The attached DRAFT document (provided here for historical purposes) has been superseded by the following publication:

Publication Number: **NIST Special Publication (SP) 800-38G**

Title: **Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption**

Publication Date: **3/29/2016**

- Final Publication: http://dx.doi.org/10.6028/NIST.SP.800-38G (which links to http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38G.pdf).
- Related Information on CSRC: http://csrc.nist.gov/publications/PubsSPs.html#800-38G and http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html
- Information on other NIST cybersecurity publications and programs can be found at: http://csrc.nist.gov/

The following information was posted with the attached DRAFT document:

July 8, 2013

SP 800-38 G

DRAFT Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption

NIST is pleased to announce that Draft NIST Special Publication 800-38G, *Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption*, is available for public comment. Format-preserving encryption (FPE) has emerged as a useful cryptographic tool, whose applications include financial-information security, data sanitization, and transparent encryption of fields in legacy databases.

Three methods are specified in this publication: FF1, FF2, and FF3. Each is a format-preserving, Feistel-based mode of operation of the AES block cipher. FF1 was submitted to NIST by Bellare, Rogaway and Spies under the name FFX[Radix]; FF2 was submitted to NIST by Vance under the name VAES3; and FF3 is the main component of the BPS mechanism that was submitted to NIST by Brier, Peyrin, and Stern. The submission documents are available at http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html.

Public comments on Draft NIST Special Publication 800-38G may be submitted to EncryptionModes @nist.gov until **September 3, 2013**.

NIST Special Publication 800-38G
Draft

Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption

Morris Dworkin

C O M P U T E R S E C U R I T Y

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-38G
Draft

Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption

Morris Dworkin
Computer Security Division
Information Technology Laboratory

July, 2013

U.S. Department of Commerce
Cameron F. Kerry, Acting Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

Authority

This publication has been developed by NIST to further its statutory responsibilities under the Federal Information Security Management Act (FISMA), Public Law (P.L.) 107-347. NIST is responsible for developing information security standards and guidelines, including minimum requirements for Federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate Federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130, Section 8b(3), *Securing Agency Information Systems*, as analyzed in Circular A-130, Appendix IV: *Analysis of Key Sections*. Supplemental information is provided in Circular A-130, Appendix III, *Security of Federal Automated Information Resources*.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on Federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other Federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-38G
Natl. Inst. Stand. Technol. Spec. Publ. 800-38G, NNN pages (Month YYYY)
http://dx.doi.org/10.6028/NIST.SP.XXX
CODEN: NSPUE2

Public comment period: *Month Day, YYYY* through *Month Day, YYYY*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: EncryptionModes@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Abstract

This Recommendation specifies three methods for format-preserving encryption, called FF1, FF2, and FF3. Each of these methods is a mode of operation of the AES algorithm, which is used to construct a round function within the Feistel structure for encryption.

Keywords

block cipher; computer security; confidentiality; cryptography; encryption; Feistel structure; format-preserving encryption; information security; mode of operation.

Acknowledgements

Much of the text in this publication is adapted from four specification documents that were submitted to NIST: Mihir Bellare, Phil Rogaway, and Terence Spies submitted the FFX framework and FFX[Radix] in [1] and [2]; Eric Brier, Thomas Peyrin, and Jacques Stern submitted BPS in [3], and Joachim Vance submitted VAES3 in [13].

In addition to the designers, the author wishes to thank his colleagues who reviewed drafts of this publication and contributed to its development, especially Elaine Barker, Meltem Somnez Turan, Lily Chen, John Kelsey, Ray Perlner, Allen Roginsky, Lawrence Bassham, Tim Hall, and Sharon Keller. The author also gratefully acknowledges the comments from the public and private sectors to improve the quality of this publication.

TABLE OF CONTENTS

TABLE OF FIGURES

1 Purpose

This publication is the seventh part in a series of Recommendations regarding the modes of operation of block cipher algorithms. The purpose of this part is to provide approved methods for format-preserving encryption (FPE).

2 Authority

This publication has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347.

NIST is responsible for developing standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems.

This recommendation has been prepared for use by Federal agencies. It may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright. (Attribution would be appreciated by NIST.)

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official.

Conformance testing for implementations of this Recommendation will be conducted within the framework of the Cryptographic Algorithm Validation Program (CAVP) and the Cryptographic Module Validation Program (CMVP). The requirements of this Recommendation are indicated by the word “shall.” Some of these requirements may be out-of-scope for CAVP or CMVP validation testing, and thus are the responsibility of entities using, implementing, installing, or configuring applications that incorporate this Recommendation.

3 Introduction

This publication specifies three block cipher modes of operation—or, simply, modes—for format-preserving encryption (FPE).

Previously approved encryption modes transform bit strings—sequences of `0`s and `1`s—into other bit strings, but these modes are not directly applicable to decimal strings, like Social Security numbers (SSNs) or credit card numbers (CCNs), or to other data formats.

Given any finite set of symbols, like the decimal numerals, FPE transforms data that is formatted as a sequence of the symbols in such a way that the encrypted form of the data has the same format and length as the original data. Thus, an FPE-encrypted SSN also appears to be an SSN.

Consequently, FPE facilitates the targeting of encryption to sensitive information, as well as the retrofitting of encryption technology to legacy applications, where a conventional encryption mode might not be feasible because it would disrupt data fields/pathways. FPE has emerged as a useful cryptographic tool, whose applications include financial-information security, data sanitization, and transparent encryption of fields in legacy databases.

The three FPE modes specified in this publication are abbreviated FF1, FF2, and FF3, to indicate that they are format-preserving, Feistel-based encryption modes. They were submitted to NIST under the names FFX[Radix], VAES3, and BPS-BC, in [2], [13], and [3], respectively. Each mode fits within a larger framework, called FFX, for constructing FPE mechanisms, that was submitted to NIST in [1]. The "X" indicates the flexibility to instantiate the framework with different parameter sets, as well as FFX's evolution from the FFSEM specification earlier submitted to NIST in [12].

The FFX framework itself is not specified in this publication; in fact, FF1, FF2, and FF3 are not presented explicitly as instantiations of FFX parameter sets, but rather as separate algorithms, in order to simplify the individual specifications.

FF1, FF2, and FF3 each employ the Feistel structure of encryption—see Sec. 5.4—which also underlies the Data Encryption Algorithm [8]. At the core of FF1, FF2, and FF3 are somewhat different Feistel round functions that are derived from an approved block cipher with 128-bit blocks, i.e., the AES algorithm[1] [6].

In addition to the formatted data for which the modes provide confidentiality, each mode also takes a "tweak" input, i.e., additional information that is not necessarily secret. This functionality can be especially important for FPE modes, because the number of possible values for the confidential data is often relatively small. In particular, the encrypted form of the data can vary in different instances when the tweak inputs are different. See Appendix C for further discussion of tweaks.

