Enhanced TSFS Algorithm for Secure Database Encryption

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—**Security of databases has become increasingly crucial in all application areas. Database encryption is an important mechanism to secure databases from attacks and unauthorised access. The Transposition-Substitution-Folding-Shifting encryption algorithm (TSFS) is a symmetric database encryption algorithm that uses three keys with an expansion technique to provide high security: it improves the efficiency of query execution time by encrypting the sensitive data only. However, it applies merely for the alphanumeric characters. This paper extends the data set of the TSFS encryption algorithm to special characters as well, 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 in decryption steps. Experiment results show that enhanced TSFS encryption algorithm outperforms Data Encryption Standard algorithm (DES) and Advanced Encryption Standard algorithm (AES) in terms of query execution time and database added size.**

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

I. INTRODUCTION

The tremendous development of technology and data storage leads organisations to depend on database systems. The organisations store huge amounts of data in the databases in safe situations in order to retrieve them in a fast and secure way. The damage on the data can happen if it suffers from attacks and unauthorised access. In the presence of security threats, the database security is becoming one of the most urgent challenges; thus, in order to achieve a high level of security, the complexity of encryption algorithm should be increased, and at the same time taking care in regard to database efficiency, ensuring performance is not affected.

There are many research studies in database security field; some of these research studies have efficient implementation. Also, many encryption algorithms have been proposed. Some of them have features but need extending to further development; one of these is the Transposition, Substitution, Folding and Shifting TSFS algorithm, known as the TSFS algorithm [1]. The TSFS algorithm provides a high degree of security by using a number of features; however, it supports only numbers and alphabetic types that are not enough to protection different types of sensitive data.

This paper provides a secure and efficient encryption method that encrypts only sensitive data without using special hardware. It enhances TSFS algorithm by extending the data set of the TSFS encryption algorithm to special characters as well, and corrects the 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 organised as follows: Section II reviews existing work on database encryption techniques; Section III introduces the ETSFS algorithm and explains its procedure, whilst Section IV introduces the implementation of ETSFS algorithm and suggested structure; Section V presents a comparative study between the algorithms, evaluates performance, and reports and discusses results. Finally, Section VI 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 & Pucheral proposed a smart card solution to protect data privacy, where database owners can access the data using a client terminal supporting by smart card devices [2]. This proposed solution is considered a secure and effective solution, but is complicated and expensive [1]. Database encryption greatly impacts database performance because, each time a query runs, a large amount of data must be decrypted. Therefore, [3] suggested that encrypting sensitive data can provide only the security needed without affecting the performance. In [1], [4] and [5] encryption algorithms were proposed depending on encryption of sensitive data only. Kaur *et al.* proposed a technique to encrypt numeric data using only a fixed data field type and length [4]. However, this algorithm does not support the encryption of character data. Agrawal *et al.* [5] 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 the indexes of database over encrypted tables, but is only applied to numeric data. Additionally, it has not investigated key management.

The DES algorithm is one of the famous encryption algorithms that use 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. [6]. It is now considered insecure for many applications; this is mainly due to the size of key, which is too small [7]. The work in [8] 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 & Sujarani [1] proposed efficient database encryption techniques using 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. Moreover, it encrypts only the sensitive data, thus limiting the added time for encryption and decryption operations. The algorithm utilises 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, the TSFS algorithm applies merely to alphanumeric characters; it does not accept special characters or symbols. More details about TSFS algorithm are provided in [9], which builds a system that generates different numbers of secret keys based on TSFS algorithm along with other algorithms to ensure high security level of encrypted data.

III. Extended TSFS 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 _). 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:

A. Transposition

Transposition transformation changes the location of the data matrix elements by using diagonal transposition which 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 data entered was: 6923@domain.Sa. Fig. 3 shows the transposition algorithm at the encryption 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.

