NIST Special Publication 800
NIST SP 800-38Gr1 2pd

Recommendation for Block Cipher Modes of Operation

Methods for Format-Preserving Encryption

Second Public Draft

Morris Dworkin
Nicky Mouha

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-38Gr1.2pd

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Special Publication 800
NIST SP 800-38Gr1 2pd

Recommendation for Block Cipher Modes of Operation

Methods for Format-Preserving Encryption

Second Public Draft

Morris Dworkin
Computer Security Division
Information Technology Laboratory

Nicky Mouha
FedWriters

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-38Gr1.2pd

February 2025

U.S. Department of Commerce
Jeremy Pelter, Acting Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

Publication History

Approved by the NIST Editorial Review Board on YYYY-MM-DD [will be added upon final publication]

How to Cite this NIST Technical Series Publication:

Dworkin M, Mouha N (2025) Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-38Gr1 2pd. https://doi.org/10.6028/NIST.SP.800-38Gr1.2pd

Author ORCID iDs

Morris Dworkin: 0000-0003-2745-9906

Nicky Mouha: 0000-0001-8861-782X

Public Comment Period
February 3, 2025 – April 4, 2025

Submit Comments
ciphermodes@nist.gov

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

Additional Information
Additional information about this publication is available at https://csrc.nist.gov/pubs/sp/800/38/g/r1/2pd, including related content, potential updates, and document history.

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

Abstract

This recommendation specifies the FF1 method for format-preserving encryption. This method is a mode of operation for an underlying, approved symmetric-key block cipher algorithm.

Keywords

block cipher; confidentiality; encryption; FF1; format-preserving encryption; information security; mode of operation.

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.

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: ciphermodes@nist.gov.

Table of Contents

List of Figures

Acknowledgments

The authors gratefully acknowledge the designers of the FF1 algorithm that is specified in this publication: Mihir Bellare, Phil Rogaway, and Terence Spies. The authors also appreciate the designers of the FF3 algorithm that was specified in the first version of this publication but removed from the current revision: Eric Brier, Thomas Peyrin, and Jacques Stern.

The work of several researchers contributed significantly to the improvement of this publication. Serge Vaudenay and Betül Durak kindly gave NIST early notification of their analysis of the FF3 method in [11], which prompted the replacement of FF3 with the FF3-1 variant in the first public draft of the revision of this publication. Similarly, Mihir Bellare, Viet Tung Hoang, and Stefano Tessaro gave NIST early notification of their analysis of the FPE modes in [3]. Subsequently, both analyses were improved — the former by Viet Tung Hoang, David Miller, and Ni Trieu [13] and the latter by Viet Tung Hoang, Stefan Tessaro, and Ni Trieu [14]. Durak and Vaudenay also published additional analysis that applied to the FPE modes in [12]. These papers motivated the larger lower limit on the number of inputs, which had previously been recommended but not required. In follow-up work, Beyne [6] described a weakness in the tweak schedule that affected both FF3 and FF3-1 but not FF1.

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

The authors also acknowledge 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.

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 specify FF1 as an approved mode of operation for format-preserving encryption (FPE).

The first version of this publication was released in March 2016. Subsequently, researchers identified vulnerabilities in [3], [6], [11], [12], [13], and [14]. In the current publication, FF3 is removed, and the previous guidance to avoid small domains for FF1 is strengthened to a requirement in order to mitigate its vulnerabilities, as summarized in Appendix G.

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, there is no natural or general way for the previously approved modes to produce encrypted data with 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 as the original data, including the length. Thus, an FPE-encrypted SSN would be a sequence of nine decimal digits.

FPE facilitates the encryption of sensitive information and 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 the security of financial information, data sanitization,[1] and the transparent encryption of fields in legacy databases.

If changes to the length of the data field are supported by the application, the data can be padded with a fixed sequence of symbols, such as adding redundant zeros to the beginning of a credit card number (CCN). Padding can provide resistance against ciphertext guessing attacks, as modifications of the FPE-encrypted data will be correctly padded upon decryption only with some probability, depending on the length of the padding. Additionally, padding the data field can hide the length of the data to mitigate plaintext guessing attacks (see Appendix A.1).

