**Format Preserving Encryption Algorithm**

**Introduction:**

1. Encryption is a process of converting the plaintext (clear text) to unreadable ciphertext. Decryption is a reverse process in which the cipher text is converted into plaintext.
2. Format Preserving Encryption is - as the name says - an encryption in which the format of the encrypted data is maintained.
3. When a plaintext is encrypted with FPE, the ciphertext then has the

same format again and has same length.

For examples:

1. Encrypt a 16-digit credit card number so that the ciphertext is another 16-digit number.

2. To encrypt an English word so that the ciphertext is another English word.

3. To encrypt an n-bit number so that the ciphertext is another n-bit number

**Modes of FPE:**

Mode—is an algorithm for the cryptographic transformation of data.There are many modes of operations for FPE but we will focus on 2 modes i.e. FF1 and FF3 each employ the Feistel structure.

**1. FF1** :

1. FF1 is Format-preserving, Feistel-based encryption mode.

2. FF1 supports flexibility in the length of the tweak.

3. Round count is ten - Round is basically set of functions which are called repeatedly in order to convert plaintext to ciphertext.

4. FF1 supports a greater range of lengths for the protected , formatted data, as well as flexibility in the length of the tweak.

**2. FF3** :

1. FF3 is also Format-preserving, Feistel-based encryption mode.

2. FF3 has fixed length of tweak.

3. Round count is eight.

4. FF3 achieves greater throughput, mainly because its round count is eight, compared to ten for FF1.

**Feistel Structure:**

FF1 and FF3 each employ the Feistel structure . so we will quickly go through the Feistel structure.

A framework for constructing an encryption mode. The framework consists

of several iterations, called rounds of a reversible transformation, in which a keyed function, called the round function, is applied to one part of the data in order to modify the other part of the data; the roles of the two parts are swapped for the next round.

The transformation consists of three steps:

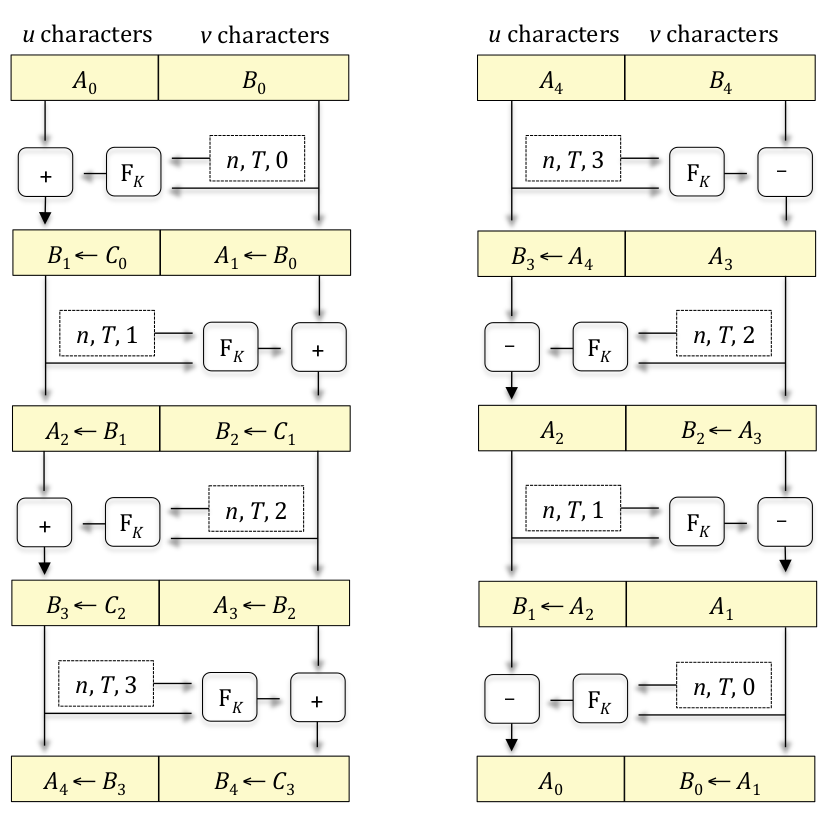
1. The data is split into two parts.

2. a keyed function, called the round function, is applied to one part of the data in order to modify the other part of the data.

3.The roles of the two parts are swapped for the next round.

The structure is illustrated in Figure 1 below, for both encryption and decryption. Four rounds are shown in Figure 1, but ten rounds are actually specified for FF1, and eight rounds for FF3.

