FPE Synopsis / Draft 3. This note is a survey of format-preserving encryption (FPE). It aims to explain FPE in an accessible but precise way. The current version has evolved, along with my understanding of FPE, since June 2008.

A Synopsis of
Format-Preserving Encryption

Phillip Rogaway *

Department of Computer Science
University of California, Davis, USA

March 27, 2010

Abstract

Format-preserving encryption (FPE) encrypts a plaintext of some specified format into a ciphertext of the *same* format—for example, encrypting a social-security number into a social-security number. In this survey we describe FPE and review known techniques for achieving it. These include FFX, a recent proposal made to NIST.

1 Introduction

Suppose you want to encrypt a 16-decimal-digit plaintext, perhaps a credit-card number, into a ciphertext that is again a 16-decimal-digit number. A shared key K is used to control the encryption. Syntactically, you seek a map $\mathcal{E}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ where $\mathcal{X}$ encodes 16-digit strings and where $\mathcal{E}_K = \mathcal{E}(K, \cdot)$ is a permutation—a forwards and backwards computable function on $\mathcal{X}$—for each $K \in \mathcal{K}$. In terms of security, you intend that $\mathcal{E}_K(X)$ not only conceal X but, also, all partial information about X.

The problem is harder than it sounds. You cannot, say, encode each $X \in \mathcal{X}$ as a 128-bit string and then apply AES, for AES will return a 128-bit string, not a 16-digit one. You can't just say to hash that 128-bit AES output, creating a 16-digit string, as that would obliterate your ability to go backwards—that is, your ability to decrypt. Classical modes of operation (CBC, CFB, CTR, ECB, OFB) don't do the job, nor do more recent modes from NIST (things like CCM, CMAC, or GCM). Indeed no contemporary cryptographic standard as yet exists that addresses the problem described, nor can a treatment be found in any published book.

And it is not only 16-decimal-digit numbers that one might like to encrypt without disruption to their format; other natural possibilities include primary account numbers (PANs) of other lengths, possibly required to have a zero Luhn-checksum [33]; various subsequences of the digits drawn from such numbers; U.S. social security numbers (SSNs); SSNs stripped of their last four digits; US postal addresses; database fields with an essentially arbitrary format; payloads in a networking packet; 512-byte disk sectors; and files of a particular kind, such as JPEG files, MPEG files, or C programs. For any of these message spaces a *format-preserving encryption* (FPE) would let you encrypt in such a way that the ciphertext will have exactly that same format as the plaintext.

The utility of FPE stems, in part, from its use in adding security to legacy protocols and systems. You can't, for example, just go in and change the PANs in a database to 128-bit strings without causing serious disruptions to programs that use those PANs, and possibly to the surrounding human context as well. Such disruptions are not only unnecessary but, in many situations, inimical to sound security practice.

It is my view that the cryptographic literature already contains good solutions for FPE but that, at least until quite recently, the ideas were scattered about, not widely known, and not cohesively described. Until quite recently, the overarching context was not spelled out all that explicitly and, to this day, relatively few cryptographers will have even heard the term "FPE". This survey aims to help fix this, and to explain the state-of-the-art to people who are not necessarily cryptographers. I will describe the FPE problem, survey known solutions, and provide some high-level commentary. My description includes the FFX mode of operation recently proposed to NIST by Bellare, Spies, and me [7].

This note is not a research paper. While you do not need to be a cryptographer to understand what I write (my

*Department of Computer Science, One Shields Ave., University of California, Davis, CA 95616, USA. Email: rogaway@cs.ucdavis.edu; URL: http://www.cs.ucdavis.edu/∼rogaway. The author is also a member of the Technical Advisory Board of Voltage Security. This survey was prepared at Voltage's suggestion. All shortcomings in this document are entirely my own.

sincere apologies if you disagree with this claim!), I do write with a cryptographically informed audience in mind. My imagined reader has encountered AES, DES, shared-key encryption, and the like. He or she may, for example, have heard a Voltage Security presentation that touched on FPE and wondered what this thing exactly is, and if it be solid science or just wishful thinking. I intend to unambiguously answer questions such as these.

My own research on FPE goes back more than a decade and includes, to date, some eight academic papers that one might reasonably place within this area [5, 6, 8, 24–26, 39, 56]. I would not say that this has been a "dominant" part of my cryptographic work, but that it is something that has been repeatedly on my mind. It has taken a while, but I am delighted to see that FPE is really catching on.

The term *format-preserving encryption* (FPE) is not my own. It is due to Terence Spies, Voltage Security's CTO [69]. I first saw the phrase on Voltage's website and immediately liked the term, and the idea associated to it, which I took to be a generalization of the kind of *integer* FPE scheme that I had previously formalized in my work with John Black [8]. There one aims to encipher on a message space of the form $\mathcal{X} = [N] = \{0, 1, \ldots, N-1\}$ for some arbitrary number N.

I didn't know it when I did my work with Black [8], but it turns out that the FPE problem goes much further back. Brightwell and Smith (1997) appear to have been the first to clearly (although informally) describe FPE and its utility [9]. They called their goal *datatype*-preserving encryption, as opposed to *format*-preserving encryption. (Both terms are good, but I prefer the second.) Even earlier, in 1981, the US National Bureau of Standards (NBS, which later became NIST) published FIPS 74, section 8 of which describes a (completely flawed) approach for enciphering an arbitrary string over a non-binary alphabet [45], the most prominent special case of FPE. In view of these references, I see FPE as a rather old problem—effectively a folklore one—of applied cryptography.

Eventually FPE did receive the attention of cryptographers. First there was the paper I mentioned with Black [8], which defined integer FPE and envisioned the use of this tool for enciphering on more general domains still. More recently, Bellare, Ristenpart, Stegers, and I gave a treatment of FPE at the level of generality that I now think most fitting [5]. That work and the current note have an intertwined history. They co-evolved to some extent, earlier versions of this note giving rise to the academic paper and the academic paper informing the current version of this note. Comparing the two, the note you are reading is considerably gentler, trying to explain matters for the benefit of non-experts. In contrast, the academic paper is narrower in scope and more technical in execution, targetting researchers in my community. Its focus is on formal definitions, theorems, and proofs.

2 The FPE Problem

This section is intended to provide a clear but informal description of the FPE problem. I try to avoid heavy mathematics and notation. A more formal treatment, for those inclined to see one, can be found in Appendix A and in reference [5].

The first thing to understand about format-preserving encryption is, naturally enough, that it *is* encryption: it's one flavor of *shared-key* (= symmetric) encryption. So the general setting is that there is a key K and, using it, you can encrypt a plaintext X to make a ciphertext Y. Anyone who has the same key K can decrypt the ciphertext Y to recover the plaintext X.

Now some encryption schemes are *randomized*, meaning that you can produce various different ciphertexts from a given plaintext depending on the computer-generated random bits—metaphorical coin tosses—that arise when you encrypt. Other encryption schemes are *stateful*, meaning that there might be a counter, say, that gets incremented with each message encrypted. When an encryption scheme takes a (non-constant) IV (initialization vector), as CBC encryption does, you know right away that it is randomized or stateful.

It is important to understand that FPE is neither randomized nor stateful; it is *deterministic*. There are no coins. No state. No IV. Every time you encrypt a particular message X with a particular key K you're going to get the exact same ciphertext Y. That's what we mean by deterministic encryption.

For a deterministic encryption scheme, I sometimes use the word *encipher* instead of *encrypt* and *decipher* instead of *decrypt*. At least to my ears, those terms help emphasize that we are in the deterministic setting. But there is nothing wrong with using the words *encrypt* and *decrypt* in the setting of deterministic encryption; that diction is very common, too.

A blockcipher, say AES, is a deterministic encryption scheme: every plaintext X has $n = 128$ bits and we convert it into a ciphertext Y that again has $n = 128$ bits, the transformation controlled by a key K. Be clear that AES itself won't let you encipher a message that has, say, 30 bits, or 50 bits, or 16 digits; it just doesn't have that ability. One might imagine some *mode of operation* that, using AES, does let you encipher a string of 30 bits or 50 bits or 16 digits. But then it's not AES we're talking about anymore—it's an AES-based mode of operation.