The FPE mode specified in this publication is called FF1. The acronym for this mode indicates that it is a format-preserving, Feistel-based encryption mode. FF1 was submitted to NIST under the name FFX[Radix] in [5]. It employs the Feistel structure (see Sec. 3.4), which also underlies the Triple Data Encryption Algorithm (TDEA) [1]. At the core of FF1 are substantially different Feistel round functions that are derived from an approved block cipher with 128-bit blocks (i.e., the Advanced Encryption Standard [AES] algorithm [17]). FF1 fits within a larger framework for constructing FPE mechanisms called FFX, which was submitted to NIST in [4]. 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, and FF1 is presented as a separate algorithm rather than an instantiation of FFX parameter sets to simplify the individual specification.

[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 [8].

In addition to the formatted data for which the mode provides confidentiality, the 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.

3. Preliminaries

3.1. Representation of Character Strings

The data inputs and outputs for FF1 are sequences of numbers that can represent both numeric and non-numeric data. 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* ≥ 2.

A character string is a finite sequence of characters from an alphabet, and 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 CCNs can be regarded as character strings in the alphabet of base-10 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, …, *radix*-1}.

The data inputs and outputs to the FF1 encryption and decryption functions must be finite sequences of numerals, or *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.

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

a→0, b→1, c→2, … x→23, y→24, z→25.

The character string hello would then be represented by the numeral string 7 4 11 11 14. Other representations are also 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 of the scope of this publication. Here, individual numerals are themselves represented in base 10. In order to display numeral sequences unambiguously when the base is greater than 10, a delimiter between the numerals is required, such as a space (as in the base 26 example above) or a comma.

FF1 uses the following convention for interpreting numeral strings as numbers. The numbers are represented by strings of numerals with *decreasing* order of significance. For example, 0025 is a string of decimal digits that represents the number 25. Algorithms for the functions that convert numeral strings to numbers and vice versa are given in Sec. 3.5.

3.2. Underlying Block Cipher and Key

The encryption and decryption functions of FF1 feature a block cipher as the main component. Thus, this FPE mechanism is a mode of operation (or simply, “mode”) 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*. The forward transformation[2] of the block cipher is denoted by CIPH_K. The inverse of CIPH_K is not needed for the modes that are specified in this publication. The underlying block cipher **shall** be approved, and the block size **shall** be 128 bits. The only block cipher that currently fits this profile is the AES block cipher [17] with key lengths of 128, 192, or 256 bits. In particular, CIPH_K is instantiated with CIPHER() with key *K*, as defined in [17].

The choice of the key length affects the security of the FPE mode (e.g., against brute-force search) and the implementation details of the AES algorithm. Otherwise, the key length does not affect the implementation of FF1, and the choice of the key length is not explicitly indicated in their specifications. Methods for generating cryptographic keys are discussed in [2]. The goal is to select the keys uniformly at random so that each possible key would 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 of the scope of this publication.

3.3. Encryption and Decryption Functions

For a given key *K* for the underlying block cipher, FF1 consists 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*, and 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*).

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

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

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, tweaks are

[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.

byte strings. The specification in Sec. 4 includes the lengths that can be supported for the tweak, plaintext, and ciphertext.

The key *K* is indicated in the above notation as an input for the encryption and decryption functions. However, the specifications in this publication list the key 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.

3.4. Feistel Structure

FFX schemes, such as FF1, 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.
3. The roles of the two parts are swapped for the next round.

Figure 1 illustrates the structure for both encryption and decryption. Four rounds are shown in Fig. 1, but 10 rounds are specified for FF1.

[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.

