Draft NIST Special Publication 800-38G
Revision 1

Recommendation for Block Cipher Modes of Operation

Methods for Format-Preserving Encryption

Morris Dworkin

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-38Gr1-draft

COMPUTER SECURITY

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

Draft NIST Special Publication 800-38G Revision 1

Recommendation for Block Cipher Modes of Operation

Methods for Format-Preserving Encryption

Morris Dworkin
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-38Gr1-draft

February 2019

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 *et seq.*, Public Law (P.L.) 113-283. 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.

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 Revision 1
Natl. Inst. Stand. Technol. Spec. Publ. 800-38G, 31 pages (February 2019)
CODEN: NSPUE2

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-38Gr1-draft

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

Public comment period: *February 28, 2019* through *April 15, 2019*

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

All comments are subject to release under the Freedom of Information Act (FOIA).

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 two methods, called FF1 and FF3-1, for format-preserving encryption. Both of these methods are modes of operation for an underlying, approved symmetric-key block cipher algorithm. Compared to the original version of this publication, the tweak size for FF3-1 is smaller than the tweak size for FF3; also, for both FF1 and FF3-1, larger domains are required, rather than merely recommended.

Keywords

Block cipher; confidentiality; encryption; FF1; FF3; FF3-1; format-preserving encryption; information security; mode of operation.

Acknowledgements

The author gratefully acknowledges the designers of the two algorithms that are specified in this publication: 1) Mihir Bellare, Phil Rogaway, and Terence Spies; and 2) Eric Brier, Thomas Peyrin, and Jacques Stern.

Serge Vaudenay and Betül Durak kindly gave NIST early notification of their analysis of the FF3 method in [7], which prompted the revision of the method in this version of the publication. Similarly, Mihir Bellare, Viet Tung Hoang, Stefano Tessaro gave NIST early notification of their analysis of the FPE modes in [1], which was subsequently improved by Hoang and Tessaro in their paper with Ni Trieu [8]. These papers motivated the larger lower limit on the number of inputs for both FF1 and FF3-1, which previously had been recommended but not required.

The author also wishes to thank his colleagues who reviewed drafts of this publication and contributed to its development, especially Elaine Barker, Nicky Mouha, Lily Chen, John Kelsey, Meltem Sonmez Turan, Kerry McKay, Allen Roginsky, Larry Bassham, Ray Perlner, Rene Peralta, Jim Foti, Sara Kerman, Andy Regenscheid, Bill Burr, and Tim Polk.

The author also acknowledges the comments from the public and private sectors to improve the quality of this publication.

Conformance Testing

Conformance testing for implementations of the functions that are specified in this publication will be conducted within the framework of the Cryptographic Algorithm Validation Program (CAVP) and the Cryptographic Module Validation Program (CMVP). The requirements on these implementations 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.

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

i) under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or

ii) without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: EncryptionModes@nist.gov.

Table of Contents

List of Figures

1 Purpose

This publication is a revision of 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 two approved methods for format-preserving encryption (FPE).

Since the original publication of these FPE modes in March of 2016, researchers identified vulnerabilities in [8], building on the work in [1], and in [7]. The present revision includes sets of technical revisions to mitigate the vulnerabilities, as summarized in Appendix F.

2 Introduction

A block cipher mode of operation—or simply, mode—is an algorithm for the cryptographic transformation of data that is based on a block cipher. The previously approved modes for encryption are transformations on binary data, i.e., the inputs and outputs of the modes are bit strings—sequences of ones and zeros. For sequences of non-binary symbols, however, there is no natural and general way for the previously approved modes to produce encrypted data that has the same format. For example, a Social Security Number (SSN) consists of nine decimal digits, so it is an integer that is less than one billion. This integer can be converted to a bit string as input to a previously approved mode, but when the output bit string is converted back to an integer, it may be greater than one billion, which would be too long for an SSN.

FPE is designed for data that is not necessarily binary. In particular, given any finite set of symbols, like the decimal numerals, a method for 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, including the length, as the original data. Thus, an FPE-encrypted SSN would be a sequence of nine decimal digits.

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. For example, database applications may not support changes to the length or format of data fields. FPE has emerged as a useful cryptographic tool, whose applications include financial-information security, data sanitization, [1] and the transparent encryption of fields in legacy databases.

The two FPE modes specified in this publication are called FF1 and FF3-1. FF3-1 is a revision of the FF3 mode that was specified in the original version of this publication; the revision of FF3, as well as a modified requirement for both FF1 and FF3-1, are described in Appendix F. The acronyms for the modes indicate that they are format-preserving, Feistel-based encryption modes. FF1 was submitted to NIST under the name FFX[Radix] in [3]. FF3 is a component of the FPE method that was submitted to NIST under the name BPS in [4]. In particular, 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 publication.

[1] The sanitization of personally identifiable information in a database—whether by FPE or other methods—does not necessarily provide strong assurance that individuals cannot be re-identified; for example, see [5].