With AES, not only is the message space the set of 128-bit strings but so too is the ciphertext space. They're the same set. One might therefore say that that AES is *format-preserving*: the format of a plaintext—that it's a string of 128 bits—is the format of the ciphertext. AES is, all by itself, a format-preserving encryption scheme, one with a message space of 128-bit strings. AES, or any other

blockcipher, is our first example of an FPE scheme.

The problem with AES (and every other conventional blockcipher) as an FPE scheme is that the message space is so particular—we can only encipher binary strings of that one, fixed length. Basically, the message space $\mathcal{X} = \{0,1\}^{128}$ is what was conventional for the cryptographic community and convenient for the AES designers. It's not necessarily what's convenient for you. The FPE problem is, in essence, to construct a cipher that is "like" AES—a deterministic, reversible, format-preserving scheme that nicely scrambles up its input—but where the message space is whatever *you* might want it to be. We should do this without sacrifice in assurance compared to AES.

As an example, an FPE scheme with a message space of 30-bit strings would deterministically encrypt a 30-bit string into a 30-bit string. An FPE scheme with a message space of 16-digit strings would deterministically encrypt a 16-digit string into a 16-digit string. An FPE scheme with a message space of valid PANs, for some prescribed notion of validity, would deterministically map one such string into another.

It is best to make an FPE scheme out of a conventional blockcipher, say AES itself. One will use the underlying blockcipher in some mode of operation. It is true that one could try to design an FPE scheme from scratch—a *de novo* construction, one might say. The cipher called Hasty Pudding is such a scheme [64]. But blockcipher design is hard and engendering confidence in any blockcipher is a long, community-based process. FPE schemes are generalizations of blockciphers, so designing a *de novo* one would have all the same barriers, and more. The only practical way that one will get a widely-accepted and widely-used FPE scheme is to base it on a conventional and widely accepted blockcipher.

Let us get back to nitty-gritty and think in terms of an FPE scheme's API. An FPE scheme is going to have some message space $\mathcal{X}$. To encrypt a value $X \in \mathcal{X}$ you'll call some procedure $\mathcal{E}$, the encryption algorithm, providing it a key K and the plaintext X. This will yield a ciphertext $Y = \mathcal{E}_K(X)$ that is again in the message space $\mathcal{X}$. To decrypt, you'll call $\mathcal{D}_K(Y)$. This will return X.

Truth be told, there is another argument that ought, in many cases, to be provided to $\mathcal{E}$ and $\mathcal{D}$. It is called the *tweak* [30]. I like to write the tweak with a superscript T, as in $Y = \mathcal{E}_K^T(X)$ or $X = \mathcal{D}_K^T(Y)$. Conceptually, each tweak names an *independent* cipher: $Y = \mathcal{E}_K^T(X)$ and $Y' = \mathcal{E}_K^{T'}(X)$ will be unrelated values if $T \neq T'$. I'll explain later why it's important to have and use tweaks.

Our formalization of FPE schemes in Appendix A actually includes yet one more argument that is provided to $\mathcal{E}$ and $\mathcal{D}$. It is the *format* that we want X to be interpreted as having. The issue is that it may not always be manifest from looking at an input X what is the format you want it to be regarded as having. Are you trying to encrypt the number 71 within the space of two-digit numbers, two-character alphanumeric strings, or what? You can't just look at the input and tell. And, more to the point, you might be interested in simultaneously supporting the encryption of an input with respect to multiple formats at the same time. The format names the message space in which X should be enciphered.

The previous paragraph not withstanding, it most often *is* the case that, for a given applications, it will be manifest, once you see X, what format you want it to be considered to have. We'll assume this from now on.

We have gotten this far without explaining the security requirement we intend for an FPE scheme. Informally, we want $\mathcal{E}_K(\cdot)$ to look like a random permutation on the message space $\mathcal{X}$. We envisage a game in which the adversary has an oracle that behaves either as a "real" encryption oracles or as "fake" encryption oracle. In the first case the oracle chooses a random key K from the key space and encrypts each query $X \in \mathcal{X}$ to the ciphertext $Y = \mathcal{E}_K(X)$, returning Y. In the second case the oracle chooses a random permutation π on $\mathcal{X}$ encrypts each query $X \in \mathcal{X}$ to the ciphertext $Y = \pi(X)$, returning Y. A good FPE scheme is one in which no reasonable adversary is able to distinguish among these two types of behaviors. This is the famous PRP (pseudorandom permutation) notion of security.

There's also a notion of a "strong" PRP. This means that encryption should resemble a random permutation even if you can decrypt points, too. For more details on PRPs and strong PRPs, see Appendix A or reference [5].

The PRP notions we have described allow one to discount many potential FPE schemes as "wrong" (that is, completely insecure). Suppose, for example, we try to make an FPE scheme $\mathcal{E}$ for 30-bit strings by saying that $\mathcal{E}_K(X)$ is X xored it with the first 30-bits of $\mathrm{AES}_K(C)$ for some 128-bit constant C. The method is wrong in that, for example, $\mathcal{E}_K(A) \oplus \mathcal{E}_K(B) = A \oplus B$, while $\pi(A) \oplus \pi(B)$ is only rarely $A \oplus B$. So a simply adversary will be able to easily win the game we have laid out: ask the oracle distinct strings A, B and see if the xor of the ciphertexts is the xor of the plaintexts. As trivial as this attack may be, it is quite enough to break the suggestion in FIPS 74 [45]. Having strong definitions of security goes a long way in helping to keep us out of trouble.

3 Survey of Schemes

TAXONOMY. Blockcipher-based techniques for FPE change in character with the domain size $N = |\mathcal{X}|$ and with gross efficiency expectations. More specifically, schemes change in character if it is or isn't practical to spend $O(N)$ time to encrypt, and if N does or doesn't exceed the number of points in the domain of the underlying

setting	size	msg space	comments
tiny-space FPE	$N \leq 2^{10}$	$\mathcal{X} = [N]$	Easy case for FPE: N is sufficiently small that it is acceptable to spend $O(N)$ time for key setup or the first encryption (one can always initialize an N-element table and do constant-time lookups into it to encrypt or decrypt). There is no application-independent cutoff for this; 2^{10} is just illustrative. Small-space techniques may also be used for tiny message spaces.
small-space FPE	$N \leq 2^{128}$	$\mathcal{X} = \Sigma^n$	This setting includes the encryption of PANs, substrings of PANs, and SSNs. We usually assume a message space of $\mathcal{X} = \Sigma^n$ for some arbitrary alphabet Σ. Natural schemes can be based on (generalized) Feistel networks. Small-space FPE is the main focus of this writeup. Proposed NIST standard FFX [7] is a small-space FPE scheme.
large-space FPE	$N \geq 2^{128}$	$\mathcal{X} = \{0,1\}^n$	Also called wide-block encryption (assuming a domain of binary strings). Standardization of EME2 and XCB in IEEE P1619.2 [27] is expected in 2010. Prototypical application is the encryption of 512-byte disk sectors.

Figure 1: **Three kinds of FPE schemes.** The distinction among small- and large-space schemes assumes that the construction is blockcipher-based (the 128 in the size column is the blockcipher's assumed block size). The message-space column indicates what we *usually* assume the message space $\mathcal{X}$ to be; the message space may be the indicated set or the union of such sets.

blockcipher we will use. The following three cases are usefully distinguished. See Figure 1.

For **tiny-space FPE** the size of the message space $N = |\mathcal{X}|$ is so small that it is feasible to spend $O(N)$ time or $O(N)$ space in order to encrypt or decrypt a point. Certainly it is application-specific how big N can be for this to be considered so, but we are imagining settings like, for example, $\mathcal{X} = [365]$, to encrypt a calendar day. Here $[N]$ is the set $\{0, \ldots, N-1\}$ (or you may take $[N]$ to be $\{1, \ldots, N\}$, the choice is not important). For a tiny-space FPE we will usually assume that the message space $\mathcal{X}$ is $[N]$ for some (not-too-big) number N.