The three modes offer somewhat similar performance profiles. FF1 supports the greatest range of lengths for the protected, formatted data and the tweak. FF2 generates a subkey for the block cipher in the Feistel round function, which can help protect the original key from side-channel analysis. FF3 offers the lowest round count, eight, compared to ten for FF1 and FF2, and is the least flexible in the tweaks that it supports.

The BPS mechanism for FPE—named after its designers, Brier, Peyrin, and Stern—was submitted to NIST in [3]; FF3 is essentially equivalent to the BPS-BC component of BPS, instantiated with a 128-bit block cipher. The full BPS mode—in particular, its chaining mechanism for longer input strings—is not approved in this Recommendation.

[1] The term "algorithm" here indicates a high-level cryptographic technique that may encompass more than one computational procedure; for example, an "encryption algorithm" like the AES algorithm has transformations for both encryption and decryption. This publication also contains ten numbered algorithms in the original sense of the word, i.e., as a list of instructions for executing a single computational procedure.

4 Definitions and Notation

4.1 Definitions

alphabet	A finite set of two or more symbols.
approved	FIPS-approved or NIST-recommended: an algorithm or technique that is either 1) specified in a FIPS or a NIST Recommendation, or 2) adopted in a FIPS or a NIST Recommendation.
base	The number of characters in a given alphabet; represented as *radix* in this publication.
bit	A binary digit: 0 or 1.
bit string	A finite, ordered sequence of bits.
block	For a given block cipher, a bit string whose length is the block size of the block cipher.
block cipher	A parameterized family of permutations on bit strings of a fixed length; the parameter that determines the permutation is a bit string called the key.
block cipher mode of operation	An algorithm for the cryptographic transformation of data that is based on a block cipher.
block size	For a given block cipher and key, the fixed length of the input (or output) bit strings.
block string	A bit string that can be represented as the concatenation of a finite, ordered sequence of blocks.
byte	A string of eight bits.
byte string	A bit string that can be represented as the concatenation of a finite, ordered sequence of bytes.
character	A symbol in an alphabet.
character string	A finite, ordered sequence of characters.
ciphertext	The numeral string that is the encrypted form of a plaintext.

decryption function	For a given block cipher and key, the function of an FPE mode that takes a ciphertext numeral string and a tweak as input and returns the corresponding plaintext numeral string as output.
designated cipher function	For a given block cipher and key, the choice of either the forward transformation or the inverse transformation.
encryption function	For a given block cipher and key, the function of an FPE mode that takes a plaintext numeral string and a tweak as input and returns a ciphertext numeral string as output.
exclusive-OR	The bitwise addition, modulo 2, of two bit strings of equal length.
Feistel structure	A framework for constructing an encryption mode. The framework consists of several iterations, called rounds, 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.
forward transformation	For a given block cipher, the permutation of blocks that is determined by the choice of a key.
inverse transformation	For a given block cipher, the inverse of the permutation of blocks that is determined by the choice of a key.
key	For a given block cipher, the secret bit string that parameterizes the permutations.
mode	See “block cipher mode of operation.”
numeral	For a given base, a nonnegative integer less than the base.
numeral string	For a given base, a character string that is comprised of numerals for the base.
plaintext	A numeral string whose confidentiality is protected by an FPE mode.
prerequisite	A required input to an algorithm that has been established prior to the invocation of the algorithm.
shall	Is required to. Requirements apply to conforming implementations.
should	Is recommended to.
tweak	The input parameter to the encryption and decryption functions whose confidentiality is not protected by the mode.

4.2 Acronyms

AES	Advanced Encryption Standard.
CAVP	Cryptographic Algorithm Validation Program.
CCN	credit card number.
CMVP	Cryptographic Module Validation Program.
FIPS	Federal Information Processing Standard.
FISMA	Federal Information Security Management Act.
FPE	format-preserving encryption.
IETF	Internet Engineering Task Force.
ITL	Information Technology Laboratory.
NIST	National Institute of Standards and Technology.
PRF	pseudorandom function.
RFC	Request For Comment.
SSN	Social Security number.

4.3 Operations and Functions

$\text{BYTELEN}(T)$	The number of bytes in a byte string, T.
$\text{CIPH}_K(X)$	The output of the designated cipher function of the block cipher under the key K applied to the block X.
$\text{LEN}(X)$	The length of the character string X.
$\text{LOG}_b(x)$	The base b logarithm of the real number $x \neq 0$.
$\text{NUM}_{radix}(X)$	The number that the numeral string X represents in base *radix*, with the most significant character first.
$\text{PRF}(X)$	The output of the function PRF applied to the block X with the

	designated cipher function of the block cipher under the key K.
$\mathrm{REV}(X)$	Given a bit string, X, the string that consists of the bits of X in reverse order.
$\mathrm{STR}^{m}_{radix}(x)$	Given a nonnegative integer x less than $radix^m$, the representation of x as a string of m characters in base $radix$, with the most significant character first.
$\lfloor x \rfloor$	The greatest integer that does not exceed the real number x.
$\lceil x \rceil$	The least integer that is not less than the real number x.
$[x]^s$	Given a nonnegative integer x less than 256^s, the representation of x as a string of s bytes.
$[i..j]$	The set of integers between two integers i and j, including i and j.
$x \bmod m$	The nonnegative remainder of the real number x modulo the positive integer m.
$X[i]$	The ith character of the string X.
$X[i \,..\, j]$	The substring of characters of a string X from $X[i]$ to $X[j]$, including $X[i]$ and $X[j]$.
$X \oplus Y$	The bitwise exclusive-OR of bit strings X and Y whose bit lengths are equal.
$X \,\|\| Y$	The concatenation of bit strings X and Y.
$\mathtt{0}^s$	The bit string that consists of s consecutive '0' bits.

4.4 Examples of Basic Operations and Functions

In this publication, the `Courier New` font indicates the two binary bits, `0` and `1`.

Given positive real numbers b and x, the base b logarithm of x, denoted $\mathrm{LOG}_b(x)$, is the unique real number for which $b^{\mathrm{LOG}_b(x)} = x$. For example, $\mathrm{LOG}_2(64) = 6$, $\mathrm{LOG}_2(10) \approx 3.32$, and $\mathrm{LOG}_{13}(169) = 2$.

Given a real number x, the floor function, denoted $\lfloor x \rfloor$, is the greatest integer that does not exceed x. For example, $\lfloor 2.1 \rfloor = 2$, and $\lfloor 4 \rfloor = 4$.

Given a real number x, the ceiling function, denoted $\lceil x \rceil$, is the least integer that is not less than x. For example, $\lceil 2.1 \rceil = 3$, and $\lceil 4 \rceil = 4$.