For the encryption function in Figure 1, the rounds are indexed from 0 to 3. The input data (and output data) for each round are two strings of characters — which will be numerals for FF1 and FF3. The lengths of the two strings are denoted by u and v, and the total number of characters is denoted by n, so that u+ v = n. During Round i, the round function, denoted by F K , is applied to one of the input strings, denoted by B i , with the length n, the tweak T, and the round number i as additional inputs. (In Figure 1, this triple (n, T, i) of additional inputs is indicated within the dotted rectangles, with the appropriate values for i). The result is used to modify the other string, denoted by A i , via modular addition , indicated by +, on the numbers that the strings represent. The string that represents the resulting number is named with a temporary variable, C i . The names of the two parts are swapped for the next round, so that the modified A i , i.e., C i , becomes B i+1 , and B i becomes A i+1 .

****

EncryptionDecryption

Figure 1: feistel Structure

The rectangles containing the two parts of the data have different sizes in order to illustrate that, u cannot equal v if n is odd. In such cases, the round function is constructed so that the lengths of its input and output strings depend on whether the round number index, i, is even or odd.

**Definitions:**

Here we will go through some definitions which are repeatedly used in implementation of FF1 algorithm.

1. Alphabet : A finite set of two or more symbols.

Ex. alphabet of lower-case English letters :

{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}

1. Base : The number of characters in a given alphabet. The base is denoted by radix.

Ex. alphabet of base ten numerals:

{0,1,2,3,4,5,6,7,8,9}

radix=10

3. Numeral : For a given base, a nonnegative integer less than the base

Ex. alphabet of base ten numerals:

{0,1,2,3,4,5,6,7,8,9}

Numeral =9

4. Numeral String:- For a given base, a finite, ordered sequence of

numerals for the base.

Ex. Given base radix=10

Base radix numeral = {0,1,2,3,4,5,6,7,8,9}

Numeral string = 12345

5. Byte : String of eight bits

Ex. byte= 0000 0101

6. Byte String : A bit string whose length is a multiple of eight bits, so

that it can be represented as the concatenation of a finite sequence of bytes.

Ex. byte string b= 00001111 11110000 - string of two bytes

7. Tweak : The input parameter to the encryption and decryption

functions whose confidentiality is not necessarily protected by the

Mode. tweaks are in the byte string format.

**Parameters requirements for FF1 Algo :**

1. radix : radix ∈ [2 .. 2^16 ]
2. minlen & maxlen: 2 <= minlen <= maxlen < (2^32)
3. Tweak : - Byte^≤maxlen where maxlen = 2^32 − 1
4. Keys : - {0,1}^128 : 128-bit AES keys
5. rnds (n) : 10 : number of rounds
6. CIPH : Cipher function - Ex. AES-based round function

**FF1 Encryption Algorithm:**

The specifications for the FF1 Encryption functions are given in Algorithms below.

The tweak, T, is optional, in that it may be the empty string, with byte length t = 0.

**Prerequisites:**

1. Designated cipher function, CIPH , of an approved 128-bit block cipher
2. Key, K, for the block cipher
3. Base, radix
4. Range of supported message lengths, [minlen .. maxlen];
5. Maximum byte length for tweaks, maxTlen

**Inputs:**

1. Numeral string, X, in base radix of length n, such that n ∈ [minlen .. maxlen]
2. Tweak T, a byte string of byte length t, such that t ∈ [0 .. maxTlen].

**Output:**

Numeral string, Y, such that LEN (Y) = n.

**Steps:**

1. u = ⎣n/2⎦; v = n – u;

u and v are non-negative integers and these values are calculated using n(length of input Numeral string to FF1)

To understand 1st step we need to understand floor division

function.

**Floor function:**

1. For a given real number ‘x’ , floor function is denoted by ⌊x⌋
2. And give greatest integer that does not exceed ‘x’.

Ex. ⌊2.1⌋ = 2 , ⌊4⌋ = 4

**2**. A = X [1 .. u]; B = X [u + 1 .. n]

‘X’ Numeral string is split into two substrings ‘A’ and ‘B’

A=X[1..u] -gives numeral string from index 1 to u

**3**. b = ⎡ ⎡v ⋅ LOG (radix)⎤/8⎤

b is parameter used in step 6.i to calculate Q value.

In step 3 two functions are used :

**Ceiling Function**: - denoted by ⎡x⎤ , for given real value of x and gives output which is the least integer that is not less than the real number x.

