The FFX Mode of Operation for Format-Preserving Encryption

Draft 1.1

Mihir Bellare[1] **Phillip Rogaway**[2] **Terence Spies**[3]

February 20, 2010

1 Introduction

This specification document describes FFX, a mechanism for *format-preserving encryption* (FPE).

Schemes for FPE enable one to encrypt Social Security numbers (SSNs), credit card numbers (CCNs), and the like, doing so in such a way that the ciphertext has the same format as the plaintext. In the case of SSNs, for example, this means that the ciphertext, like the plaintext, consists of a nine decimal-digit string. Similarly, encryption of a 16-digit CCN results in a 16-digit ciphertext. FPE is rapidly emerging as a useful cryptographic tool, with applications including financial-information security, data sanitization, and transparently encrypting fields in a legacy database.

The encryption algorithm of FFX takes in a key K, a plaintext X, and a tweak T. The plaintext is taken over an arbitrary alphabet Chars. Assuming that $n = |X|$ is a supported length, FFX.Encrypt will produce—deterministically—a ciphertext $Y = \text{FFX.Encrypt}_K^T(X) \in \mathsf{Chars}^n$. One can recover X from Y by way of $X = \text{FFX.Decrypt}_K^T(X)$.

FFX mode is flexible and customizable. In particular, unrepresented in the explicitly named arguments to FFX.Encrypt and FFX.Decrypt is the fact that FFX depends on a number of *parameters*. Once chosen, it is assumed that they are held fixed for the lifetime of a given user-generated key. The parameters used in FFX include the number of Feistel rounds $\mathsf{rnds}(n)$, the desired degree of imbalance $\mathsf{split}(n)$ in the Feistel network, and the round function F.

As example instantiations of FFX, we consider enciphering (i) binary strings of 8–128 bits, or (ii) decimal strings of 4–36 digits. In Appendices A and B we specify parameter sets, denoted A2 and A10, to enable these task. Both employ a round function derived from AES. The fully instantiated schemes would be denoted FFX-A2 and FFX-A10.

[1] **University of California, San Diego**, Dept. of Computer Science & Engineering, 9500 Gilman Drive, La Jolla, CA 92093, USA. e-mail: mihir@cs.ucsd.edu, url: http://cseweb.ucsd.edu/∼mihir/. Bellare's work on this spec has been done in collaboration with **Semtek Innovative Solutions Corp**; please see the acknowledgments on p. 6.

[2] **University of California, Davis**, Dept. of Computer Science, One Shields Avenue, Davis, CA 95616, USA. e-mail: rogaway@cs.ucdavis.edu, url: http://www.cs.ucdavis.edu/∼rogaway/. Rogaway's work on this spec has been done in collaboration with **Voltage Security**; please see the acknowledgments on p. 6.

[3] **Voltage Security**, 4005 Miranda Ave, Suite 210, Palo Alto, CA 94304, USA. email: terence@voltage.com. Spies is the Chief Technology Officer at Voltage.

The name *FFX* is meant to suggest *Format-preserving, Feistel-based encryption.* The X reflects there being multiple instantiations (that is, parameter choices). It further reflects that FFX is an outgrowth of, and extension to, the FFSEM specification earlier submitted to NIST by Spies [28]. The current draft replaces that contribution. Compared to it, FFX is more general, adding in support for tweaks, non-binary alphabets, and non-balanced splits. Cycle-walking can now be avoided in the setting of primary practical importance, encrypting decimal strings.

While this document does not attempt to explain or survey all of the cryptographic results relating to FFX, their existence—and indeed the entire history of results concerning Feistel networks—underlies our mechanism's selection. In general, contemporary cryptographic results and experience indicate that FFX achieves cryptographic goals including nonadaptive message-recovery security, chosen-plaintext, and even PRP-security against an adaptive chosen-ciphertext attack. The quantitative security depends on the number of rounds used, the imbalance, and the adversary's access to plaintext/ciphertext pairs. One assumes that the underlying round function is a good pseudorandom function (PRF).

While FFX can, in principle, be used to encipher character strings of arbitrary length, the mechanism is intended for message spaces smaller than that of AES (2^{128} points). For enciphering longer strings, other techniques would seem to be preferable. In particular, EME2 [9] can encipher binary strings of any length $n > 128$.

An earlier version of this specification document, version 1.0, was provided to NIST in November 2009. The substantive change we have made since that version is the addition of a parameter profile for binary strings, A2.

2 Definition of FFX

See Figure 1 for an illustration of FFX, Figure 2 for a description of the parameters on which FFX depends, and Figure 3 for the the definition of FFX in terms of these parameters. We expect all parameter choices to be fixed for the lifetime of a given key. Encryption and decryption must use the same parameters.

3 Notation

Throughout this document, a *number* means a nonnegative integer. Plaintexts and ciphertexts are regarded as strings over an alphabet $\mathsf{Chars} = \{0, 1, \ldots, \mathsf{radix} - 1\}$. Members of the alphabet are called *characters.* The number of characters radix in Chars is referred to as the *radix* of the alphabet. Example radix values are 2, 10, and 26, corresponding to bits, digits, and uppercase English letters. It is required that $\mathsf{radix} \geq 2$.

If a user wishes to encrypt over a non-numeric alphabet, say $\{\mathsf{a}, \ldots, \mathsf{z}\}$, she must set up a bijective mapping between this alphabet and $\mathsf{Chars} = \{0, \ldots, \mathsf{radix} - 1\}$ via which her inputs and outputs can be regarded as numeric values, as required for our algorithms.

A *string* is a finite sequence of characters from Chars. By $|\,X\,|$ we denote the length of string X, the number of characters in it. For example, $X = 00326$ is a string of length $|\,00326\,| = 5$. Note

