

Secure Format Preserving Encryption for Multiple Data Fields

S.Vidhya

{vidhya.s@kgcas.com}

Associate Professor, Department of Computer Science, KG College of Arts and Science,
Coimbatore, Tamilnadu, India

Abstract. In Format Preserving Encryption the cipher text will have the similar length, structure and type as the plain text. By maintaining the original format, FPE ensures that the encrypted cipher text can be seamlessly used in existing systems and processes without requiring significant modifications. The limited randomness leads to security vulnerabilities, such as statistical analysis attacks or frequency-based attacks. FPE algorithms are typically used in real-world applications and systems where security is of utmost importance. In this paper a more secure Format Preserving Encryption algorithm by interleaving multiple fields is proposed.

Keywords: Format Preserving Encryption, FPE, Order preserving Encryption, Cryptography, Database encryption.

1 Introduction

Format Preserving Encryption (FPE) in a database refers to the application of FPE techniques specifically to protect sensitive data stored within a database system. FPE in databases is designed to encrypt the data while preserving its original format and maintaining its usability within the database ecosystem.

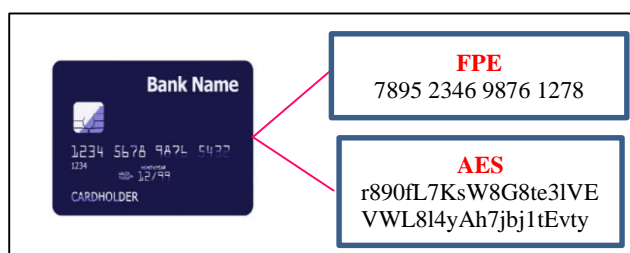


Fig. 1. Credit card encryption using FPE and traditional encryption scheme.

It is mainly applicable to small fixed length domains. Unlike traditional encryption techniques that render data unreadable, FPE enables the ability to search and query encrypted data.

Applications can perform operations such as sorting, filtering, indexing and searching on encrypted data, allowing for efficient analysis of data and retrieval without compromising security. The initial proposal of FPE was made by Michael Brightwell et al [1]. Later three major schemes are proposed such as cycle-walking cipher, prefix cipher and generalized Feistel cipher along with a technique that utilized a Rank then-Encipher (RTE) approach and an unbalanced Feistel structure. RTE enables FPE to be used for different format. The NIST endorsed FFX, VAES3, and BPS algorithms and the corresponding standards are FF1, FF2, and FF3. However, FF2 did not meet NIST's security requirements and was consequently eliminated from FFX modes. The 64-bit tweak parameter employed in FF3 was reduced to 56 bits, resulting in the revised version known as FF3-1. Currently, the standards for Format Preserving Encryptions are FF1 and FF3-1. [1].

2 FF1 and FF3-1

2.1 Encryption and Decryption

FF1 and FF3-1 use block cipher encryption to maintain the format of plaintext. FF3-1 uses AES block cipher in Counter (CTR) mode as the default option. Although the specific block cipher employed in FF1 may differ, the standard recommends the Advanced Encryption Standard (AES) as the default choice [2]. The subsequent notations illustrate the encryption and decryption functions, along with the corresponding parameters, of the FF1 and FF3-1 schemes.

PT – Plaintext, CT – Cipher text, Key – Secret key, TW – Tweak

Encrypt (Key, TW, PT) = CT, Decrypt (Key, TW, CT) = PT

2.2 Feistel Structure

The construction employed in block ciphers, known as a Feistel network structure, serves as the foundation for the FF1 and FF3-1 modes. In both FF1 and FF3-1, the Feistel network operates on a block of data by dividing it into two halves, applying a round function to each half, and then combining the halves. This process is repeated for all the rounds.

Data Division. The input data block is equally divided into two parts and referred as the left partition (L) and the right partition (R).

Round Function. The input for the round function is round key and right partition R to produce the output (R').

Swap and Feedback. At the end of each round, the right and left portions are interchanged. The new right (R') becomes left and left partition becomes right.

Final Round. After a certain number of rounds, a final round is performed without swapping the halves.

Output. The final outcome of the Feistel network is the concatenation of the left partition (L) and the right partition (R), producing the cipher text.

2.3 FF1

The FF1 algorithm is defined to operate on binary strings of arbitrary length and supports various data formats, such as numbers and alphanumeric strings [3].

