Masking using Format Preserving Encryption

Jaswinder Kaur, Maninder Kaur

Department of Computer Science and Engineering, Department of Electronics and Communication Engineering
Doaba Institute of Engineering & Technology, Ghataur, Kharar, India

er.jaswinkaur@gmail.com, maninderecediet@gmail.com

Abstract: For every organization the most important thing is sensitive data. Data masking and data encryption are used to achieve this. In this paper we introduce an efficient algorithm to mask the sensitive data using format preserving encryption.

Keywords: Format Preserving Masking (FPM); Format Preserving Encryption (FPE); Credit Card Numbers (CCN); Social Security Numbers (SSN)

I. INTRODUCTION

The survey of data privacy domain showed that the predominant percentage of all security invasions is from the internal threats, making the cost of internal security breaches nearly half a time more than that of the external breaches [1]. With the use of online applications and software, the organizations are moving to third party services to store their data and associated applications to secure online transactions, and to protect data from anomaly administrators and malicious attackers by encryption and masking of Credit card numbers (CCN), Social Security Numbers (SSN).

Format preserve masking (FPM) or Data masking technique masks or conceals the sensitive data in a database from unauthorized usage. The masked data looks similar to real data and can be used in test and development environments. The efficiency of the data masking technique is that the original data cannot be reconstructed from the modified data unless the masking technique is known [7]. It plays a vital role in real time data usage. Many challenges are also faced while securing the sensitive data like the cost of modifying existing databases, sensitive information like SSN and CCN are used as a primary key in database changes in this field may require significant schema changes and applications related to specific data format will require a format change. To preserve the original format of the data, Format preserving encryption (FPE) has been used.

The rest of the paper is organized as follows. In Section II the data masking techniques are discussed in brief. The proposed technique is discussed in Section III. Conclusions are drawn in Section IV.

II. DATA MASKING TYPES

Several data masking techniques are listed below. In this section we provide a brief introduction to various data masking techniques.

A. Substitution

In substitution we replace the existing data in the database with some random values that look real and are of the same type as the original data [2]. The modified data and the original data may or may not be relatable. It also has its own advantage, that the look of the data remains intact even after substitution.

B. Shuffling

Shuffling is similar to substitution except that the substitution takes place between rows of the database. Shuffling is done till there is no two related data present in the same row [2]. The advantage of shuffling over substitution is that generating of random unique values is not necessary.

C. Number and Data Varinace

Number and data variance is used for masking of numeric or date fields. The original values are varied within a specific range in this technique. The advantage of this technique is that the look of the data does not change since the modified value has some percentage of the real value [2]. It prevents bypassing of the records using the number and date fields.

A. Encryption

Algorithmic approach is practiced for modifying the data in case of encryption. Encrypted data does not look like the original data because of the existence of special characters in encrypted data. Encryption using key-algorithms reveals the data with the key [3]. So when the key is in the hands of unauthorized persons, they can easily decipher the data.

B. Nulling Out/ Deleting

In this technique, we just delete sensitive data and replace the fields with NULL values [3]. This technique is not so useful for the databases in test and development environments and only can be used for the database without those environments.

C. Masking Outs

Masking out replaces some parts of the data with specific characters like X or *. Proper care should be taken in masking out appropriate data by not masking required information. If the required information is masked then the entire field becomes useless [3].
III. PROPOSED TECHNIQUE

The proposed implementation is based on using the FPE technique to achieve FPM. FPE uses AES-128 Algorithm and two additional steps to get the format preserved [6]. We have used this technique to achieve masking and added an additional step to achieve more efficiency.

Firstly the 16 digit CCNO digit is encrypted then the encrypted value is converted to real time scenario looking value [4]. This implies that at end, the encrypted value is converted to get the 16 digit plain text CCNO as shown in “Fig. 1”.

![Fig. 1. Overview of FPM](image)

The AES parameters are:
- 128 bit plain text
- 128 bit key

This will produce a 128 bit cipher text within 16 rounds. The number of possible keys that can be used for the encryption is \(2^{128}=3.4 \times 10^{38}\). So even if we have a robotic hand with a keyboard that could try out \(2^{55}\) keys per seconds would also take approx. 149 billion years to break the cipher text. Before beginning, “Fig. 2” depicts the various representations forms.