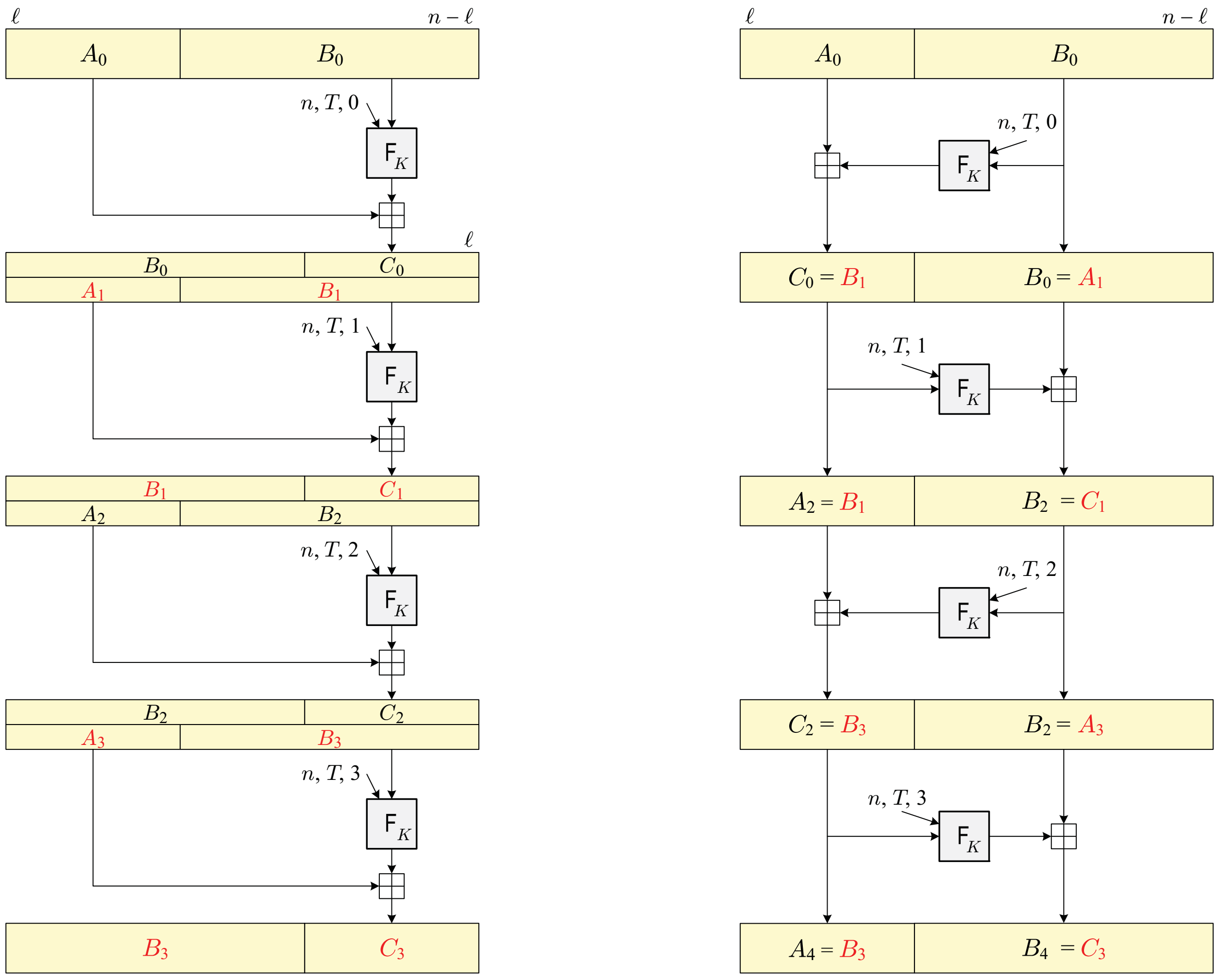

Figure 1: **Illustration of FFX encryption when** $\mathsf{method} = 1$ **(left) and** $\mathsf{method} = 2$ **(right).** The first four rounds are shown. The divided boxes on the left are used to illustrate the re-partitioning of a string; for example, string B_0C_0 is exactly the string A_1B_1, where $|C_0| = |A_1| = \ell$ and $|B_0| = |B_1| = n - \ell$. No such re-partitioning occurs on the right, but strings get two names instead. All boxed strings are over $\mathsf{Chars} = \{0, 1, \ldots, \mathsf{radix} - 1\}$, while T is a byte string, $n \geq 2$ is a number, and $1 \leq \ell \leq n - 1$ is the imbalance.

that leading zeros are counted just like any other character. By Chars^* we mean the set of strings over Chars having any length. If $X, Y \in \mathsf{Chars}^*$ are strings we let XY or $X \parallel Y$ denote their concatenation. The i-th digit of a string X will be denoted $X[i]$, for any $i \in \{1, \ldots, |X|\}$. For $1 \leq i \leq j \leq |X|$ we let $X[i\,..\,j] = X[i] \cdots X[j]$.

The function $\boxplus$ takes a pair of equal-length strings and returns a string of the same length. Two possibilities are allowed: *characterwise addition* and *blockwise addition*. For characterwise addition, $a_1 \cdots a_n \boxplus b_1 \cdots b_n = c_1 \cdots c_n$ where $c_i = (a_i + b_i) \bmod \mathsf{radix}$. For blockwise addition, $c_1 \cdots c_n$ is instead the unique string such that $\sum c_i \mathsf{radix}^{n-i} = \left(\sum a_i \mathsf{radix}^{n-i} + \sum b_i \mathsf{radix}^{n-i}\right) \bmod \mathsf{radix}^n$. For example, when $\mathsf{radix} = 10$, characterwise addition would have $439 \boxplus 724 = 153$ while blockwise addition would result in $439 \boxplus 724 = 163$. The function $\boxminus$ correspondingly takes a pair of equal-length strings and returns a string of the same length. It is determined by saying that $X \boxminus Y$ is the

parameter	**description**
radix	The *radix*, a number $\mathsf{radix} \geq 2$ that determines the alphabet $\mathsf{Chars} = \{0, \ldots, \mathsf{radix} - 1\}$. Plaintexts and ciphertexts are strings of characters from Chars.
Lengths	The set of *permitted message lengths.* For a plaintext to be encrypted, or for a ciphertext to be decrypted, its length must be in this set.
Keys	The *key space*, a finite nonempty set of binary strings.
Tweaks	The *tweak space*, a nonempty set of strings. Conceptually, different tweaks name unrelated encryption mappings.
addition	The *addition operator*, either 0 (characterewise addition) or 1 (blockwise addition). Determines the meaning of the operators $X \boxplus Y$ and $X \boxminus Y$ that add or subtract equal-length strings over the alphabet $\mathsf{Chars} = \{0, 1, \ldots, \mathsf{radix} - 1\}$.
method	The *Feistel method*, either 1 or 2. The value determines which of the two prominent Feistel variants will be used.
split (n)	The *imbalance*, a function that takes a permitted length $n \in \mathsf{Lengths}$ and returns a number $1 \leq \mathsf{split}(n) \leq n/2$.
rnds (n)	The *number of rounds*, a function that takes a permitted length $n \in \mathsf{Lengths}$ and returns an even number $\mathsf{rnds}(n)$.
F	The *round function*, a function that takes in a key $K \in \mathsf{Keys}$, a permitted length $n \in \mathsf{Lengths}$, a tweak $T \in \mathsf{Tweaks}$, a round number $i \in \{0, \ldots, \mathsf{rnds}(n) - 1\}$, and a string $B \in \mathsf{Chars}^*$. It returns a string $\mathsf{F}_K(n, T, i, B) \in \mathsf{Chars}^*$. If $\mathsf{method} = 1$ or i is even then $\lvert B \rvert = n - \mathsf{split}(n)$ and $\lvert \mathsf{F}_K(n, T, i, B) \rvert = \mathsf{split}(n)$. If $\mathsf{method} = 2$ and i is odd then $\lvert B \rvert = \mathsf{split}(n)$ and $\lvert \mathsf{F}_K(n, T, i, B) \rvert = n - \mathsf{split}(n)$.

Figure 2: **Parameters of FFX.** To have a fully-specified scheme, each of these parameters must be defined.

unique string Z such that $Y \boxplus Z = X$. As an example, still with $\mathsf{radix} = 10$, we have $32 \boxminus 15 = 27$ for characterwise addition and $32 \boxminus 15 = 17$ for blockwise addition. Note that when the radix is two, characterwise addition and subtraction are the same as XOR.

We expect radix and Lengths to be determined by the needs of the application, not by security considerations. The choice of Keys will typically flow from the underlying cryptographic primitive employed; for example, the key space might consist of AES keys if one uses AES to construct the round function. The set Tweaks should be large enough to accommodate all non-secret information that may be associated to a plaintext. Users are strongly encouraged to employ tweaks whenever possible, as their judicious use can significantly enhance security. See Appendix F. The addition parameter specifies the group over which addition is performed. Efficiency considerations determine its choice; we do not expect the value to be security relevant.

In specifying a collection of parameters one must choose rnds, as well as method and split, in order to balance performance requirements and security considerations. See Appendix H for a discussion. To avoid known attacks, we require that $\mathsf{rnds}(n) \geq 8$ if $n = 2 \cdot \mathsf{split}(n)$ or if $\mathsf{method} = 2$ and $n = 2 \cdot \mathsf{split}(n) + 1$, and we require that $\mathsf{rnds}(n) \geq 4n/\mathsf{split}(n)$ otherwise. We emphasize that these values are minimums, not recommended values. We insist that $\mathsf{radix}^n \geq 100$. This last requirement is to prevent the meet-in-the-middle attack discussed in Appendix H.