Given two integers i and j with $i \le j$, the set of integers between i and j, including i and j, is denoted $[i..j]$. For example, [2 .. 5] = {2, 3, 4, 5}.

Given a real number x, and a positive integer m, the remainder of x modulo m, denoted x mod m, is $x - m\lfloor x/m \rfloor$. For example, 20 mod 7 = 6, and -3 mod 7 = 4.

Given a positive integer s, $\mathtt{0}^s$ denotes the string that consists of s '0' bits. For example, $\mathtt{0}^8$ = 00000000.

The concatenation operation on character strings is denoted ||. For example, 001 || 10111 = 00110111, and if the base is 16, for example, 1 3 13 7 || 0 8 10 = 1 3 13 7 0 8 10.

Given bit strings of equal length, the exclusive-OR (XOR) operation, denoted $\oplus$, specifies the addition, modulo 2, of the bits in corresponding bit positions. For example, 10011 $\oplus$ 10101 = 00110.

Given a character string X, the length of X is denoted LEN(X). For example, LEN(%$$) = 3, and LEN(00010) = 5.

Given a byte string X, the length of X in bytes is denoted BYTELEN(X). For example, BYTELEN(1011100110101100) = 2.

Given a character string X, and any index i, i.e., with $1 \le i \le$ LEN(X), the ith character of X is denoted $X[i]$. For a pair of indices (i, j), with $1 \le i \le j \le$ LEN(X), the substring of characters from $X[i]$ to $X[j]$ is denoted $X[i .. j]$. For example, if X = &$*#@@%, then X[2]=$, X[3..5]=*#@.

Given a base, *radix*, and a length, m, there are $radix^m$ distinct numeral strings. Given an integer x such that $0 \le x < radix^m$, the integer-to-string function, denoted $\mathrm{STR}^m_{radix}(x)$, is the representation of x as a numeral string of length m in base *radix*, with the most significant character first, i.e., on the left. For example, $\mathrm{STR}^4_{12}(559)$ is the string of four numerals in base 12 that represents 559, namely, 0 3 10 7. An algorithm for computing $\mathrm{STR}^m_{radix}(x)$ is given in Sec. 5.5.

A separate notation is given for the conversion of integers to byte strings: $[x]^s = \mathrm{STR}^{8s}_2(x)$. For example, $[1]^1$ = 00000001.

Given a (non-empty) numeral string X in base *radix* of length m, the string-to-integer function, denoted $\mathrm{NUM}_{radix}(X)$, is the integer x such that $\mathrm{STR}^m_{radix}(x) = X$. In other words, $\mathrm{NUM}_{radix}(X)$ is the non-negative integer less than $radix^{\mathrm{LEN}(X)}$ whose most-significant-character-first representation in base *radix* is X. For example, NUM_2(00011010) = 26. An algorithm for computing $\mathrm{NUM}_{radix}(X)$ is given in Sec. 5.5.

Given a bit string, X, the string REV(X) is the concatenation of the bits of X in reverse order. For example, REV(00001) = 10000.

5 Preliminaries

5.1 Character Strings

A finite set of two or more symbols is called an alphabet, and the elements of an alphabet are called characters. A character string is a finite sequence of characters from an alphabet. Individual characters may repeat in the string. Thus, for example, given the 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},

the words "`hello`" and "`cannot`" are character strings, but "`Hello`" and "`can't`" are not, because the symbols "`H`" and "`'`" are not in the alphabet.

The number of characters in an alphabet is called the base, denoted *radix*, so that $radix \geq 2$ by definition. For a given base, a numeral is a nonnegative integer that is less than the base.

The base is a parameter of FF1, FF2, and FF3, and their specifications assume that the alphabet is the set of numerals for the base:

{0, 1, …, *radix*-1}.

Therefore, to apply FF1, FF2, or FF3 to character strings from an arbitrary alphabet, the string must first be converted, via a one-to-one correspondence, into a string of numerals for the base of the alphabet. For example, the natural conversion from lower-case English letters to base 26 numerals is

a→0, b→1, c→2, … x→23, y→24, z→25,

although other one-to-one correspondences are possible. The choice and implementation of conversion methods from particular alphabets to their appropriate alphabets of numerals is outside the scope of this publication.

The interpretation of a numeral string as a number depends on the base and the ordering convention; algorithms for functions that convert numeral strings to numbers and vice versa are given in Sec. 5.5. Individual numerals, and other numbers in the specifications, are represented in base 10.

The basic operations and functions on strings are defined with examples in Sec. 4.4.

5.2 Underlying Block Cipher and Key

The encryption and decryption functions of FF1, FF2, and FF3 feature a block cipher as the main component; thus, each of these FPE mechanisms is a mode of operation (mode, for short) of the block cipher.

For any given key, *K*, the underlying block cipher of the mode is a permutation, i.e., an invertible transformation on bit strings of a fixed length; the fixed-length bit strings are called blocks, and the length of a block is called the block size. For an FPE mode, as part of the choice of the underlying block cipher with the key, either the forward transformation or the inverse transformation[2] is specified as the designated cipher function, denoted CIPH_K. The inverse of CIPH_K is not needed for the modes that are specified in this publication.

For each of the three modes, the underlying block cipher shall be approved, and the block size shall be 128 bits. Currently, the AES block cipher, with key lengths of 128, 192, or 256 bits, is the only block cipher that fits this profile.

The choice of the key length affects the security of the FPE modes, e.g., against brute-force search, and also affects the details of the implementation of the AES algorithm. Otherwise, the key length does not affect the implementation of FF1, FF2, and FF3, and the choice of the key length is not explicitly indicated in their specifications Methods for generating cryptographic keys are discussed in [9]; the goal is to select the keys uniformly at random, i.e., for each possible key to occur with equal probability.

The key shall be kept secret, i.e., disclosed only to parties that are authorized to know the protected information. Compliance with this requirement is the responsibility of the entities using, implementing, installing, or configuring applications that incorporate this Recommendation. The management of cryptographic keys is outside the scope of this publication.

5.3 Encryption and Decryption Functions

For a given key, denoted *K*, for the designated block cipher, FF1, FF2, and FF3 each consist of two related functions: encryption and decryption. The encryption function takes as input a numeral string called the plaintext, denoted *X*, and an additional parameter, called the tweak, denoted *T*; the function returns a numeral string called the ciphertext, denoted *Y*, with the same length as *X*. Similarly, the inputs to the decryption function are a numeral string *X* and a tweak *T*; the output is a numeral string *Y* of the same length as *X*.

For FF1, the encryption function and the decryption function are denoted FF1.Encrypt(*K*, *T*, *X*) and FF1.Decrypt(*K*, *T*, *X*), with analogous notation for FF2 and FF3.