For **small-space FPE** the size of the message space $N = |\mathcal{X}|$ is *at most* 2^w where w is the blocksize of the blockcipher underlying our FPE scheme. These days we almost always think of using AES as our blockcipher, so $w = 128$ bits and $N = 2^{128} \approx 10^{38.5}$ becomes the cutoff for "small." Note that N can be vastly smaller than 2^{128} and still be huge, not small, by human standards (eg, the universe is believed to be fewer than 2^{86} nanoseconds old, "small" by our reckoning). For a small-space FPE scheme the number of domain points is small only in the sense that a domain point (or an encoding of one) won't "fill out" an entire AES block.

For a small-space FPE scheme we will usually assume that the message space consists of strings of some given length n. The strings might be binary, decimal, alphanumeric, or something else. We write $\mathcal{X} = \Sigma^n$ for the strings of length n over the (finite) alphabet Σ.

Small-space FPE schemes can usually be used on tiny message spaces, suggesting that a small-space *vs.* large-space distinction would already make for an adequate taxonomy. One reason for maintaining the tiny/small distinction is that known tiny-space FPE techniques have a different qualitative character than their small-space counterparts. One reason for actually using tiny-space FPE techniques is that, on tiny domains, small-space schemes will typically have worse or effectively absent provable-security guarantees (although no known attacks). In other words, scheme selection focused on quantitative provable-security results will suggest using tiny-space FPE techniques where efficiency considerations allow it. One argument in favor of ignoring tiny-space techniques and using small-space schemes even on tiny domains is to minimize complexity and the number of cryptographic schemes an application employs.

For **large-space FPE** the size of the message space $N = |\mathcal{X}|$ is *at least* 2^w where w is again the blocksize of the blockcipher we want to use to make our scheme (so, once again, assume that $w = 128$). For a large-space FPE scheme we will usually assume that the message space consists of *binary* strings of one or more permitted lengths. Permitted lengths are at least 128 bits. Under these assumptions, a large-space FPE is the same as a *wide-block encryption scheme*. Special cases include enciphering a binary string of exactly 512 bytes (eg, a disk sector) and enciphering *any* binary string of length $n \geq 128$, getting a binary string of the exact same length.

In Figs. 2, 3, and 4 we survey some tiny-space, small-space, and large-space FPE schemes. I have tried to identify the most important schemes and references.

In the remainder of this survey I will focus mostly on small-space FPE schemes. The reasons are as follows. First, this is the setting that matters for key applications like enciphering SSNs, PANs, and substrings of PANs. Second, small-space schemes can be used also on tiny-space domains. Third, large-space schemes are better understood and further along in standardization [27], so they would seem to need less of our attention.

In the remainder of this section we explain some of the

method	can encrypt on	description	security
Knuth shuffle [15, 18, 29, 40, 59]	$\mathcal{X} = [N]$ where N is small	Shuffle $[N]$ by repeatedly choosing an element from a decreasing prefix and moving it to the end. Slow setup then fast, table-driven encryption. Map X to its position after the shuffle.	Provably secure with ideal bounds.
Permutation numbering	$\mathcal{X} = [N]$ where N is *very* small	Map key K to a number $k \in [N!]$ and encrypt with the k'th permutation π_k on $[N]$. One AES call enough to usually determine permutation on $N \leq 34$ points (32-bit arithmetic enough for $N \leq 12$, 64-bit arithmetic enough for $N \leq 20$).	Provably secure with ideal bounds.
Prefix cipher [8]	$\mathcal{X} = [N]$ where N is small	Use ordering of $E_K(0), \ldots, E_K(N-1)$ to determine permutation. Slow setup then fast, table-driven encryption. Simpler but uses more coins than Knuth shuffle.	Provably secure with ideal bounds.

Figure 2: **Tiny-space FPE schemes.** Encryption or key-setup take time proportional to $N = |\mathcal{X}|$, restricting feasible values of N.

schemes mentioned in the tables. Discussion of FFX is in a section of its own, Section 4, but the methods it employs are described below.

FIPS 74 SCRAMBLING. The earliest treatment we know for small- or large-space FPE is in [45, Section 8]. The method enciphers an n-character string over some alphabet $\Sigma = \{0, 1, \ldots, d-1\}$ by deterministically generating from the key an n-character pad, again from $\Sigma = \{0, 1, \ldots, d-1\}$, and then characterwise modulo-d adding it to the plaintext to generate the ciphertext. For example, suppose the plaintext is $X = 123456$ and that the alphabet consists of the decimal digits $\Sigma = \{0, 1, \ldots, 9\}$. Suppose that the user's key K determines the pad $P = 749621$, say by enciphering the all-zeros string with $\mathrm{DES}_K(\cdot)$ and taking the (64-bit) result modulo 10^6. Then the ciphertext will be $123456 \boxplus 749621 = 862077$. From the point of view of Section 2 (and probably from any modern perspective), the method is easily attacked. Knowledge of one plaintext/ciphertext pair will let one decrypt any ciphertext of the same or shorter length.

BRIGHTWELL-SMITH SCRAMBLING. Brightwell and Smith were the first to really describe and motivate the FPE problem, and they also sketch an *ad hoc* solution for small-space (or even large-space) FPE [9]. They again assume the message is a sequence of characters—the string-based setting on which we too will focus. The scheme they sketch is cryptographically naïve, but the recognition of the problem was highly insightful. We will not describe the Brightwell-Smith scheme, which, in any case, was complex and not fully pinned down.

PREFIX CIPHER. After the preceding examples, let's give a simple example of a clearly *correct* scheme for FPE. Consider the tiny domain $\mathcal{X} = \{0, 1, 2, 3, 4\}$ having just five possible plaintexts. Under the control of a key K, say having 128 bits, we have to decide which of the five points 0 encrypts to, which of the remaining five points 1 encrypts to, and so on. One way to do this would be to compute $Y_0 = \mathrm{AES}_K(0)$, $Y_1 = \mathrm{AES}_K(1)$, $Y_2 = \mathrm{AES}_K(2)$, $Y_3 = \mathrm{AES}_K(3)$, $Y_4 = \mathrm{AES}_K(4)$, and $Y_5 = \mathrm{AES}_K(5)$. The argument to AES means the 128-bit string that encodes the given number. Use the relative ordering of $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5$ to determine the desired permutation. For example, if $Y_2 < Y_4 < Y_0 < Y_1 < Y_3$ then we would set $E_K(0) = 2$ (as 0 lands in position 2, with positions numbered 0 to 4), $E_K(1) = 3$, $E_K(2) = 0$, $E_K(3) = 4$, and $E_K(4) = 1$. It's not the most efficient scheme; you need to make five AES calls to encrypt one of five different points. But the usual assumption on AES, that it's a good PRP, is enough to know that our constructed FPE is a good PRP. Clearly this solution does not scale to encrypting one of 10^{16} points, for example, motivating more sophisticated techniques.

FPE BY SHUFFLING CARDS. Some techniques in small-space FPE are based on a card-shuffling metaphor. One thinks of the message space as consisting of a deck of playing cards $\mathcal{X} = [N] = \{0, \ldots, N-1\}$. Each card $X \in \mathcal{X}$ has its number imprinted on it. We use the key K as the specification of a recipe to shuffle this deck of cards. For example, key K could serve as the input to a pseudorandom generator and one could use the resulting bits as the needed randomness for some card-shuffling technique, like the customary riffle shuffle. After the shuffle each card X is in some position in $[N] = \{0, \ldots, N-1\}$. This position is regarded as the encryption of X under the key K. In other words, the point X encrypts to its final

