The IFX Mode of Operation for Format-Preserving Encryption Over Non-Uniform Symbols

Kai Johnson

November 13, 2016

1. Introduction

Format-preserving encryption (FPE) algorithms such as FFSEM or FFX and its derivatives encrypt small messages of uniform, non-binary symbols while preserving the same set of symbols in their output. FPE algorithms for uniform symbols may be used to encrypt credit card numbers, Social Security numbers, or other messages where the symbols used in the message all share a common format.

This document describes an integer-based format-preserving encryption algorithm called IFX, which encrypts messages of non-uniform symbols. For example, IFX could be used to encrypt vehicle license plate numbers which consist of a specific sequence of digits and letters while preserving this sequence in the ciphertext output.

The IFX method is based on a Feistel network similar to method two of FFX, but differing in its encoding of the input and output, in the method of splitting messages for input into the Feistel network, and in the specifics of the data used in the Feistel network. In concept, IFX encodes the input message as a pair of integers, then uses these integers as input to the Feistel network. Then, IFX decodes the resulting pair of integers into a message in the same non-uniform format as the input.

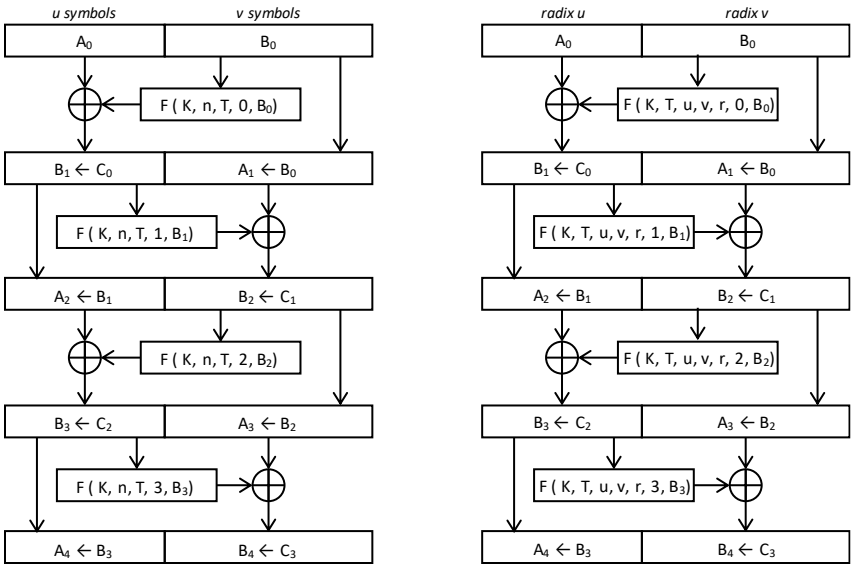

Figure 1: Comparison of FFX method two encryption and IFX encryption. The Feistel network in FFX operates on vectors of symbols of differing length but with a uniform radix for all the symbols. In IFX, the Feistel network operates instead on two integers, each with a different radix. The inputs to FFX, A_0 and B_0, have corresponding permutations of radix^u and radix^v. In comparison, the inputs to IFX, A_0 and B_0, are integers values of radix u and v, respectively. The decryption algorithms for FFX and IFX both operate similarly to encryption, but with the order of the rounds reversed, the roles in relation to the round function F reversed, and with modular addition replaced with modular subtraction.

The IFX method is initialized with a vector of integer radices that describes the input and output formats. A license plate format, for example, might consist of a vector of {10, 26, 26, 26, 10, 10, 10} to represent a sequence of symbols with one digit followed by three letters followed by three digits.

In its general form, IFX accepts arbitrary input formats, with radix vectors of arbitrary length greater than one containing radices greater than one. Specific implementations may forgo this flexibility in favor of hard-coding a single input format.

The IFX encryption algorithm takes a key K, a plaintext vector X, and tweak vector T, where K is a valid key for the underlying AES block cipher, X consists of integers corresponding to the radices specified in the initialization, and T consists of an arbitrary number of bytes. The output of the encryption algorithm is a ciphertext vector Y consisting of integers corresponding to the radices specified in the initialization. Given the same key K and tweak T, the IFX decryption algorithm outputs the plaintext X corresponding to the ciphertext Y.