For a given tweak, the decryption function is the inverse of the encryption function, so that

$$\text{FF1.Decrypt}(K, T, \text{FF1.Encrypt}(K, T, X)) = X.$$

Thus, when a ciphertext is the input to the decryption function, along with the same tweak that was used to generate the ciphertext, the output is the corresponding plaintext.

[2] The forward transformation and the inverse transformations are sometimes referred to as the "enrypt" and "decrypt" functions, respectively, of the block cipher; however, in this publication, those terms are reserved for functions of the FPE modes.

The tweak may be any information that can be associated to the input numeral string and does not need to be kept secret. Although implementations may fix the value of the tweak, the use of variable tweaks is strongly recommended as a security enhancement; see Appendix C. In FF1 and FF3, tweaks are bit strings; in FF2, tweaks may also be character strings in a different base. The tweak lengths that can be supported differ for each of the three modes; see the individual specifications in Sec. 6.

The key, K, is indicated in the notation as an input for the encryption and decryption functions; however, in the specifications in this publication, the key is listed as a prerequisite, i.e., an input that is usually established prior to the invocation of the function.[3] Several other prerequisites are omitted from the above notation, such as the underlying block cipher, the designation of CIPH_K, and the base for the numeral strings.

5.4 Feistel Structure

FFX schemes, including FF1, FF2, and FF3, are based on the Feistel structure for encryption. The Feistel structure consists of several iterations, called rounds, of a reversible transformation, which consists of three elements: 1) the data is split into two parts; 2) a keyed function, F_K, called the round function, is applied to one part of the data in order to modify the other part of the data; and 3) the roles of the two parts are swapped for the next round. The structure is illustrated in Figure 1 below. Four rounds are shown in this example, but ten rounds are actually specified for FF1 and FF2, and eight rounds for FF3.

The two parts of the input numeral string, denoted A_0 and B_0, consist of u and v characters, respectively; the total number of characters, $u+v$, is denoted n. The rounds are indexed from 0 to 3 in the figure. During round i, the round function F_K is applied to B_i, with the tweak T, the round number i, and the length n, as additional inputs, and the result is used to modify A_i, via modular addition[4], indicated by +, on the numbers that the strings represent[5]. The string that represents the resulting number is named with a temporary variable, C_i. The roles of the 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}.

The rectangles containing the two parts of the data in each round are different sizes to illustrate that, if n is odd, then u cannot equal v. In order to accommodate 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.

The Feistel structure for decryption is almost identical; the only changes are 1) modular addition is replaced by modular subtraction, and 2) the order of the round indices is reversed. For example, in Figure 1, the arguments to F_K would begin at n, T, 3 at the top, and end at n, T, 0 at the bottom.

[3] The distinction doesn't affect the execution of the function: all inputs are required, independent of when they were established or provided to the implementation.
[4] In principle, the addition operation can be any other reversible operation on strings that preserves their length; for example, the FFX specification [1] supports an option for characterwise addition.
[5] The ordering convention for the interpretation of strings is different for FF3 than for FF1 and FF2.

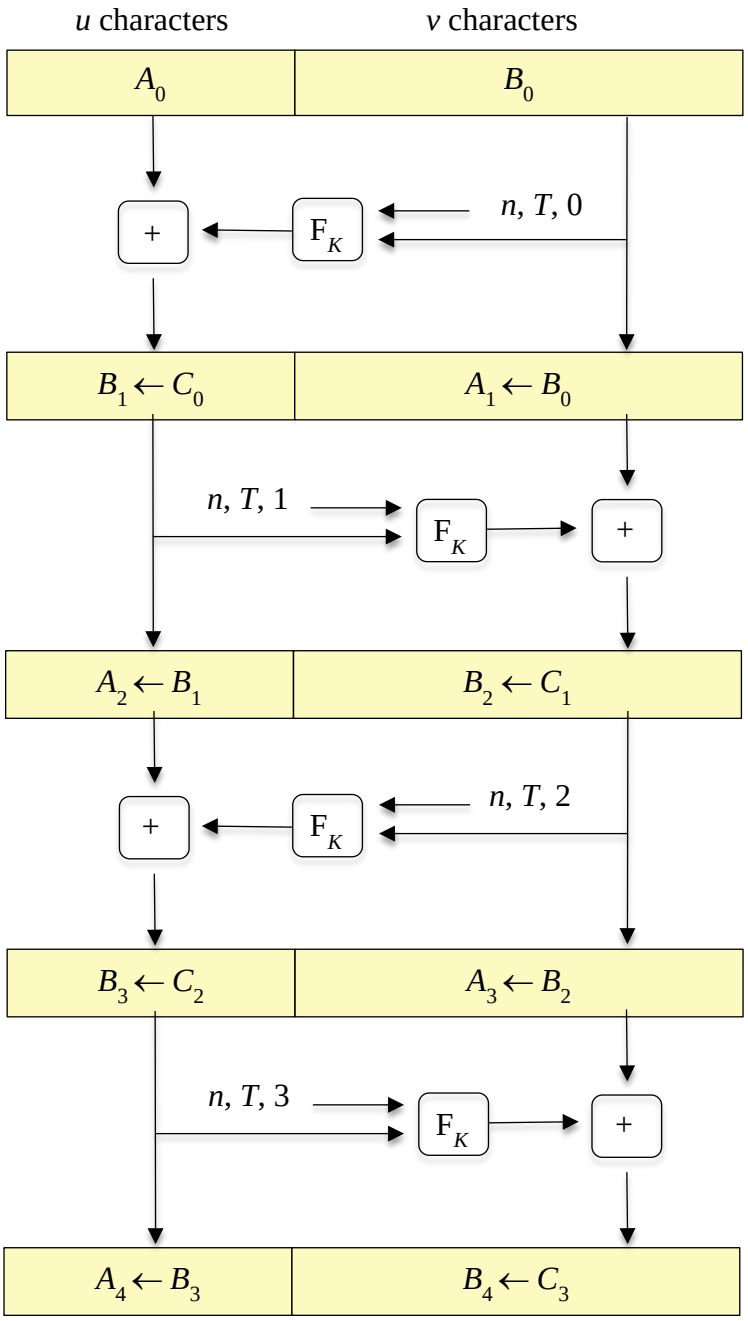

Figure 1: Illustration of the Feistel Structure

5.5 Component Functions

This section gives algorithms for the component functions that are called in the specifications of FF1, FF2, and FF3. The conversion functions $\text{NUM}_{radix}(X)$ and $\text{STR}^{m}_{radix}(x)$—defined in Sec. 4.4 with examples—are specified in Algorithms 1 and 2. These functions support the ordering convention for the numeral strings in FF1 and FF2, namely, that the first (i.e., left-most) numeral of the string is the most-significant numeral.