The round function $\mathsf{F}_K(n, T, i, B)$ must be constructed from a blockcipher E or a hash function H. We recommend AES for the former. Options for an AES-based round function include

```
**algorithm** FFX.Encrypt$(K, T, X)$
**if** $K \notin$ Keys **or** $T \notin$ Tweaks **or** $X \notin$ Chars$^*$ **or** $|X| \notin$ Lengths **then return** $\perp$
$n \leftarrow |X|$;  $\ell \leftarrow$ split$(n)$;  $r \leftarrow$ rnds$(n)$

**if** method $= 1$ **then**
**for** $i \leftarrow 0$ **to** $r - 1$ **do**
    $A \leftarrow X[1 .. \ell]$; $B \leftarrow X[\ell + 1 .. n]$
    $C \leftarrow A \boxplus \mathsf{F}_K(n, T, i, B)$
    $X \leftarrow B \,\|\, C$
**return** $X$
**end if**

**if** method $= 2$ **then**
$A \leftarrow X[1 .. \ell]$;  $B \leftarrow X[\ell + 1 .. n]$
**for** $i \leftarrow 0$ **to** $r - 1$ **do**
    $C \leftarrow A \boxplus \mathsf{F}_K(n, T, i, B)$
    $A \leftarrow B$;  $B \leftarrow C$
**return** $A \,\|\, B$
**end if**
```

```
**algorithm** FFX.Decrypt$(K, T, Y)$
**if** $K \notin$ Keys **or** $T \notin$ Tweaks **or** $Y \notin$ Chars$^*$ **or** $|Y| \notin$ Lengths **then return** $\perp$
$n \leftarrow |Y|$;  $\ell \leftarrow$ split$(n)$;  $r \leftarrow$ rnds$(n)$

**if** method $= 1$ **then**
**for** $i \leftarrow r - 1$ **downto** $0$ **do**
    $B \leftarrow Y[1 .. n - \ell]$;  $C \leftarrow Y[n - \ell + 1 .. n]$
    $A \leftarrow C \boxminus \mathsf{F}_K(n, T, i, B)$
    $Y \leftarrow A \,\|\, B$
**return** $Y$
**end if**

**if** method $= 2$ **then**
$A \leftarrow Y[1 .. \ell]$;  $B \leftarrow Y[\ell + 1 .. n]$
**for** $i \leftarrow r - 1$ **downto** $0$ **do**
    $C \leftarrow B$;  $B \leftarrow A$
    $A \leftarrow C \boxminus \mathsf{F}_K(n, T, i, B)$
**return** $A \,\|\, B$
**end if**
```

Figure 3: **Definition of FFX.** The meaning of $\boxplus$ and $\boxminus$ is determined by the parameter addition, which may be either 0 (characterwise addition) or 1 (blockwise addition). Here and elsewhere, Chars $= \{0, 1, \ldots, \text{radix}\}$ is the underlying alphabet and Lengths $= \{\text{minlen}, \ldots, \text{maxlen}\}$ are the permitted message lengths.

the CBC MAC [10] and CMAC [20]. When using a hash function, the PRF construction could be based on HMAC [19].

For example parameter collections see Appendices A and B. We believe these parameter collections to be useful and illustrative, but they are not meant to be exclusive.

An implementation of FFX is considered to be in conformance with a specified parameter collection $\mathcal{A}$ if the message space associated to the implementation is an arbitrary but specified subset of that associated to $\mathcal{A}$. Of course the functionality of the implementation on the domain where it is defined must be identical to that called for by $\mathcal{A}$. As an example, an implementation of FFX-A10 that requires the plaintext or ciphertext to be exactly 16 decimal digits, or that requires the tweak to be 4 to 8 bytes, could be in conformance with FFX-A10 even though FFX-A10 allows encryption over a broader message space and tweak space. This is no different from, say, allowing a conforming implementation of SHA-1 to assume that its input is a byte string, whereas the specification for SHA-1 allows any bit string, instead.

4 Acknowledgments

The authors gratefully acknowledge the advice, assistance, and participation of

Lance Nakamura — Semtek Innovative Solutions Corporation, VP of Product Development
Joachim Vance — Semtek Innovative Solutions Corporation, Director of Product Security
Viet Tung Hoang — University of California at Davis, Dept. of Computer Science
Ben Morris — University of California at Davis, Dept. of Mathematics
Tom Ristenpart — University of California at San Diego, Dept. of Comp. Science & Eng.
Till Stegers — University of California at Davis, Dept. of Computer Science

Nakamura and Vance closely examined all aspects of the design, implemented numerous versions of the spec as it has evolved, and provided Bellare with valuable feedback. Alternating Feistel might not have been supported without their insight into the relative software performance of method-1 and method-2 schemes. In work closely tied to the development of this spec, Morris and then Hoang developed coupling-based analyses for generalized Feistel schemes, including method-1 and method-2 Feistel networks with arbitrary splits and non-binary alphabets [8, 16]. Hoang's insights and careful readings of this spec are much appreciated. Stegers worked on the earliest versions of this spec, including doing an implementation. Ristenpart provided insightful readings and criticism. We would like to additionally thank Niels Ferguson Pierre-Alin Fouque, Jacques Patarin, and Hovav Shacham for all of their feedback.

Mihir Bellare gratefully acknowledges his support from and collaboration with Semtek, while Phil Rogaway gratefully acknowledges his support from and collaboration with Voltage.

References

[1] M. Bellare. New Proofs for NMAC and HMAC: Security without collision-resistance. *CRYPTO 2006*, LNCS vol. 4117, Springer, pp. 602–619, 2006.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *CRYPTO 1996*, LNCS vol. 1109, Springer, pp. 1–15, 1996.

[3] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. *Selected Areas in Cryptography* (SAC 2009), LNCS 5867, Springer, 2009. Also ePrint report 2009/251.

[4] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. *RSA Data Security Conference, Cryptographer's Track* (RSA CT '02). LNCS vol. 2271, pp. 114–130, Springer, 2002.

[5] M. Brightwell and H. Smith. Using datatype-preserving encryption to enhance data warehouse security. *20th National Information Systems Security Conference Proceedings* (NISSC), pp. 141–149, 1997. csrc.nist.gov/nissc/1997/proceedings/141.pdf.

[6] L. Fibíková. Provable secure scalable block ciphers. Ph.D. dissertation, University of Duisburg–Essen, 2003. Available through http://www.d-nb.de/.

[7] L. Granboulan and T. Pornin. Perfect block ciphers with small blocks. *Fast Software Encryption* (FSE 2007), LNCS 4593, Springer, pp. 452-465, 2007.

[8] V. Hoang and P. Rogaway. On generalized Feistel networks. Manuscript, February 2010.

[9] IEEE P1619.2. Draft standard architecture for wide-block encryption for shared storage media. 2008. Available from https://siswg.net.

[10] ISO/IEC 9797-1:1999. Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher. International Organization for Standardization / International Electrotechnical Commission, 1999.

[11] M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. *CRYPTO 2002*, LNCS vol. 2442, Springer, pp. 31–46, 2002

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

[13] S. Lucks. Faster Luby-Rackoff ciphers. *Fast Software Encryption* (FSE 1996), LNCS vol. 1039, Springer, pp. 180–203, 1996.

[14] H. Luhn. Computer for verifying numbers. US Patent #2,950,048. August 23, 1960.

[15] U. Maurer and K. Pietrzak. The security of many-round Luby-Rackoff pseudo-random permutations. *EUROCRYPT 2003*, LNCS vol. 2656, Springer, pp. 544–561, 2003.