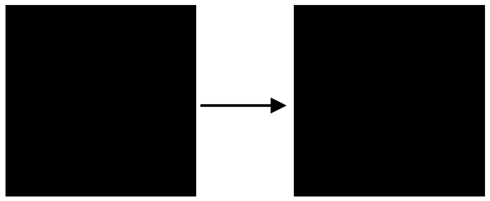

Fig. 2. Transposition example.

```
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.

B. Substitution

The second technique is the substitution transformation. It replaces one data matrix element with another by applying a certain function [1]. If the element represents 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 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 substitution process described in [1] has confusion. Confusion occurs 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, the result of the substitution process is then 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, such as when the plan text size is 16 digits, for example, the cipher text size will be 17 digits if one element only changes, which contradicts the TSFS algorithm's feature. The second problem is, since the inverse operation decrypts the data digit by digit, it will then 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 alphabet, 10 if p is number and 7 if p is symbol.

The decryption function D is as follows:

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

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 as there is no data with a minus sign representation. Accordingly, one more step has been added to the ETSFS algorithm implementation to determine whether the result includes the minus sign. Subsequently, the following is applied:

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

Fig. 4 shows the result of applying substitution on the output of the Transposition example and Fig. 5 shows the substitution algorithm at encryption side.

C. Folding

The Folding transformation shuffles one of the data matrix elements with another in the same entered data, like the 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 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. 6 shows an example of folding whilst Fig. 7 shows the folding algorithm at encryption side.

D. Shifting

The last part of the algorithm is the shifting transformation, which provides a simple way of encrypting using 16-array element of numeric digits to change 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 the 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 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 array is 15, the shifting process in [1] then returns 15, causing the same two problems described in substitution transformation. Accordingly, the ETSFS algorithm separates each type from the 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 symbol: the symbols used in emails (-, ., @, _) and symbols used in IP address (/, :). The last two 16-arrays are used in an alphabetic context, with one for capital letters and the other for small letters in order to enhance TSFS algorithm and make it sensitive to the type of letter. The process is illustrated in Fig. 8, whilst Fig. 9 shows the shifting algorithm at the encryption side.

The previous encryption process is considered as the result of the first round of ETSFS algorithm. The output of the first round goes as an input to the second round and the output of the second round goes as an input to the third round. This process continues up to the 12th round, whilst 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, two keys are selected 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.

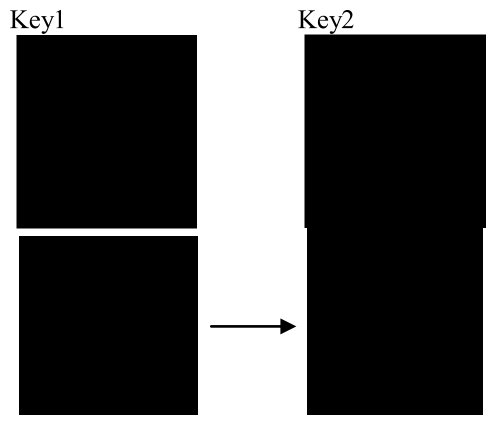

Fig. 4. Substitution example.

```
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. 5. Substitution algorithm.