The function REV(X))—defined in Sec. 4.4 with an example—is specified in Algorithm 3. It is used to adapt $\text{NUM}_{radix}(X)$ and $\text{STR}^{m}_{radix}(x)$ to the opposite ordering convention, for the numeral strings in FF3.

The PRF function, specified in Algorithm 4, essentially invokes the Cipher Block Chaining encryption mode [7] on the input bit string and returns the final block of the ciphertext; this function is the pseudorandom core of the Feistel round function for FF1.Encrypt and FF1.Decrypt.

Algorithm 1: $\text{NUM}_{radix}(X)$

Prerequisite:
Base, *radix*.

Input:
Numeral string, X.

Output:
Number, x.

Steps:
1. Let $x = 0$.
2. For i from 1 to LEN(X), let $x = x \cdot radix + X[i]$.
3. Return x.

Algorithm 2: $\text{STR}^{m}_{radix}(x)$

Prerequisites:
Base, *radix*;
String length, m.

Input:
Number, x, such that $x < radix^{m}$.

Output:
Numeral string, X.

Steps:
1. For i from 1 to m:
   i. $X[m+1-i] = x \bmod radix$;
   ii. $x = \lfloor x / radix \rfloor$.
2. Return X.

Algorithm 3: REV (*X*)

Input:
Character string, *X*.

Output:
Character string, *Y*.

Steps:
1. For *i* from 1 to LEN(*X*) let *Y*[*i*]=*X*[LEN(*X*)+1-*i*].
2. Return *Y*[0 .. LEN (*X*)-1].

Algorithm 4: PRF(*X*)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, *K*, for the block cipher;

Input:
Nonempty bit string, *X*, such that LEN(*X*) is a multiple of 128.

Output:
Block, *Y*.

Steps:
1. Let m = LEN(*X*)/128.
2. Partition *X* into *m* blocks $X_1,\ldots, X_m$, so that $X = X_1 \| \ldots \| X_m$ and LEN(X_i) = 128 for all *i* from 1 to *m*.
3. Let $Y_0 = \mathtt{0}^{128}$, and for *j* from 1 to *m* let Y_j = CIPH$_K$($Y_{j-1} \oplus X_j$).
4. Return Y_m.

6 Mode Specifications

The specifications of FF1, FF2, and FF3 are presented in this section, organized into prerequisites, inputs, outputs, steps, and descriptions of the steps. In addition to the key and designated cipher function, each mode has prerequisites for the choices of the base, *radix*, and of the range of lengths, [*minlen* .. *maxlen*], for the numeral string inputs that the implementation supports. FF1 and FF2 also have a prerequisite for the choice of the maximum tweak length, *maxTlen*, that the implementation supports, and FF2 has an additional parameter, *tweakradix*, for the choice of the base for tweak strings. For all of these parameters, the choices that are allowed for the individual modes are described within the prerequisites.

The parameter choices may affect interoperability. The behavior of an implementation when presented with incorrect inputs is out of the scope of this Recommendation.

6.1 FF1

The specifications for the FF1.Encrypt and FF1.Decrypt functions are given in Algorithms 5 and 6 below. The tweak, *T*, is optional in that it may be the empty string, with byte length t=0.

The parameters *radix*, *minlen*, *maxlen*, and *maxTlen* in FF1.Encrypt and FF1.Decrypt shall meet the following requirements:

- $radix \in [2 .. 2^{16}]$;
- $radix^{minlen} \geq 100$;
- $minlen \geq 2$;
- $maxlen < 2^{32}$; and
- $maxTlen < 2^{32}$.

Algorithm 5: FF1.Encrypt(*K*, *T*, *X*)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, *K*, for the block cipher;
Base, *radix*, for the character alphabet;
Range of supported message lengths, [*minlen* .. *maxlen*];
Maximum byte length for tweaks, *maxTlen*.

Inputs:

Character string, *X*, in base *radix* of length *n* such that $n \in [minlen .. maxlen]$;

Tweak *T*, a byte string of byte length *t*, such that $t \in [0 .. maxTlen]$.

Output:
Character string, *Y*, such that $\text{LEN}(Y) = n$.

Steps:
1. Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2. Let $A = X[1 .. u]$; $B = X[u + 1 .. n]$.
3. Let $b = \lceil \lceil v \, \text{LOG}_2(radix) \rceil / 8 \rceil$; $d = 4\lceil b/4 \rceil + 4$
4. Let $P = [1]^1 \| [2]^1 \| [1]^1 \| [radix]^3 \| [10]^1 \| [u \bmod 256]^1 \| [n]^4 \| [t]^4$.
5. For i from 0 to 9:
   i. Let $Q = T \| [0]^{(-t-b-1) \bmod 16} \| [i]^1 \| [\text{NUM}_{radix}(B)]^b$.
   ii. Let $R = \text{PRF}(P \| Q)$.
   iii. Let *S* be the first *d* bytes of the following string of $\lceil d/16 \rceil$ blocks:
   $R \| \text{CIPH}_K(R \oplus [1]^{16}) \| \text{CIPH}_K(R \oplus [2]^{16}) \| \ldots \| \text{CIPH}_K(R \oplus [\lceil d/16 \rceil - 1]^{16})$.
   iv. Let $y = \text{NUM}_2(S)$.
   v. If *i* is even, let $m = u$; else, let $m = v$.
   vi. Let $c = (\text{NUM}_{radix}(A) + y) \bmod radix^m$.
   vii. Let $C = \text{STR}^m_{radix}(c)$.

viii. Let $A = B$.
ix. Let $B = C$.

6. Return $A \| B$.

Description

The "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 5i and 5iii, are defined in Step 3.[6] A fixed block, P, used as the initial block for the invocation of the function PRF in Step 5ii, is defined in Step 4. An iteration loop for the ten Feistel rounds of FF1 is initiated in Step 5, 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 5i. The function PRF is applied to the concatenation of P and Q in Step 5ii, 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 5iii. (In Figure 1**Error! Reference source not found.**, S corresponds to the output of F_K.) In Steps 5iv to 5vii, 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 5iv, S is converted to a number, y. In Step 5v, the length, m, of A for this Feistel round is determined. In Step 5vi, y is added to the number represented by the substring A and the result is reduced modulo the mth power of *radix*, yielding a number, c, which is converted to a numeral string in Step 5vii. In Steps 5viii and 5ix, 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 6.

Algorithm 6: FF1.Decrypt(K, T, X)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, K, for the block cipher;
Base, *radix*, for the character alphabet;
Range of supported message lengths, [*minlen* .. *maxlen*];
Maximum byte length for tweaks, *maxTlen*.

Inputs:

Numeral string, X, in base *radix* of length n such that $n \in$ [*minlen* .. *maxlen*];

Tweak byte string, T, of byte length t such that $t \in$ [0 .. *maxTlen*].

