International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 4 | Issue 2 | ISSN : 2456-3307

Database Encryption Using TSFS Algorithm

J. Raja Sekhar[1] , G. Sivaranjani [2]
[1]Student, Department of Computer Science, Rayalaseema Institute of Information and Management Sciences, Tirupati, India
[2]Associate Professor, Department of Computer Science, Rayalaseema Institute of Information and Management Sciences, Tirupati , India

ABSTRACT:

Security of databases has become increasingly crucial in all application areas. Database encryption is an important mechanism to secure databases from attacks and unauthorized 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: Security, Transposition, Substitution, Folding, Shifting.

INTRODUCTION

The main objective is to provide security for database for the web-based application. Client of this web application is a college. College Authority wants to maintain a website for helping students and also wants to maintain information about student's attendance, marks, registration details and also wants to percentages of final year students for placements in different companies.Data security has consistently been a major issue in web applications. Database security has paramount importance in industrial, civilian and government domains.

Organizations are storing big amount of data in database for data mining and other types of analysis. Some of this data is considered sensitive and has to be protected from disclosure.

Database systems are usually deployed deep inside the company network and thus insiders have the easiest opportunity to attack and compromise them, and then steal the data. So, data must be protected from inside attackers also.

Many conventional database security systems are proposed for providing security for database, but still the sensitive data in database are vulnerable to attack because the data are stored in the form of plaintext only.

Fig.1 Database security

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, multitiered 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. 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. 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.

II. 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.

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. Encryption algorithm.

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. 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. 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

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

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:

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

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:

$$D(E(x)) = M - |D(E(x))|$$

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.

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]+ numeric(data[i,j]) mod M)+k2[i,j]) mod M; end for
    end for
    data = temp;
    return data;
```

End substitution

Fig: Substitution algorithm

Algorithm inverse Substitution (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 inverse Substitution

Fig: Inverse Substitution

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.

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: 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 inverse Folding

Fig: 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.

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.

CONCLUSION

The security of this data has become an important issue for the Organization. The best solution centered 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.

REFERENCES

1. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikanth, Yirong Xu,2002, Hippocratic databases, Proceedings of the 28th international conference on Very Large Data Bases.
2. L Liu, J. Gai,2008, A new lightweight database encryption scheme transparent to applications, Proceedings of the 6th IEEE International Conference on Industrial Informatics.
3. K Kaur, K. Dhindsa, G. Singh, 2009, Numeric to numeric encryption: using 3KDEC algorithm, Proceedings of IEEE International Conference on Advance Computing. [4] D.
4. Manivannan, R. Sujarani, 2010, Light weight and secure database encryption using TSFS algorithm, Proceedings of the International Conference on Computing Communication and Networking Technologies.
5. Hanan A. Al-Souly, Abeer S. Al-Sheddi, Heba A. Kurdi., 2013 Lightweight Symmetric Encryption Algorithm for Secure Database, (IJACSA) International Journal of Advanced Computer Science and Applications. [6] S. Bhatnagar, Securing Data-At-Rest, Literature by Tata Consultancy Services.
6. Gang Chen, “A Database Encryption Scheme for Enhanced Security and Easy Sharing Computer Supported Cooperative Work in Design”, 2006. CSCWD „06. 10th International Conference on Publication Year: 2006.