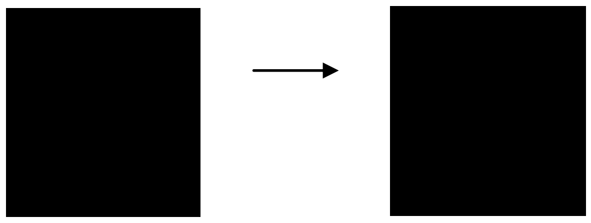

Fig. 6. Folding example.

```
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. 7. Folding algorithm.

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. 8. 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. 9. Shifting algorithm.

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. 10. The three tiers separate the functions into interface, processing and data management functions. The multi-tier architecture allows developers to create flexible and reusable applications. 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 in order to increase the security. Finally, data management-tire stores the data.

Depending on the architecture suggested, the implementation structure was developed, as shown in Fig. 11. This figure illustrates all classes and their connections with each other. It shows the attributes of each class and functions headers. The classes include:

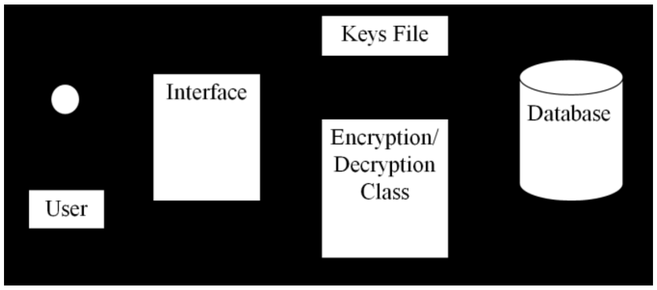

Fig. 10. Implementation architecture.

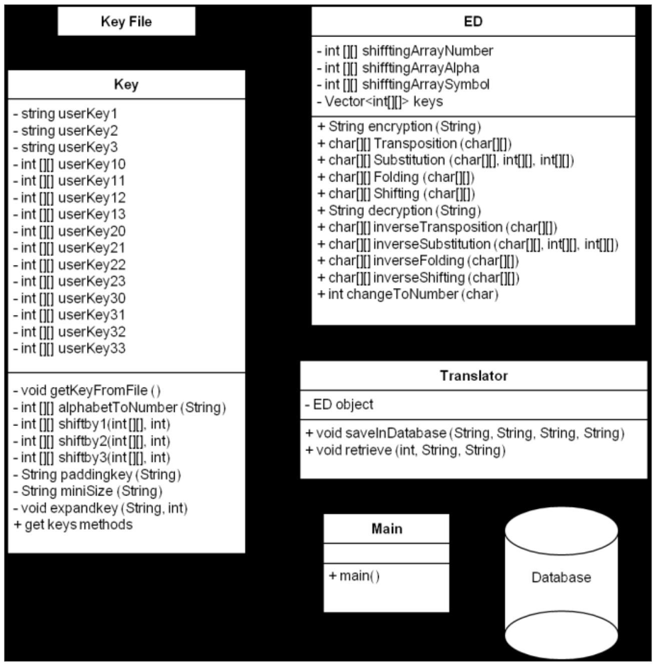
Key File
ED
- int [][] shifftingArrayNumber
- int [][] shifftingArrayAlpha
- int [][] shifftingArraySymbol
- Vector<int[][]> keys
+ String encryption (String)
+ char[][] Transposition (char[][])
+ char[][] Substitution (char[][], int[][], int[][])
+ char[][] Folding (char[][])
+ char[][] Shifting (char[][])
+ String decryption (String)
+ char[][] inverseTransposition (char[][])
+ char[][] inverseSubstitution (char[][], int[][], int[][])
+ char[][] inverseFolding (char[][])
+ char[][] inverseShifting (char[][])
+ int changeToNumber (char)
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
Translator
- ED object
+ void saveInDatabase (String, String, String, String)
+ void retrieve (int, String, String)
Main
+ main()
Database

Fig. 11. Implementation UML diagram.

1) Main Class: In general, at the beginning, the user can enter the information that will be encrypted. In this implementation, the main class reads the 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, which is then saved in the database. Another function for the interface is to retrieve the data form the database by using the translator part.

2) Translator Class: This paper has 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. Thus, the translator part 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. The other method uses 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.

Encryption/Decryption (E/D) Class: This class is the most important class for this work. It uses several methods to prepare the size and format of the data that will be encrypted/ decrypted, and then applies the process of TSFS algorithm to encrypt/decrypt the data using keys stored in the key class, then returns the results to the translator class to apply the query.

3) Key Class: Key class reads the initial three keys from a file and accordingly expands the keys to generate twelve subkeys by shifting the rows as described in [1].

V. COMPARATIVE STUDY

This section presents a comparative study between the DES algorithm, AES algorithm and 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.

A. Experiment Setup