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

[6] When B is encoded as a byte string in Step 5i, b is the number of bytes in the encoding. The definition of d ensures that the output of the Feistel round function is at least 4 bytes longer than this encoding of B, which minimizes any bias in the modular reduction in Step 5vi.

Steps:

1. Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2. Let $A = X[1 .. u]$; $B = X[u + 1 .. n]$.
3. Let $b = \lceil \lceil v \cdot \text{LOG}_2(radix) \rceil / 8 \rceil$; $d = 4\lceil b/4 \rceil + 4$
4. Let $P = [1]^1 \| [2]^1 \| [1]^1 \| [radix]^3 \| [10]^1 [u \bmod 256]^1 \| [n]^4 \| [t]^4$.
5. For i from 9 to 0:
   i. Let $Q = T \| [0]^{(-t-b-1) \bmod 16} \| [i]^1 \| [\text{NUM}_{radix}(B)]^b$.
   ii. Let $R = \text{PRF}(P \| Q)$.
   iii. Let S be the first d bytes of the following string of $\lceil d/16 \rceil$ blocks:
   $R \| \text{CIPH}_K(R \oplus [1]^{16}) \| \text{CIPH}_K(R \oplus [2]^{16}) \| \ldots \| \text{CIPH}_K(R \oplus [\lceil d/16 \rceil - 1]^{16})$.
   iv. Let $y = \text{NUM}_2(S)$.
   v. If i is even, let $m = u$; else, let $m = v$.
   vi. Let $c = (\text{NUM}_{radix}(A) - y) \bmod radix^m$.
   vii. Let $C = \text{STR}^m_{radix}(c)$.
   viii. Let $A = B$.
   ix. Let $B = C$.
6. Return $A \| B$.

Description:
The FF1.Decrypt algorithm is almost identical to the FF1.Encrypt algorithm; the only differences are in Step 5, where the order of the indices is reversed, and in Step 5vi, where modular addition is replaced by modular subtraction.

6.2 FF2

The specifications for the FF2.Encrypt and FF2.Decrypt functions are given in Algorithms 7 and 8 below. The tweak, *T*, is optional in that it may be the empty string, with byte length t=0.

The parameters *radix*, *tweakradix*, *minlen*, *maxlen*, and *maxTlen* in FF2.Encrypt and FF2.Decrypt shall meet the following requirements:

- $radix \in [2 .. 2^8]$;
- $tweakradix \in [2 .. 2^8]$;
- $radix^{minlen} \geq 100$;
- $minlen \geq 2$;
- $maxlen \leq 2\lfloor 120/\text{LOG}_2(radix) \rfloor$ if *radix* is a power of 2;
- $maxlen \leq 2\lfloor 98/\text{LOG}_2(radix) \rfloor$ if *radix* is not a power of 2; and
- $maxTlen < \lfloor 104/\text{LOG}_2(tweakradix) \rfloor$.

Algorithm 7: FF2.Encrypt(*K*, *T*, *X*)

Prerequisites:
Approved, 128-bit block cipher, CIPH;

Key, *K*, for the block cipher;
Base, *radix*, for the character alphabet;
Base, *tweakradix*, for the tweak character alphabet;
Range of supported message lengths, [*minlen* .. *maxlen*]
Maximum supported tweak length, *maxTlen*.

Inputs:
Numeral string, *X*, in base *radix* of length *n* such that $n \in [minlen \,..\, maxlen]$;
Tweak numeral string, *T*, in base *tweakradix* of length *t* such that $t \in [0 \,..\, maxTlen]$.

Output:
Numeral string, *Y*, such that $\text{LEN}(Y) = n$.

Steps:
1. Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2. Let $A = X[1 \,..\, u]$; $B = X[u + 1 \,..\, n]$.
3. If $t>0$, $P = [radix]^1 \,||\, [t]^1 \,||\, [n]^1 \,||\, [\text{NUM}_{tweakradix}(T)]^{13}$; else $P = [radix]^1 \,||\, [0]^1 \,||\, [n]^1 \,||\, [0]^{13}$.
4. Let $J = \text{CIPH}_K(P)$
5. For i from 0 to 9:
   i. Let $Q \leftarrow [i]^1 \,||\, [\text{NUM}_{\text{radix}}(B)]^{15}$
   ii. Let $Y \leftarrow \text{CIPH}_J(Q)$.
   iii Let $y \leftarrow \text{NUM}_2(Y)$.
   iv. If *i* is even, let $m = u$; else, let $m = v$.
   v. Let $c = (\text{NUM}_{radix}(A) + y) \bmod radix^m$.
   vi. Let $C = \text{STR}^m_{radix}(c)$.
   vii. Let $A = B$.
   viii. Let $B = C$.

6. Return $A \,||\, B$.

Description:
The "split" of the numeral string *X* into two substrings, *A* and *B*, is performed in Steps 1 and 2. If *n* is even, $\text{LEN}(A)=\text{LEN}(B)$; otherwise, $\text{LEN}(A)=\text{LEN}(B)-1$. A fixed block, *P*, is defined in Step 3 from encodings of the base, *radix*, and the tweak, *T*. The block cipher under the key *K*, i.e., CIPH_K, is applied to *P* in Step 4 to produce a 128-bit subkey, *J*. An iteration loop for the ten Feistel rounds of FF2 is initiated in Step 5, executing eight substeps for each round, as follows:

The substring, *B*, and the round number, *i*, are encoded as a binary string, *Q*, in Step 5i. The block cipher under the subkey *J* is applied to *Q* in Step 5ii to produce a block, *Y*. (In **Error! Reference source not found.**, *Y* corresponds to the output of F_K.) In Steps 5iii to 5vi, *Y* 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 *Y* with *A* is indicated by the "+" operation.) In particular, in Step 5iii, *Y* is converted to a number, *y*. In Step 5iv, the length, *m*, of *A* for this Feistel round is determined, and in Step 5v, the number *y* is reduced modulo the *m*th power of the base *radix*. In Step 5v, the number *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 5vi. In Steps 5vii and 5viii, 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 FF2. After the tenth round, the concatenation of *A* and *B* is returned as the output in Step 6.

Algorithm 8: FF2.Decrypt(*K*, *T*, *X*)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, *K*, for the block cipher;
Base, *radix*, for the character alphabet;
Base, *tweakradix*, for the tweak character alphabet;
Range of supported message lengths, [*minlen* .. *maxlen*]
Maximum supported tweak length, *maxTlen*.

Inputs:
Numeral string, *X*, in base *radix* of length *n* such that $n \in [minlen \,..\, maxlen]$;
Tweak numeral string, *T*, in base *tweakradix* of length *t* such that $t \in [0 \,..\, maxTlen]$.

Output:
Numeral string, *Y*, such that $\text{LEN}(Y) = n$.