2. Notation

Variable names use capital letters for vectors and lower case letters for scalars. Literal values of vectors are written as lists of elements in braces, e.g. { 1, 2, 3, 4 }.

IFX uses two's complement integers of constrained magnitude (e.g. 2 or 4 byte integers) and unconstrained magnitude. The integer types may be inferred from the context where they are not explicitly specified.

$x..y$ Sequence of integers greater than or equal to x and less than or equal to y.

$x + y$ Addition of the integers x and y.

$x - y$ Subtraction of the integer y from x.

$x \times y$ Multiplication of the integers x and y.

$x\ div\ y$ Integer division of the integer x by y, i.e. the floor of the real number value of x divided by y.

$x\ mod\ y$ Nonnegative remainder of the integer x modulo the positive integer y.

$\sqrt{x}$ The square root of x.

$\lfloor x \rfloor$ Largest integer less than or equal to the real value of x.

$\lceil x \rceil$ Smallest integer greater than or equal to the real value of x.

$\max(X)$ Largest integer in the vector of integers X.

$length(X)$ Number of elements in the vector X.

$X[n]$ The n[th] element of the vector X, where n is in the range 0 to $length(X) - 1$.

$X[n..m]$ Vector containing the n[th] through m[th] elements of the vector X.

$X \mid\mid Y$ Concatenation of the vectors X and Y.

$descending(X)$ Vector containing the elements of X sorted in descending order.

$primes(x)$ Vector of x elements where the n[th] element contains a vector of the prime factors of n. See Appendix A for possible implementations.

$bitlength(x)$ Number of bits required to represent the integer x.

$ciph(K, IV, X)$ Result of AES encryption in CBC mode using key K and initialization vector I of the byte vector X.

3. Component Functions

$product(X)$ Product of the integers in a vector.

Inputs:

X, a vector of positive integers of arbitrary length

Outputs:

y, an unconstrained integer

Steps:

1. $y \leftarrow 1$
2. For each element x of X
    a. $y \leftarrow y \times x$

$factors(X)$ Prime factors of the integers in a vector. See Appendix A for alternative implementations.

Inputs:

W, a vector of integers of arbitrary length, where each integer is greater than or equal to two

Outputs:

G, a vector of integers of arbitrary length

Steps:

1. $E \leftarrow primes(\max(W))$
2. $G = \{\}$
3. For each w in W
    a. $G \leftarrow G \mid\mid E(w)$

$num(X)$ Unconstrained integer representation of a vector of integer values.

Prerequisites:

W, an arbitrary length vector of radices supplied to IFX.Initialize

Inputs:

X, a vector of $length(W)$ integers such that $0 \leq X[i] < W[i]$ for all i in $0..length(W)$

Outputs:

y, an unconstrained integer

Steps:

1. $y \leftarrow 0$
2. For each i in $0..length(X) - 1$
    a. $y \leftarrow y \times W[i] + X[i]$

$rounds(u, v)$ Number of Feistel rounds required for inputs with radices u and v.

Inputs:

u and v, unconstrained integers

Output

r, a constrained integer

Steps:

1. $x \leftarrow bitlength(v - 1)$
2. $y \leftarrow bitlength(u - 1)$
3. If $x \leq y, r \leftarrow 4 \times \lceil (x + y)/x \rceil$; else $r \leftarrow 4 \times \lceil (x + y)/y \rceil$

$bytes(x)$ Vector of bytes containing the two's compliment representation of an integer.

Inputs:

x, a constrained or unconstrained integer

Outputs:

X, a vector of bytes

Steps:

1. $X \leftarrow \{\}$
2. Do
    a. $X \leftarrow \{x \bmod 256\} \,||\, X$

b. $x \leftarrow x \; div \; 256$

While $x \neq 0$

$integer(X)$ Two's compliment integer represented by a byte vector.

Inputs:

X, a vector of bytes

Outputs:

x, an unconstrainted integer

Steps:

1. $y \leftarrow 0$
2. For each i in $0..length(X) - 1$
   a. If $i = 0, y \leftarrow y \times 256 \; + \; X[i]$; else $y \leftarrow y \times 256 \; + \; X[i] \; mod \; 256$

$padding(x)$ Vector of x bytes containing zeros.