The experiment compared ETSFS algorithm with DES and AES algorithms, which have open source code. So as to implement and 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, so as to obtain a fixed and fair comparison between the algorithms, same entered data and same functions responsible for accessing the database were used in all algorithms. Moreover, each algorithm was tested with the following data sizes: 100, 500, 1,000, 1,500 and 2,000 rows. For each size, the experiment was repeated three times, with the average value for each timer then calculated so as to eliminate the effect of the computer processing issues and insure near fair real value.

B. Evaluation Metrics

Several experiments have been conducted on the algorithms. Two evaluation metrics are considered: execution time (Second) that encryption/decryption processes is taken and the size (Kilobyte) of database after storing the encrypted data to know the impact of encryption process on the database overall size. For execution speed, three timers were used in each algorithm for three types of query:

- Insert: we calculated the insertion execution time to determine how much time the insertion operation consumes throughout the encryption processes. Insert query example: *INSERT INTO person VALUES (encrypted name, encrypted phone, encrypted mail, and job).*
- Select All: to determine how much time the query takes to select all the rows and decrypt the encrypted fields through the decryption processes. A Select All query example: *SELECT name, phone, mail, job FROM person*.
- Select with Condition: to establish how much time the query takes to encrypt the data in the condition, then compares this data with the encrypted data inside the database to retrieve the required data, and accordingly decrypt the encrypted selected fields with decryption processes. Select with condition query example: *SELECT name, phone, mail, job FROM person WHERE name = encrypted name*.

C. Results and Discussion

Execution time: The experiment's results of execution time are represented in the first three figures. Fig. 12 shows the relationship between the execution time of the insertion operation with encryption processes, and the number of tuples for each algorithm. Fig. 13 shows the relationship between decryption time during searching in database using the Select All query to retrieve the complete table, and the number of tuples. Fig. 14 also shows the relationship between decryption time but using Select with Condition queries, and the number of tuples for each algorithm. From the results, it is obvious that AES algorithm consumes the longest time for encryption, and ETSFS algorithm consumes the least time for that. That is mean, ETSFS algorithm has the best execution performance compared to DES and AES algorithms when applying insert queries with encryption. On the contrary, DES algorithm consumes the least time when select queries have been applied. 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. Generally, the encryption process observed consumes more execution time than the decryption process. 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.