![Fig. 2. Number Representation](image)

In total of 16 rounds, each round consists of four major transformations. But before these four steps the 128-bit plain text is converted to 16 bytes. These bytes are represented as 4X4 order matrix which is popularly known as state array. As shown in “Fig. 3”, \(b_0-b_{15}\) denotes the 16 bytes and the obtained matrix is called as state matrix.

![Fig. 3. State Matrix](image)

A. Sub Bytes

Each \(A_{(i,j)}\) in the state matrix is replaced by \(S(A_{(i,j)})\) using the Rijndael S-box. The first transformation, sub bytes is used at the encryption site. In order to substitute a byte we interpret the byte as two hexadecimal digits [4]. The sub bytes operation involves 16 independent byte-to-byte substitutions. This is shown in “Fig. 4”.

![Fig. 4. Sub Bytes](image)
In state array the first four bytes are used to identify the row while the lower order four bytes are used to identify the columns. For example the number 41 in the state array is substituted by 83 (4th row, 1st column). Similarly all the 16 elements in the state array are replaced by another 16 elements using the S-BOX. The SubByte transformation is shown in “Fig. 5”.

Note that the S-box is generated by determining the multiplicative inverse for a given number in GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1), Rijndael’s finite field. Zero, which has no inverse, is mapped to zero. This transformation is known as the “Nyberg S-box” after its inventor Kaisa Nyberg. The multiplicative inverse as shown in “Fig. 6” is then transformed using the following affine transformation where \[x_0, \ldots, x_7\] is the multiplicative inverse as a vector.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7
\end{bmatrix}
\]

Fig. 4. Rijndael S-box

Fig. 5. SubByte Transformation

\[
\begin{bmatrix}
b_0 & b_1 & b_2 & b_3 \\
b_4 & b_5 & b_6 & b_7 \\
b_8 & b_9 & b_{10} & b_{11} \\
b_{12} & b_{13} & b_{14} & b_{15}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_0 & b_1 & b_2 & b_3 \\
b_4 & b_5 & b_6 & b_7 \\
b_{10} & b_{11} & b_{12} & b_{13} \\
b_{14} & b_{15} & b_0 & b_1
\end{bmatrix}
\]

Fig. 6. Multiplicative Inverse

B. Permutation

Permutation is achieved by shifting rows as shown in “Fig. 8, 9”. In Masking, we will be using the same concept of permutation as used in encryption which is called as shifting rows. Here the first row is never shifted, the second row is shifted by one byte, then the second by two, the third by three and so on. Hence, concluding on the concept to shift the nth row by n times.

Fig. 7. Row Shifting

Fig. 8. Row Shifting Example

C. Column Mixing

This step takes four bytes as input value and produces the output as a similar four byte value as shown in “Fig. 9”. In column mixing we use a Mix-column Function (f_{mix}). Here each input byte affects all the four output bytes on the result produced. Together with the shift-rows, column mixing produces the diffusion process in the cipher text. To modify the bits inside the byte we need a inter byte transformation. These bits are transformed based on the bits inside the nearby bytes to mix bytes to produce the diffusion at each bit level. Unlike the permutation step, column mixing operates at the column level. It does a transformation for each column of the matrix array into a new column.

Fig. 9. Column Mixing

As shown in “Fig. 10”, we can see the process of column mixing will require the matrix multiplication task.
Fig. 10. Matrix Multiplication

D. Adding Key

Similar to the Column mixing step the Adding key too proceeds using one column at a time. It adds a round keyword with each state column matrix. Matrix addition operation is used in Add Round key phase as shown in “Fig. 11”. The sub-key is added by combining each byte of the state with the co-related byte of the sub-key using the bitwise XOR operation.

Fig. 11. Adding Key

After the 16 rounds have been completed using the above described 4-rounds in each of the 16th state. Three additional steps are implemented to preserve the format and type of the plaintext. There are in total 3 additional steps that includes two steps to preserve the format and an additional third step to make the increase it security value.

E. EX-OR Operation

The resultant 128 bit obtained from the above steps is divided into group of 8 bits. So in total we obtain 16 group of 8-bit each. For each group we perform the EX-OR operation [5]. The truth table for EX-OR is depicted below:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 12. EX-OR Gate Outputs

For our process the higher ordered four bytes are EX-OR(ed) with the lower order four bytes. As a result, at the end of this process we get a 64-bit block as shown in “Fig. 13”. So, applying the EX-OR operation to the remaining groups we get 8 hex digits as the result: 9F7A538F9C587F. The 16 digit cipher text is 3975532936553799.