method	can encrypt on	description	security
Alternating Feistel [1, 32]	$\mathcal{X} = \{0,1\}^n$ for any desired n	Like unbalanced Feistel but based on ladder drawing of Feistel. Round functions' domain and range alternate between contracting and expanding.	Provably secure to nearly the size of the domain of the round function with larger domain [8, 26, 32].
Brightwell-Smith scrambling [9]	$\mathcal{X} = \Sigma^n$ for any desired Σ and n	Characters successively modified by a multi-step process involving modular additions, rippling, and the use of DES in OFB mode. Later elaborated and named DTP [34].	Cryptographically unsophisticated scheme that seems unlikely to be secure. No provable security results are claimed.
Classical Feistel [16]	$\mathcal{X} = \{0,1\}^{2n}$ for any desired n	Classical ($\sim$1971), extensively studied approach. Variable number of rounds (eg, 3–32). Can use a high-assurance PRF as the round function.	With enough rounds, provably secure to about 2^n queries [36, 49]. With enough rounds, known attacks are completely ineffective [50]. Also true of all Feistel variants described here.
FIPS 74 scrambling [45]	$\mathcal{X} = \Sigma^n$ for any desired n	Characterwise-modular addition of plaintext and a key-dependent, DES-based pad.	Trivially attackable.
FFSEM [68]	$\mathcal{X} = \{0,1\}^{2n}$ for any desired n; or $\mathcal{X} = [N]$ for any desired N	Former NIST submission from Voltage Security that combines classical Feistel and cycle walking. Has been replaced by FFX.	Inherits classical Feistel results.
FFX [7]	$\mathcal{X} = \Sigma^n$ for any desired n	NIST submission uses alternating or unbalanced Feistel. Highly parametrized. Parameter sets A2 and A10 concretize enciphering binary and decimal strings.	Provably secure with bounds associated to alternating or unbalanced Feistel [26, 39, 42].
Numeric Feistel [5, 8]	$\mathcal{X} = [N]$ where $N = a\,b$ for $a, b \geq 2$	Like unbalanced or alternating Feistel but everything is done mod a or mod b instead of using binary strings. Intended to make cycle walking rare.	Provably secure with about the same bounds as the analogous string-based methods [5, 8, 26].
Recursive-merge shuffle [21]	$\mathcal{X} = [N]$ where N is arbitrary	Trace the trajectory of (just) card x in a shuffle suggested by [14, 42]. Samples from hypergeometric distribution to do so. Impractically slow.	Provably secure with ideal bounds.
Thorp shuffle [71]	$\mathcal{X} = [N]$ for even N	Split deck in half and let cards fall left-right or right-left depending on a coin. Coincides with maximally unbalanced Feistel if $N = 2^n$.	Provably secure to $N^{1-\epsilon}$ queries (with $O(1/\epsilon)$ passes) [26, 39]. Weak security notions already achieved with two passes [39].
Unbalanced Feistel [63]	$\mathcal{X} = \{0,1\}^n$ for any n	Generalized Feistel scheme where split is not into even halves. Source-heavy and target-heavy subtypes.	Provably secure to nearly the size of the domain of the round function [26, 42]. Weak attacks [51, 52]. Source-heavy preferred.

Figure 3: **Small-space FPE schemes.** Most of the included scheme are "generic," the scheme based on a pseudorandom function that can, in turn, be built from a blockcipher.

method	can encrypt on	description	security
ABL4 [38]	$\mathcal{X} = \{0,1\}^*$	Unbalanced Feistel with a layer of ECB. Slow, eclipsed by later work.	Provable security claimed, but no known writeup.
CMC [25]	$\mathcal{X} = \{0,1\}^{mw}$ where $m \geq 2$	An encrypt-mix-encrypt scheme where encryption is CBC and the mix is very lightweight. Not parallelizable.	Provably secure with good bounds.
EME [24]	$\mathcal{X} = \{0,1\}^{mw}$ where $m \geq 1$	An encrypt-mix-encrypt scheme where encryption is ECB and the masking is lightweight.	Provably secure with good bounds.
EME2 [22, 27]	$\mathcal{X} = \{0,1\}^{\geq w}$	Refinement of EME to extend the domain. Formerly named EME*. In draft standard IEEE P1619.2.	Provably secure with good bounds.
HCH [11]	$\mathcal{X} = \{0,1\}^{>w}$	Follow-on to HCTR, an efficiency and security improvement to it.	Provably secure with good bounds.
HCTR [10, 72]	$\mathcal{X} = \{0,1\}^{\geq w}$	A hash-counter-hash scheme where the hash is relatively heavy.	Provably secure with good bounds.
HEH [61] **HEH*** [60]	$\mathcal{X} = (\{0,1\}^w)^+$ $\mathcal{X} = \{0,1\}^{\geq w}$	A hash-encrypt-hash follow-on to TET; improves efficiency, makes VIL.	Provably secure with good bounds.
HMC [41]	$\mathcal{X} = \{0,1\}^{>w}$	Follow-on work to HCTR improves bounds and efficiency.	Provably secure with good bounds.
NR [42, 43]	$\mathcal{X} = \{0,1\}^{mw}$ where $m \geq 1$	A hash-encrypt-hash scheme with a relatively heavy hash and an encrypt of ECB mode, say. Not fully specified.	Provably secure with good bounds.
PEP [12]	$\mathcal{X} = \{0,1\}^{mw}$ where $m \geq 1$	Like TET, follow-on to NR. Poor speed, eclipsed by other modes.	Provably secure with good bounds.
TES [62]	$\mathcal{X} = \{0,1\}^{\geq 2w}$ where $m \geq 1$	A hash-encrypt-hash scheme using only the forward direction of the blockcipher. Tweakable, VIL.	Provably secure with good bounds.
TET [23]	$\mathcal{X} = \{0,1\}^{mw}$ where $m \geq 1$	A hash-encrypt-hash scheme that concretizes NR. Tweakable, parallelizable, FIL.	Provably secure with good bounds.
Type-1 Feistel [73] **Type-2 Feistel** [73] **Type-3 Feistel** [73]	$\mathcal{X} = \{0,1\}^{mw}$ for desired m	Generalized Feistel schemes used in blockciphers like CAST-256, RC6, and MARS.	Provably secure with good bounds [26, 73].
VIL [6]	$\mathcal{X} = \{0,1\}^{\geq w}$	Early scheme that helped introduced and formalize the problem.	Provably secure for CPA security, but does not achieve CCA security.
XCB [37]	$\mathcal{X} = \{0,1\}^{\geq 2w}$ or $\mathcal{X} = \{0,1\}^{\geq w}$ with nonce AD	A hash-counter-hash scheme with a relatively heavy hash. In draft standard IEEE P1619.2.	Provably secure with good bounds, although the writeup leaves some questions about the precise claims.

Figure 4: **Large-space FPE schemes.** All of these are blockcipher-based. The underlying blockcipher is assumed to have a blocksize of w bits (eg, $w = 128$) and we are doing length-preserving enciphering on an indicated set of strings. Some schemes not included in this enumeration (either broken, not blockcipher-based, or with unclear claims) include BEAR and LION [1], Mercy [13, 19], and PSPC [73].

position in the deck.

When analyzing shuffling-based techniques we can think of the coins that will be produced from the key as being chosen uniformly at random. The fact that they are *actually* to be derived from some short key K won't compromise the shuffle if cryptographically sound techniques are used to map the key to the needed coins.

Now for "most" ways to shuffle cards—again think of the usual riffle shuffle—the position that a card X lands in depends on a great many coins. It will depend on where the *other* cards ended up. You will thus need at least $O(N)$ time[1] to figure out where a single card goes; basically, you must figure out where *all* cards go to know where *any* card goes. But there turn out to be clever ways to shuffle a deck of cards where the trajectory of a card X hardly depends on the trajectory of other cards. For such a shuffle you can follow where a card moves, as you shuffle the deck, by looking at just a few coin tosses per shuffle. The amount of time to encrypt will thus depend principally on the number of rounds (shuffles), not the size of the domain (number of cards). Moni Naor calls such a shuffle *oblivious* [42, p. 62], [58, p. 17]. Naor noticed the potential utility of oblivious shuffling to cryptography some 20 years back. The Knuth shuffle (Figure 2) is not oblivious, so it only works to make a tiny-space FPE scheme. In contrast, the Thorp and recursive-merge shuffles (Figure 3) are oblivious shuffles.

LARGE-SPACE SCHEMES. There have been a great many blockcipher-based solutions offered for the problem, as enumerated in Figure 4. One reason is that the problem was taken up by the Security in Storage Working Group (SISWG), which will put out a standard known as IEEE P1619. The Working Group had initially focused on sector-level disk encryption. Another reason for the interest in wide-block encryption schemes is the wide range of solutions that are amenable to provable-security analysis: designing and proving secure a new one is fun, challenging, but not *too* challenging to succeed.