Each of these FPE modes fits within a larger framework, called FFX, for constructing FPE mechanisms; FFX was submitted to NIST in [2]. The "X" indicates the flexibility to instantiate the framework with different parameter sets, as well as FFX's evolution from its precursor, the Feistel Finite Set Encryption Mode.

The FFX framework itself is not specified in this publication; in fact, FF1 and FF3-1 are not presented explicitly as instantiations of FFX parameter sets, but rather as separate algorithms, in order to simplify the individual specifications.

FF1 and FF3-1 each employ the Feistel structure—see Sec. 4.4—which also underlies the Triple Data Encryption Algorithm (TDEA) [15]. At the core of FF1 and FF3-1 are somewhat different Feistel round functions that are derived from an approved block cipher with 128-bit blocks, i.e., the Advanced Encryption Standard (AES) algorithm [12].

In addition to the formatted data for which the modes provide confidentiality, each mode also takes an additional input called the "tweak," which is not necessarily secret. The tweak can be regarded as a changeable part of the key, because together they determine the encryption and decryption functions. Tweaks that vary can be especially important for implementations of FPE modes, because the number of possible values for the confidential data is often relatively small, as discussed in Appendix A and Appendix C.

FF1 and FF3-1 offer somewhat different performance advantages. FF1 supports a greater range of lengths for the protected, formatted data, as well as flexibility in the length of the tweak. FF3-1 achieves greater throughput, mainly because it has eight rounds, compared to ten for FF1.

3 Definitions and Notation

3.1 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 Federal Information Processing Standard (FIPS) or a NIST Recommendation.
base	The number of characters in a given alphabet. The base is denoted by *radix*.
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 whose length is a multiple of a given block size, so that it can be represented as the concatenation of a finite sequence of blocks.
byte	A string of eight bits.
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.
character	A symbol in a given alphabet.
character string	A finite, ordered sequence of characters from a given alphabet.
ciphertext	In this publication, the numeral string that is the encrypted form of a plaintext numeral string.
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 (XOR)	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 forward transformation .
key	For a given block cipher, the secret bit string that parameterizes the permutation.

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 finite, ordered sequence of numerals for the base.
plaintext	In this publication, 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 necessarily protected by the mode.

3.2 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.
PRP	pseudorandom permutation.
RFC	Request for Comment.
SSN	Social Security number.

3.3 Operations and Functions

BYTELEN(X)	The number of bytes in a byte string, X, which may be represented as a bit string. For example, BYTELEN(`1011100110101100`) = 2.
CIPH$_K$(X)	The output of the designated cipher function of the block cipher under the key K applied to the block X.
LEN(X)	The number of numerals/bits in a numeral/bit string X. For example, LEN(`010`) = 3.
LOG(x)	The base 2 logarithm of the real number $x > 0$. For example, LOG(64) = 6 and LOG(10) ≈ 3.32.
NUM(X)	The integer that a bit string X represents when the bits are valued in decreasing order of significance. For example, NUM(`10000000`) = 128. An algorithm for computing NUM(X) is given in Sec. 4.5.
NUM$_{radix}$(X)	The number that the numeral string X represents in base *radix* when the numerals are valued in decreasing order of significance. For example, NUM$_5$(`00011010`) = 755. An algorithm for computing NUM$_{radix}$(X) is given in Sec. 4.5.
PRF(X)	The output of the function PRF applied to the block X; PRF is defined in terms of a given designated cipher function.
REV(X)	Given a numeral string, X, the numeral string that consists of the numerals of X in reverse order. For example, in base ten, REV(`13579`) = `97531`.
REVB(X)	Given a byte string, X, the byte string that consists of the bytes of X in reverse order. For example, REVB($[1]^1 \| [2]^1 \| [3]^1$) = $[3]^1 \| [2]^1 \| [1]^1$.
STR$^m_{radix}$(x)	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. For example, 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 STR$^m_{radix}$(x) is given in Sec. 4.5.
$\lfloor x \rfloor$	The floor function: given a real number x, the greatest integer that does not exceed x. For example, $\lfloor 2.1 \rfloor = 2$, and $\lfloor 4 \rfloor = 4$.
$\lceil x \rceil$	The ceiling function: given a real number x, the least integer that is not less than x. For example, $\lceil 2.1 \rceil = 3$, and $\lceil 4 \rceil = 4$.
$[x]^s$	Given a nonnegative integer x less than 256^s, the representation of x as a string of s bytes. For example, $[5]^1$ = `00000000 00000101`.

