algorithm encryption (String data,
                      Array[12] keys )
Pre: data is plain text.
     keys is array that contains 12 4x4-key matrices.
Post: encryptedData is data after encrypting.

        Matrix[4,4] dataMatrix;
        String encryptedData;
        if (data length < 16)
                padd data by adding *'s;
        else if (data length > 16)
                cut the data after 16;
        end if
        dataMatrix = data;
        key  = expandKeys (keys);
        for (int i=0; i<12; i++)
                dataMatrix = transposition (dataMatrix);
                dataMatrix = substitution (dataMatrix, keys(i),
keys((i+1)mod 12));
                dataMatrix = folding (dataMatrix);
                dataMatrix = shifting (dataMatrix);
        end for
        encryptedData = dataMatrix;
        return encryptedData

End encryption
```

Fig. 1. Encryption algorithm.

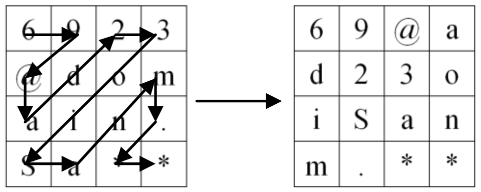

Fig. 2. Transposition example.

```
Algorithm transposition (Matrix data)
Pre: data is 4x4 matrix that contains the data should be encrypted.
Post: data is data after changing symbols location.

        Matrix temp;
        temp[0,0] = data[0,0];
        temp[0,1] = data[0,1];
        temp[0,2] = data[1,0];
        temp[0,3] = data[2,0];
        temp[1,0] = data[1,1];
        temp[1,1] = data[0,2];
        temp[1,2] = data[0,3];
        temp[1,3] = data[1,2];
        temp[2,0] = data[2,1];
        temp[2,1] = data[3,0];
        temp[2,2] = data[3,1];
        temp[2,3] = data[2,2];
        temp[3,0] = data[1,3];
        temp[3,1] = data[2,3];
        temp[3,2] = data[3,2];
        temp[3,3] = data[3,3];
        data = temp;
        return data;

End transposition
```

Fig. 3. Transposition algorithm.

```
Algorithm inverseTransposition (Matrix data)
Pre: data is 4x4 matrix, which contains the data should be decrypted.
Post: data is data after retrieving symbols location.

        Matrix temp;
        temp[0,0] = data[0,0];
        temp[0,1] = data[0,1];
        temp[0,2] = data[1,1];
        temp[0,3] = data[1,2];
        temp[1,0] = data[0,2];
        temp[1,1] = data[1,0];
        temp[1,2] = data[1,3];
        temp[1,3] = data[3,0];
        temp[2,0] = data[0,3];
        temp[2,1] = data[2,0];
        temp[2,2] = data[2,3];
        temp[2,3] = data[3,1];
        temp[3,0] = data[2,1];
        temp[3,1] = data[2,2];
        temp[3,2] = data[3,2];
        temp[3,3] = data[3,3];
        data = temp;
        return data;

End inverseTransposition
```

Fig. 4. Inverse transposition algorithm.

B. *Substitution*

The second algorithm is substitution transformation. It replaces one data matrix element with another by applying certain function [1]. If the element represents an alphabetic character, it then will be replaced with another character. If the element represents a number, it will be replaced with a number, and if it represents a symbol, it will be replaced with a symbol. The encryption function [1] E for any given letter x is

$$E(x) = (((k1+p) \bmod M + k2) \bmod M \qquad (1)$$

Where p is the plain matrix element, k1 and k2 are the keys elements that have the same position of p, and M represents the size of modulo operation. The ETSFS algorithm takes three values for the modulus size instead of one value as in the TSFS algorithm. The described substitution process in [1] has confusion. Confusion happens if the data is composed of alphabetic and numeric digits, and the modulus size (M) will be 26 for any digit, as illustrated in the next example. If one element in the data was 4, k1=5, k2=5, M = 26, then the result of substitution process is 14 as the paper presents. This result causes two problems. The first problem, is that the length of the data will be changed and increased; for example, when the plan text size is 16 digits, the cipher text size will be 17 digits if one element only changes, and that contradicts the TSFS algorithm's feature. The second problem, since the inverse operation decrypts the data digit by digit also, is that then it will deal with each element in the cipher text individually (1 then 4). As a result, the decrypted data will be different from the data that have been encrypted. Therefore, the ETSFS algorithm gives M the following values: 26 if p is aliphatic, 10 if p is numerical and 7 if p is symbolic. The decryption function [1] D is:

$$D(E(x)) = (((E(x) - k2) \bmod M) - k1) \bmod M \qquad (2)$$

Since most of the programming languages such as Java and C++ deal with the modulus as the remainder of an integer division, some of the results may have minus sign, and this will create a problem because there is no data that have minus sign representation. So, one more step has been added to the ETSFS algorithm implementation to check if the result includes the minus sign, and then apply:

$$D(E(x)) = M - |D(E(x))| \qquad (3)$$

The following Fig. 5 shows the result of substitution. From the same example in fig. 5, if we implemented the decryption operation (2) on the first element, the result would be -4, so the ETSFS algorithm applies function (3) to get the correct result, which is 6. Fig. 6 and 7 show the substitution encryption algorithm and its inverse respectively.

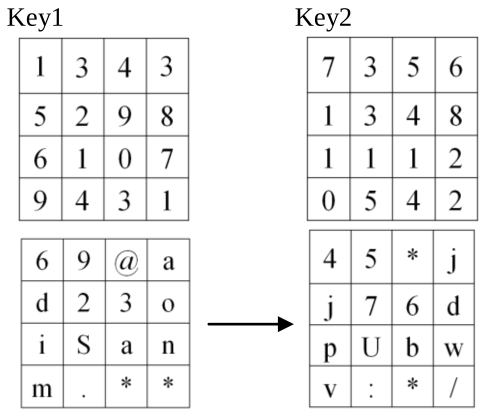

Fig. 5. Substitution example.