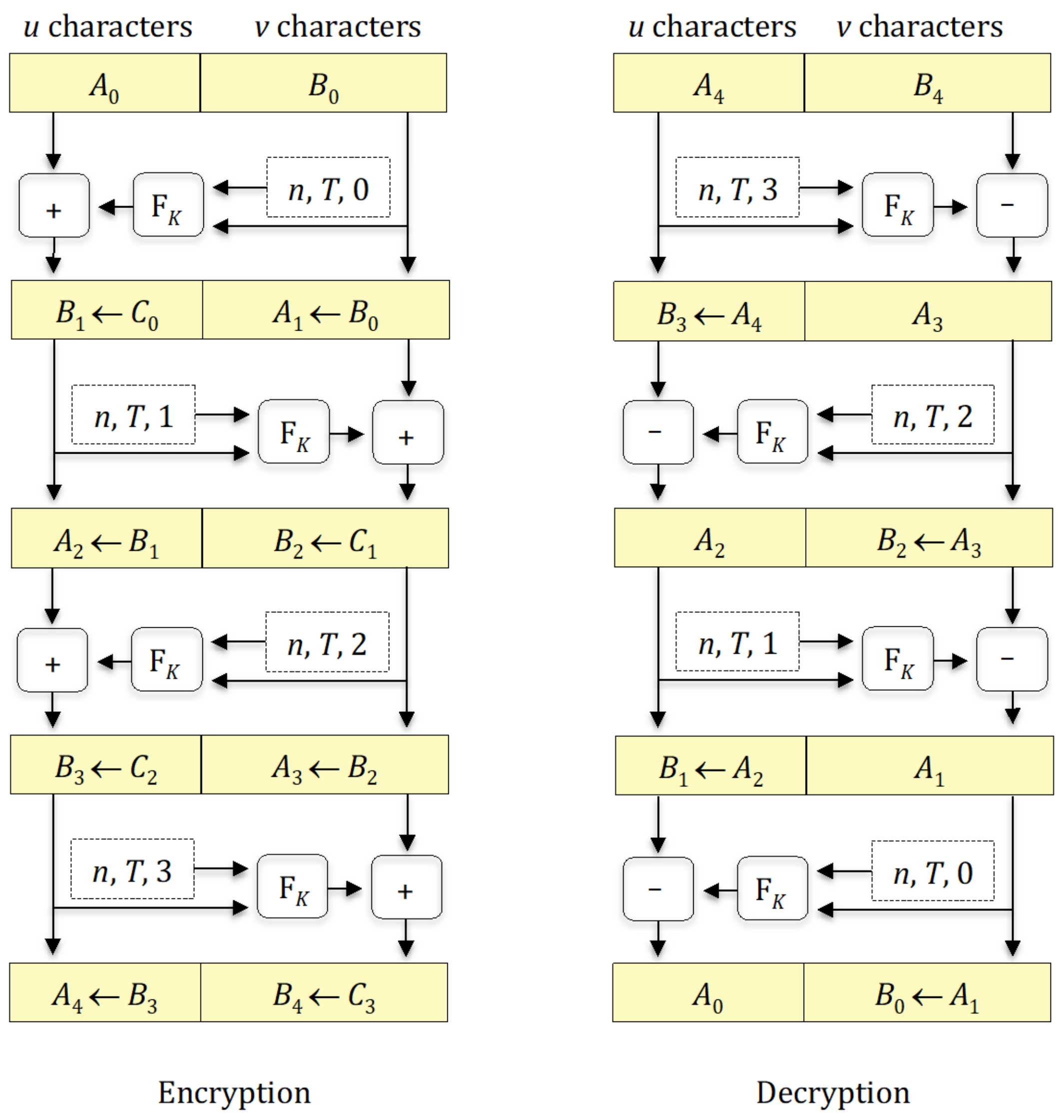

Fig. 1. Example of Feistel structure

For the encryption function example in Fig. 1, the rounds are indexed from 0 to 3. The input and output data for each round are two strings of characters, which will be numerals for FF1. 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.[4] The result is used to modify the other string (denoted by A_i) via modular addition[5] (indicated by +) on the numbers that the strings represent. The string that represents the resulting number is named with a temporary variable C_i. The names of the two parts are swapped for the next round so that the modified A_i (i.e., C_i) becomes B_{i+1}, and B_i becomes A_{i+1}.