$[i\,..\,j]$	The set of integers between two integers i and j, including i and j. For example, $[2\,..\,5] = \{2, 3, 4, 5\}$.
$x \bmod m$	The nonnegative remainder of the integer x modulo the positive integer m, i.e., $x - m\lfloor x/m \rfloor$. For example, 13 mod 7 = 6, and −3 mod 7 = 4.
$X[i]$	Given a numeral/bit string X and an index i such that $1 \le i \le \text{LEN}(X)$, the i^{th} numeral/bit of X. For example, in base ten, if $X = 798137$, then $X[2] = 9$.
$X[i\,..\,j]$	The substring of the string X from $X[i]$ to $X[j]$, including $X[i]$ and $X[j]$. For example, in base ten, if $X = 798137$, then $X[3\,..\,5] = 813$.
$X \oplus Y$	The bitwise exclusive-OR of bit strings X and Y whose bit lengths are equal. For example, `10011` $\oplus$ `10101` = `00110`.
$X \,\|\, Y$	The concatenation of numeral strings X and Y. For example, `001` || `1011` = `0011011`, and `3 1` || `31 8 10` = `3 1 31 8 10`.
0^s	The bit string that consists of s consecutive '0' bits. For example, 0^8 = `00000000`.

4 Preliminaries

4.1 Representation of Character Strings

The data inputs and outputs for FF1 and FF3-1 are sequences of numbers that can represent both numeric and non-numeric data, as discussed below.

A finite set of two or more symbols is called an *alphabet*. The symbols in an alphabet are called the *characters* of the alphabet. The number of characters in an alphabet is called the *base*, denoted by *radix*; thus, $radix \ge 2$.

A character string is a finite sequence of characters from an alphabet; individual characters may repeat in the string. In this publication, character strings (and bit strings) are presented in the `Courier New` font.

Thus, for 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},

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

SSNs or Credit Card Numbers (CCNs) can be regarded as character strings in the alphabet of base ten numerals, namely, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The notion of numerals is generalized to any given base as follows: the set of base *radix* numerals is

$$\{0, 1, \ldots, radix\text{-}1\}.$$

The data inputs and outputs to the FF1 and FF3-1 encryption and decryption functions must be finite sequences of numerals, i.e., *numeral strings*. If the data to be encrypted is formatted in an alphabet that is not already the set of base *radix* numerals, then each character must be represented by a distinct numeral in order to apply FF1 or FF3-1.

For example, the natural representation of lower-case English letters with base 26 numerals is

$$a \rightarrow 0,\ b \rightarrow 1,\ c \rightarrow 2,\ \ldots\ x \rightarrow 23,\ y \rightarrow 24,\ z \rightarrow 25.$$

The character string `hello` would then be represented by the numeral string `7 4 11 11 14`. Other representations are possible.

The choice and implementation of a one-to-one correspondence between a given alphabet and the set of base *radix* numerals that represents the alphabet is outside the scope of this publication.

In this publication, individual numerals are themselves represented in base ten. In order to display numeral sequences unambiguously when the base is greater than ten, a delimiter between the numerals is required, such as a space (as in the base 26 example above) or a comma.

FF1 and FF3-1 use different conventions for interpreting numeral strings as numbers. For FF1, numbers are represented by strings of numerals with *decreasing* order of significance; for FF3-1, numbers are represented by strings of numerals in the reverse order, i.e., with *increasing* order of significance. For example, "`0025`" is a string of decimal digits that represents the number twenty-five for FF1 and the number five thousand two hundred for FF3-1. Algorithms for the functions that convert numeral strings to numbers and vice versa are given in Sec. 4.5.

4.2 Underlying Block Cipher and Key

The encryption and decryption functions of FF1 and FF3-1 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 by CIPH_K. The inverse of CIPH_K is not needed for the modes that are specified in this publication.

For both modes, the underlying block cipher shall be approved, and the block size shall be 128 bits. Currently, the AES block cipher [12], 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 and FF3-1, and the choice of the key length is not

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

explicitly indicated in their specifications. Methods for generating cryptographic keys are discussed in [16]; 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 the functions that are specified in this publication. The management of cryptographic keys is outside the scope of this publication.

4.3 Encryption and Decryption Functions

For a given key, denoted by K, for the designated block cipher, FF1 and FF3-1 each consist of two related functions: encryption and decryption. The inputs to the encryption function are a numeral string called the plaintext, denoted by X, and a byte string, called the tweak, denoted by T; the function returns a numeral string called the ciphertext, denoted by 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 is denoted by FF1.Encrypt(K, T, X), and the decryption function is denoted by FF1.Decrypt(K, T, X), with analogous notation for FF3-1.

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

$$\text{FF1.Decrypt}(K, T, \text{FF1.Encrypt}(K, T, X)) = X,$$
$$\text{FF3-1.Decrypt}(K, T, \text{FF3-1.Encrypt}(K, T, X)) = X.$$

The tweak does not need to be kept secret; often, it is some readily available data that is associated with the plaintext. Although implementations may fix the value of the tweak, variable tweaks should be used as a security enhancement; see Appendix C. In FF1 and FF3-1, tweaks are byte strings. The specifications in Sec. 5 include the lengths that can be supported for the tweak, as well as for the plaintext/ciphertext.

The key, K, is indicated in the above 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.

4.4 Feistel Structure

FFX schemes, including FF1 and FF3-1, are based on the Feistel structure. The Feistel structure consists of several iterations, called *rounds*, of a reversible transformation. 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; and 3) the roles of the two parts are swapped for the next round. The structure is illustrated in Figure 1 below,

[3] The distinction does not affect the execution of the function: all information is required, independent of when they were established or provided to the implementation.

for both encryption and decryption. Four rounds are shown in Figure 1, but ten rounds are actually specified for FF1, and eight rounds for FF3-1.