[16] B. Morris, P. Rogaway, and T. Stegers. How to encipher messages on a small domain: deterministic encryption and the Thorp shuffle. *CRYPTO 2009.* LNCS, Springer, 2009.

[17] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1), pp. 29-66, 1999.

[18] National Bureau of Standards [USA]. FIPS PUB 74. Guidelines for implementing and using the NBS Data Encryption Standard. 1981.

[19] National Institute of Standards [USA]. FIPS 198. The keyed-hash message authentication code (HMAC). 2002.

[20] National Institute of Standards [USA]. NIST Special Publication 800-38B. Recommendation for cipher block modes of operation: the CMAC mode for authentication. M. Dworkin, 2005.

[21] J. Patarin. Generic attacks on Feistel schemes. Cryptology ePrint report 2008/036. January 24, 2008. Also *ASIACRYPT 2001*, LNCS vol. 2248, Springer, pp. 222–238, 2001.

[22] J. Patarin. Security of random Feistel schemes with 5 or more rounds. *CRYPTO 2004*, LNCS vol. 3152, Springer, pp. 106–122, 2004.

[23] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, 13(3), pp. 315–338, 2000.

[24] PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard: Requirements and Security Assessment Procedures, Version 1.2.1. July 2009.

[25] B. Schneier and J. Kelsey. Unbalanced Feistel networks and block-cipher design. *Fast Software Encryption* (FSE 1996), LNCS vol. 1039, Springer, pp. 121–144, 1996

[26] R. Schroeppel. Hasty Pudding Cipher specification. Unpublished manuscript, available at http://richard.schroeppel.name:8015/hpc/hpc-spec. June 1998 (revised May 1999).

[27] Social Security Number Verification Service (SSNVS). Web page maintained by the U.S. Social Security Administration, http://www.socialsecurity.gov/employer/stateweb.htm.

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

[29] E. Thorp. Nonrandom shuffling with applications to the game of Faro. *Journal of the American Statistical Association*, 68, pp. 842–847, 1973.

parameter	value	comment
radix	2	alphabet is Chars $= \{0,1\}$
Lengths	[minlen .. maxlen] where minlen $= 8$, maxlen $= 128$	permissible message lengths
Keys	$\{0,1\}^{128}$	128-bit AES keys
Tweaks	$\text{Byte}^{\le M}$ where $M = 2^{64}-1$	tweaks are arbitrary byte strings
addition	0	characterwise addition (xor)
method	2	alternating Feistel
split (n)	$\lfloor n/2 \rfloor$	maximally balanced Feistel
rnds (n)	$\begin{cases} 12 & \text{if } 32 \le n \le 128, \\ 18 & \text{if } 20 \le n \le 31, \\ 24 & \text{if } 14 \le n \le 19, \\ 30 & \text{if } 10 \le n \le 13, \text{ and} \\ 36 & \text{if } 8 \le n \le 9 \end{cases}$	from entropy-based heuristic
F	defined below	AES-based round function

```
100 algorithm $\mathsf{F}_K(n, T, i, B)$
101 $\text{VERS} \leftarrow 1;\ t \leftarrow |T|_8$
102 $P \leftarrow [\text{VERS}]^2 \,\|\, [\mathsf{method}]^1 \,\|\, [\mathsf{addition}]^1 \,\|\, [\mathsf{radix}]^1 \,\|\, [n]^1 \,\|\, [\mathsf{split}(n)]^1 \,\|\, [\mathsf{rnds}(n)]^1 \,\|\, [t]^8$
103 $Q \leftarrow T \,\|\, [0]^{-t-9 \bmod 16} \,\|\, [i]^1 \,\|\, 0^{64-|B|} \,\|\, B$
104 $Y \leftarrow \text{CBC-MAC}_K(P \,\|\, Q)$
105 if $\text{EVEN}(i)$ then $m \leftarrow \mathsf{split}(n)$ else $m \leftarrow n - \mathsf{split}(n)$
106 return $Y[129-m \,..\, 128]$
```

Figure 4: **Parameter collection A2.** Mechanism FFX-A2 enciphers binary strings of up to 128 bits. It does so using maximally balanced Feistel and a round function based on the AES CBC-MAC.

A Parameter Collection A2

Notation. We will employ the following new notation. **(1)** $[s]^i$ is the i-byte string that encodes the number $s \in [0 \,..\, 2^{8i}-1]$ (for example, $[6]^1 = 00000110$). **(2)** Byte denotes $\{0,1\}^8$, the set of 8-bit bytes. **(3)** $|T|_8 = |T|/8$ is the length, in bytes, of the byte string T. **(4)** $\text{EVEN}(i)$ is the predicate that is TRUE if i is even and FALSE otherwise. **(5)** When $K \in \{0,1\}^{128}$ and $X \in \{0,1\}^*$ and $|X|$ is divisible by 128 , algorithm $\text{CBC-MAC}_K(X)$ is defined as follows. First, let $X_1 \cdots X_m \leftarrow X$ where $|X_i| = 128$ and let $Y \leftarrow 0^{128}$. Then, for $j \leftarrow 1$ **to** m, set $Y \leftarrow \text{AES}_K(Y \oplus X_i)$. Finally, return Y.

Specification. We specify the parameter collection we call A2. When FFX is instantiated with parameter collection A2 one obtains the scheme FFX-A2. With it one can encipher binary strings of 8 to 128 bits. The underlying mechanism in FFX-A2 is a maximally-balanced alternating Feistel scheme with an AES-based round function. The definition of A2 is given in Figure 4.

Implementation notes. An implementation of F needs fewer AES calls than a quick inspection of the code might suggest. Suppose first that an application will, with a given key K, encipher messages of only one length n using tweaks of only one byte length $t \le 7$. Then the value of P (line 102) is static, so $P' = \text{AES}_K(P)$ can be precomputed. Writing $\text{CBC-MAC}_K(C, X)$ for the procedure identical to $\text{CBC-MAC}_K(X)$ except that $C \in \{0,1\}^{128}$, rather than 0^{128}, is the initial value of Y, we note that $\text{CBC-MAC}_K(P \,\|\, Q) = \text{CBC-MAC}_K(P', Q)$. As P' is known and $|Q| = 128$ we need only one AES call per round. At the other extreme, if n and t vary and t is arbitrary one can still compute the $P' = \text{AES}_K(P)$ value appropriate to a given enciphering or deciphering operation and then $Q' = \text{CBC-MAC}_K(P', T')$ where T' is the longest prefix of T that is a multiple of 128 bits. Each round will then need a single AES operation, to compute $\text{CBC-MAC}_K(Q', Z)$ for some $|Z| = 128$. In conclusion, the amortized number of AES calls to encipher or decipher an n-bit string with FFX-A2 is $\mathsf{rnds}(n)$ if n and $t \le 7$ are static, and never more than $\mathsf{rnds}(n) + 1 + \lceil (|T|_8 - 7)/16 \rceil$.

Security notes. The round function F is constructed in such a way that the set of inputs on which the CBC-MAC is invoked is prefix-free. (A set of strings is prefix-free if for any distinct x, y in the set, x is not a prefix of y.) The CBC-MAC is known to be a good PRF when it is invoked on a set of prefix-free inputs, assuming AES is a good PRP [23].

Including VERS, method, $\mathsf{addition}$, radix, $\mathsf{split}(n)$, and $\mathsf{rnds}(n)$ in the input to the CBC-MAC (lines 102 and 104) is not strictly necessary, given that the first four values are fixed in A2 and the next two are deterministic functions of n. All the same, the explicit inclusion of these values makes for a more robust design.