Table 1. Parameters in FF1

minlen	2	maxlen	2^{32}	radix	$2..2^{16}$	tweak T	Optional t=0	round	10
--------	---	--------	----------	-------	-------------	---------	--------------	-------	----

In the encryption algorithm mentioned above, the plaintext X is divided into two partitions, A and B . The internal function employs the length of B (b), along with d and P (the initial block), for the subsequent encryption process. Additionally, the tweak value (T) and the round number (i) are utilized to generate Q from B . R is then formed using the opening blocks P and Q , followed by encryption to obtain S . The modulo the radix value (m) is appended with the decimal strings which are derived from S and A . The resulting value, linking the final A and B values, represents the encrypted value. The FF1-Decrypt algorithm is identical to the FF1-Encrypt algorithm, with differences occurring in Step 6: 1) the sequence of the indices is inverted, 2) the values of A and B are interchanged, and 3) modular subtraction is used instead of modular addition in Step 6vi.

2.4 FF3-1

FF3-1 is mainly used for small domains. It additionally uses Tweak to increase the domain space. The tweaks should be generated randomly for high entropy in each half. [4].

Table 2. Parameters in FF3-1

minlen	2	maxlen	$2 \times \lceil \log_{\text{radix}}(2^{96}) \rceil$	radix	$2..2^{16}$	tweak T	56 bits	round	8
--------	---	--------	--	-------	-------------	---------	---------	-------	---

The algorithm, resembling FF1, partitions the plain text X into two equal portions, A and B . The value of random generated tweak is 54 bits. TL is formed by concatenating $T[0..27]$ with 04 , while TR is created by combining $T[32..55]$, $T[28..31]$, and 04 .

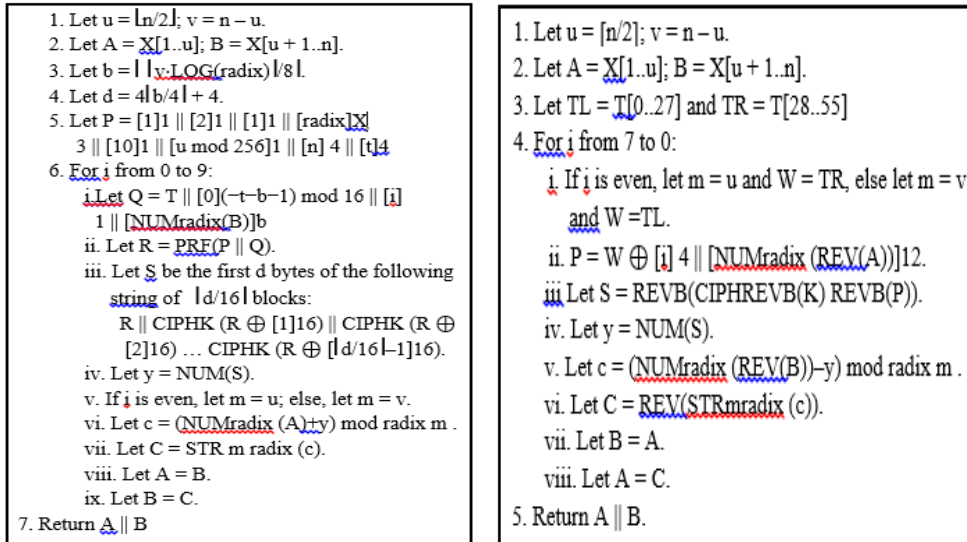


Fig. 2. FF3-1 Encryption and Decryption algorithm

The i represents the round number. If i is an even number, m is assigned to u , and W takes the value of TR . Conversely, if i is odd, m is assigned as v , and W is set to TL . A four-byte value is constructed using W and i . By reversing the radix-notation number B and padding it with leading zeroes until 12 bytes of data are created, P is generated by combining the two. C is produced by utilizing S and A in a manner similar to FF1. FF3-1. The value of C is derived by repeatedly executing the round function by 8 times. The cipher text is obtained by combining the final A and B values.

3 Proposed Algorithm

FF1 and FF3-1 are deterministic encryption modes, meaning that the same plaintext will always encrypt to the same cipher text. The lack of probabilistic encryption introduces the possibility of frequency or pattern analysis on the cipher text [5]. The proposed algorithm increases the complexity of existing standard FF3-1 algorithm by interleaving multiple fields to create the plaintext.

3.1 Format Preserving Encryption for multiple fields

The proposed algorithm is derived from FF3-1 with an additional layer of customization and security to the encryption scheme. The proposed algorithm is tested by encrypting credit card number along with expiry month and date.