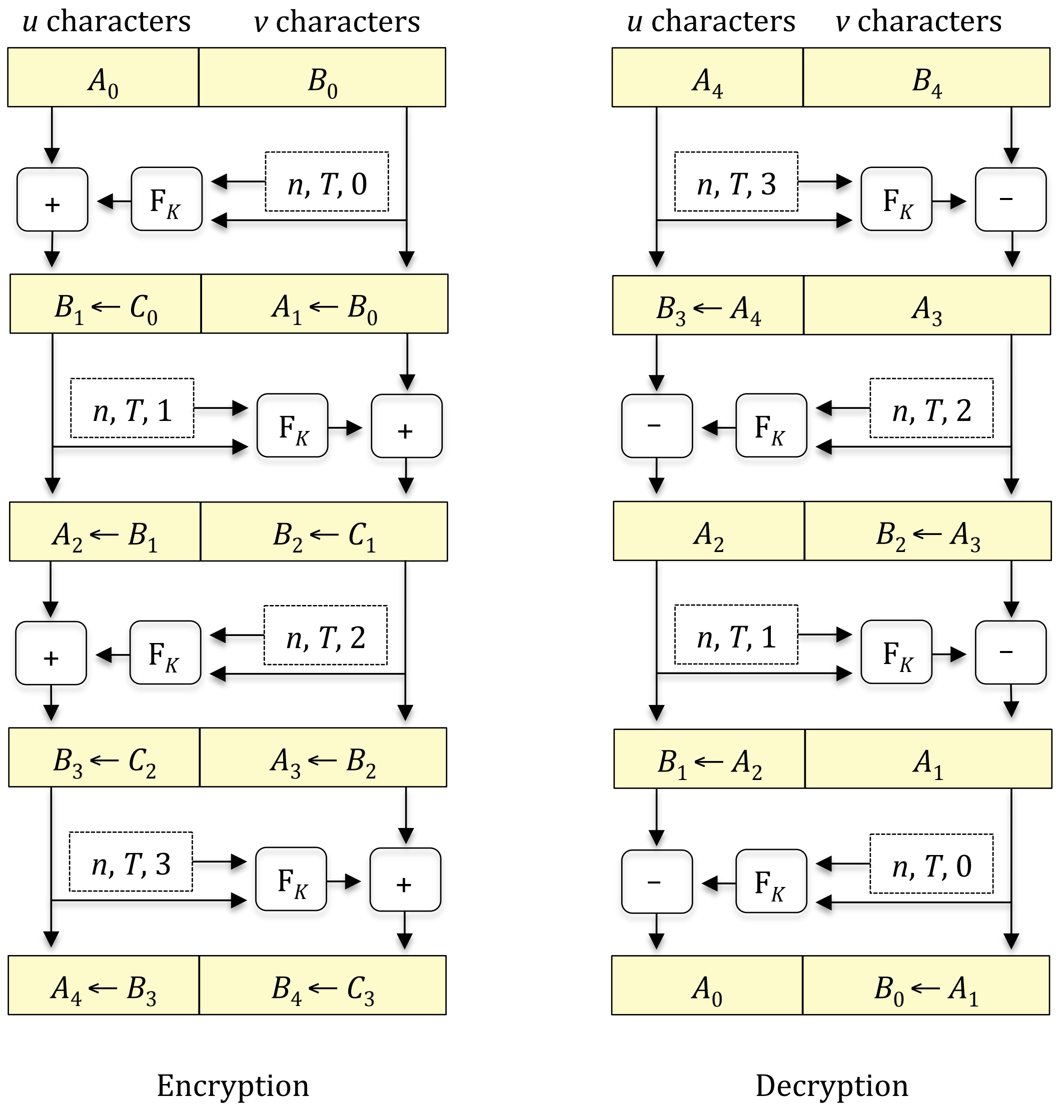

Figure 1: Feistel Structure

For the encryption function example 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 FF3-1. 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[4], indicated by +, on the numbers that the strings

[4] For some applications of the Feistel structure—but not FF1 and FF3-1—the "+" operation may be a different reversible operation on strings that preserves their length; for example, the FFX specification in [2] supports an option for character-wise addition.

represent[5]. 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}.

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.

The Feistel structure for decryption is almost identical to the Feistel structure for encryption. There are three differences: 1) the order of the round indices is reversed; 2) the roles of the two parts of the data in the round function are swapped as follows: along with n, T, and i, the input to F_K is A_{i+1} (not B_i), and the output is combined with B_{i+1} (not A_i) to produce A_i (not B_{i+1}); and 3) modular addition (of the output of F_K to A_i) is replaced by modular subtraction (of the output of F_K from B_{i+1}).