```
Algorithm substitution (Matrix data,
                        Matrix key1,
                        Matrix key2)
Pre: data is 4x4 matrix.
     key1 and key2 are 4x4 matrix used to encrypt data.
Post: data is data after applying substitution encryption method.

        Matrix temp;
        int M;
        for (int i=0; i<4; i++)
             for (int j=0; j<4; j++)
                  if (data[i,j] is alphabet)
                           M=26;
                  else if (data[i,j] is number)
                           M=10;
                  else if (data[i,j] is symbol)
                           M=7;
                  end if
                  temp[i,j]= (((k1[i,j]+ numric(data[i,j]) mod M)+k2[i,j]) mod M;
             end for
        end for
        data = temp;
        return data;

End substitution
```

Fig. 6.   Substitution algorithm.

```
Algorithm inverseSubstitution (Matrix data,
                               Matrix key1,
                               Matrix key2)
Pre: data is 4x4 matrix of data get from inverse Transposition technique.
      key1 and key2 4x4 matrix used to decrypt data.
Post: data is data after retrieving changes.

        Matrix temp;
        int M;
        for (int i=0; i<4; i++)
             for (int j=0; j<4; j++)
                  if (data[i,j] is alphabet)
                           M=26;
                  else if (data[i,j] is number)
                           M=10;
                  else if (data[i,j] is symbol)
                           M=7;
                  end if
                  num=(numric(data[i,j])-k2[i,j]-k1[i,j]) mod M
                  if (num<0)
                           num = M - |num|
                  end if
             end for
        end for
        data = temp;
        return data;

End inverseSubstitution
```

Fig. 7.   Inverse substitution algorithm.

C. *Folding*

The third algorithm is folding transformation. It shuffles one of the data matrix elements with another in the same entered data, like a paper fold. The data matrix is folded horizontally, vertically and diagonally [1]. The horizontal folding is done by exchanging the first row with the last row. The vertical one is done by exchanging the first column with the last column. The diagonal fold is done by exchanging the inner cells, the upper-left cell with the down-right cell and the upper-right cell with the down-left cell. Fig. 8 shows the example after folding, while Fig. 9 shows the folding encryption algorithm. Next, Fig. 10 shows the folding decryption algorithm.

$$\begin{array}{|c|c|c|c|}\hline 4 & 5 & * & j \\ \hline j & 7 & 6 & d \\ \hline p & U & b & w \\ \hline v & : & * & / \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline / & : & * & v \\ \hline d & b & U & j \\ \hline w & 6 & 7 & p \\ \hline j & 5 & * & 4 \\ \hline \end{array}$$

Fig. 8.   Folding example.

```
Algorithm folding (Matrix data)
Pre:  data is 4x4 matrix of data get from substitution technique.
Post: data is data matrix after applying folding technique.

              Matrix temp;
              temp[0,0] = data[3,3];
              temp[0,1] = data[3,1];
              temp[0,2] = data[3,2];
              temp[0,3] = data[3,0];
              temp[1,0] = data[1,3];
              temp[1,1] = data[2,2];
              temp[1,2] = data[2,1];
              temp[1,3] = data[1,0];
              temp[2,0] = data[2,3];
              temp[2,1] = data[1,2];
              temp[2,2] = data[1,1];
              temp[2,3] = data[2,0];
              temp[3,0] = data[0,3];
              temp[3,1] = data[0,1];
              temp[3,2] = data[0,2];
              temp[3,3] = data[0,0];
              data = temp;
              return data;

End folding
```

Fig. 9.   Folding algorithm.

```
Algorithm inverseFolding (Matrix data)
Pre:  data is 4x4 matrix of data get from inverse substitution technique.
Post: data is data matrix after applying inverse folding technique.