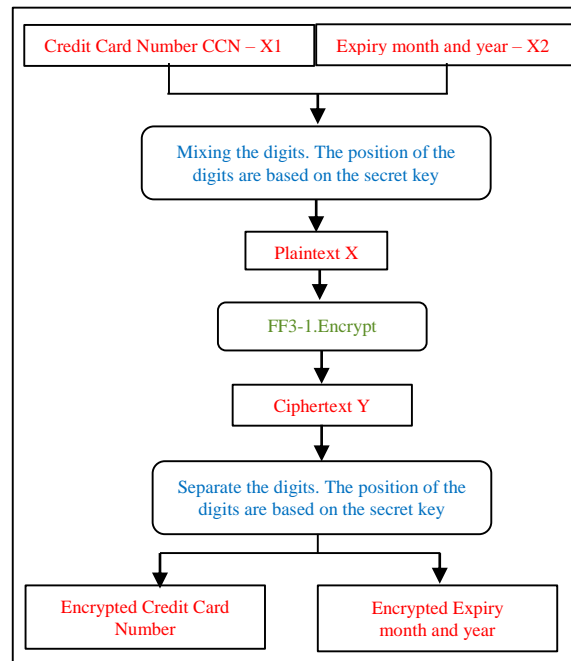


Fig. 3. Format Preserving Encryption for multiple fields

3.2 Python procedures for FPE for multiple fields

The python procedure `interleave_digits` is used to mix the digits of two columns based on the secret key. The hexadecimal digits in the keyset is converted into decimal digits to perform arithmetic operations. In order to avoid the uniform distribution of positions, multiplication and modulus operations are applied to each digit.

```

def interleave_digits(X1, X2,k):
#X1is the credit card number
#X2 is the expiry month and year
#k is the secret key
X1_digits = list(map(int, str(X1)))
X2_digits = list(map(int, str(X2)))
X2_len = len(X2_digits)
X_len = X1_len + X2_len
X = []
for digit in k:
    keyset.add(int(digit, 16))
X = [0] * (X1_len +X2_len)
# To avoid duplicate positions
keyset = list(keyset)
# To avoid uniform distribution
positions = [(element * keyset[-1]) % X_len) for element in
keyset]
positions = positions[:X2_len]
  
```

```

index = 0
for i in X2_digits:
    X[(positions[index])] = i
    index = index + 1
i=0
for index in range(0,X_len):
    if index not in positions:
        X[index]=X1_digits[i]
        i=i+1
Return(X)

```

The credit card number is represented as X1 and the six digits expiry month and year is X2. Insert each digit in X2 to X1 based on the key. The insertion position depends on the secret key. The hexadecimal secret key is converted into decimal digits and stored in the python set in order to remove the duplicate digits. The resultant list is stored on X which is used as input for FF3-1 algorithm.

The python procedure separate_digits is used to divide the encrypted multiple fields into two fields. The resultant Cipher texts Y is the output of Format Preserving Encryption of FF3-1 scheme in which the format and datatype of the cipher texts are equivalent to the original fields Credit card number and expiry month and year.

```

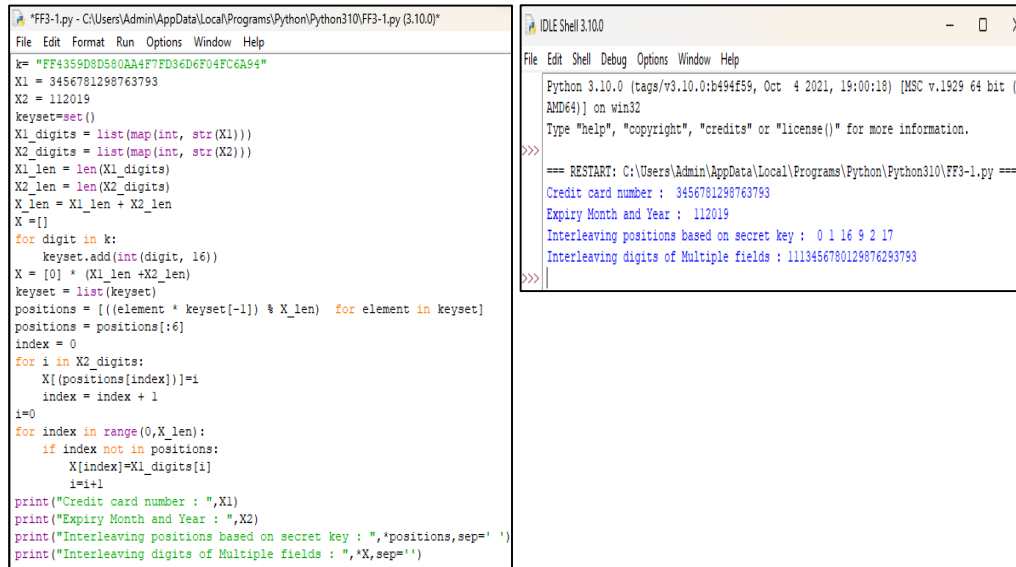
def separate_digits(Y,k):
# Y is the encrypted value
keyset = set(str(int(k,h=16)))
positions = list(keyset)
Y1 = []
Y2 = []
Y_digits = list(map(int, str(Y)))
for i in range(0,len(Y_digits)):
    if i in positions:
        Y2.append(int(Y_digits[i]))
    else:
        Y1.append(int(Y_digits[i]))
return Y1, Y2