Let us explain our choice for the number of rounds. Here there are two questions: why do we use more rounds for small n, and why the particular numbers? In fact, we do not know that more rounds are actually necessary for small n; as far as we know, using $\mathsf{rnds}(n) = 10$, say, for all n, would be perfectly fine. But, instead of this, we selected round counts according to the following heuristic. Let the *induced entropy* be the number of bits of entropy in the outputs of the round functions across all rounds, assuming truly random round functions. This is at least $\mathsf{rnds}(n) \cdot \mathsf{split}(n)$. Our A2 parameter collection uses four rounds plus enough additional rounds to provide at least 128 bits of induced entropy. Also, we never use fewer than 12 rounds, and we round up to the next multiple of 6.

As we will explain in Appendix E, the authors do not regard the Patarin attacks [21] on $n \ge 6$ rounds as "real" attacks against FFX, since they need more than $\mathsf{radix}^n - 2$ queries per tweak, and, having done so, only distinguish populations of PRPs from populations of random permutations. All the same, the entropy-based heuristic above leads to round counts large enough to defeat even the Patarin (non-)attack.

Let $q_2(n)$ denote the "CCA threshold" for FFX-A2, meaning the largest number of queries such that even a computationally unbounded adversary asking $q_2(n)$ encryption or decryption queries will have PRP-advantage less than or equal to 0.5 in the model in which the round function is replaced by a uniform random function. Recent work by Hoang and Rogaway [8] implies bounds of, for example, $q_2(32) > 580$ and $q_2(50) > 37000$. Note that if tweaks are used then one will need to have, for example, more than 580 or 37000 identical tweaks, for these two cases, until one can no longer prove the inexistence of an adversary getting advantage exceeding 0.5 (assuming the underlying PRP's security).

Bounds more sophisticated than those used above are known from Patarin [22], but there are some difficulties with using them to get concrete numbers for FFX-A2, beginning with the fact that the bounds are stated only asymptotically.

The best attack we know against FFX-A10 is, by far, to do exhaustive key-search on AES, expending about 2^{128} time. The m-in-m attack mentioned above would take in excess of 2^{1024} time. Overall, we regard the selected round counts as being extremely conservative from multiple points of view.

B Parameter Collection A10

Notation. We will use the notation introduced in Appendix A, plus the following additional number-to-string and string-to-number conversion routines: **(a)** $\textsc{num}_{\mathsf{radix}}(x)$ takes a nonempty string $x \in \{0, \ldots, \mathsf{radix}-1\}^*$ and converts it to the corresponding number, where the number is interpreted in the given radix, most-significant character first. For example, $\textsc{num}_2(1100) = 12$. **(b)** $\textsc{str}^m_{10}(y)$ takes a number $y \in [0 \mathinner{..} 10^m - 1]$ and returns the m-digit string that represents it, base-10. For example, $\textsc{str}^3_{10}(37) = 037$ (the string). More generally, $\textsc{str}^m_{\mathsf{radix}}(y)$ takes a number $y \in [0 \mathinner{..} \mathsf{radix}^m - 1]$ and returns the m-character string that represents it in the the given radix, most significant character first.

Specification. We specify the parameter collection we call A10. When FFX is instantiated with parameter collection A10 one obtains the scheme FFX-A10. With it one can encipher decimal strings of 4 to 36 digits. This covers U.S. Social Security numbers, credit card numbers, and commonly considered subsequences of credit card numbers. The underlying mechanism in FFX-A10 is a maximally-balanced alternating Feistel scheme with an AES-based round function.

Implementation notes. While the value B in Figures 3 and 5 is a decimal string of 18 or fewer digits, an implementation would probably want to represent B by the equivalent 8-byte unsigned integer. Under this representation, the $[\textsc{num}_{10}(B)]^8$ and $[\cdot]^8$ conversions at line 203 effectively vanish, as does the $\textsc{str}^m_{10}(z)$ conversion at line 210. Note that a 4-byte unsigned integer will suffice for representing B if $n \leq 18$.

An implementation of F needs fewer AES calls than a quick inspection of the code might suggest. Suppose first that an application will, with a given key K, encipher messages of only one length n using tweaks of only one byte length $t \leq 10$. Then the value of P (line 202) is static, so $P' = \text{AES}_K(P)$ can be precomputed. Writing CBC-MAC$_K(C, X)$ for the procedure identical to CBC-MAC$_K(X)$ except that $C \in \{0,1\}^{128}$ is the initial value of Y, we note that CBC-MAC$_K(P \,\|\, Q)$ = CBC-MAC$_K(P', Q)$. As P' is known and $|Q| = 128$ we need only one AES call per round. At the other extreme, if n and t vary and t is arbitrary one can still compute the $P' = \text{AES}_K(P)$ value appropriate to a given enciphering or deciphering operation and then Q' = CBC-MAC$_K(P', T')$ where T' is the longest prefix of T that is a multiple of 128 bits. Each round will then need a single AES operation, CBC-MAC$_K(Q', Z)$ for some $|Z| = 128$. In conclusion, the amortized number of AES calls to encipher or decipher an n-digit string with FFX-A10 is $\mathsf{rnds}(n)$ if n and $t \leq 10$ are static, and never more than $\mathsf{rnds}(n) + 1 + \lceil (|T|_8 - 10)/16 \rceil$.

Security notes. The round function F is constructed in such a way that the set of inputs on which the CBC-MAC is invoked is prefix-free. (A set of strings is prefix-free if for any distinct x, y in the set, x is not a prefix of y.) The CBC-MAC is known to be a good PRF when it is invoked on a set of prefix-free inputs, assuming AES is a good PRP [23].

Let U_d denote the uniform distribution on $\mathbb{Z}_{10^m}$ and let M_m be the distribution given by picking R

parameter	**value**	**comment**
radix	10	alphabet is $\mathsf{Chars} = \{0, 1, 2, \ldots, 8, 9\}$
Lengths	$[\mathsf{minlen} \,..\, \mathsf{maxlen}]$ where $\mathsf{minlen} = 4$, $\mathsf{maxlen} = 36$	permitted message lengths
Keys	$\{0,1\}^{128}$	128-bit AES keys
Tweaks	$\text{BYTE}^{\leq M}$ where $M = 2^{64} - 1$	tweaks are arbitrary byte strings
addition	1	blockwise addition
method	2	alternating Feistel
split (n)	$\lfloor n/2 \rfloor$	maximally balanced Feistel
rnds (n)	$\begin{cases} 12 & \text{if } 10 \leq n \leq 36, \\ 18 & \text{if } 6 \leq n \leq 9, \text{ and} \\ 24 & \text{if } 4 \leq n \leq 5 \end{cases}$	from entropy-based heuristic
F	given below	AES-based round function

```
200 algorithm $\mathsf{F}_K(n, T, i, B)$
201 $\text{VERS} \leftarrow 1$;  $t \leftarrow |T|_8$
202 $P \leftarrow [\text{VERS}]^2 \,\|\, [\mathsf{method}]^1 \,\|\, [\mathsf{addition}]^1 \,\|\, [\mathsf{radix}]^1 \,\|\, [n]^1 \,\|\, [\mathsf{split}(n)]^1 \,\|\, [\mathsf{rnds}(n)]^1 \,\|\, [t]^8$
203 $Q \leftarrow T \,\|\, [0]^{-t-9 \bmod 16} \,\|\, [i]^1 \,\|\, [\text{NUM}_{10}(B)]^8$
204 $Y \leftarrow \text{CBC-MAC}_K(P \,\|\, Q)$
205 $Y' \leftarrow Y[1 \,..\, 64]$;  $Y'' \leftarrow Y[65 \,..\, 128]$
206 $y' \leftarrow \text{NUM}_2(Y')$;  $y'' \leftarrow \text{NUM}_2(Y'')$
207 if $\text{EVEN}(i)$ then $m \leftarrow \mathsf{split}(n)$ else $m \leftarrow n - \mathsf{split}(n)$
208 if $m \leq 9$ then $z \leftarrow y'' \bmod 10^m$
209 else $z \leftarrow (y' \bmod 10^{m-9}) \cdot 10^9 + (y'' \bmod 10^9)$
210 return $\text{STR}_{10}^m(z)$
```

Figure 5: **Parameter collection A10.** Mechanism FFX-A10 enciphers decimal strings of up to 36 digits. It does so using maximally-balanced Feistel and a round function based on the AES CBC-MAC.

at random in $\mathbb{Z}_{2^{64}}$ and returning $R \bmod 10^m$. The statistical distance between these distributions is at most $10^m/2^{66}$, which as at most 2^{-36} given that the maximum value of m here is 9. This means that line 210 is returning a m-digit string that is very close to uniform; the bias would be at most 2^{-35}.