              Matrix temp;
              temp [0,0] = data[3,3];
              temp [0,1] = data[3,1];
              temp [0,2] = data[3,2];
              temp [0,3] = data[3,0];
              temp [1,0] = data[1,3];
              temp [1,1] = data[2,2];
              temp [1,2] = data[2,1];
              temp [1,3] = data[1,0];
              temp [2,0] = data[2,3];
              temp [2,1] = data[1,2];
              temp [2,2] = data[1,1];
              temp [2,3] = data[2,0];
              temp [3,0] = data[0,3];
              temp [3,1] = data[0,1];
              temp [3,2] = data[0,2];
              temp [3,3] = data[0,0];
              data = temp;
              return data;

End inverseFolding
```

Fig. 10. Inverse folding algorithm.

D. *Shifting*

The last part of the algorithm is the shifting transformation, which provides a simple way to encrypt using a 16-array element of numeric digits to exchange a letter with another. Each element of the array must contain the numeric representation of the data. Each digit must appear only once in each element of the array. The digits can appear in any order [1]. In shifting process, the algorithm replaces each element in the data matrix by its position within its array element. The ETSFS algorithm uses four 16-arrays instead of one array as the TSFS algorithm uses, because the described shifting process in [1] has confusion. For example, if an element in the plain text is 4 and its position within the array is 15, then the shifting process in [1] returns 15, which is causing the same two problems that were described in substitution transformation. So, the ETSFS algorithm separates each type from other. The ETSFS algorithm uses four 16-arrays, one for numeric, one for symbols, but because it is difficult to enumerate all symbols in this project; the suggested ETSFS algorithm considers only two types of symbols. Symbols that are used in emails (-, ., @, _) and symbols that are used in IP addresses (/, :). The last two 16-arrays are used for alphabetic, where one for capital letters and the other for small letters. We used that to enhance TSFS algorithm and make it is sensitive for the type of letter. The process illustrated in Fig. 11. Fig. 12 and 13 show the shifting encryption algorithm and its inverse respectively.

The previous encryption process is considered as the result of the first round of the ETSFS algorithm. The output of the first round goes as an input to the second round and the output the second round goes as an input to the third round. This process continues up to the 12th round and the output of this round is the cipher text of the given plain text and that cipher text is stored in the database. For keys, in each round, it selects two keys for encryption. In encryption, each round (i) selects the key (i) and the key (i+1), at round 12 it selects key (12) and key (1). In decryption, the keys are selected in reverse order. The Fig. 14 shows the steps of expand keys as [1] suggested.

IV. Implementation

A Java-based project has been built to test the ETSFS algorithm correctness and performance. The implementation uses three-tier architecture, as represented in Fig. 15. The three-tier separate the functions into interface, processing and data management functions. The multi-tier architecture allows developers to create flexible and reusable applications. In addition, this architecture provides "encryption as a service" to facilitate the interaction between the interface and the encryption/decryption model, and makes the process of encryption or decryption transparent to application [2]. In this paper, the interface-tire is used to enter and retrieve data from the database. The processing-tier is used to garner the data or query from the interface-tier and then to complete the encryption or decryption processes to apply the query over the secure database. It stores the keys in a separate file instead of storing them in the database to increase the security. Finally, a data management-tire stores the data.

Depending on the suggested architecture, the implementation structure was developed as shown in Fig. 16. This Figure illustrates all classes and their connections with each other. It shows the attributes of each class and functions headers. The classes include:

A. Maic Class

In general, at the beginning, the user can enter the information that will be encrypted. In this implementation, the main class reads data from a file to obtain equivalent results when measuring the performance. The interface part is responsible for taking the data from file and sending it to the translator part to save it in the database. Another function for the interface is to retrieve the data form the database by using the translator part.

I/P	**Array Element**	**O/P**
/	0 1 2 3 4 5 6	/
:	1 2 3 4 5 6 0	/
*	2 3 4 5 6 0 1	@
v	3 4 5 6 7 8 9 10 11 12 13 14 15 …23 24 25 0 1 2	s
d	4 5 6 7 8 9 10 11 12 13 14 15 16 … 24 25 0 1 2 3	z
b	5 6 7 8 9 10 11 12 13 14 15 16 … 24 25 0 1 2 3 4	w
U	6 7 8 9 10 11 12 13 14 15 16 … 24 25 0 1 2 3 4 5	O
.	.	.
.	.	.
4	1 5 4 6 0 7 2 8 3 9	2

Fig. 11. Shifting example.

```
Algorithm shifting (Matrix data,
                    MAtrix arrayNumber,
                    Matrix arrayAlpha,
                    Matrix arraySymbol )
Pre: data is 4x4 matrix of data gets from folding technique.
    arrayNumber is 16x10 dimension array used for numeric data.
    arrayAlpha is 16x26 dimension array used for alphabetic data.
    arraySymbol is 16x7 dimension array used for symbol data.
Post: data is data matrix after applying shifting technique.