```

The output of the FF3-1 algorithm is encrypted value Y which is the combination of encrypted values of credit card number and expiry month and year. The y is separated into Y1 and Y2 based on the positions which are derived from the secret key.

3.3 Experiments

The procedures `interleave_digits()` and `separate_digits()` are tested with 16 digits credit card number and 6 digits expiry month and date and 128 bit secret key which is represented by hexadecimal value[6]. The added procedures increase the complexity of the FF3-1 algorithm. The above procedure is executed on Python's idle 3.10.0 version with the inputs.



```
*FF3-1.py - C:\Users\Admin\AppData\Local\Programs\Python\Python310\FF3-1.py (3.10.0)*
File Edit Format Run Options Window Help
k= "FF4359D8D580AA4F7FD36D6F04FC6A94"
X1 = 3456781298763793
X2 = 112019
keyset=set()
X1_digits = list(map(int, str(X1)))
X2_digits = list(map(int, str(X2)))
X1_len = len(X1_digits)
X2_len = len(X2_digits)
X_len = X1_len + X2_len
X=[]
for digit in k:
    keyset.add(int(digit, 16))
X = [0] * (X1_len + X2_len)
keyset = list(keyset)
positions = [(element * keyset[-1]) % X_len for element in keyset]
positions = positions[:6]
index = 0
for i in X2_digits:
    X[positions[index]] = i
    index = index + 1
i=0
for index in range(0, X_len):
    if index not in positions:
        X[index] = X1_digits[i]
        i=i+1
print("Credit card number : ", X1)
print("Expiry Month and Year : ", X2)
print("Interleaving positions based on secret key : ", positions, sep=' ')
print("Interleaving digits of Multiple fields : ", X, sep='')

IDLE Shell 3.10.0
Python 3.10.0 (tags/v3.10.0:b494e59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\Admin\AppData\Local\Programs\Python\Python310\FF3-1.py ===
Credit card number : 3456781298763793
Expiry Month and Year : 112019
Interleaving positions based on secret key : 0 1 16 9 2 17
Interleaving digits of Multiple fields : 1113456780129876293793
>>>
```

Fig. 4. Procedure `interleave_digits()` coding and **Fig. 5.** output in Python's IDLE

The input credit card number, expiry month and date, the interleaving positions based on the secret key and the interleaved output which is acted as the input for FF3-1 are printed. The output of the procedure `interleave_digits` is applied to the FF3-1 scheme.

The encrypted value is given as input for `separate_digits()` procedure with 128 bit secret key to separate the encrypted values for credit card number and expiry month and year. The outcome of the FF3-1, the cipher text is applied to the procedure `separate_digits` to get the correct number of encrypted digits of both Credit card number and expiry month and date. The format of the encrypted fields are same as original fields. During the decryption process, the same sequence of procedures are applied to get the correct plaintext. The encrypted credit card number and expiry month and date are applied to the procedure `interleave_digits()`. The output of the procedure is acted as input for FF3-1 decryption algorithm. The output of the decryption is applied to the procedure `separate_digits()` to get the original credit card number and expiry month and date.

```

Reverse condng.py - C:\Users\Admin\AppData\Local\Programs\Python\Python310\Reverse condng.py (3.10.0)
File Edit Format Run Options Window Help
k= "FF4359D8D580AA4F7FD36D6F04FC6A94"
Y = 4601862737748642371923
Y = list(map(int, str(Y)))
Y_len = len(Y)
keyset=set()
for digit in k:
    keyset.add(int(digit, 16))
Y1 = []
Y2 = []
keyset = list(keyset)
positions = [(element * keyset[-1] & Y_len) for element in keyset]
positions = positions[:6]
for i in positions:
    Y2.append(Y[i])
for i in range(0, Y_len):
    if i not in positions:
        Y1.append(Y[i])
print("The ciphertext of Multiple fields Y : ", 'Y, sep='')
print("After separation")
print("Interleaving positions based on secret key : ", 'positions, sep=' ')
print("Encrypted credit card number : ", 'Y1, sep='')
print("Encrypted month and year : ", 'Y2, sep='')