4.5 Component Functions

This section gives algorithms for the component functions that are called in the specifications of FF1 and FF3-1. The conversion functions $\mathrm{NUM}_{radix}(X)$, $\mathrm{NUM}(X)$, and $\mathrm{STR}^m_{radix}(x)$ are defined in Sec. 3.3, including examples, and they are specified in Algorithms 1-3 below. These functions support the ordering convention for the numeral/bit strings in FF1, namely, that the first (i.e., left-most) numeral/bit of the string is the most-significant numeral/bit

In FF3-1, the numeral strings follow the opposite ordering convention, as do the byte strings for the block cipher. In order to adapt $\mathrm{NUM}_{radix}(X)$, STR^m_{radix} (x), and CIPH_K (X) for the FF3-1 specifications, the functions $\mathrm{REV}(X)$ and $\mathrm{REVB}(X)$ are defined in Sec. 3.3 and specified in Algorithms 4 and 5.

The $\mathrm{PRF}(X)$ function, specified in Algorithm 6, essentially invokes the Cipher Block Chaining encryption mode [14] 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.

In order to simplify the specifications of $\mathrm{NUM}(X)$, $\mathrm{REVB}(X)$, and $\mathrm{PRF}(X)$, the byte or block strings in Algorithms 2, 5, and 6 are represented as bit strings.

Algorithm 1: NUM_{radix} (X)

Prerequisite:
Base, *radix*.

Input:
Numeral string, X.

Output:
Number, x.

[5] The ordering convention for interpreting strings as numbers is different for FF3-1 than for FF1.

Steps:

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

Algorithm 2: $\text{NUM}(X)$

Input:
Byte string, X, represented in bits.

Output:
Integer, x.

Steps:

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

Algorithm 3: $\text{STR}^m_{radix}(x)$

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

Input:
Integer, x, such that $0 \le x < radix^m$.

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 4: $\text{REV}(X)$

Input:
Numeral string, X.

Output:
Numeral string, Y.

Steps:

1. For i from 1 to $\text{LEN}(X)$, let $Y[i] = X[\text{LEN}(X)+1-i]$.
2. Return $Y[1..\text{LEN}(X)]$.

Algorithm 5: REVB(X)

Input:
Byte string, X, represented in bits.

Output:
Byte string, Y, represented in bits.

Steps:
1. For i from 0 to $\text{BYTELEN}(X)-1$ and j from 1 to 8, let $Y[8i+j] = X[8\cdot(\text{BYTELEN}(X)-1-i)+j]$.
2. Return $Y[1\,..\,8\cdot\text{BYTELEN}(X)]$.

Algorithm 6: 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 $m = \text{LEN}(X)/128$.
2. Let $X_1, \ldots, X_m$ be the blocks for which $X = X_1 \| \ldots \| X_m$.
3. Let $Y_0 = 0^{128}$, and for j from 1 to m let $Y_j = \text{CIPH}_K(Y_{j-1} \oplus X_j)$.
4. Return Y_m.

5 Mode Specifications

The specifications of the encryption and decryption algorithms for FF1 and FF3-1 are presented in Sections 6.1 and 6.2, organized into prerequisites, inputs, outputs, steps, and descriptions of the steps. In addition to the key and designated cipher function, the prerequisites for each mode are the choices of 1) the base, *radix*, and 2) the range of lengths, [*minlen*..*maxlen*], for the numeral string inputs that the implementation supports. FF1 also has a prerequisite for the choice of the maximum tweak length, *maxTlen*, that the implementation supports. For each mode, the requirements on the values for the prerequisites are specified prior to the encryption and decryption algorithms.

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

For each specification, the 128-bit input and output blocks of the designated block cipher, CIPH_K, are represented as strings of 16 bytes.

5.1 FF1

The specifications for the FF1.Encrypt and FF1.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*, *minlen*, and *maxlen* in FF1.Encrypt and FF1.Decrypt shall meet the following requirements:

- $radix \in [2..2^{16}]$,
- $radix^{minlen} \geq 1\,000\,000$, and
- $2 \leq minlen \leq maxlen < 2^{32}$.

Algorithm 7: FF1.Encrypt(K, T, X)

Prerequisites:
Designated cipher function, CIPH, of an approved 128-bit block cipher;
Key, K, for the block cipher;
Base, *radix*;
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 T, a byte string of byte 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. Let $b = \lceil \lceil v \cdot \text{LOG}(radix) \rceil / 8 \rceil$.
4. Let $d = 4\lceil b/4 \rceil + 4$.
5. Let $P = [1]^1 \,||\, [2]^1 \,||\, [1]^1 \,||\, [radix]^3 \,||\, [10]^1 \,||\, [u \bmod 256]^1 \,||\, [n]^4 \,||\, [t]^4$.
6. 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 $y = \text{NUM}(S)$.
   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 $B = C$.
7. 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, $\text{LEN}(A)=\text{LEN}(B)-1$. 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.