  Matrix temp;
  char charOfData;
  loop from i=0 to i=3 do
       loop from j=0 to j=3 do
          charOfData = data[i,j];
          if (charOfData is number )
               loop from k=0 to k=9 do
                  if (arrayNumber[(3xi)+i+j][k]== charOfData )
                           temp[i,j] = k;
                           break;
                  end if
              end loop

          elseIf (charOfData is alphabet)
              loop from k=0 to k=25 do
                  if (arrayAlpha[(3xi)+i+j][k]== charOfData )
                           temp[i,j] = Alpha that have order (k);
                  end if
              end loop
          else
              loop from k=0 to k=6 do
                  if (arraySymbol[(3xi)+i+j][k]== charOfData )
                           temp[i,j] = Symbol that have order (k);
                  end if
              end loop
          end if
      end loop
  end loop
  data = temp;
  return data;

End shifting
```

Fig. 12. Shifting algorithm.

```
Algorithm inverseShifting (Matrix data,
                           Matrix arrayNumber,
                           Matrix arrayAlpha,
                           Matrix arraySymbol )
Pre: data is 4x4 matrix of data get from inverse folding technique.
    arrayNumber is 16x10 dimension array used for numeric data.
    arrayAlpha is 16x26 dimension array used for alphabetic data.
    arraySymbol is 16x7 dimension array used for symbol data.
Post: data is data matrix after applying inverse shifting technique.

  Matrix temp;
  char charOfData;
  int index;
  loop from i=0 to i=4 do
       loop from j=0 to j=4 do
          charOfData = data[i,j];
          if (charOfData is number)
               index = the order of charOfData number in number order;
               temp [i,j] = arrayNumber[(3xi)+i+j][index];
          elseIf (charOfData is alphabet)
               index = the order of charOfData alphabet in alphabetical
order;
               temp [i,j] = arrayAlpha[(3xi)+i+j][index];
          else
               index = the order of charOfData symbol in symbol order;
               temp [i,j] = arraySymbol[(3xi)+i+j][index];
          end if
      end loop
  end loop
  data = temp;
  return data;

End inverseShifting
```

Fig. 13. Inverse shifting algorithm.

```
Algorithm expandKeys (String [3] initialKeys)
Pre:  initialKeys contains of Key1, Key2 and Key3 as initial keys.
Post: expandedKeys is array of size 12, contains 12 expanded keys Matrcis.

  Matrix[12] expandedKeys;
  Matrix[4,4] tempKey;
  for (int i=0; i<3; i++)
      if (initialKeys[i] length < 16)
          padd data by adding 0's;
      else if (initialKeys[i] length > 16)
          cut the data after 16;
      end if
      change initialKeys to number based on the position in alphabets a-z;
      tempKey = initialKeys[i];
      for (int j=0; j<4; i++)
          for (int h=0; h<4; i++)
               expandedKeys [j+ix4].row(h) = tempKey.row(h) after
shifting by (j+h)%4 times;
          end for
      end for
  end for
  return expandedKeys;