Including the values VERS, method, addition, radix, split(n), and rnds(n) in the input to the CBC-MAC is not strictly necessary, given that the first four values are fixed and the next two are deterministic functions of n. All the same, the explicit inclusion of these values makes for a more robust design.

Let us explain our choice for the number of rounds. Here there are two questions: why do we use more rounds for small n, and why the particular numbers? In fact, we do not know that more rounds are actually necessary for small n; as far as we know, using $\mathsf{rnds}(n) = 10$, say, for all n, would be perfectly fine. But, instead of this, we selected round counts according to the following heuristic. Let the *induced entropy* be the number of bits of entropy in the outputs of the round functions across all rounds, assuming truly random round functions. This is at least $\mathsf{rnds}(n) \cdot \mathsf{split}(n) \cdot \log_2(\mathsf{radix})$. Our A10 parameter collection uses four rounds plus enough additional rounds to provide at least 128 bits of induced entropy. Also, we never use fewer than 12 rounds, and we round up to the next multiple of 6.

As we will explain in Appendix E, the authors do not regard the Patarin attacks [21] on $n \geq 6$ rounds as "real" attacks against FFX, since they need more than $\mathsf{radix}^n - 2$ queries per tweak, and, having done so, only distinguish populations of PRPs from populations of random permutations. All the same, the heuristic above leads to round counts large enough to defeat even the Patarin (non-)attack.

Let $q_{10}(n)$ denote the "CCA threshold" for FFX-A10, meaning the largest number of queries such that even a computationally unbounded adversary asking $q_{10}(n)$ encryption or decryption queries will have PRP-advantage less than or equal to 0.5 in the model in which the round function is replaced by a uniform random function. Recent work by Hoang and Rogaway [8] implies bounds of, for example, $q_{10}(6) \geq 62$ and $q_{10}(16) > 77000$. Note that if tweaks are used then one will need to have, for example, more than 77000 16-digit plaintexts with identical tweaks until we can no longer prove the inexistence of an adversary getting advantage exceeding 0.5 (assuming the underlying PRP's security).

Bounds more sophisticated than those used above are due to Patarin [22], but there are some difficulties with using them to get concrete numbers for FFX-A10: the bounds are only for the balanced setting, for binary strings, and they are stated asymptotically.

The best attack we know against FFX-A10 is to do exhaustive key-search on AES, expending about 2^{128} time. The m-in-m attack mentioned above would take in excess of 2^{2600} time. Overall, we regard the selected round counts as being extremely conservative from multiple points of view.

C Brief History

The origins of the FPE problem go back over 25 years. In 1981, the US National Bureau of Standards (NBS, later to become NIST) published FIPS 74 [18], an appendix of which describes an approach for enciphering arbitrary strings over an arbitrary alphabet. Brightwell and Smith (1997) appear to be the first authors to clearly and more generally describe the FPE problem and its utility [5]. They called it "datatype-preserving encryption." Black and Rogaway [4] brought the problem to the attention of the cryptographic community in 2002, providing definitions and a number of solutions. Spies gave the primitive its currently-used name, circa 2003.

Bellare, Ristenpart, Rogaway, and Stegers provide a comprehensive treatment of the FPE problem, including definitions and solutions employing what they call type-1 and type-2 Feistel networks [3]. The current document employs these same constructs. The Feistel construction itself goes back to later versions of the IBM cipher Lucifer. The method-1 ("unbalanced") Feistel variant is described by Schneier and Kelsey [25], while the method-2 ("alternating") Feistel variant is described by Lucks [13]. References [3, 4, 8, 15, 16, 22] provide some of the relevant security results on the Feistel construction.

FFX evolved from the FFSEM specification due to Spies [28]. The FFSEM scheme was, in turn, based on the paper by Black and Rogaway [4].

D High-Level Design Choices

Why feistel? The authors believe that, for FPE, there is, at present, no serious alternative to some form of Feistel. The approach benefits from being a classically known and extensively studied. For all their prescience in identifying FPE, the actual mechanistic ideas described by Brightwell and Smith do not possess any such history and are rather incoherent [5]. Methods to achieve uniform shuffles on large domains, starting with a random function, remain completely impractical [7]. Developing a new confusion/diffusion primitive seems out of that question in terms of delivering assurance and leveraging existing AES experience and implementations, while resurrecting an old confusion/diffusion primitive, say Hasty Pudding [26], is unrealistic for the same reasons. Only Feistel combines decades of history and a corpus of significant academic work.

Why the parameters? The generality embodied by the parameter choices facilitates continuing innovation while at the same time ensuring that any NIST-approved FPE mode embodies sound cryptographic practice. We name specific collections of parameters, A2 and A10, so that, despite the permitted generality, implementers can employ a fully-instantiated "off the shelf" mode.

The classical Feistel construction is balanced. When one considers making it unbalanced, two alternative ways of doing so arise naturally, represented here by method-1 and method-2. We have allowed both because both have appeared in the literature [13, 25], and, depending on the parameter choices and implementation, one might offer better performance than the other, so implementers benefit by having a choice. At the present time there are no convincing security reasons to prefer one choice over the other, although such reasons could arise in the future.

Why imbalance? Some imbalance is made necessary by the fact that inputs could have odd length. Greater imbalance is allowed so as to keep within the scope of the standard the Thorp Shuffle (where $\mathsf{method} = 1$ and $\mathsf{split}(n) = 1$) and similar schemes whose analysis continues [8, 16].

The final round. In DES, which uses balanced Feistel, the returned value would (in terms of our method-1 construction) be $B \| A$ rather than $A \| B$. That is, the last round is different, not "crossing the wires" at the end. The historical reason for this choice was to allow the encryption algorithm to also be used for decryption. We have not followed this convention because it is simpler for all rounds (including the last) to be the same and, regardless, the trick does not work for method-2 schemes in the sense that altering the order of outputs in the final round still does not allow the encryption algorithm to be used for decryption when the number of rounds is even, as it is for us. The hardware-based mindset that may have lead DES to this choice does not seem important enough in our setting to warrant the increased complexity and lack of uniformity that would result.

E Security Definition

In this section we describe the security notion we expect FFX to meet. The material here is rather technical, but it is not needed for understanding the definition of FFX or how to use it.