Algorithm 8: FF1.Decrypt(K, T, X)

Prerequisites:
Designated cipher function, CIPH, of an approved 128-bit block cipher;
Key, K, for the block cipher;
Base, *radix*;
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 T, a byte string of byte 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. Let $b = \lceil \lceil v \cdot \text{LOG}(radix) \rceil / 8 \rceil$.
4. Let $d = 4\lceil b/4 \rceil + 4$
5. Let $P = [1]^1 \,\|\, [2]^1 \,\|\, [1]^1 \,\|\, [radix]^3 \,\|\, [10]^1 \,\|\, [u \bmod 256]^1 \,\|\, [n]^4 \,\|\, [t]^4$.

---

[6] When B is encoded as a byte string in Step 6i, 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 four bytes longer than this encoding of B, which minimizes any bias in the modular reduction in Step 6vi.

6. For i from 9 to 0:
   i. Let $Q = T \,||\, [0]^{(-t-b-1) \bmod 16} \,||\, [i]^1 \,||\, [\text{NUM}_{radix}(A)]^b$.
   ii. Let $R = \text{PRF}(P \,||\, Q)$.
   iii. Let S be the string of 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 $y = \text{NUM}(S)$.
   v. If i is even, let $m = u$; else, let $m = v$.
   vi. Let $c = (\text{NUM}_{radix}(B) - y) \bmod radix^{m}$.
   vii. Let $C = \text{STR}^{m}_{radix}(c)$.
   viii. Let $B = A$.
   ix. Let $A = C$.
7. Return $A \,||\, B$.

Description:
The FF1.Decrypt algorithm is similar to the FF1.Encrypt algorithm; the differences are in Step 6, where: 1) the order of the indices is reversed, 2) the roles of A and B are swapped, and 3) modular addition is replaced by modular subtraction, in Step 6vi.

5.2 FF3-1

The specifications for the FF3-1.Encrypt and FF3-1.Decrypt functions are given in Algorithms 9 and 10 below. The parameters *radix*, *minlen*, and *maxlen* in FF3-1.Encrypt and FF3-1.Decrypt shall meet the following requirements:

- $radix \in [2..2^{16}]$,
- $radix^{minlen} \geq 1\,000\,000$, and
- $2 \leq minlen \leq maxlen \leq 2\lfloor \log_{radix}(2^{96}) \rfloor$.

Algorithm 9: FF3-1.Encrypt(K, T, X)

Prerequisites:
Designated cipher function, CIPH, of an approved 128-bit block cipher;
Key, K, for the block cipher;
Base, *radix*;
Range of supported message lengths, $[minlen..maxlen]$.

Inputs:
Numeral string, X, in base *radix* of length n, such that $n \in [minlen..maxlen]$;
Tweak bit string, T, such that $\text{LEN}(T) = 56$.

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 $T_L = T[0..27] \,||\, 0^4$ and $T_R = T[32..55] \,||\, T[28..31] \,||\, 0^4$.
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 $P = W \oplus [i]^4 \,||\, [\text{NUM}_{radix}(\text{REV}(B))]^{12}$.
   iii Let $S = \text{REVB}(\text{CIPH}_{\text{REVB}(K)}\text{REVB}(P))$.
   iv. Let $y = \text{NUM}(S)$.
   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 in Step 3 into a 32-bit left tweak, T_L, and a 32-bit right tweak, T_R. An iteration loop for the eight Feistel rounds of FF3-1 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, is applied to P using the byte-reversed ordering convention, to produce a block, S. (In Figure 1, S corresponds to the output of F_K.) In Steps 4iv to 4vi, 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, although this operation is different than for FF1 in that FF3-1 uses the opposite ordering convention for the conversion of strings to numbers and vice versa.) In particular, in Step 4iv, S 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 4vi. 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 FF3-1. After the eighth round, the concatenation of A and B is returned as the output in Step 5.

Algorithm 10: FF3-1.Decrypt(K, T, X)

Prerequisites:
Designated cipher function, CIPH, of an approved 128-bit block cipher;
Key, K, for the block cipher;
Base, *radix*;
Range of supported message lengths, [*minlen*..*maxlen*].

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

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

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 $T_L = T[0..27] \,||\, 0^4$ and $T_R = T[32..55] \,||\, T[28..31] \,||\, 0^4$.
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. $P = W \oplus [i]^4 \,||\, [\text{NUM}_{radix}(\text{REV}(A))]^{12}$.
   iii Let $S = \text{REVB}(\text{CIPH}_{\text{REVB}(K)}\text{REVB}(P))$.
   iv. Let $y = \text{NUM}(S)$.
   v. Let $c = (\text{NUM}_{radix}(\text{REV}(B)) - y) \bmod radix^m$.
   vi. Let $C = \text{REV}(\text{STR}^m_{radix}(c))$.
   vii. Let $B = A$.
   viii. Let $A = C$.
5. Return $A \,||\, B$.

Description:
The FF3-1.Decrypt algorithm is similar to the FF3-1.Encrypt algorithm; the differences are in Step 4, where: 1) the order of the indices is reversed, 2) the roles of A and B are swapped, and 3) modular addition is replaced by modular subtraction, in Step 4v.