End expandKeys
```

Fig. 14. Expand keys algorithm.

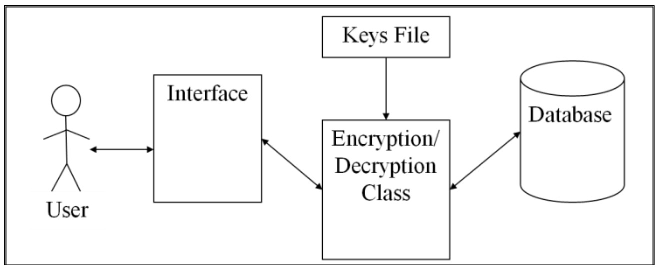

Fig. 15. Implementation architecture.

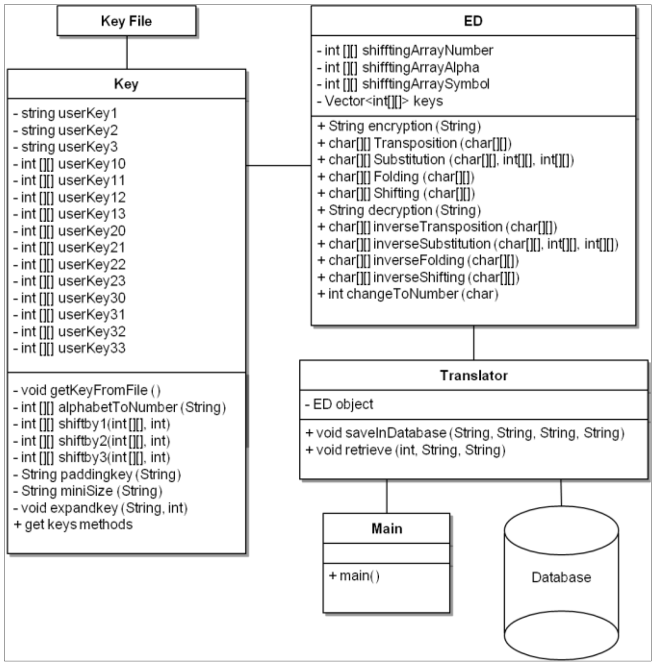
Key File
Key
- string userKey1
- string userKey2
- string userKey3
- int [][] userKey10
- int [][] userKey11
- int [][] userKey12
- int [][] userKey13
- int [][] userKey20
- int [][] userKey21
- int [][] userKey22
- int [][] userKey23
- int [][] userKey30
- int [][] userKey31
- int [][] userKey32
- int [][] userKey33
- void getKeyFromFile ()
- int [][] alphabetToNumber (String)
- int [][] shiftby1(int [][], int)
- int [][] shiftby2(int [][], int)
- int [][] shiftby3(int [][], int)
- String paddingkey (String)
- String miniSize (String)
- void expandkey (String, int)
+ get keys methods
ED
- int [][] shifftingArrayNumber
- int [][] shifftingArrayAlpha
- int [][] shifftingArraySymbol
- Vector<int[][]> keys
+ String encryption (String)
+ char[][] Transposion (char[][])
+ char[][] Substitution (char[][], int[][], int[][])
+ char[][] Folding (char[][])
+ char[][] Shifting (char[][])
+ String decryption (String)
+ char[][] inverseTransposition (char[][])
+ char[][] inverseSubstitution (char[][], int[][], int[][])
+ char[][] inverseFolding (char[][])
+ char[][] inverseShifting (char[][])
+ int changeToNumber (char)
Translator
- ED object
+ void saveInDatabase (String, String, String, String)
+ void retrieve (int, String, String)
Main
+ main()
Database

Fig. 16. Implementation UML diagram.

B. Translator Class

The middle part contains three parts. One part is used for keys; the other one, which is the translator, is used to deal with the database, and the last part is used for encryption/decryption operations. During the experiment, the translator class receives the data from the interface part by the SaveInDatabase() method, and draws a connection with the database so as to apply insert query in the database after encrypting the received data. Here the function depended on the application and the table used in the database. We used one table named "person". It contains four columns: name, phone, mail and job. The first three columns contained sensitive data. So, SaveInDatabase() method receives four data and encrypts the first three data by using the encryption/decryption (ED) class, then applies the insert query to add this data into the database.

Also, the translator has another method used to apply the select query from the database. In this project, we focused only on the encryption/decryption algorithms rather than how the query should be translated or mapped to a query that can be applied on the encrypted database. So, we just used two direct specific select queries for selecting the data, one for retrieving the complete table and the other for selecting a query depending on a condition which is the information of a person how has specific name. After retrieving the data from the database, the translator class decrypts it by using the ED class, and then printed it in the screen. In the second query, before the method does select operation from the database, this method must encrypt the data that is used in the condition by using the ED class to do the comparison operations over encrypted data.

C. Encryption/Decryption Class

This class is the most important class for this work. After the ED class receives the data from the translator class, it checks on some of points. It checks the length. The length must be equal to 16; if it is less than 16, the ED class pads the data with (*) symbol up to 16 digits. In this project, we choose the (*) symbol for padding because it is not used in the data. If it is more than 16, the ED class deducts of the excess. After that, the ED class changes the data form to 4x4 array form to apply the algorithm on an array form. Then the ED class applies the same following scenario twelve times. First, it applies transposition transformation and changes the location of the elements in 4x4 data array by using diagonal transposition, and the result array of this step considers as input to the second step. In the second step, the ED class applies substitution transformation and replaces one data with another by using two keys in each round. Each element in the data array is one of three types: number, alphabet or symbol, and each of them has different modulo. If the element is alphabet or symbol, first it changes to a number based on the position in the alphabets a-z (a-0, b-1, ... z-25), then the ED class applies the equation of encryption on this number. We selected the symbols' positions as following (*, -, ., /, :, @, _) depended on precedence in ASCII codes. The result array of this step considers as input for the third step. In the third step, the ED class applies folding transformation and shuffles each element in 4x4 data array with another one in the same array, and the result array of this step considers as input for the fourth step. Fourth step, the ED class applies shifting transformation and replaces each digit of data array by its position within its array element. Each element in the data array is one of three types: number, alphabet or symbol, and each of them have different arrays set. There are four defined 16-arrays, one for numeric, two for alphabetic where one for capital letter and other one for small letter and last array for symbols. If the element in the data array is alphabet or symbol, first the ED class changes the element into a number based on the position in the alphabets a-z (a-0, b-1, ... z-25), then the ED class replaces the number with its position within its array set. The result array of this step considers as input for the first step again in the next round. After the twelve rounds finish and the data encrypted, the ED class sends the result to the translator part to store it in the database.