Steps:
1. Let $u = \lfloor n/2 \rfloor$; $v = n - u$.
2. Let $A = X[1 \,..\, u]$; $B = X[u + 1 \,..\, n]$.
3. If $t>0$, $P = [radix]^1 \,||\, [t]^1 \,||\, [n]^1 \,||\, [\text{NUM}_{tweakradix}(T)]^{13}$; else $P = [radix]^1 \,||\, [0]^1 \,||\, [n]^1 \,||\, [0]^{13}$.
4. Let $J = \text{CIPH}_K(P)$
5. For i from 9 to 0:
   i. Let $Q \leftarrow [i]^1 \,||\, |\, [\text{NUM}_{radix}(B)]^{15}$
   ii. Let $Y \leftarrow \text{CIPH}_J(Q)$.
   iii Let $y \leftarrow \text{NUM}_2(Y)$.
   iv. If *i* is even, let $m = u$; else, let $m = v$.
   v. Let $z = y \bmod radix^m$.
   vi. Let $c = (\text{NUM}_{radix}(A) - y) \bmod radix^m$.
   vii. Let $C = \text{STR}^m_{radix}(c)$.
   viii. Let $A = B$.
   ix. Let $B = C$.
6. Return $A \,||\, B$.

Description:
The FF2.Decrypt algorithm is almost identical to the FF2.Encrypt algorithm: the only differences are in Step 5, where the order of the indices is reversed, and in Step 5v, where modular addition is replaced by modular subtraction.

6.3 FF3

The specifications for the FF3.Encrypt and FF3.Decrypt functions are given in Algorithms 9 and 10 below.

The parameters *radix*, *minlen*, and *maxlen* in FF3.Encrypt and FF3.Decrypt shall meet the following requirements:

- $radix \in [2 .. 2^{16}]$;
- $radix^{minlen} \geq 100$;
- $minlen \geq 2$; and
- $maxlen \leq 2\lfloor \log_{radix}(2^{96}) \rfloor$.

Algorithm 9: FF3.Encrypt(*K*, *T*, *X*)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, *K*, for the block cipher;
Base, *radix*, for the character alphabet such that $radix \in [2 .. 2^{16}]$;
Range of supported message lengths, [*minlen* .. *maxlen*], such that $minlen \geq 2$ and $maxlen \leq 2\lfloor \log_{radix}(2^{96}) \rfloor$].

Inputs:
Numeral string, *X*, in base *radix* of length *n* such that $n \in [minlen .. maxlen]$;
Tweak bit string, *T*, such that $\text{LEN}(T) = 64$.

Output:
Numeral string, *Y*, such that $\text{LEN}(Y) = n$.

Steps:
1. Let $u = \lceil n/2 \rceil$; $v = n - u$.
2. Let $A = X[1 .. u]$; $B = X[u + 1 .. n]$.
3. Let $T_L = T[0..31]$ and $T_R = T[32..63]$
4. For *i* from 0 to 7:
   i. If *i* is even, let $m = u$ and $W = T_R$, else let $m = v$ and $W = T_L$.
   ii. Let $P = \text{REV}([\text{NUM}_{radix}(\text{REV}(B))]^{12}) \| W \oplus \text{REV}([i]^4)$.
   iii Let $Y = \text{CIPH}_K(P)$.
   iv. Let $y = \text{NUM}_2(\text{REV}(Y))$.
   v. Let $c = (\text{NUM}_{radix}(\text{REV}(A)) + y) \bmod radix^m$.
   vi. Let $C = \text{REV}(\text{STR}^m_{radix}(c))$.
   vii. Let $A = B$.
   viii. Let $B = C$.
5. Return $A \| B$.

Description:
The "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.[7] The tweak, T, is partitioned into a 32-bit left tweak, T_L, and a 32-bit right tweak, T_R, in Step 3. An iteration loop for the eight Feistel rounds of FF3 is initiated in Step 4, executing eight substeps for each round, as follows:

In Step 4i, the parity of the round number, i, determines the length, m, of the substring A, and whether T_L or T_R will be used as W in Step 4ii, in which a 32-bit encoding of i, XORed with W, is concatenated with a 96-bit encoding of B to produce a block, P. In Step 4iii, the block cipher under the key, i.e., CIPH_K, is applied to P, to produce a block, Y. (In Figure 1, Y corresponds to the output of F_K.) In Steps 4iv to 4vii, Y 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 Y with A is indicated by the "+" operation, although this operation is different than for FF1 and FF2 in that FF3 uses the opposite ordering convention for the conversion of strings to numbers and vice versa.) In particular, in Step 4iv, Y is converted to a number, y. In Step 4v, the number y is added to the number represented by the substring A and the result is reduced modulo the mth power of *radix*, yielding a number, c, which is converted to a numeral string in Step 5vi. In Steps 4vii and 4viii, 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 FF3. After the eighth round, the concatenation of A and B is returned as the output in Step 5.

Algorithm 10: FF3.Decrypt(K, T, X)

Prerequisites:
Approved, 128-bit block cipher, CIPH;
Key, K, for the block cipher;
Base, *radix*, for the character alphabet such that *radix* $\in [2 .. 2^{16}]$;
Range of supported message lengths, [*minlen* .. *maxlen*], such that *minlen* $\geq$ 2 and *maxlen* $\leq 2\lfloor \log_{radix}(2^{96}) \rfloor$.

Inputs:
Numeral string, X, in base *radix* of length n such that $n \in$ [*minlen* .. *maxlen*];
Tweak bit string, T, such that LEN(T) = 64.

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

Steps:
1. Let $u = \lceil n/2 \rceil$; $v = n - u$.
2. Let $A = X[1 .. u]$; $B = X[u + 1 .. n]$.
3. Let $T_L = T[0..31]$ and $T_R = T[32..63]$
4. For i from 7 to 0:
   i. If i is even, let $m = u$ and $W = T_R$, else let $m = v$ and $W = T_L$.
   ii. Let $P = \text{REV}([\text{NUM}_{radix}(\text{REV}(B))]^{12}) \,||\, W \oplus \text{REV}([i]^4)$.

[7] If n is odd, A is one character longer than B, in contrast to FF1 and FF2, where B is one character longer than A.

iii Let $Y = \text{CIPH}_K(P)$.
iv. Let $y = \text{NUM}_2(\text{REV}(Y))$.
vi. Let $c = (\text{NUM}_{radix}(\text{REV}(A)) - y) \bmod radix^m$.
vii. Let $C = \text{REV}(\text{STR}^m_{radix}(c))$.
viii. Let $A = B$.
ix. Let $B = C$.
5. Return $A \| B$.

Description:
The FF3.Decrypt algorithm is almost identical to the FF3.Encrypt algorithm: the only differences are in Step 4, where the order of the indices is reversed, and in Step 4v, where modular addition is replaced by modular subtraction.