[4] In Fig. 1, this triple (n, T, i) of additional inputs is indicated within the dotted rectangles with the appropriate values for i.

[5] For some applications of the Feistel structure other than FF1, the "+" operation may be a different reversible operation on strings that preserves their length. For example, the FFX specification in [4] supports an option for character-wise addition.

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}).
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}.

3.5. Component Functions

This section gives algorithms for the component functions that are called in the specification of FF1. The conversion functions $\text{NUM}_{radix}(X)$, $\text{NUM}(X)$, and $\text{STR}^{m}_{radix}(x)$ are defined in Appendix E and specified in Algorithms 1–3 below. These functions support the ordering convention for the numeral strings in FF1, namely that the first (i.e., leftmost) numeral of the string is the most significant numeral.

The $\text{PRF}(X)$ function specified in Algorithm 4 essentially invokes the Cipher Block Chaining encryption mode [9] 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 $\text{NUM}(X)$ and $\text{PRF}(X)$, the byte or block strings in Algorithms 2 and 4 are represented as bit strings.

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 $\text{LEN}(X)$, 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 $0 \leq x < radix^m$.

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: $\text{PRF}(X)$

Prerequisites:
Forward cipher function, CIPH, of an approved 128-bit block cipher;
Key, K, for the block cipher.

Input:
Block string, X.

Output:
Block, Y.

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

4. Specification of FF1

The specifications of the encryption and decryption algorithm for FF1 are organized into prerequisites, inputs, outputs, steps, and descriptions of the steps. In addition to the key and forward cipher function, the prerequisites 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. Given a length *n* in this range, the number of different input strings to the encryption and decryption functions (i.e., the *domain* size) is $radix^n$. FF1 also has a prerequisite for the choice of the maximum tweak length *maxTlen* that the implementation supports. The requirements on the values for the prerequisites are specified prior to the encryption and decryption algorithms. The 128-bit input and output blocks of the forward block cipher CIPH_K are represented as strings of 16 bytes.

The parameter choices may affect interoperability. The implementation **shall** only accept inputs that are formatted correctly for the given parameters according to this recommendation.

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*, and *maxlen* in FF1.Encrypt and FF1.Decrypt **shall** meet the following requirements:

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

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

Prerequisites:
Forward 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 $b = \lceil \text{BITLEN}(radix^v - 1)/8 \rceil$.
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 10 Feistel rounds of FF1 is initiated in Step 6, executing nine substeps for each round. 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.[7] 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.[8] 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.

---

[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.
[7] In Fig. 1, S corresponds to the output of F_K.
[8] In Figure 1, combining S with A is indicated by the "+" operation.

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

Prerequisites:
Forward 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 $b = \lceil \text{BITLEN}(radix^v - 1)/8 \rceil$.
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 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. Conformance

Implementations of FF1.Encrypt or FF1.Decrypt may be tested for conformance to this recommendation under the auspices of NIST's Cryptographic Algorithm Validation Program [20].

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

In order to claim conformance with this recommendation, an implementation of FF1 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 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 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.

The use of floating-point arithmetic may lead to incorrect results due to a possible loss of precision. To avoid such issues, the algorithms in this standard **shall** not be implemented with floating-point representations nor floating-point arithmetic.

References

[1] Barker E, Mouha N (2017) Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-67r2. https://doi.org/10.6028/NIST.SP.800-67r2

[2] Barker E, Roginsky A, Davis R (2020) Recommendation for Cryptographic Key Generation. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-133r2. https://doi.org/10.6028/NIST.SP.800-133r2

[3] Bellare M, Hoang VT, Tessaro S (2016) Message-Recovery Attacks on Feistel-Based Format Preserving Encryption. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2016)* (ACM, Vienna, Austria), pp. 444–455. https://doi.org/10.1145/2976749.2978390

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

[5] Bellare M, Rogaway P, Spies T (2010) 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. Available at https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffx/ffx-spec2.pdf