6 Conformance

Implementations of FF1.Encrypt, FF1.Decrypt, FF3-1.Encrypt, or FF3-1.Decrypt may be tested for conformance to this Recommendation under the auspices of NIST's Cryptographic Algorithm Validation Program [12].

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

In order to claim conformance with this Recommendation, an implementation of FF1 or FF3-1 may support as few as one value for the base.

Two implementations can only interoperate when they support common values for the base. Moreover, FF1 and FF3-1 have 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 also has a parameter, *maxTlen*, that indicates the maximum supported length of a tweak string. 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.

Appendix A: Parameter Choices and Security

The values of the parameters, e.g., *radix*, *minlen*, and *maxlen* affect the security that FF1 and FF3-1 can offer, because, as for any FPE method, encrypted data may be vulnerable to guessing attacks when the number of possible inputs is sufficiently small.

In particular, for a base *radix* numeral string S, there are $radix^{\text{LEN}(S)}$ possible values. For any ciphertext C, the corresponding plaintext has the same length; therefore, an attacker can guess the plaintext with probability $1/radix^{\text{LEN}(C)}$ by selecting a numeral string of LEN(C) at random. Repeated guesses increase the attacker's probability of success proportionately: with g distinct guesses, the probability is $g/radix^{\text{LEN}(C)}$.

For example, SSNs are base 10 numeral strings of length 9, so there are one billion possibilities. If an attacker could guess a thousand different values for an SSN, one of the guesses would be correct with probability $1000/10^9$, i.e., one in a million.

The original specifications of FF1and FF3 only imposed a modest absolute minimum of 100 on the number of possible inputs in order to preclude a generic meet-in-the-middle attack on the Feistel structure [17]. However, in order to mitigate guessing attacks and the analytic attacks described in [1] and [8], the number of possible inputs, namely $radix^{minlen}$, is required to be greater than or equal to 1 000 000, for both FF1 and FF3-1. In order to further limit the effectiveness of guessing attacks, implementations should also limit the number of guesses that an attacker can mount, if possible.

In order to prevent attacks against one instance of encryption from applying to other instances, implementations should enforce the use of different tweaks for different instances, as discussed in Appendix C. Usually, tweaks are non-secret information that can be associated with instances of encryption. For FF3-1, the tweak length is fixed, but for FF1 the maximum tweak length parameter, *maxTlen*, should be chosen to accommodate the desired tweaks for the implementation.

Two other potential parameters of the Feistel structure are fixed for FF1 and FF3-1, namely, the number of Feistel rounds and the imbalance, i.e., the values of the lengths u and v in Figure 1. Both of these parameters were set with consideration to both performance and security requirements. See Appendix H of [2] for a discussion.

Appendix B: Security Goal

The designers of FFX aimed to achieve strong-pseudorandom permutation (PRP) security for a conventional block cipher [10]. In the FFX proposal to NIST [2], the designers of FFX cited the history of cryptographic results concerning Feistel networks as underlying their selection of the FFX mechanism. They asserted that, under the assumption that the underlying round function is a good pseudorandom function (PRF), contemporary cryptographic results and experience indicate that FFX achieved several cryptographic goals, including nonadaptive message-recovery security, chosen-plaintext security, and even PRP-security against an adaptive chosen-ciphertext attack. The quantitative security would depend on the number of rounds used, the imbalance, and the adversary's access to plaintext-ciphertext pairs. See [2] for details.

Appendix C: Tweaks

Tweaks have been supported in stand-alone block ciphers, such as Schroeppel's Hasty Pudding [18], and the notion was later formalized and investigated by Liskov, Rivest, and Wagner [9]. Tweaks are important for FPE modes, because FPE may be used in settings where the number of possible 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 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 the hundred CCNs that shared a given value for the middle-six digits 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.

Similarly, in the encrypted database, about a hundred CCNs would be expected to share each possible value for the ciphertext, i.e., the middle-six digits. If the hundred CCNs that produce a given ciphertext had been encrypted with the same tweak, then the corresponding plaintexts would also be the same. This outcome would be undesirable because the compromise of the confidentiality of any of the hundred CCNs would reveal the others.

If, however, the leading six digits and the trailing four digits of the CCN had been used as the tweak, then the corresponding plaintexts would almost certainly be different. Therefore, for example, learning that the decryption of 111111-770611-1111 is 111111-123456-1111 would not reveal any information about the decryption of 999999-770611-9999, because the tweak in that case was different.

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

Extensive tweaking means that fewer plaintexts are encrypted under any given tweak. This corresponds, in the security model that is described in [2], to fewer queries to the target instance of the encryption.

Appendix D: Examples

Examples for FF1 and FF3-1 are available at the examples page on NIST’s Computer Security Resource Center website: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values.

Appendix E: References

[1] M. Bellare, V. T. Hoang, and S. Tessaro, "Message-recovery attacks on Feistel-based Format Preserving Encryption," in ACM CCS '16, pages 444–455, ACM Press, 2016, https://doi.org/10.1145/2976749.2978390.