7 Conformance

An implementation may claim conformance to any of the following six functions:

FF1.Encrypt	FF2.Encrypt	FF3.Encrypt
FF1.Decrypt	FF2.Decrypt	FF3.Decrypt

Component functions such as PRF are not approved for use independent of these eight functions.

In order to claim conformance with this Recommendation, an implementation of FF1, FF2, or FF3 must support at least one set of values for its parameters.

Two implementations can only interoperate when they support a common value for the base. Similarly, two implementations of FF2 can only interoperate when they also support a common value for the tweak base.

Moreover, each mode has two parameters, *minlen* and *maxlen*, that determine the lengths for the numeral strings that are supported by an implementation of the encryption or decryption function for the mode. FF1 and FF2 also have a parameter, *maxTlen*, that determines the lengths of the tweak strings they support. The selection of these parameters may also affect interoperability.

For every algorithm that is specified in this Recommendation, a conforming implementation may replace the given set of steps with any mathematically equivalent set of steps. In other words, different procedures that produce the correct output for any input are permitted.

8 References

[1] M. Bellare, P. Rogaway, T. Spies, The FFX Mode of Operation for Format-Preserving Encryption, Draft 1.1, Natl. Inst. Stand. Technol. [Web page], http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf, February, 2010.

[2] M. Bellare, P. Rogaway, T. Spies, Addendum to "The FFX Mode of Operation for Format-Preserving Encryption": A parameter collection for enciphering strings of arbitrary radix and length, Draft 1.0, Natl. Inst. Stand. Technol. [Web page], http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec2.pdf, September 2010.

[3] E. Brier, T. Peyrin, and J. Stern, BPS: a Format-Preserving Encryption Proposal Natl. Inst. Stand. Technol. [Web page], http://csrc.nist.gov/groups/ST/toolkit/BCM/documents /proposedmodes/bps/bps-spec.pdf.

[4] M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. CRYPTO 2002, LNCSvol. 2442, Springer, pp. 31–46, 2002

[5] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandomfunctions. SIAM Journal on Computing, 17(2), pp. 373–386, 1988

[6] National Institute of Standards and Technology. FIPS 197, The Advanced Encryption Standard (AES). 2001.

[7] National Institute of Standards and Technology. NIST Special Publication 800-38A: *Recommendation for Block Cipher Modes of Operation—Methods and Techniques*, December, 2001.

[8] National Institute of Standards and Technology. NIST Special Publication 800-67 Revision 1: *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, January, 2012.

[9] National Institute of Standards and Technology. NIST Special Publication 800-133: *Recommendation for Cryptographic Key Generation*, December, 2012.

[10] J. Patarin. Generic attacks on Feistel schemes. Cryptology ePrint report 2008/036. January24, 2008. Also ASIACRYPT 2001, LNCS vol. 2248, Springer, pp. 222–238, 2001.

[11] R. Schroeppel. Hasty Pudding Cipher specification. Unpublished manuscript, [Web page] athttp://richard.schroeppel.name:8015/hpc/hpc-spec, June 1998 (revised May 1999).

[12] T. Spies. Feistel finite set encryption. NIST submission, February 2008.

[13] J. Vance, VAES3 scheme for FFX: An addendum to "The FFX Mode of Operation for Format-Preserving Encryption": A parameter collection for encipher strings of arbitrary radix with subkey operation to lengthen life of the enciphering key, Natl. Inst. Stand. Technol. [Web page], http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ proposedmodes/ffx/ffx-ad-VAES3.pdf.

Appendix A: Parameter Choices

The length of the key affects its resistance to brute-force search. The requirement for each mode that $radix^{minlen} \geq 100$ precludes a generic meet-in-the-middle attack on the Feistel structure [10]. A requirement on *maxlen* for FF2—namely, that $maxlen \leq 2\lfloor 98/\text{LOG}_2(radix)\rfloor$ if *radix* is not a power of 2—minimizes the bias in the generation of *z*.

Otherwise, the choices of the mode parameters, e.g., *radix*, *minlen*, and *maxlen*, are determined by the needs of the application, not by security considerations.

Two potential parameters of the Feistel structure are fixed for FF1, FF2, and FF3, namely, the number of Feistel rounds and the imbalance, i.e., the lengths into which the input numeral string is split: these were chosen in each case to balance performance requirements and security considerations. See H of [1] for a discussion.

For FF1 and FF2, the maximum tweak length parameter, *maxTlen*, should be chosen to accommodate non-secret information that may be associated to a plaintext. The security rationale for tweaks is discussed in Appendix C.

Appendix B: Security Goal

The designers of FFX aimed to achieve strong-pseudorandom permutation (PRP) security for a conventional block cipher [5]. In the FFX proposal to NIST [1], the designers of FFX cite the history of cryptographic results concerning Feistel networks as underlying their selection of the FFX mechanism. They assert that, under the assumption that the underlying round function is a good pseudorandom function (PRF), contemporary cryptographic results and experience indicate that FFX achieves cryptographic goals including nonadaptive message-recovery security, chosen-plaintext, and even PRP-security against an adaptive chosen-ciphertext attack. The quantitative security depends on the number of rounds used, the imbalance, and the adversary's access to plaintext/ciphertext pairs. See [1] for details.

Appendix C: Tweaks

Tweaks have been supported in stand-alone block ciphers, such as Schroeppel's Hasty Pudding [11], and the notion was later formalized and investigated by Liskov, Rivest, and Wagner [4]. Tweaks are important for FPE modes, because FPE may be used in settings where the number of character strings is relatively small. In such settings, the tweak should vary with each instance of the encryption whenever possible.

For example, suppose that in an application for CCNs the leading six digits and the trailing four digits need to be available, unencrypted, to the application, so that only the remaining six digits in the middle of the CCNs are encrypted. There are a million different possibilities for these middle-six digits, so, in a database of 100 million CCNs, about a hundred distinct CCNs would be expected to share each possible value for these six digits. If these hundred CCNs were encrypted with the same tweak, then their ciphertexts would be the same.

If, however, the other ten digits had been the tweak for the encryption of the middle-six digits, then the hundred ciphertexts would almost certainly be different. Therefore, for example, learning that CCN 123456-123456-9876 encrypts to 123456-770611-9876 would not allow the decryption of 111111-770611-9999, as the mapping of 123456 to 770611 is specific to the surrounding digits 123456/9876.

In general, it is recommended to use information that is available and statically associated to a plaintext as a tweak for that plaintext. Ideally, the non-secret tweak associated to a plaintext is associated only to that plaintext.

Extensive tweaking means that fewer plaintexts are enciphered under any given tweak. This corresponds, in the security model that is described in [1], to fewer queries to the target instance of the encryption. The relevant metric is the maximum number of plaintexts that are encrypted with the same tweak, which is likely to be significantly less than the total number of plaintexts enciphered.