Ex. ⎡2.1⎤ = 3 , ⎡4⎤ = 4

**LOG(radix):** base 2 logarithm of x

Ex. LOG(10) = 3.32... radix=10

LOG(36)= 1.556.. radix=36

**4**. d = 4 ⎡b/4⎤ + 4

d is a parameter used to calculate value of ‘S’(in step 6.iii)

**5**. P = [1]1 || [2]1 || [1]1 || [radix]3 || [10]1 || [u mod 256]1 || [n]4 || [t]4

P is another parameter used in step 6.ii as a input to

pseudorandom function.

P should be 16 bytes

Here in step 5 ‘ || ‘ - represent concatenation operator

Ex. 1 2 3 || 4 5 = 1 2 3 4 5

[x] s :- ([x]^s): Given a nonnegative integer x less than 256 s , the representation of x as a string of s bytes

Ex. [2]1=2

[radix]3=[10]3=[0,0,10] - string of ‘s’ bytes

**6**. For i from 0 to 9 : - As total rounds in FF1=10

**6.1**. Q = T || [0]^(−t−b−1) mod 16 || [i]^1 || [ NUM(radix)(B)]^b

In this step i represent round count i.e from 0 to 9 and

NUM(radix)(B) used to convert Numeral string B to ‘number’.

T is byte string tweak (if tweak is present then we need to

consider ASCII value of each and every char ) and 2nd parameter is used to append number of zeros into Q depending upon value of ‘t’(length of tweak) and value b (calculated in step 2)

**NUM(radix)(B) :** The number that the numeral string X

represents in base radix when the numerals are valued in decreasing order of significance.

Algorithm for converting given numeral string to number is given as follows:

**Algorithm : NUM radix (X)**

Prerequisite:

Base, radix.

Input:

Numeral string, X.

Output:

Number, x.

Steps:

1. Let x = 0.
2. For i from 1 to LEN (X), let x = x ⋅ radix + X [i].
3. Return x

**6.ii :** R = PRF (P || Q)

R parameter value is calculated using pseudorandom

function . Algorithm for pseudorandom function is given below:

**Algorithm : PRF (X)**

Prerequisites:

Designated cipher function, CIPH , of an approved 128-bit block cipher;

Key, K, for the block cipher.

Input:

Block string, X.

Output:

Block, Y.

Steps:

1. Let m = LEN (X)/128.
2. Let X 1 ,.., X m be the blocks for which X = X 1 || ... || Xm
3. Let Y 0 = 0 128 , and for j from 1 to m let

Y j = CIPH K (Y j–1 ⊕ X j ).

4. Return Ym .

**6.iii :** Let S be the first d bytes of the following string of ⎡d/16⎤ blocks:

R || CIPH K (R ⊕ [1]^16 ) || CIPHK (R ⊕ [2]^16 ) ... CIPHK (R ⊕ [⎡d/16⎤ – 1]^16 ).

S value is calculated here in step 6.iii using concataning R value with the outputs of block cipher function(CIPHK) with key k and only d bytes are considered in value of S.

**6.iv :** y = NUM(S)

In this step bit string value S is converted to number using NUM(S) function.

Algorithm for NUM(x) is given below:

**Algorithm : NUM (X)**

Input:

Byte string, X, represented in bits.

Output:

Integer, x.

Steps:

1. Let x = 0.
2. For i from 1 to LEN (X), let x = 2 x + X [i].
3. Return x

**6.v :** If i is even, let m = u; else, let m = v

Depending on value of round m value get changed.

‘m’ value get affected only in case if length of input numeral string is odd.

**6.vi :** c = ( NUM(radix)(A) + y) mod radix^m

As discussed above NUM(radix)(A) converts Numeral string A to a number and added to y value which is calculated in step 6.iv

**6.vii :** C = STRradix(c)

In this step STRradix(c) used to convert number c value to ‘Numeral string’.

**STRradix(c)** : Given a nonnegative integer x less than radix m , the representation of x as a string of m numerals in base radix, in decreasing order of significance.

**Algorithm : STR radix (x)**

Prerequisites:

Base, radix;

String length, m.

Input:

Integer, x, such that 0 ≤ x < radix m .

Output:

Numeral string, X.

Steps:

1. For i from 1 to m:

i. X [m + 1– i] = x mod radix ;

ii. x = ⎣x/radix⎦.

2. Return X.

**6.viii :** A = B

**6.ix :** B = C

As mentioned in feistel structure after modifying value of one part need to swap value in order to continue the loop.

**7.** Return A || B

After completion of 10 rounds value of step 7 is returned as cipher text.