The notion we target is a conventional one: strong-PRP security for a conventional (fixed length, not tweakable) blockcipher [12]. It should be recognized from the start that the notion is too strong to model any "real-world" attack, but it has the advantage both of familiarity and of tightly implying the more realistic security notions one may ultimately care about. To review, let $\mathcal{E}\colon \mathsf{Keys} \times \mathsf{Chars}^n \to \mathsf{Chars}^n$ be a blockcipher, meaning that $\mathcal{E}_K(\cdot) = \mathcal{E}(K, \cdot)$ is a permutation on Chars^n and $n \geq 1$ is fixed. Consider an adversary A that has two oracles, E and D. The adversary may ask any encryption query $E(X)$ or decryption query $D(Y)$, where $X, Y \in \mathsf{Chars}^n$. We consider two ways of answering these queries. In the first, key K is uniformly chosen from Keys and then $E(X)$ and $D(Y)$ queries are answered by $\mathcal{E}_K(X)$ and $\mathcal{E}_K^{-1}(Y)$, respectively. In the second, a permutation π on Chars^n is chosen uniformly at random and then $E(X)$ and $D(Y)$ queries are answered by $\pi(X)$ and $\pi^{-1}(Y)$. The adversary's *advantage* is the probability that A outputs the bit "1" if run in the first setting minus the probability that it outputs "1" when run in the second setting. Informally, we regard a blockcipher $\mathcal{E}$ as *secure* if for any adversary A that runs in a reasonable amount of time and asks some number $q \leq N - 2$ queries to its oracles, where $N = |\mathsf{Chars}|^n$, adversary A's advantage is small. Limiting q to $N - 2$ instead of N is because balanced Feistel constructions give rise to only even permutations (that is, permutations that are the product of an even number of transpositions), giving rise to a trivial distinguishing test if N or $N - 1$ queries are asked.

To apply this definition to FFX one should assume that the adversary fixes the tweak T and message length n used for queries. This is a safe thing to assume because FFX includes T and n within the scope of its round function F, so we know that it is never advantageous, up to the PRF-ness of the round function F, for the adversary to use queries of varying T or n-values. By avoiding the inclusion of T and n in adversarial queries we eliminate the artificial boosting of advantage associated to taking an attack achieving a tiny advantage ε for breaking a PRP and creating an attack with advantage nearly 1 for a tweakable family of PRPs just by running the original attack on $\Theta(\varepsilon^{-2})$ differently-tweaked PRPs and finishing with a majority vote. Such boosting of advantage does not represent any genuine security degradation and does not help for breaking real-world security properties of a PRP, such as its unpredictability.

We note that Patarin shows that, for $r \geq 6$, even n, and a binary alphabet, there is an attack employing about $2^{\frac{n}{2}(r-4)}$ time for distinguishing a family of n-bit, r-round Feistel constructions from a comparable family of n-bit random permutations [21]. Under our definition, this is simply not an attack, as our adversary has only a single permutation.

F Why use a Tweak?

FPE will be used in settings where the number $N = \mathsf{radix}^n$ of strings of a certain allowed length n is quite small. In such settings, use of a tweak can significantly enhance security, and is strongly recommended whenever this is possible. Let us explain.

Suppose we are enciphering the middle-six-digits of a 16-digit CCN; the remaining ten digits are to be left in the clear. If we use a deterministic and tweakless scheme, there is a danger that an adversary might be able to create, by noncryptographic means, an unnecessarily useful dictionary of plaintext/ciphertext pairs (X, Y), where X is a 6-digit number and Y is its encryption. Each plaintext/ciphertext pair (X, Y) that the adversary somehow obtains (acquired, for example, by a phishing attack) would let the adversary decrypt every CCN that happens to have those same middle-six digits. Note that in a database of 100-million entries we'd expect about 100 CCNs to share any given middle-six digits. Learning k CCNs and possessing an encrypted database ought

```
algorithm CycleWalking.Encrypt($K, T, X$)
if $K \notin$ Keys or $T \notin$ Tweaks or $X \notin$ Chars$^*$ or
   $|X| \notin$ Lengths or not VALID($X$) then return $\perp$
repeat
    $X \leftarrow$ FFX.Encrypt($K, T, X$)
until VALID($X$)
return $X$
```

```
algorithm CycleWalking.Decrypt($K, T, Y$)
if $K \notin$ Keys or $T \notin$ Tweaks or $Y \notin$ Chars$^*$ or
   $|Y| \notin$ Lengths or not VALID($Y$) then return $\perp$
repeat
    $Y \leftarrow$ FFX.Decrypt($K, T, Y$)
until VALID($Y$)
return $Y$
```

Figure 6: Cycle walking. Plaintexts satisfying a predicate VALID are enciphered to plaintexts satisfying the same predicate by way of repeated application of the FFX.Encrypt operation.

not to give you $100k$ more CCNs for free.

The problem is not a cryptographic failure, but a failure to use a good tool well. The middle-six digits ought to have been tweaked by the remaining ten. If this had been done then learning that CCN 1234-123456-9876 encrypts to 1234-770611-9876, say, would not let one decrypt 1111-770611-9999, as the mapping of 123456 to 770611 is specific to the surrounding digits 1234/9876.

In general, it is desirable to use all information that is available and statically associated to a plaintext as a tweak for that plaintext. In the most felicitous setting of all, the non-secret tweak associated to a plaintext is associated *only* to that plaintext.

Extensive tweaking means that fewer plaintexts are enciphered under any given tweak. This corresponds, in the PRP model we have adopted, to fewer queries to the target instance. The relevant metric is the maximum number of plaintexts enciphered with the same tweak, which is likely to be significantly less than the total number of plaintexts enciphered.

G Enciphering on Arbitrary Domains

Mechanism FFX enciphers strings within the set Chars^n for some alphabet of characters Chars and some allowed message length $n \in \mathsf{Lengths}$. Often it is useful to encipher points within a message space that is *not* of this form. When this the case, FFX should be used in conjunction with one or both of the following techniques: *cycle walking* and *dense encoding.*

Cycle walking and dense encoding are well-known techniques, having the status of folklore. Using them on top of FFX does not damage security compared to a more direct application of FFX.

G.1 Cycle Walking

As an illustrative example, consider the problem of enciphering calendar days $\{0, \ldots, N-1\}$ within some epoch, say 01/01/1900 to 12/31/2099. The number of possible days $N = 73,049$ is not a power $n \geq 2$ of some radix radix. Now one could certainly consider the points we wish to encipher as being binary-strings, say ones of length $17 = \lceil \log_2(73049) \rceil$ bits, but it is not all binary strings of length 17 that should be considered valid—only the ones representing numbers between 0 and $N-1$. While FFX will allow us to encipher a 17-bit string into a 17-bit string, how would one encipher on a space consisting of just the *particular* 17-bit strings that satisfy the desired predicate VALIDDAY(X)?

As a second example, consider the problem of enciphering a valid Social Security number (SSN) into a valid SSN. While there is no universally agreed-upon notion of validity for SSNs, some numbers—for example, those beginning with 000 or beginning with a number in excess of 772—have not been issued by the U.S. Social Security Administration and may therefore be considered invalid [27]. Let $\textsc{valid}\mathrm{SSN}(X)$ be some fixed predicate that, given a nine-decimal-digit string X, returns either TRUE or FALSE according to some fixed validity criterion. One might wish to encipher points on the message space that consists of all nine-digit strings X that satisfy $\textsc{valid}\mathrm{SSN}(X)$.

In both examples, there is a universe of strings Chars^n on which we can FFX-encipher and there is a subset of these that satisfy some fixed predicate VALID. To encrypt within the subset of valid strings, one can use the mechanism of Figure 6. The mechanism works by repeatedly enciphering or deciphering until obtaining a valid string. A valid string must eventually be obtained because repeated enciphering or deciphering amounts to "walking along a cycle" and at least one point on this cycle, the initial one, satisfies the validity predicate.

In order for cycle walking to be efficient, it is important that VALID is TRUE a reasonably large fraction of the time. In our first example, the valid strings comprise $73049/2^{17} \approx 56\%$ of the total; in the second example, the fraction depends on how strict a validity condition is established. If valid strings comprise a fraction p of the total, the expected number of iterations of the algorithm is around $1/p$, indicating that for good performance p should be made as large as possible.