[6] Beyne T (2021) Linear Cryptanalysis of FF3-1 and FEA. Advances in Cryptology (CRYPTO 2021), eds Malkin T and Peikert C (Springer, Santa Barbara, CA), Lecture Notes in Computer Science 12825, pp. 41–69. https://doi.org/10.1007/978-3-030-84242-0_3

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

[8] de Montjoye Y-A, Radaelli L, Singh VK, Pentland A "S" (2015) Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science* 347(6221), pp. 536–539. https://doi.org/10.1126/science.1256297

[9] Dworkin M (2001) Recommendation for Block Cipher Modes of Operation: Methods and Techniques. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-38A. https://doi.org/10.6028/NIST.SP.800-38A

[10] Dworkin M, Perlner R (2015) Analysis of VAES3 (FF2). *Cryptology ePrint Archive preprint*, Report 2015/306. Available at https://eprint.iacr.org/2015/306

[11] Durak FB, Vaudenay S (2017) Breaking the FF3 Format-Preserving Encryption Standard over Small Domains. *Advances in Cryptology (CRYPTO 2017)*, eds Katz J, Shacham H (Springer, Santa Barbara, CA), *Lecture Notes in Computer Science* 10402, pp. 679–707. https://doi.org/10.1007/978-3-319-63715-0_23

[12] Durak FB, Vaudenay S (2018) Generic Round-Function-Recovery Attacks for Feistel Networks over Small Domains. *Applied Cryptography and Network Security (ACNS 2018)*, eds Preneel B, Vercauteren F (Springer, Leuven, Belgium), *Lecture Notes in*

Computer Science 10892, pp. 440–458. Available at https://doi.org/10.1007/978-3-319-93387-0_23

[13] Hoang VT, Miler D, Trieu N (2019) Attacks only Get Better: How to Break FF3 on Large Domains. *Advances in Cryptology (EUROCRYPT 2019)*, eds Ishai Y, Rijmen V (Springer, Darmstadt, Germany), *Lecture Notes in Computer Science* 11477, pp. 85–116. https://doi.org/10.1007/978-3-030-17656-3_4

[14] Hoang VT, Tessaro S, Trieu N (2018) The Curse of Small Domains: New Attacks on Format-Preserving Encryption. *Advances in Cryptology (CRYPTO 2018)*, eds Schacham H, Boldyreva A (Springer, Santa Barbara, CA), *Lecture Notes in Computer Science* 10991, pp. 221–251. https://doi.org/10.1007/978-3-319-96884-1_8

[15] Liskov M, Rivest RL, Wagner D (2002) Tweakable Block Ciphers. *Advances in Cryptology (CRYPTO 2002)*, ed Yung M (Springer, Santa Barbara, CA), *Lecture Notes in Computer Science* 2442, pp. 31–46. https://doi.org/10.1007/3-540-45708-9_3

[16] Luby M, Rackoff C (1988) How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing* 17(2) pp. 373–386. https://doi.org/10.1137/0217022

[17] National Institute of Standards and Technology (2001; Updated 2023) Advanced Encryption Standard (AES). (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publications (FIPS PUBS) NIST FIPS 197, November 2001; Updated May 2023, https://doi.org/10.6028/NIST.FIPS.197-upd1

[18] National Institute of Standards and Technology (2014) Explanation of changes to Draft SP 800-38G. Available at https://csrc.nist.gov/news/2014/explanation-of-changes-to-draft-sp-800-38G

[19] National Institute of Standards and Technology (2019) Public Comments on the Draft SP 800-38G Rev. 1. Available at https://csrc.nist.gov/CSRC/media/Publications/sp/800-38g/rev-1/draft/documents/sp800-38gr1-draft-comments-received.pdf

[20] National Institute of Standards and Technology (2023) Cryptographic Algorithm Validation Program (CAVP). Available at https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program

[21] Patarin J (2008) Generic Attacks on Feistel Schemes. *Cryptology ePrint Archive preprint*, Report 2008/036. Available at https://eprint.iacr.org/2008/036

[22] Schroeppel R (1999) Hasty Pudding Cipher Specification (Revised). Available at http://richard.schroeppel.name/hpc/hpc-spec

Appendix A. Parameter Choices and Security

The quantity $radix^{minlen}$ is a lower bound on the domain size, which affects the resistance that FF1 can offer to plaintext guessing attacks (discussed in Appendix A.1) and a variety of cryptanalytic attacks (discussed in Appendix A.2).