Blockcipher-based wide-block encryption schemes begin with Naor and Reingold [42, 43]. Those authors introduce a paradigm that has come to be called the *hash-encrypt-hash* approach. The hashing here refers to a kind of universal hash function that is actually a permutation, while the encryption is just ECB. Naor and Reingold do not fully specify any scheme, but subsequent schemes that build on their work, like TET [23], have done so.

Unhappy with the complexity of the hashing required by hash-encrypt-hash schemes, Shai Halevi and I introduced a different, *encrypt-mix-encrypt* approach [25]. Follow-on schemes include EME and EME2 [22, 24, 27]. Such schemes have two layers of encryption and not much else.

The latest approach, a *hash-ctr-hash* construction, is a variant of *hash-encrypt-hash*. It began with the precursor to XCB [37]. The inner encryption is CTR mode and the hash seems to be somewhat cheaper than it was with the NR-style schemes. XCB does the hash using $\mathrm{GF}(2^{128})$ multiplications in a manner reminiscent of GCM [48].

Given their likely standardization [27], users needing a wide-block scheme are probably best off selecting EME2 or XCB.

CLASSICAL FEISTEL. To a cryptographer, using a *Feistel network* is probably the most natural approach for solving the small-space FPE problem. The technique is simple, old, and extensively studied. It was popularized by the U.S. Data Encryption Standard (DES) (1974) [44] but dates back to even earlier than this [16, 17, 66].

The "classical" version of a Feistel network works like this. Suppose you want to encipher a binary string X. Assume that X has an *even* number of bits. Partition it into a left-hand side L and a right-hand side R. Take the right-hand side R and apply to it some key-dependent *round function* F_K^1 to get a processed right-hand side R^* that has the same length as the original right-hand side R. Next you xor together the already-mentioned left-hand side L and the processed right-hand side R^* to get the *new* right-hand side R'. The old right-hand side becomes the new left-hand side L'. This is round-1 of the classical Feistel network, mapping a left and right side, (L, R), to a new left and right side, (L', R'). Each subsequent round i works the same way except that you use variant F_K^i of the round function. You can use as many rounds as you wish. See Figure 5 for an illustration showing three rounds.

The Feistel construction always engenders a reversible function—that is, it "works" to make a blockcipher—no matter what you use for the round functions F_K^i. In particular, the round functions don't have to be permutations. Look at any one round of Figure 5, say the last. To go backwards one step works like this. You have an (L', R') pair, so you known the former right-hand side $R = L'$. Thus—assuming you know the key K—you can compute the processed right-hand side $R^* = F_K^i(R)$. Now we know that $R' = L \oplus R^*$, which implies that $L = R' \oplus R^*$. Thus we can compute the former left-hand side L. We have manged to go backwards one round. Repeat this for every round and you have reversed the entire encryption process. The fact that you can reverse it *means* that we have made a blockcipher.

Let us see how one might use the classical Feistel structure to encipher a 6-digit number, say the number 123456. Again see Figure 5. To encode a 6-digit number needs 20 bits, as $2^{19} < 10^6 < 2^{20}$, so we begin by writing 123456 as a 20-bit binary number, getting 0001111000 1001000000. The first 10 bits are made

[1] I really should write $\Omega(N)$ time, not $O(N)$ time, but I fear that the distinction is likely to confuse more people than it would inform. The same for when I spoke of tiny-space techniques using $O(N)$ time or memory to encrypt.

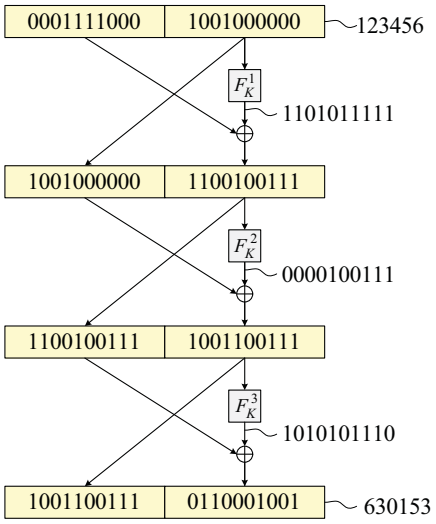

Figure 5: **Classical Feistel.** Three rounds of a Feistel network on a 20-bit string. Here 0001111000 1001000000 gets enciphered to 1001100111 0110001001 using the round function F. Its subscript is the key and its superscript is the round number.

the initial left-hand side L; the next 10 bits are the initial right-hand side R. We now go through the Feistel construction. In the illustration, just three rounds are shown. More rounds than this should used; the figure is just to get across how the construction works. The 10-bit numbers that result from each F_K^i application have been randomly generated in the figure; we have not specified what PRF is actually to be used. After we are done running the input through the Feistel network we get, in this example, the binary string $1001100111\,0110001001$, which, in decimal, is the number 630153. Thus we have enciphered the six-digit string 123456 into the six digit string 630153.

CYCLE WALKING. The astute reader may ask: but what if our final 20-bit number had exceeded 999999? Then we would *not* have enciphered our 6-digit number to a 6-digit number. Certainly getting a 7-digit result *is* possible; since $2^{20} = 1048576$ there are 48576 twenty-bit numbers that are too big for six decimal digits. The chance that we will get such a number as our final output, if everything is assumed to be random, is $(2^{20} - 10^6)/2^{20} = 48576/1048576 \approx 4.6\%$.

Well, if the ciphertext exceeds 999999 there is a simple solution: *just re-encipher it*. Keep re-enciphering until you get a 6-digit string.

I call the method just described *cycle walking*. It works, in general, like this. Suppose you would like to encipher on some message space $\mathcal{X}$; in other words, you aim to make an FPE scheme $E\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. Suppose what you know how to do is to encipher on a *superset* of $\mathcal{X}$; you have in hand an FPE scheme $E'\colon \mathcal{K} \times \mathcal{X}' \to \mathcal{X}'$ that works to encipher on $\mathcal{X}' \supseteq \mathcal{X}$. Then the cycle-walking method defines E in the following way:

algorithm $E_K(X)$ // *Encipher* $X \in \mathcal{X} \subseteq \mathcal{X}'$ *in* $\mathcal{X}$
repeat
  $X \leftarrow E'_K(X)$ // *Encipher* X *in* $\mathcal{X}'$
until $X \in \mathcal{X}$
return X

Correspondingly, decryption D of points in $\mathcal{X}$ using the decryption algorithm D' for points in $\mathcal{X}' \supseteq \mathcal{X}$ works like this:

algorithm $D_K(Y)$ // *Decipher* $Y \in \mathcal{X} \subseteq \mathcal{X}'$ *in* $\mathcal{X}$
repeat
  $Y \leftarrow D'_K(Y)$ // *Decipher* Y *in* $\mathcal{X}'$
until $Y \in \mathcal{X}$
return Y

But what, you might ask, if you *never* land in $\mathcal{X}$—what if the **repeat** . . . **until** just runs forever?! In fact, this cannot happen [8]. The reason is that we *start* on a point $X \in \mathcal{X}$ and encryption with a particular key traces out a directed cycle of points in $\mathcal{X}'$. So we know that the cycle we are walking on contains *some* point of $\mathcal{X}$, and so we must eventually hit *some* point of $\mathcal{X}$ as we walk around it.

No problem if you didn't follow the explanation just given. The point is that it works. Indeed it is possible to show [5], in a technical sense, that cycle walking not only terminates, but that it preserves, in the encryption scheme E, the hardness of the encryption scheme E'.

For cycle walking to be efficient, you need to have an efficient way to test if $X \in \mathcal{X}$. In addition, it better be the case that $\mathcal{X}'$ isn't *too* much larger than $\mathcal{X}$. That's because the expected number of times you're going to have to repeat the E' encryption (or the D' decryption) is the ratio of the domain sizes, $|\mathcal{X}'|/|\mathcal{X}|$. For example, the expected number of repetitions to encipher six digits using 20 bits is $2^{20}/10^6 = 1048756/1000000 \approx 1.05$. That's not too bad. But it won't work to, say, encipher a 16-digit PAN by cycle walking within the space of 128-bit strings, since 10^{16} is tiny compared to 2^{128}, meaning that you'd need an absurd number of repetitions.