In recent work [3] it is shown that, in a formal model, it is not damaging to release the cycle-walking timing information. (That is, the number of times that the repeat/until loop of Figure 6 is executed.)

G.2 Dense Encoding

It is sometimes useful to re-encode strings from a relatively sparse set of strings into a dense set of strings. To illustrate the idea, suppose one wishes to encipher not all credit card numbers (CCNs) of a given length n but, instead, all of those with a Luhn checksum of zero [14]. In other words, we are assuming that a plaintext presented for encryption will have a Luhn checksum of zero and that we wish to ensure that we encrypt it into a ciphertext that again has this same property. To accomplish this, take the n-digit CCN X we wish to encrypt and verify that it does indeed have the correct Luhn checksum. Assuming that it does, strip the last digit from X, resulting in an $(n-1)$-digit string X' (by the definition of the Luhn checksum, the string X' uniquely determines X). Encipher X' into an $(n-1)$-digit string Y' using FFX.Encrypt. Now expand Y' to an n-digit string with the correct Luhn checksum by appending the necessary final digit. Return this string as the ciphertext Y for X.

The above approach is more efficient than cycle walking. In general, encoding a sparse set of strings one wishes to encipher (all n-digit strings with a Luhn checksum of zero) into a dense set of strings (all $(n-1)$-digit strings) is a desirable first step for format-preserving encryption on a sparse space.

H Number of Rounds

The choice of method, rnds, and split impact security. Let us discuss some of the issues involved in their selection.

Note first that many different FPE security goals are possible; which one is targeted can impact the round count selection. One can consider, for example, message-recovery security, designated-point security, PRP-security against a chosen-plaintext attack, and PRP-security against a chosen-ciphertext attack. The attacks can be adaptive or non-adaptive. The underlying scheme might be regarded as tweakable or not. Which goal is sufficient depends on the application, but we have singled out one very strong goal in Appendix E that we believe adequate to cover real-world applications of FPE.

Once a goal has been decided upon, another question is the quantitative level of security desired. We are seeking security up to $q(n) = \mathsf{radix}^n - 2$ queries on length-n inputs, and we want the time to break the scheme to be completely prohibitive—as bad as doing exhaustive key search on the underlying blockcipher. We expect this to usually be 128 bits, and we consider attacks taking more than 2^{128} time as being irrelevant.

For any choice of split and method, there are two sources of information on how many rounds $r = \mathsf{rnds}(n)$ are needed to achieve this goal. One is existing attacks. These give us a lower bound on the needed number of rounds. From the other direction, proofs of security give us a number of rounds proven sufficient to preclude attacks. Ideally, these numbers would match. But, at the moment, there is a significant gap between them. The severity of this gap depends on the parameters.

Going with the lower bound is not conservative enough since better attacks might emerge. Going with the upper bound is conservative but one pays a performance penalty—assuming the security bound is ever good enough. In such situations, we need to pick and recommend a number of rounds in between the lower and upper bounds.

For six or more rounds, the best generic attack on Feistel is Patarin's meet-in-the-middle (henceforth "m-in-m") attack [21]. The minimal round counts and domain size specified in Section 2 ensure that this attack takes more than 2^{128} time. Let us now give the details for the m-in-m attack.

Fix n and, for concision, let $d = \mathsf{radix}$, $a = \mathsf{split}(n)$, and $r = \mathsf{rnds}(n)$. First assume $\mathsf{method} = 1$. A round function is a map $f\colon \mathsf{Chars}^{n-a} \to \mathsf{Chars}^a$. If $\mathbf{f} = (f_1, \ldots, f_\ell)$ is an ℓ-vector of round functions and $X, Y \in \mathsf{Chars}^n$ then we let $\mathbf{f}(X) = f_\ell(\cdots(f_1(X))\cdots)$ and $\mathbf{f}^{-1}(Y) = f_1^{-1}(\cdots(f_\ell^{-1}(Y))\cdots)$.

We describe the attack as a PRP one, so that the adversary has an oracle E, this being either $\mathcal{E}_K$ or a random permutation π on Chars^n. The adversary queries any distinct points $X_1, \ldots, X_q \in \mathsf{Chars}^n$ to get back $Y_1, \ldots, Y_q$. It then makes tables T_1, T_2 whose entries are defined by

$$T_1[\mathbf{f}] = (\mathbf{f}(X_1), \ldots, \mathbf{f}(X_q)) \text{ and } T_2[\mathbf{f}] = (\mathbf{f}^{-1}(Y_1), \ldots, \mathbf{f}^{-1}(Y_q))$$

for all $(r/2)$-vectors $\mathbf{f}$ of round functions. If there exist $\mathbf{f}_1, \mathbf{f}_2$ such that $T_1[\mathbf{f}_1] = T_2[\mathbf{f}_2]$ then the adversary returns 1, else 0.

For the attack to succeed (defined, say, as the adversary getting advantage is at least 1/2), the number of queries q must be large enough. A ballpark estimate is $q = \lceil \frac{ad^{n-a}r}{2n} \rceil$, but we stress this is an estimate only. We will, conservatively, estimate the running time of the attack by assuming $q = 1$ and asking that, even then, the running time should be at least 2^{128}.

A round function is described by a table having d^{n-a} rows, with each entry being in Chars^a. The size of such a table is ad^{n-a}. An $(r/2)$-vector of round functions is thus described by a table of size $ard^{n-a}/2$. Each of the tables T_1, T_2 has $N = d^{ad^{n-a}r/2}$ rows of this type, so, even neglecting the storage of the table entries, the storage and time to make the tables is at least $2d^{ad^{n-a}r/2} \cdot ard^{n-a}/2 \geq d^{ad^{n-a}r/2+(n-a)}$. The running time of the attack is thus at least $t = d^{ad^{n-a}r/2+(n-a)}$. Let

$$r_m = \frac{(256/\log_2(d)) - 2(n-a)}{ad^{n-a}} \; .$$

Then $r \geq r_m$ guarantees $t \geq 2^{128} = d^{128/\log_2(d)}$. We have written a program to compute r_m for given d, n, a. Our requirement $\mathsf{radix}^n \geq 100$ guarantees that $r_m \leq 8$ for all choices of a. This doesn't necessarily mean that $r_m > 8$ when $\mathsf{radix}^n < 100$; our requirement of $\mathsf{radix}^n \geq 100$ was chosen because it is simple to state and sufficient to ensure $r_m \leq 8$.

Now let's turn to $\mathsf{method} = 2$. A round function for the i-th round maps as $f\colon \mathsf{Chars}^{n-a} \to \mathsf{Chars}^a$ if i is even and as $f\colon \mathsf{Chars}^a \to \mathsf{Chars}^{n-a}$ if i is odd, for $0 \leq i \leq r-1$. Let $r_1 = \lceil r/4 \rceil$ and $r_2 = \lfloor r/4 \rfloor$. For T_1, each $(r/2)$-vector of round functions is described by a table of size $ar_1d^{n-a} + (n-a)r_2d^a$, and T_1 has $N = d^{ad^{n-a}r_1+(n-a)d^ar_2}$ rows. Neglecting the second table T_2, the storage and time to make the first is at least $d^{ad^{n-a}r_1+(n-a)d^ar_2} \cdot (d^{n-a} + d^a)$. The running time of the attack is thus at least $t = d^{ad^{n-a}r_1+(n-a)d^ar_2+(n/2)}$. Assuming $r/2$ is even, let

$$r_m = \frac{2((256/\log_2(d)) - n)}{ad^{n-a} + (n-a)d^a} \; .$$

Then $r \geq r_m$ guarantees $t \geq 2^{128} = d^{128/\log_2(d)}$. Based on our program, our requirement $\mathsf{radix}^n \geq 100$ continues to guarantee that $r_m \leq 8$ for all choices of a.