Inputs:

x, a constrained integer

Outputs:

X, a vector of bytes

Steps:

1. $X \leftarrow \{ 0, 0, 0, \ldots, 0 \}$, where the number of elements is determined by x.

str(y) Representation an unconstrained integer as a vector of integer values.

Prerequisites:

W, an arbitrary length vector of radices supplied to IFX.Initialize

Input:

y, an unconstrained integer

Output:

Y, a vector of $length(W)$ integers such that $0 \; \leq \; Y[i] \; < \; W[i]$ for all i in $0..length(W)$

Steps:

1. $Y \leftarrow \{\ldots\}, where \; the \; length \; of \; Y \; is \; the \; same \; as \; the \; length \; of \; W$
2. For i in $length(W) - 1..0$

a. $Y[i] \leftarrow y \; mod \; W[i]$
b. $y \leftarrow y \; div \; W[i]$

4. IFX Algorithms

$IFX.Initialize(W)$ Initialize the IFX algorithms with a vector of radices.

Inputs:

W, an arbitrary length vector of radices, where each radix is an integer greater than or equal to two

Outputs:

w, an unconstrained integer

u and v, unconstrained integers such that $u \times v = w$

Steps:

1. $w \leftarrow product(W)$
2. $G \leftarrow descending(factors(W))$
3. $u \leftarrow 1; v \leftarrow 1$
4. For each element g of G
   a. If $u \times g \leq \lfloor \sqrt{w} \rfloor, u \leftarrow u \times g$; else $v \leftarrow v \times g$

$IFX.Encrypt(K, T, X)$ Encrypt a plaintext vector using the supplied key and tweak.

Prerequisites:

W, an arbitrary length vector of radices supplied to IFX.Initialize

u and v, unconstrained integers such that $u \times v = product(W)$

Inputs:

K, an AES key

T, a vector of bytes of arbitrary length

X, a plaintext vector of $length(W)$ integers such that $0 \leq X[i] < W[i]$ for all i in $0..length(W)$

Outputs:

Y, a ciphertext vector of $length(W)$ integers such that $0 \leq Y[i] < W[i]$ for all i in $0..length(W)$

Steps:

1. $x \leftarrow num(X)$
2. $a \leftarrow x \; div \; v$

3. $b \leftarrow x \bmod v$
4. $r \leftarrow rounds(u, v)$
5. $R \leftarrow bytes(r)$
6. $U \leftarrow bytes(u)$
7. $V \leftarrow bytes(v)$
8. $s \leftarrow length(T) + length(U) + length(V) + length(R)$
9. $S \leftarrow bytes(s)$
10. $O \leftarrow R \,||\, S \,||\, padding(-length(R) - length(S) - length(T) - length(U) - length(V) \bmod 16) \,||\, T \,||\, U \,||\, V$
11. $I \leftarrow \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$
12. $P \leftarrow ciph(K, I, O)$
13. $P \leftarrow P[length(P) - 16..length(P) - 1]$
14. For i in $0..r - 1$
    a. If i is even, $d \leftarrow u$; else $d \leftarrow v$
    b. $I \leftarrow bytes(i)$
    c. $B \leftarrow bytes(b)$
    d. $Q \leftarrow I \,||\, padding(-length(I) - length(B) \bmod 16) \,||\, B$
    e. $F \leftarrow ciph(K, P, Q)$
    f. $f \leftarrow num(F)$
    g. $c \leftarrow (a + f) \bmod d$
    h. $a \leftarrow b$
    i. $b \leftarrow c$
15. $y \leftarrow a \times v + b$
16. $Y \leftarrow str(y)$

$IFX.Decrypt(K, T, Y)$ Encrypt a plaintext vector using the supplied key and tweak.