We comment that cycle walking reveals side-channel information when it makes manifest how quickly a point in the space $\mathcal{X}$ is found. Recent work establishes what seems intuitively clear: that release of this timing information is not damaging [5]. Of course this statement has a precise technical meaning, which I am not describing here.

NONBINARY ALPHABETS. The slowdown associated to cycle walking can be undesirable. If you try to encrypt $\mathcal{X}$ by embedding it in the smallest message space $\mathcal{X}'$ that has at least as many points as $\mathcal{X}$ and that can be represented by binary strings of even length, then $\mathcal{X}'$ might be as much as

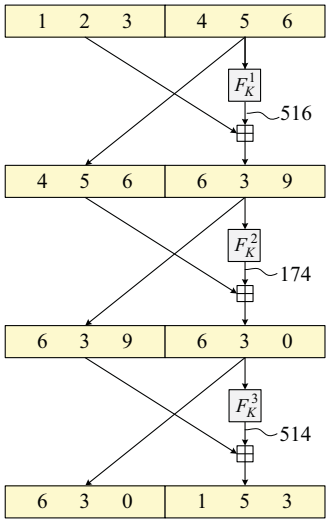

Figure 6: **Decimal Feistel.** Three rounds of a Feistel network on a 6-digit string. Here 123456 gets enciphered to 630153 using the round function F.

much as four times larger than $\mathcal{X}$. You'd have to repeat the encryption process E' (or the decryption process D') an expected four times. This might be a problem. Possibly worse, this is only an *expected* number of repetitions—the actual number could, on a given run, be considerably greater. For example, if $\mathcal{X}'$ is four times larger than $\mathcal{X}$ then 6.25% of the time you will need 20 or more repetitions. The unpredictability of the running time might be particularly undesirable in applications on low-end hardware where a customer is waiting for some transaction to complete.

One way to deal with this issue is to use the Feistel construction *directly* on the non-binary input string. This nice idea was first suggested, to my knowledge, by Terence Spies. See Figure 6, which shows three rounds of Feistel being used to encipher the string 123456. Rather than convert the plaintext into a binary string, as was done in Figure 5, we apply the construction directly to the decimal string. The initial left-hand side is the three-digit string 123 and the initial right-hand side is the three-digit string 456. The round function now maps three digits to three digits (instead of mapping 10 bits to 10 bits), doing this in a way that depends on the key K and the round number i. It no longer makes sense to xor ($\oplus$) a left-hand side L with a processed right-hand side R^* as xor is an operation only defined for binary strings. So, instead, we must use some other operator, denoted $\boxplus$ in Figure 6. What might this operator be?

If we are working with a decimal alphabet there are two natural realizations of $\boxplus$ that we might use. The first possibility for $\boxplus$ is *characterwise addition*. This means that, character by character, we add up, modulo 10, the corresponding characters of the equal-length strings L and R^*. Carries are ignored. So, with this type of addition, $456 \boxplus 174 = 520$, since $4+1 = 5 \pmod{10}$, $5+7 = 2 \pmod{10}$, and $6+4 = 0 \pmod{10}$. The second possibility for $\boxplus$ is *blockwise addition*. This means that we add up the two strings as though they were equal-length numbers, letting carries propagate except for the final carry, which is ignored. Said differently, we add up two m-digit numbers using ordinary addition modulo 10^m. In this case we'd have $456 \boxplus 174 = 630$ since $456 + 174 = 630 \pmod{10^3}$. An inspection of Figure 6 reveals that we have there assumed blockwise addition.

To the best of anyone's knowledge, there is no security difference between using characterwise and blockwise addition in a Feistel network. There will be an efficiency difference in some settings.

The above examples were for a decimal alphabet. But of course the same technique works for any radix. As an example, if we want to encrypt a string drawn from the 26-character alphabet $\{\mathrm{A}, \ldots, \mathrm{Z}\}$ then we can give the characters their natural numbering (A=0, $\cdots$, Z=25) and get, with characterwise addition, HELLO $\boxplus$ THERE $=$ ALPCS, since $(7{+}19)(4{+}7)(11{+}4)(11{+}17)(14{+}4) = (0)(11)(15)(2)(18)$. Here $(i + j)$ means the character numbered $i + j \pmod{26}$.

One pragmatic difficulty in realizing Feistel over a non-binary alphabet is that we will need a round function F that operates from and to this non-binary alphabet. Since off-the-shelf cryptographic tools operate on binary strings, we must construct the desired round function from a conventional pseudorandom function on and to binary strings. In Section 4 we sketch how this can be done using AES, the CBC MAC, and a bit of modular arithmetic.

UNBALANCED FEISTEL. So far we have seen Feistel-based methods that encipher even-length strings. But what if the input is of odd length? How can we encipher a 15-digit string, or a 5-digit one?

One possibility is to use cycle walking, embedding the message space of odd-length strings in a message space of even-length ones over the same or another alphabet. In terms of security, this works fine. But we have already discussed the efficiency downside of cycle walking. Is there a way to work directly with an odd-length input?

Two well-known generalizations of the Feistel construction can be used to encipher odd-length strings: *unbalanced Feistel* [63] and *alternating Feistel* [1, 32]. We now describe the former.

In the classical Feistel network each left-hand string L and right-hand string R have the *same* length. In that sense, the construction is *balanced*. It is also possible to have intermediate left- and right-hand values of *different* lengths $a = |L|$ and $b = |R|$. Such *unbalanced* Feistel networks were first discussed by Schneier and Kelsey [63]. See Figure 7 for an illustration of the tech-

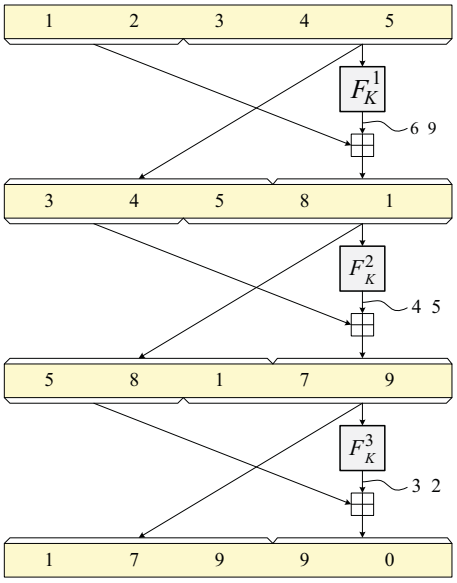

Figure 7: **Unbalanced Feistel.** Three rounds of an unbalanced Feistel network using a round function that maps three characters to two characters. Here 123456 gets encrypted to 17990.

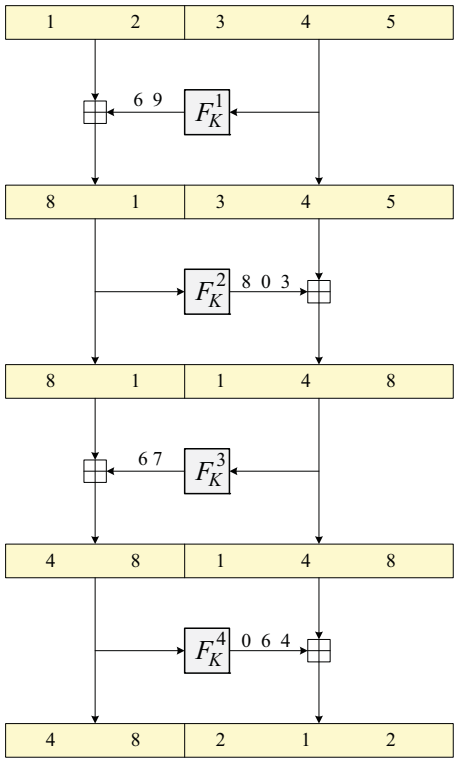

Figure 8: **Alternating Feistel.** Four rounds of an alternating Feistel network using a round function that alternates between mapping three characters to two characters and the other way around. Here string 123456 gets encrypted to 48212.

nique using a five-digit input. We use left-hand sides of $a = |L| = 2$ digits and right-hand sides of $b = |R| = 3$ digits. The round function F correspondingly maps $b=3$ characters to $a = 2$. After each application of it the old (three-digit) right-hand side becomes the new (three-digit) left-hand side. The new (two-digit) right-hand side is the sum (with respect to the operator $\boxplus$) of the old left-hand side and the processed right-hand side.