Also, the ED class has other function is responsible of decryption operations. This function receives the encrypted data from the translator part. The ED class first prepares the data into 4x4 array form to apply the algorithm on array form. After that, it sends the data array to the decryption methods in reverse sequence, started with inverse shifting, inverse folding, inverse substitution then inverse transposition. All of these methods follow the same scenario that applies in encrypted part but with inverse operations. This scenario also applies twelve times, the output of the each round considered as input to the second round. After the twelve rounds finish and the data decrypted, the ED class sends the result to the translator class.

D. Key Class

Key class reads three initial keys from the file. Then it checks on some points. Checking on the length, the length must be equal to 16. If it is less than 16, key class pads the key with 0s. If it is more than 16, key class deducts off excess. Then key class converts all digit of the keys to numbers based on position in the alphabets a-z (a-0, b-1, ... z-25). After that, it changes the keys form to 4x4 array form to apply the operations on array form.

After that, the keys are expanded based on shifting the rows into twelve keys, where each key is expanded into four keys. Each output key is stored in 4x4 array form, and the expanding technique implements as the following: the first output key has four rows where row 0 is not shifted, row 1 is shifted by one time, row 2 is shifted two times and row 3 is shifted three times. In the second output key, row 0 is shifted one time, row 1 is shifted 2 times, row 2 is shifted three times, and row 3 is not shifted. In third output key, are row 0 is shifted two times, row 1 is shifted three times, row 2 is not shifted and row 3 is shifted one time. In the last output key, row 0 is shifted three times, row 1 is not shifted, row 2 is shifted one time and row 3 is shifted two times.

V. Comparative Study

This section presents a comparative study between the DES algorithm, the AES algorithm and the ETSFS algorithms. It explains the experiment in order to evaluate their performance to establish the best algorithm amongst all possible algorithms. It then reports the results and discusses them.

The DES algorithm uses too short key length. Within the rapid advances, DES is breakable, for example in 1998 the Electronic Frontier Foundation built a DES Cracker for less than $250,000 that can decode DES messages in less than a week [12]. The AES algorithm is used as a standard for data encryption. It is stronger and faster than the DES algorithm [13]. DES and AES algorithms have open source code, and they are supported by Java libraries.

A. Experiment Setup

The experiment compared the ETSFS algorithm with the DES and AES algorithms, which have open source code. To test the algorithms, the following materials were used:

- Programming language: Java.
- Application platform: NetBeans IDE 6.9.1.
- Development: Java Development Kit (JDK) 1.6.
- Database management system: MySQL Server 5.6.
- Java external Library: Connector-java 5.1.23-bin.jar to connect the java with MySQL server.
- Visual database design tool: MySQL Workbench 5.2 CE used for database design, modeling and SQL development.
- Operating system: Windows Vista Home Premium, 32-bit.
- Hardware computer: Dell XPS M1330 laptop, Intel(R) Core(TM)2 Duo CPU T7300 2.00GHz and 3.00 GB for RAM.

In the experiment, to obtain fixed and fair comparison between the algorithms, same entered data and same functions that are responsible for accessing the database were used in all algorithms. Furthermore, each algorithm was tested with the following data size: 100, 500, 1000, 1500 and 2000 rows. For each size, the experiment was repeated three times and then the average value for each timer was calculated to eliminate the effect of the computer processing issues and insure near fair real value.

B. Evaluation Metrics

Execution time (Second): The evaluation performance of encryption/decryption processes conducted in terms of the execution time of insert and select SQL queries. In this project, we followed the same approach presented in [14] to calculate the execution time. We used a timer to calculate the execution time of the query from the beginning of its work until it finishes successfully. Three timers were used in each algorithm to calculate the execution times for three types of quires:

- **Insert:** We calculated the insertion execution time to know how much time does the insertion operation consumes with encryption processes. Insert query example: *INSERT INTO person VALUES (encrypted name, encrypted phone, encrypted mail, job).*
- **Select all:** To know how much time does the query take to select all the rows and decrypt the encrypted fields with decryption processes. Select all query example: *SELECT name, phone, mail, job FROM person*.
- **Select with condition:** To know how much time the query takes to encrypt the data in the condition, then compare this data with the encrypted data inside the database to retrieve the required data, and decrypt the encrypted selected fields with decryption processes. Select with condition query example: *SELECT name, phone, mail, job FROM person WHERE name = encrypted name*.

1) Database size (Kilobyte): One of the most important factors to determine about the database is its overall size [15]. So, we used the database size criterion to compare the database sizes between the three algorithms after storing the encrypted data. We used it to know the impact of the encryption process on the database overall size.

C. Results and Discussion

In this section, we stated the results of our experiments in terms of execution time and database size aspects. For each criterion, we presented the average values for each one of the three different algorithms with different sizes of data. Also, we interpreted the findings and results, and showed the relationships among the algorithms.

1) Execution time:

- **Insert queries execution time:** First, Fig. 17 shows the relationship between encryption time during insertion the data in the database and the number of tuples for each algorithm. In general, when the size of data increases, the time also increases. From the results, it is obvious that the AES algorithm consumes the longest time for encrypting and the ETSFS algorithm consumes the least time for that. The speed of the AES encryption depends on the number of rounds and the key generation in each time it does the encrypt function. Also, each round in the AES algorithm needs round key from the key expansion algorithm. So the AES algorithm depends on the speed of the round key generation [16]. In contrast, the ETSFS algorithm needs the key generation only one time before doing the encrypt function. That is mean the ETSFS algorithm has the best execution performance when applying insert queries that include encryption processes.

- **Select all query execution time and select with condition quires execution time:** Fig. 18 shows the relationship between decryption time during searching in database with using a query to retrieve the complete table and the number of tuples, and Fig. 19 shows the relationship between decryption time but with using the queries that depend on a condition and the number of tuples for each algorithm. The DES algorithm consumes least time when applying the select queries. The instability of the relationship may occur owing to the time taken in connecting and disconnecting the database or file operations. Moreover, hardware issues may cause inconsistent relations. The AES algorithm consumes more time than the ETSFS algorithm for selecting queries because in the AES algorithm, the decryption module depends mainly on the key expansion module [16]. Also, the execution of the query that retrieves the complete table consumes little more time than those that depend on the condition because the former retrieves more tuples than the latter; this tuples need more decryption operations.

2) Database Size: In term of storage, Fig. 20 shows the relationships between the number of tuples and the size of the database in kilobyte after storing the encrypted data with the use of the three algorithms with different data sizes. Where the data size increases, the difference is evident between the algorithms. The results show that the ETSFS algorithm has consumed the smallest space among other algorithms. This happened because the size of the encrypted data in ETSFS algorithm does not increase more so than the original size; rather, it keeps the size as it is. However, AES and DES algorithms extend the data size to multiple of 128-bit and 64-bit respectively [12]. The AES algorithm needs largest storage space to store the data, followed by DES algorithm, which needs larger storage space than the ETSFS algorithm.

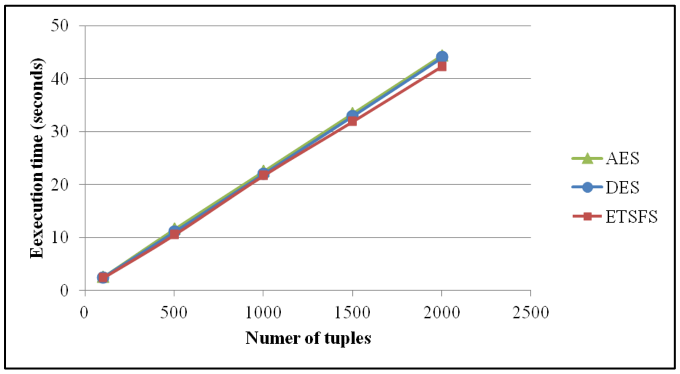

Fig. 17. The relationship between the insert operations execution time and the number of inserted rows for the three algorithms.

The experiment proves that the ETSFS algorithm can secure the data successfully when the data sets of the ETSFS algorithm are increased, and also that the substitution and shifting techniques are corrected. The insert queries execution time is not affected because the algorithm encrypts the sensitive data only, and the database size is not affected because the algorithm does not increase the size of encrypted data. In general, the improved performance comes without compromising query processing time or database size.