Database size: Fig. 15 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. The results show that the ETSFS algorithm has consumed the smallest space amongst other algorithms; this occurred because the size of the encrypted data in ETSFS algorithm does not increase more so than the original one; rather, it keeps its size as it is.

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 without affecting the performance.

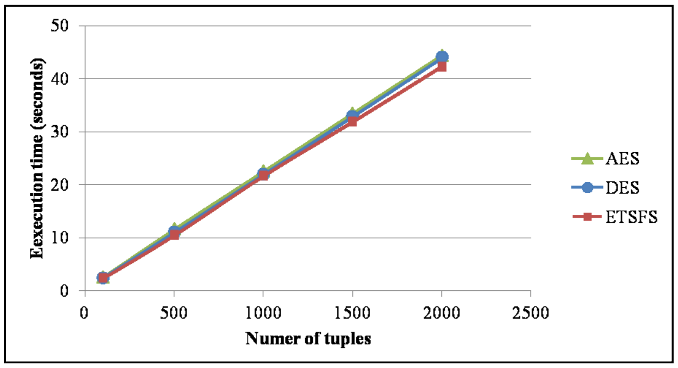

Fig. 12. The relationship between the insert operations execution time and the number of inserted rows for the three algorithms.

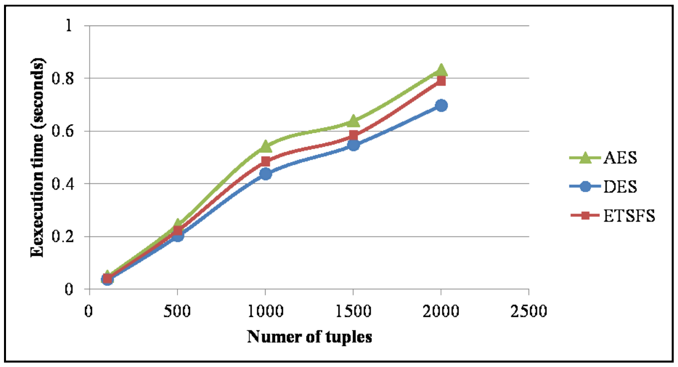

Fig. 13. The relationship between the execution time of select all rows operation and the number of rows for the three algorithms.

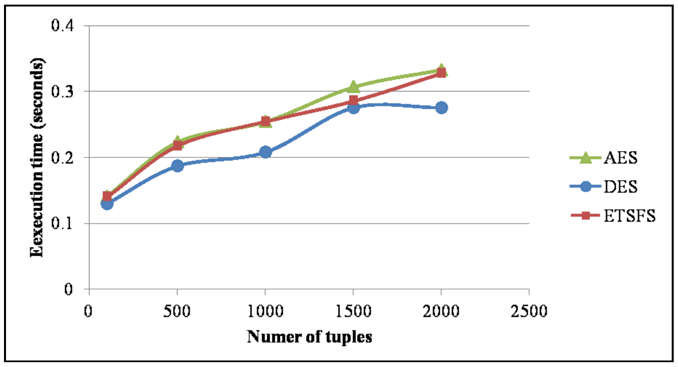

Fig. 14. The relationship between the execution time of select operations that depend on condition and the number of rows in the database for the three algorithms.

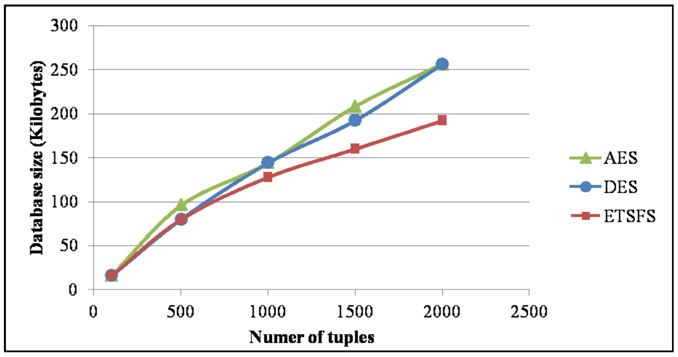

Fig. 15. The relationship between the database size after applying encryption and the number of rows for the three algorithms.

VI. Conclusion And Future Work

Data-storing and exchanging between computers is growing fast across the world. The security of this data becomes an important issue for the world. The best solution centred on securing the data is using cryptography, besides 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 the increasing on the data size, and also to correct 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 the query processing time or the 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 amongst 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 and R. Sujarani. “Light weight and secure database encryption using TSFS algorithm”. in *Proc. Int. Conf. Computing Communication and Networking Technologies*, July 2010, pp. 1-7.

[2] L. Bouganim and P. Puncheral. "Chip-Secured data access: Confidential data on untrusted servers". in *Proc. Of the 28th International Conference on Very Large Databases*, August 2002, pp. 131-142.

[3] Z. Yang, S. Sesay, J. Chen and D. Xu. "A Secure Database Encryption Scheme". in *Proc. of Consumer Communications and Networking Conference*, January 2005, pp. 49-53.

[4] K. Kaur, K. Dhindsa and G. Singh. “Numeric to numeric encryption: using 3KDEC algorithm”. in *Proc. IEEE Int. Conf. Advance Computing*, March 2009, pp. 1501-1505.

[5] A. Rakesh, K. Jerry and S. Ramakrishnan. "Order preserving encryption for numeric data". in *Proc. ACM SIGMOD Int. Conf. Management of Data*, June 2004, pp.563-574.

[6] Z. Yong-Xia, "The Technology of Database Encryption". in *Proc. 2nd Int. Conf. MultiMedia and Information Technology*, April 2010, pp. 268-270.

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

[8] I. Widiasari. “Combining advanced encryption standard (AES) and one time pad (OPT) encryption for data security”. *The International journal of Computer Applications*, vol. 57, pp. 1-8 , November 2012.

[9] C. Raj, M. Kumar, D. Manivannan and A. Umamakeswari, “Design and development of secret session key generation using embedded crypto device-ARM-LPC 2148”. *Journal of Artificial Intelligence*, vol.6, pp.134-144, 2013.