Two other potential parameters of the Feistel structure are fixed for FF1, namely the number of Feistel rounds and the imbalance (i.e., the values of the lengths *u* and *v* in Fig. 1). Both of these parameters were set with consideration to both performance and security requirements. See Appendix H of [4] for a discussion.

A.1. Plaintext Guessing Attacks

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 one specific plaintext with probability $1/radix^{\text{LEN}(C)}$ by selecting a numeral string of LEN(*C*) at random. Repeated guesses proportionally increase the attacker's probability of success: with *g* distinct guesses, the probability is $g/radix^{\text{LEN}(C)}$.

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

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 where feasible, as discussed in Appendix C. Usually, tweaks are non-secret information that can be associated with instances of encryption. For FF1, the maximum tweak length parameter *maxTlen* should be chosen to accommodate the desired tweaks for the implementation.

A.2. Analytic Attacks

For any symmetric key cryptographic algorithm, the length of the secret key should be specified to be large enough to render the brute force search of the key as extremely unlikely to succeed. Ideally, no other significant cryptanalytic attack would require fewer computational resources (i.e., time complexity) in order to succeed with comparable probability. The amount of data protected by the secret key does not play a significant role in the brute force search. The attacker may only require one or two plaintext-ciphertext pairs.

Research in recent years [3][6][11][12][13][14] has shown that cryptanalytic attacks exist for the FPE modes in this publication when the domain is sufficiently small.[9] In this case, the computational resources required for the attacks are not prohibitive. The security against the attacks in those papers depends on the data complexity (i.e., difficulty of compromising enough data protected by the key), often with a particular pattern or structure (e.g., plaintext-

[9] The fundamental problem seems to be that — unlike the previously approved encryption modes — for FF1, the attacker may plausibly compromise a significant fraction of the domain.

ciphertext pairs). Hoang summarized the data complexity for the attacks in the above papers in a public comment [19] on an earlier draft of this publication. In particular, for the domain size of 10^6 required in this publication, the data complexity for an attack to recover a single target message would be at least 2^{77} for FF1, which is prohibitive.

Appendix B. Security Goal

The designers of FFX intended to achieve strong pseudorandom permutation (PRP) security for a conventional block cipher [16]. In the FFX proposal to NIST [4], the designers of FFX cited the history of cryptographic results concerning Feistel networks as underlying their selection of the FFX mechanism. Under the assumption that the underlying round function is a good pseudorandom function (PRF), they asserted that 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 [4].

Appendix C. Tweaks

Tweaks have been supported in stand-alone block ciphers (e.g., Schroeppel's Hasty Pudding Cipher [22]), and the notion was later formalized and investigated by Liskov, Rivest, and Wagner [15]. 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 100 distinct CCNs would be expected to share each possible value for these six digits. If the 100 CCNs that shared a given value for the middle six digits were encrypted with the same tweak, then their ciphertexts would be the same. However, if the other 10 digits had been the tweak for the encryption of the middle six digits, then the 100 ciphertexts would almost certainly be different.

Similarly, about 100 CCNs in the encrypted database would be expected to share each possible value for the ciphertext (i.e., the middle six digits). If the 100 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 100 CCNs would reveal the others.

However, if 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, learning that the decryption of 111111-770611-1111 is 111111-123456-1111, for example, 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 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. In the security model described in [4], this corresponds to fewer queries to the target instance of the encryption.

Appendix D. Examples

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

Appendix E. List of Symbols, Abbreviations, and Acronyms

AES
Advanced Encryption Standard

BITLEN(*x*)
Given a positive integer *x,* the bit length of its representation $\text{STR}_2^m(x)$ as a bit string, where *m* is the unique integer such that $2^{m-1} \leq x < 2^m$. For example, BITLEN(64) = 7 and BITLEN(10) = 4.