[2] M. Bellare, P. Rogaway, and T. Spies, *The FFX Mode of Operation for Format-Preserving Encryption*, Draft 1.1, February 20, 2010, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffx/ffx-spec.pdf.

[3] M. Bellare, P. Rogaway, and 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, September 3, 2010, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffx/ffx-spec2.pdf.

[4] E. Brier, T. Peyrin, and J. Stern, *BPS: a Format-Preserving Encryption Proposal*, [April 2010], https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/bps/bps-spec.pdf.

[5] Y-A. de Montjoye, L. Radaelli, V. Kumar Singh, and A. Pentland, "Unique in the shopping mall: On the reidentifiability of credit card metadata," *Science*, vol. 347 no. 6221 (January 30, 2016), pp. 536-539, https://doi.org/10.1126/science.1256297.

[6] M. Dworkin and R. Perlner, *Analysis of VAES3 (FF2)*, Report no. 2015/306, IACR Cryptology ePrint Archive, April 2, 2015, https://eprint.iacr.org/2015/306

[7] F. B. Durak and S. Vaudenay, "Breaking the FF3 Format-Preserving Encryption Standard Over Small Domains" in *Advances in Cryptology—CRYPTO 2017*, Lecture Notes in Computer Science vol. 10402, Springer, pp. 679–707, https://doi.org/10.1007/978-3-319-63715-0_23.

[8] V.T. Hoang, S. Tessaro, N. Trieu, "The Curse of Small Domains: New Attacks on Format-Preserving Encryption" in *Advances in Cryptology—CRYPTO 2018*, Lecture Notes in Computer Science 10991, Springer, Cham., pp. 221–251, https://doi.org/10.1007/978-3-319-96884-1_8.

[9] M. Liskov, R. Rivest, and D. Wagner, "Tweakable block ciphers," in *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science 2442, Berlin: Springer, pp. 31–46, September 13, 2002, https://doi.org/10.1007/3-540-45708-9_3.

[10] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM Journal on Computing*, vol. 17 no. 2 (1988), pp. 373–386, https://doi.org/10.1137/0217022.

[11] National Institute of Standards and Technology, *Explanation of changes to Draft SP 800-38G*, June 27, 2014, https://csrc.nist.gov/news/2014/explanation-of-changes-to-draft-sp-800-38G.

[12] National Institute of Standards and Technology, *Cryptographic Algorithm Validation Program (CAVP)*, https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program.

[13] National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 197, *The Advanced Encryption Standard (AES)*, November 2001, https://doi.org/10.6028/NIST.FIPS.197.

[14] National Institute of Standards and Technology. NIST Special Publication (SP) 800-38A, *Recommendation for Block Cipher Modes of Operation—Methods and Techniques*, December 2001, https://doi.org/10.6028/NIST.SP.800-38A.

[15] National Institute of Standards and Technology. NIST Special Publication (SP) 800-67 Revision 2, *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, January 2012, https://doi.org/10.6028/NIST.SP.800-67r2.

[16] National Institute of Standards and Technology. NIST Special Publication (SP) 800-133, *Recommendation for Cryptographic Key Generation*, December 2012, https://doi.org/10.6028/NIST.SP.800-133.

[17] J. Patarin, *Generic attacks on Feistel schemes*, Report no. 2008/036, IACR Cryptology ePrint Archive, January 24, 2008, https://eprint.iacr.org/2008/036.

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

Appendix F: Revision History

A third mode, FF2—submitted to NIST under the name VAES3—was included in the initial draft of this publication. As part of the public review of Draft NIST Special Publication (SP) 800-38G and as part of its routine consultation with other agencies, NIST was advised by the National Security Agency in general terms that the FF2 mode in the draft did not provide the expected 128 bits of security strength. NIST cryptographers confirmed this assessment via the security analysis in [6] and announced the removal of FF2 in [11].

For both FF1 and FF3-1, the domain size, i.e., the number of possible input strings, is the quantity $radix^{minlen}$. In response to the analysis in [8], the lower bound that is required for the domain size in the specifications of both FF1 in Sec. 5.1 and FF3-1 in Sec. 5.2 was raised from one hundred in the original publication to one million in Rev. 1.

The name "FF1" is unchanged from the original version of this publication, because the lower bound on the domain size only affects which parameter combinations are approved, not the specification of the encryption and decryption functions. FF3-1 has a different name than FF3 because, in addition to the new lower bound on the domain size, the encryption and decryption functions of FF3 were revised.

In particular, in response to the analysis in [7] on FF3, the size of the tweak specified in Sec. 5.2 was reduced from 64 bits for FF3 to 56 bits for FF3-1, which entailed the modification of the definitions of the strings T_L and T_R in Step 3 of Algorithm 9 and Step 3 of Algorithm 10. The modified definitions of these two strings can equivalently be implemented by taking a 64-bit tweak, reordering some of its bits in a particular manner, and then forcing the bits in eight particular bit positions to be zero. For tweaks with certain properties—for example, if non-zero bits only occur in the leading 28 bit positions—the specification of FF3-1 is backwards compatible with the original specification of FF3.