In Figure 7 we have persisted in using blockwise addition, so, for example, $12 \boxplus 69 = 81$. As before, character-wise addition is fine, too.

Relative to the input length n, the value $a = |L|$ quantifies the imbalance. We call it the *split*. Balanced Feistel networks can be considered a special case of unbalanced Feistel networks where the input has even length n and the split is $a = n/2$. The unbalanced Feistel scheme we illustrated in Figure 7 is *almost* balanced—with a split of $a = \lfloor n/2 \rfloor$, it's as balanced as you can get with an odd-length input. One can also consider more unbalanced schemes. At one extreme ($a = 1$) you would encipher an n-character string using a round function that mapped $b=n-1$ characters to $a=1$ character. The $n-1$ character R-value would effectively be shifted one position to the left with each round, a new character, computed from all the rest, joining in at the end. It would take n-rounds—what one might call one "pass"—to work your way one time through the string.

Schneier and Kelsey call an unbalanced Feistel network *source-heavy* when $a < b$; the round function's input is longer than its output, so it is said to be *contracting*. The scheme is *target-heavy* if $a > b$; the round function is then *expanding*. Security results for unbalanced Feistel networks can be found in [26, 39, 42], while attacks (not very effective) can be found in [51, 52]. For source-heavy schemes, quantitative security results improve as the imbalance increases (ie, the split goes down), assuming you adequately compensate for the imbalance with a sufficient number of additional rounds. On the other hand, security considerations suggest that one avoid target-heavy schemes with large split $a=|L|$.

We introduced unbalanced Feistel networks as a way to deal with odd-length strings. But you can now see that the schemes aren't just for that: unbalanced Feistel networks are possible for inputs of any length, and over any alphabet.

ALTERNATING FEISTEL. An oddity of unbalanced Feistel is the implicit re-partitioning of intermediate strings. Again refer to Figure 7. The round-1 output of $345 \,\|\, 81$ must be re-conceptualized as $34 \,\|\, 581$ so that the round-2 round function can be applied to the 581 and the resulting output can be added to the 34. This might seem to be only a "viewpoint shift," not actual work. But, in an implementation, it likely *does* correspond to actual work: we will likely have a pair of integer variables holding $(345,\ 81)$

and need to create from them a pair of integer variables holding $(34,\ 581)$. This will take some arithmetic.

Alternating Feistel, depicted in Figure 8, avoids repartitioning and, therefore, the corresponding implicit arithmetic. This is not the reason it was invented, but it is one reason for liking this generalized Feistel scheme. The method dates to Anderson and Biham [1] and, independently, to Lucks [32]. Two kinds of round functions are alternately used: a contracting one and an expanding one. Even-numbered rounds work one way and odd-numbered rounds work the other way. The operator $\boxplus$ can again be realized by characterwise or blockwise addition.

Our illustration was again for the "nearly-balanced" setting; for a five-character string, the left- and right-hand pieces were as close to being balanced as possible. This isn't the only possibility. For example, one could, with a five-character string, have the compressing round function map four characters to one, and have the expanding round function map one character to four. A parameter a, the *split*, indicates the desired imbalance, the original n-character string being divided into a length-a left-hand piece L and a length-$(n-a)$ right-hand piece R.

Note that balanced Feistel is again a special case; it can be regarded as an alternating Feistel scheme with an even number of characters n and a split of $a = n/2$. Both unbalanced and alternating Feistel networks can be regarded as upwardly-compatible generalizations of the classical Feistel design.

param	**description**
radix	The *radix*, a number 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 functions.
addition	The *addition operator*, either characterewise addition or blockwise addition. Determines the meaning of $\boxplus$.
method	The *Feistel method*, either balanced, unbalanced, or alternating (the first is actually regarded as a special case of either of the other two).
split	The *imbalance*, a number (actually, a function) specifying the degree of imbalance in our unbalanced or alternating Feistel network.
rnds	The *number of rounds*, a number (actually, a function) specifying the number of rounds to use.
F	The *round function*, on $K \in \mathsf{Keys}$, $n \in \mathsf{Lengths}$, $T \in \mathsf{Tweaks}$, $i \in \{0, \ldots, \mathsf{rnds} - 1\}$, and $B \in \mathsf{Chars}^*$, it returns a string $\mathsf{F}_K(n, T, i, B) \in \mathsf{Chars}^*$ of the appropriate length.

Figure 9: **Parameters of FFX.** To have a fully-specified scheme, each of these parameters must be defined.

4 FFX Mode

OVERVIEW. To help bring small-space FPE to practice, Mihir Bellare, Terence Spies, and I have written a specification document for a Feistel-based design [7]. Our mode of operation is called FFX (Format-preserving, Feistel-based encryption). It is derivative of an earlier proposal, FFSEM, authored by Spies [68]. Using FFX one can encipher strings of any length over any desired alphabet. FFX employs a pseudorandom function as its underlying cryptographic primitive. We expect this to be built from AES.

There were a large number of concerns to try to balance in designing FFX. In the end, we decided on a structure like this. The specification document defines the general structure for a Feistel-based FPE scheme. It is general in the sense that it depends on a number of *parameters*. In all, nine parameters are used; see Figure 9. To have a fully concrete scheme one must pin down all parameters. The situation is analogous to the specification of HMAC, say, which does not mandate a particular hash function or MAC length. The basic mechanism underlying FFX is a balanced, unbalanced, or alternating Feistel network over an arbitrary alphabet—what we already described in Section 3. Formally, balanced Feistel networks are just regarded as a special case of either of the other two kinds.

param	**A2**	**A10**
radix	2	10
Lengths	$\{8, 9, \ldots, 128\}$	$\{4, 5, \ldots, 36\}$
Keys	$\{0,1\}^{128}$	$\{0,1\}^{128}$
Tweaks	BYTE^*	BYTE^*
addition	xor	blockwise
method	alternating Feistel	alternating Feistel
split	maximally balanced	maximally balanced
rnds	12 or more	12 or more
F	AES-based	AES-based

Figure 10: **Schemes FFX-A2 and FFX-A10.** The former allows the encryption of binary strings, the latter, decimal strings. For a full description of these modes, see the spec [7].

Because of the open-endedness of FFX, we specify a couple of *parameter collections*, A2 and A10, which select values for all parameters. Scheme FFX-A2 can encipher binary strings of 8–128 bits. Scheme FFX-A10 can

encipher decimal strings of 4–36 digits. Both modes use an alternating Feistel network that is as close to balanced as possible (for messages with an even number of characters, the scheme *is* a balanced Feistel network). The round functions for FFX-A2 and FFX-A10 use AES operating in the CBC MAC mode of operation. This is the classical way to turn a blockcipher into a pseudorandom function—a method standardized by ANSI, ISO, and NIST [2, 28, 46] that has strong provable-security guarantees [3, 4, 55]. See Figure 10 for a description of FFX-A2 and FFX-A10 parameters.

As we've explained Feistel schemes already, little more need be said to understand the structure of FFX. Still, in making an "industrial strength" specification there were some significant matters to attend to, as we now describe.

TWEAKS. The FFX scheme is *tweakable*: a byte string T, the tweak, can be provided when one encrypts or decrypts a string. Encryption takes the form $Y = \mathcal{E}_K^T(X)$; decryption, $X = \mathcal{D}_K^T(Y)$. As indicated in Section 2, each tweak T conceptually names its "own" permutation, independent from those associated to all other tweaks.