Prerequisites:

W, an arbitrary length vector of radices supplied to IFX.Initialize

u and v, unconstrained integers such that $u \times v = product(W)$

Inputs:

K, an AES key

T, a vector of bytes of arbitrary length

Y, a plaintext vector of $length(W)$ integers such that $0 \leq Y[i] < W[i]$ for all i in $0..length(W)$

Outputs:

X, a ciphertext vector of $length(W)$ integers such that $0 \leq X[i] < W[i]$ for all i in $0..length(W)$

Steps:

1. $y \leftarrow num(Y)$

2. $a \leftarrow y \; div \; v$
3. $b \leftarrow y \; mod \; v$
4. $r \leftarrow rounds(u, v)$
5. $R \leftarrow bytes(r)$
6. $U \leftarrow bytes(u)$
7. $V \leftarrow bytes(v)$
8. $s \leftarrow length(T) + length(U) + length(V) + length(R)$
9. $S \leftarrow bytes(s)$
10. $O \leftarrow R \;||\; S \;||\; padding(-length(R) - length(S) - length(T) - length(U) - length(V) \; mod \; 16) \;||\; T \;||\; U \;||\; V$
11. $I \leftarrow \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$
12. $P \leftarrow ciph(K, I, O)$
13. $P \leftarrow P[length(P) - 16..length(P) - 1]$
14. For i in $r - 1..0$
    a. If i is even, $d \leftarrow u$; else $d \leftarrow v$
    b. $c \leftarrow b$
    c. $b \leftarrow a$
    d. $I \leftarrow bytes(i)$
    e. $B \leftarrow bytes(b)$
    f. $Q \leftarrow I \;||\; padding(-length(I) - length(B) \; mod \; 16) \;||\; B$
    g. $F \leftarrow ciph(K, P, Q)$
    h. $f \leftarrow num(F)$
    i. $a \leftarrow (c - f) \; mod \; d$
15. $x \leftarrow a \times v \; + \; b$
16. $X \leftarrow str(x)$

5. Determining the Number of Feistel Rounds

FFX suggests at a minimum that $rnds(n) \; \geq \; 4n/split(n)$, where n is the number of uniform elements in the plaintext. IFX uses non-uniform elements, so we cannot simply derive n by counting the number of elements. IFX also uses an algebraic function to split the plaintext, rather than simply counting elements on either side as FFX does, so we cannot simply count $split(n)$.

In the context of IFX, we define n to be the log base 2 of the unconstrained integer w, which is the product of the radices in W. That is, we consider the input as a bit vector, and consider n to be the number of bits. Likewise, in IFX we consider $split(n)$ to be the log base 2 of the smaller of the two component radices, u and v. That is, the number of bits in the smaller portion of the input.

Rather than compute the true log base 2 of the unconstrained and possibly very large integers u, v and w, we use the binary logarithm as a near approximation. Thus, following the guidance in FFX, the minimum number of rounds for IFX is $4 \times lb(w) \; / \; lb(min(u, v))$.

Except in cases where W contains a radix with a large prime factor, IFX does an efficient job of splitting the information in W into nearly equal parts. In many cases, IFX will operate on a balanced or nearly balanced split for the Feistel network, which requires the minimal number of rounds.

Note that the calculation of $rounds(u, v)$ produces only a minimum value for the number of Feistel rounds. The actual number of rounds required to make the effort to break the scheme comparable to an exhaustive key search on the underlying block cipher is to be determined in further analysis of the IFX algorithm.

Appendix A

Alternative implementations: use a precomputed table, find prime factors for radices individually. This algorithm uses a modified sieve of Eratosthenes to collect prime factors. While this is feasible in the general case for small values of x, it is impractical for large values of x.

$primes(x)$ Generate a vector of x elements where the nth element contains a vector of the prime factors of n.

Input:

x, a constrained integer greater than 1

Output:

E, a vector of x elements where the nth element contains a vector of the prime factors of n

Steps:

1. $E \leftarrow \{\,\}_0, \{\,\}_1, \{\,\}_2, \ldots, \{\,\}_x\}$
2. $r \leftarrow \lceil \sqrt{x} \rceil$
3. For i in $2..r$
    a. If $E[i] \neq \{\,\}$, next i
    b. $j \leftarrow i$
    c. While $j \times i \leq x$
        i. $E[j \times i] \leftarrow E[j \times i] \,||\, \{i\}$
        ii. $j \leftarrow j + 1$
4. For i in $2..x$
    a. If $E[i] = \{\,\}$
        i. $E[i] = \{i\}$
    b. Else
        i. $j \leftarrow i/product(E[i])$
        ii. If $j > 1, E[i] \leftarrow E[i] \,||\, E[j]$