```

Fig.6. Procedure separate_digits() coding

```

IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Admin\AppData\Local\Programs\Python\Python310\Reverse condng.py
The ciphertext of Multiple fields Y : 4601862737748642371923
After separation
Interleaving positions based on secret key : 0 1 16 9 2 17
Encrypted credit card number : 1862737486421923
Encrypted month and year : 463707

```

Fig. 7. Procedure separate_digits() coding and the output in Python's IDLE

The encrypted value Y is separated in to encrypted credit card number Y1 and encrypted expiry month and year Y2. It is the reverse of the procedure interleave_digits(). Based on the secret key the cipher text Y is separated into two fields. The encrypted values are same format and data type as plaintext such credit card number and expiry month and year. The proposed algorithm can also be applied to encrypt Social Security Number, Aadhar number and other small domains. In all the applications, more than one fields are encrypted in a single encryption and also ensure the format of the chipper text is same as plaintext. The complexity of the proposed algorithm is increased to provide added security for FPE. By increasing the number of encrypted fields, the security of the proposed algorithm is also increased.

4 Performance analysis

The proposed algorithm has the following features.

- More than one column is encrypted in a single encryption which reduces the overhead and time.
- Using a single secret key, multiple fields can be encrypted.

- Mixing multiple columns, the proposed algorithm is more secure than FF3-1.
- The algorithm is non-deterministic hence the same plaintext never produces the same ciphertext.

One of the key benefits of performing the encryption for multiple fields is the possibility of avoiding pattern-based attack which is more common in Format Preserving Encryption. The proposed algorithm minimizes patterns and correlations in the ciphertext [7]. It's also crucial to ensure that the domain size (the set of possible plaintext values) is sufficiently large to make it computationally infeasible for attackers to guess the plaintext based on observed patterns.

5 Conclusion

FPE encrypts data in a way that maintains the original format, such as preserving the length, character set, or structure of the data. This requirement introduces additional complexity because the encryption process must respect the format constraints while ensuring security. FPE must provide a high level of security, including properties like confidentiality, integrity, and resistance against attacks such as brute-force, known-plaintext, or chosen-plaintext attacks. Achieving these security goals requires more complex cryptographic algorithms and techniques. Due to these additional requirements and complexities, designing and implementing format-preserving encryption can be more challenging than traditional encryption algorithms. The proposed algorithm provides an additional layer of customization and security to the encryption scheme FF3-1.

References

- [1] Wonyoung Jang, Sun-Young Lee. "A formatpreserving encryption FF1, FF3-1 using lightweight block ciphers LEA and, SPECK" Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020
- [2] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. NIST Special Publication, 800:38G..
- [3] F. B. Durak and S. Vaudenay, Breaking the FF3 Format-Preserving Encryption Standard Over Small Domains, CRYPTO 2017.
- [4] Orr Dunkelman, , Abhishek Kumar, Eran Lambooj , and Somitra Kumar Sanadhya "Cryptanalysis of Feistel-Based Format-Preserving Encryption", International Association for Cryptologic Research, 2020.
- [5] Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. NIST Special Publication SP 800-38G Rev. 1(2019)
- [6] F. Betül Durak, Henning Horst, Michael Horst and Serge Vaudenay "FAST: Secure and High Performance Format-Preserving Encryption and Tokenization", Advances in Cryptology – ASIACRYPT 2021 (pp.465-489).
- [7] Ke Luo and Wei Gao, "A study of Format Preserving Encryption algorithm Incorporating Robust Chinese Remainder Theorem", 3rd International Conference on Artificial Intelligence and Advanced Manufacture, October 2021.