F. Conversion

Here we apply the 2421 coding to the hexa-digits, we get the exact 16 decimal digit and each digit is in the valid form i.e exists from 0 to 9 as shown in “Table 1”. An ordinary hex to decimal conversion cannot be applied here. In hex the letter A to F represent a two digit number. So instead of ordinary conversion we apply the 2421 decimal conversion method to get the valid decimal digit.

<table>
<thead>
<tr>
<th>TABLE I. CONVERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 ---- 1001 ---- 2+0+0+1 ---- 3</td>
</tr>
<tr>
<td>F ---- 1111 ---- 2+4+2+1 ---- 9</td>
</tr>
<tr>
<td>7 ---- 0111 ---- 0+4+2+1 ---- 7</td>
</tr>
<tr>
<td>A ---- 1010 ---- 2+0+2+1 ---- 5</td>
</tr>
<tr>
<td>5 ---- 0101 ---- 0+4+0+1 ---- 5</td>
</tr>
<tr>
<td>3 ---- 0011 ---- 0+0+2+1 ---- 3</td>
</tr>
<tr>
<td>8 ---- 1000 ---- 2+0+0+1 ---- 2</td>
</tr>
<tr>
<td>F ---- 1111 ---- 2+4+2+1 ---- 9</td>
</tr>
<tr>
<td>9 ---- 1001 ---- 2+0+0+1 ---- 3</td>
</tr>
<tr>
<td>C ---- 1100 ---- 2+4+0+0 ---- 6</td>
</tr>
<tr>
<td>B ---- 1011 ---- 2+0+2+1 ---- 5</td>
</tr>
<tr>
<td>5 ---- 0101 ---- 0+4+0+1 ---- 5</td>
</tr>
<tr>
<td>8 ---- 1000 ---- 2+0+0+1 ---- 3</td>
</tr>
<tr>
<td>D ---- 1101 ---- 2+4+0+1 ---- 7</td>
</tr>
<tr>
<td>F ---- 1111 ---- 2+4+2+1 ---- 9</td>
</tr>
<tr>
<td>F ---- 1111 ---- 2+4+2+1 ---- 9</td>
</tr>
</tbody>
</table>

The 16 digit cipher text is 3975532936553799.

G. Shift ‘Mask’ Times

In the final step, we sum up the obtained resultant from the conversion step.
3+9+7+5+5+3+2+9+3+6+5+5+3+7+9+9 = 90 = 9+0 = 9 (mask)

This value obtained after adding up is called the mask.

- If the mask value is ‘EVEN’ then operation to be performed is right-shift the number of times as is the mask value.
- If the mask value is ‘ODD’ then operation to be performed is left-shift the number of times as is the mask value.

In our example the mask value is 9 which is odd so we will perform the left shift 9 times. The value 3975532936553799 will be 3655379939755329 after the final step.

At the end of the last step the plain input text and the final obtained text are in the same format and data type. Hence no need to change the database structures, queries and application programs to handle this final obtained masked value.

IV. CONCLUSION

In this paper we have proposed an algorithm for FPM using FPE to secure the sensitive data. The table “Table II” and “Fig. 14” shows the performance of the existing FPE techniques and Proposed FPE to encrypt 128 bit plaintext. There is no iterations and table build time in proposed FPE technique. So it requires limited time for masking.

TABLE II. CONVERSION

<table>
<thead>
<tr>
<th>Technique Used</th>
<th>Time Required (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix Method</td>
<td>4500</td>
</tr>
<tr>
<td>Cycle Walking Method</td>
<td>3050</td>
</tr>
<tr>
<td>Feistel and Cycle Walking Method</td>
<td>10,000</td>
</tr>
<tr>
<td>Proposed Technique</td>
<td>450</td>
</tr>
</tbody>
</table>

Fig. 14. Time Requirements

The proposed technique is very useful for real time applications like masking CCN or SSN. This algorithm retains the format and data type of plaintext even after masking. There is no need to change the database structure, queries and application programs after masking.

REFERENCES