CAVP
Cryptographic Algorithm Validation Program

$\text{CIPH}_K(X)$
The output of the forward cipher function of the block cipher under the key *K* applied to the block *X*.

CMVP
Cryptographic Module Validation Program

FPE
Format-Preserving Encryption

IETF
Internet Engineering Task Force

LEN(*X*)
The number of numerals [bits] in a numeral [bit] string *X*. For example, LEN(`010`) = 3.

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. 3.5.

$\text{NUM}_{radix}(X)$
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 $\text{NUM}_{radix}(X)$ is given in Sec. 3.5.

PRF
Pseudorandom Function

PRF(*X*)
The output of the function PRF applied to the block *X*. PRF is defined in terms of the forward cipher function.

PRP
Pseudorandom Permutation

$\text{STR}_{radix}^m(x)$
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, $\text{STR}_{12}^4(559)$ is the string of four numerals in base 12 that represents 559, namely, 0 3 10 7. An algorithm for computing $\text{STR}_{radix}^m(x)$ is given in Sec. 3.5.

$\lfloor x/y \rfloor$
Integer division, rounded down: given positive integers *x* and *y*, the nonnegative integer *q* such that $x = q\,y + r$ for a nonnegative integer $r < y$. For example, $\lfloor 5/2 \rfloor = 2$, and $\lfloor 6/2 \rfloor = 3$.

$\lceil x/y \rceil$
Integer division, rounded up: given positive integers *x* and *y*, the nonnegative integer *q* such that $x = q\,y - r$ for a nonnegative integer $r < y$. For example, $\lfloor 5/2 \rfloor = 3$, and $\lfloor 6/2 \rfloor = 3$.

$[x]^s$

Given a nonnegative integer x less than 256^s, the representation of x as a string of s bytes. For example, $[5]^2$ = `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* mod *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 $1 \leq i \leq \text{LEN}(X)$, the i^{th} numeral [bit] of X. For example, in base 10, 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 10, 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`.

Appendix F. Glossary

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. 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.

domain
The set of inputs to a function.

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 (i.e., rounds) in which a keyed function (i.e., 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
See *block cipher*.

mode
See *block cipher mode of operation*.

numeral
For a given base, a non-negative 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.

tweak
The input parameter to the encryption and decryption functions whose confidentiality is not necessarily protected by the mode.

Appendix G. Change Log

A second mode — FF2 (submitted to NIST under the name VAES3) — was included in the initial draft of this publication in 2016. As part of the public review of SP 800-38G ipd and routine consultation with other agencies, NIST was advised by the National Security Agency (NSA) 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 [10] and announced the removal of FF2 in [18].

A third mode — FF3 (submitted to NIST under the name BPS [7]) — was included in the first version of this publication. An attack by Beyne [6] on both FF3 and FF3-1 (a backward-compatible variant with a restricted tweak input that was proposed in SP 800-38Gr1 ipd) led to the removal of FF3.

The original specifications of FF1 imposed only a modest absolute minimum of 100 on the domain size in order to preclude a generic meet-in-the-middle attack on the Feistel structure [21] with a recommendation for $radix^{minlen}$ to be greater than or equal to 1 000 000. In response to the published analysis in [3][12][14], this recommendation was strengthened to a requirement in Revision 1.

The name "FF1" has not changed from the first 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.

The original specification contained a LOG() function (in step 3 of Algorithms 5 and 6) that could sometimes result in incorrect values due to a loss of precision when implemented using floating-point arithmetic. For this reason, the current specification has been rewritten in terms of the BITLEN() function, and floating-point arithmetic is disallowed for implementations of the algorithms in this standard.

The original specification defined CIPH_K as the "designated cipher function," which permitted CIPH_K to be instantiated as either the forward transformation or the inverse transformation of the underlying block cipher. The current specification requires CIPH_K to be instantiated as the forward transformation to promote interoperability and consistency with [5].