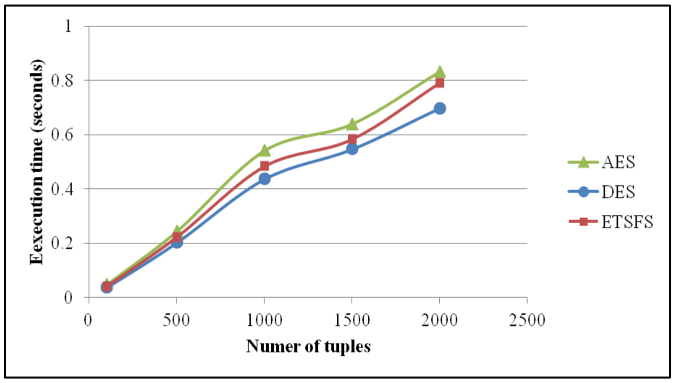

Fig. 18. The relationship between the execution time of select all rows operation and the number of rows for the three algorithms.

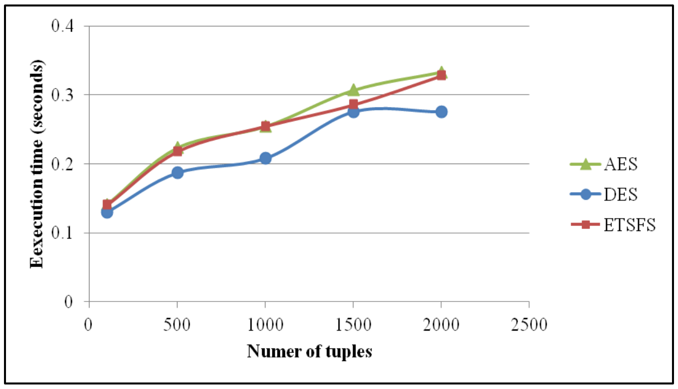

Fig. 19. The relationship between the execution time of select operations that depend on condition and the number of rows in the database for the three algorithms.

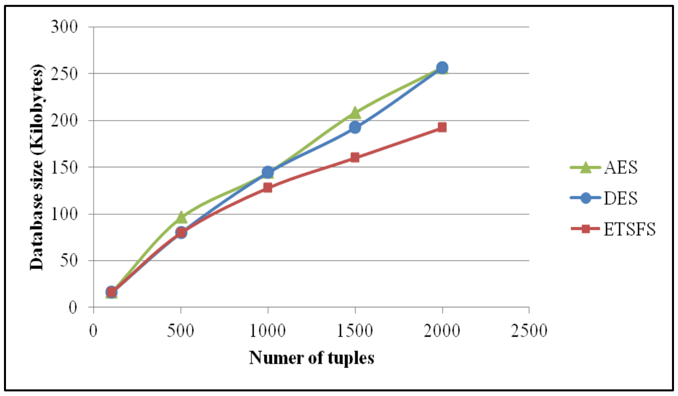

Fig. 20. The relationship between the database size after applying encryption and the number of rows for the three algorithms.

VI. Conclusion And Future Work

Data-storing and exchanging between computers is growing fast across the world. The security of this data has become an important issue for the world. The best solution centred on securing the data is using cryptography, along with other methods. This paper proposes the enhancement of the TSFS algorithm to support the encryption of special characters, correct substitution process by providing more than one modulo factor to differentiate between data types and prevent increasing the data size, as well as correcting the shifting process for the same reasons by providing four 16-arrays. The experimental results have shown that the ETSFS algorithm successfully encrypted important symbols, as well as alphanumeric data. The improved performance comes without compromising query processing time or database size. Using well-established encryption algorithms as benchmarks, such as DES and AES, the proposed ETSFS algorithm was shown to have consumed the smallest space and encryption time compared to the other algorithms.

Due to time constraints, it was difficult to cover all special symbols in this paper; however, the ETSFS algorithm can be extended to include other symbols with slight modification to the encryption/decryption processes. For future work, it is intended that this algorithm be improved so as to accommodate any size of data, rather than only 16 digits. Furthermore, it is intended to further evaluate the security of ETSFS algorithm by establishing the number of operations and the time attackers need to recover the keys and accordingly hack the encrypted data.

References

[1] D. Manivannan, R .Sujarani, Light weight and secure database encryption using TSFS algorithm, Proceedings of the International Conference on Computing Communication and Networking Technologies, 2010, pp. 1-7.

[2] L. Liu, J. Gai, A new lightweight database encryption scheme transparent to applications, Proceedings of the 6th IEEE International Conference on Industrial Informatics, 2008, pp.135-140.

[3] L. Bouganim, P. Puncheral, Chip-Secured data access: Confidential data on untrusted servers, Proceedings of the 28th International Conference on Very Large Databases, 2002, pp. 131-142.

[4] Z. Yang, S. Sesay, J. Chen, D. Xu, A Secure Database Encryption Scheme, Proceedings of Consumer Communications and Networking Conference, 2005, pp. 49-53.

[5] K. Kaur, K. Dhindsa, G. Singh, Numeric to numeric encryption: using 3KDEC algorithm, Proceedings of IEEE International Conference on Advance Computing, 2009, pp. 1501-1505.

[6] A. Rakesh, K. Jerry, S. Ramakrishnan, Order preserving encryption for numeric data, Proceedings of ACM SIGMOD International Conference on Management of Data, 2004, pp.563-574.

[7] M. Hwang, W. Yangb, Multilevel secure database encryption with subkeys, Data and Knowledge Engineering 22 (1997) 117-131.

[8] Z.Yong-Xia, The Technology of Database Encryption, Proceedings of the 2nd International Conference on MultiMedia and Information Technology, 2010, pp. 268-270.

[9] S. Bhatnagar, Securing Data-At-Rest, Literature by Tata Consultancy Services.

[10] I. Widiasari, Combining advanced encryption standard (AES) and one time pad (OPT) encryption for data security, The International Journal of Computer Applications 57 (2012) 1-8.

[11] M. Raj, M. Kumar, D. Manivannan, A. Umamakeswari, Design and development of secret session key generation using embedded crypto device-ARM-LPC 2148, Journal of Artificial Intelligence 6 (2013) 134-144.

[12] H. Alanazi, B. Zaidan, A. Zaidan, H. Jalab, M. Shabbir, Y. Al-Nabhani. New comparative study between DES, 3DES and AES within nine factors, Journal of Computing 2 (2010) 152-157.

[13] A. Mousa, E. Nigm, S. El-Rabaie, O. Faragallah, Query processing performance on encrypted databases by using the REA algorithm, The International Journal of Network Security 14 (2012) 280-288.

[14] A. Sterbenz, P. Lipp, Performance of the AES candidate algorithms in java, Proceedings of the AES Candidate Conference, 2000, pp.161-165.

[15] Oracle, Siebel Performance Tuning Guide, Available from: http://docs.oracle.com/cd/E14004_01/books/PDF/PerformTun.pdf 2012.

[16] T. Subashri, R. Arunachalam, B. Kumar, V. Vaidehi, Pipelining architecture of AES encryption and key generation with search based memory, The International journal of VLSI design & communication systems (VLSICS) 1 (2010) 1-13.