**Description of above algorithm:**

he “split” of the numeral string X into two substrings, A and B, is performed in Steps 1 and 2. If n is even, LEN (A)= LEN (B); otherwise, LEN (A)= LEN (B)–1. The byte lengths b and d, which are used in Steps 6i and 6iii, respectively, are defined in Steps 3 and 4. 6 A fixed block, P, used as the initial block for the invocation of the function PRF in Step 6ii, is defined in Step 5. An iteration loop for the ten Feistel rounds of FF1 is initiated in Step 6, executing nine substeps for each round, as follows:

The tweak, T, the substring, B, and the round number, i, are encoded as a binary string, Q, in Step 6i. The function PRF is applied to the concatenation of P and Q in Step 6ii, to produce a block, R, which is either truncated or expanded to a byte string, S, with the appropriate number

of bytes, d, in Step 6iii. (In Figure 1, S corresponds to the output of F K .) In Steps 6iv to 6vii, S is combined with the substring A to produce a numeral string C in the same base and with the same length. (In Figure 1, the combining of S with A is indicated by the “+” operation.) In particular,

in Step 6iv, S is converted to a number, y. In Step 6v, the length, m, of A for this Feistel round is determined. In Step 6vi, y is added to the number represented by the substring A, and the result is reduced modulo the m th power of radix, yielding a number, c, which is converted to a numeral

string in Step 6vii. In Steps 6viii and 6ix, the roles of A and B are swapped for the next round: the substring B is renamed as the substring A, and the modified A (i.e., C) is renamed as B.

This completes one round of the Feistel structure in FF1. After the tenth round, the concatenation

of A and B is returned as the output in Step 7.

**Test Vector :**

**Test Vector 1:**

PT is <0123456789>

Radix = 10

Input (length = 10): "0123456789"

No Tweak

Key is 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

**Intermediate values:**

X is 0 1 2 3 4 5 6 7 8 9 - numeral string

Tweak is <empty>

Step1 : u = ⎣n/2⎦; v = n – u

u is 5, v is 5

Step2: A = X [1 .. u]; B = X [u + 1 .. n]

A is 0 1 2 3 4 - A numeral string

B is 5 6 7 8 9 - B numeral string

Step 3: b = ⎡ ⎡v ⋅ LOG (radix)⎤/8⎤

LOG(10)=3.32=4 - ceil division

v.4=ceil\_div(20/8)=ceil\_div(2.5)=3

b is 3

Step 4: d = 4 ⎡b/4⎤ + 4

d is 8

Step5: P = [1]1 || [2]1 || [1]1 || [radix]3 || [10]1 || [u mod 256]1 || [n]4 ||

[t]4

[radix]3 = [10]3 =[0,0,10] -Represented in byte string upto

3 bytes

[n]4 =[10]4 =[0,0,0,10]

[t]4 = [0]4 =[0,0,0,0] - length of tweak ‘t’ is zero

P is [ 1, 2, 1, 0, 0, 10, 10, 5, 0, 0, 0, 10, 0, 0, 0, 0 ]

- after concatenating all values

- it should be 16 bytes

Step6.i : Q = T || [0](−t−b−1) mod 16 || [i]1 || [ NUM(radix)(B)]b

T is empty - if present we need to convert each digit/char to

its ascii

- 2nd block is to padd 0’s into Q : (-0-3-1)%16=12 : [0]12

- i represent round count

[ NUM(radix)(B)]b= [ NUM(10)(B)]3 - convert numeral string

to integer

NUM(10)(5,6,7,8,9)=[56789]3 = [0,221,213]

-conversion to byte string

(use supportive code

1.numeral\_str\_to\_num.c - To convert numeral string to number

2. binary\_format.c - To convert number to byte string)

Q is [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 221,213 ]

Step6.ii: R = PRF (P || Q)

R is [ 195, 184, 41, 161, 232, 100, 43, 120, 204, 41,148,

123, 59, 147, 219, 99 ]

Step 6.iii: S is c3b829a1e8642b78 - first d bytes

Step 6.iv: y = NUM (S)

y is 14103068008476060536

Step 6.v: m is 5

Step6.vi : c = ( NUM(radix)(A)+y) mod radixm

c is 61770 - here c is a number

Step6.vii: C = STRm radix (c)

C is 61770 - here C is Numeral string

Step 6.viii: A is 5 6 7 8 9 - value of B is assigned to A

Step6.ix: B is 6 1 7 7 0 - value of C i.e modified value of A

assigned to B

This loop will continue till it completes 10 rounds after that A and B concatenate and returned.