Why is the tweak needed? Consider the encryption of the middle six digits of a 16-digit PAN. The envisaged application is unable to encrypt the entire PAN because, for example, legacy architectural considerations mandate that the first six digits and the last four digits be presented in the clear. One might then worry that our message space has become too small: multiple PANs in a large database *will* have the same middle-six digits. Without using a tweak, these will encrypt to the same thing. The adversary may then assemble a dictionary mapping known six-digit plaintexts to their corresponding ciphertexts. The dictionary could be used to decrypt unknown ciphertexts.

The exposure suggested above is unnecessary: what *my* middle-six digits encrypt to in the context of *my* PAN need tell an adversary nothing about what *your* middle-six digits encrypts to in the context of *your* PAN. To ensure this is so, use the non-secret ten digits of the PAN as the tweak T for encrypting the remaining six-digit plaintext. Now identical six-digit plaintexts will *not* give rise to identical six-digit ciphertexts (except by chance or for identical PANs). In general, the string Y that a plaintext X encrypts to with tweak T will tell an adversary nothing about the preimage of a ciphertext Y' that was encrypted using some *other* tweak T'.

Summarizing, when you encrypt a plaintext X using an FPE scheme, it is desirable—sometimes even essential—to provide as a tweak T everything that the recipient will know about X and its context. Providing such a tweak will in most cases be enough to overcome the (unavoidable) fact that, with any deterministic encryption scheme, identical inputs do map to identical outputs.

In the definition of FFX, the tweak T is included in the scope of the round function F. This function is expected to be pseudorandom, so including T within its scope has the effect of creating an unrelated round function for unequal tweaks.

Designers of recent wide-block encryption schemes (eg, EME2 and XCB) have also been careful to accommodate tweaks in their designs.

ROUND FUNCTION. In both FFX-A2 and FFX-A10, the selected round function begins by applying the AES CBC MAC, keyed by the underlying key K, to a string that encodes the round number i, the input string B, and the tweak T. (Some other values, like the radix and input length, are also thrown in.) The result is a 128-bit string. But the round function is not supposed to output a 128-bit string—we need, instead, to create a string of the desired length m over the desired character set. For example, the round function for FFX-A10 will, given a five-digit input, need to generate $m = 2$ or $m = 3$ digits, according to the parity of the round, as was shown in Figure 8.

For FFX-A2, whose round function must create a binary string, we can excise the needed number of bits a from the 128-bit AES CBC MAC output. We will need $4 \le m \le 64$ bits, since FFX-A2 can encipher strings of 8–128 bits. On the other hand, FFX-A10 can encipher strings of 4–36 digits, so our round function must be able to generate outputs of 2–18 digits. To create $2 \le m \le 9$ digits we take the last 64 bits of the AES CBC MAC output modulo 10^m. To create $10 \le m \le 18$ digits we similarly process the first 64 bits, too. The decimal strings we generate in this way will be nearly as unpredictable as the 128-bit strings from which they come (a formal statement corresponding to this, quantifying the *bias*, is not hard to state and prove).

The round functions for FFX-A2 and FFX-A10 were designed so that, even if a tweak is long, you will only have to pay to process it one time. Achieving this sort of efficiency property can be done using standard "crypto-engineering" kinds of tricks (the tweak T is simply put near the "beginning" of the CBC MAC). For full details on FFX and its predefined parameter collections, see the mode's specification document [7].

NUMBER OF ROUNDS. Considerable attention was paid to the minimum number of rounds allowed in FFX and the number of rounds selected for FFX-A2 and FFX-A10. We considered provable-security results, attacks, round-selection heuristics, efficiency, and simplicity. We selected very conservative values. For the balanced and nearly-balanced settings, FFX is required to use at least eight rounds. Mechanisms FFX-A2 and FFX-A10 use more: 12–36 rounds and 12–24 rounds, respectively. In both cases the actual number of rounds used depends on the length n of the input (the spec contains a small table). We actually use *more* rounds for *shorter* inputs. While the authors do not know this to be necessary, as the in-

put length decreases quantitative security results do get worse, while the best attacks do improve, suggesting that, at least for now, the conservative choice is to devote more rounds for shorter strings. For further discussion, see the spec [7].

5 Discussion

RANK-THEN-ENCIPHER FPE. One reason that we focus on numeric message spaces $\mathcal{X} = [N]$ or string message spaces $\mathcal{X} = \Sigma^n$ is due to the *rank-then-encipher* approach for encrypting on an arbitrary finite set. Let's illustrate the idea with a concrete example.

Suppose you want to encipher 16-digit strings with a zero Luhn checksum. Each string represents a (valid-checksum) PAN. To encrypt a PAN X, grab its first 15 digits, X'; encipher X' as a 15-digit string to get a 15-digit ciphertext Y'; then convert this 15-digit string Y' to a valid 16-digit PAN Y by appending to Y' whatever final digit is necessary to produce the correct checksum.

More abstractly, the rank-then-encipher approach works like this. There is some finite message space $\mathcal{X}$ on which you would like to do FPE. You already *know* how to do FPE on some *other* message space $\mathcal{X}'$ having the same number of points as $\mathcal{X}$. So, to encipher a point $X \in \mathcal{X}$ map it to a corresponding point $X' \in \mathcal{X}'$, encipher that point in $\mathcal{X}'$ to get a ciphertext $Y' \in \mathcal{X}'$, then map Y' to its corresponding point Y in $\mathcal{X}$. The maps between $\mathcal{X}$ and $\mathcal{X}'$ must be bijective—that is, each point in $\mathcal{X}$ is matched up with with one and only one point in $\mathcal{X}'$.

We call the above *rank-then-encipher encryption*. The set $\mathcal{X}'$ is usually of the form $\mathcal{X}' = [N]$, or something easily regarded as equivalent, in which case the map from $\mathcal{X}$ to $\mathcal{X}'$ is a *ranking*—a numbering—of the points in $\mathcal{X}$.

Rank-then-encipher encryption can be considered a "meta-technique" for building an FPE scheme. By a meta-technique I mean a method that turns an FPE scheme on one message space into an FPE scheme on another, closely related one. Cycle walking was another meta-technique that we looked at. See Figure 11.

The rank-then-encipher approach lets us create FPE schemes on the kinds of message spaces that arise in practice. Theoretical work informs us that there *are* message spaces where FPE is possible but where the rank-then-encipher approach won't work [5]. But the known examples are highly artificial—message spaces on which one would never *really* want to encrypt.

Finding ranking and unranking functions (the maps between $\mathcal{X}$ and $\mathcal{X}'$) is not a cryptographic problem; there is no cryptographic significance as to how you do it. Choose as simple and efficient a method as can be found.

PROOFS AND ATTACKS. It is beyond the scope of this note to explain or survey all the provable-security results associated to Feistel schemes; we instead refer the reader to the papers referenced in Figure 3. It is fair to say, however, that provable-security results for Feistel have been steadily improving, both in quantitative power and in scope. Known security proofs provide quantitatively useful guarantees. Still, there remains a huge gap between what we can prove about the security of Feistel constructions and what the best known attacks actually accomplish. Part of the issue is that an information-theoretic analysis of Feistel networks simply cannot take us as far as one might like, as there are (absurdly inefficient) information-theoretic attacks on Feistel constructions when the number of queries exceeds the size of the domain of the round function.

In most protocol-design domains I do not take that absence of known and effective attacks as evidence of their inexistence. But Feistel is different. It has been the most well-known approach for making blockciphers for 35 years. Effective attacks on Feistel-based constructions almost never attempt to attack the Feistel structure *itself*; they look, instead, for defects in the round function used. In all that time that Feistel has been around, there has never been suggested an effective attack on the Feistel design itself. When random round functions are used and the number of rounds is six or more, the best attack known, in terms of time complexity, is the (relatively obvious) meet-in-the-middle attack [50] that runs in doubly-exponential time: for r rounds of balanced Feistel on $2n$ bits, the time exceeds $2^{0.5\,nr2^n}$ steps. For enciphering decimal strings the number 2 is replaced by 10 and the time exceeds $10^{0.5\,nr10^n}$ computational steps. To be concrete, attacking just eight rounds of a balanced Feistel construction employing a random round function and applied to a string of just six decimal digits would take, in the best attack known, $10^{12,000}$ time (and 4001 oracle queries). Assuming we have made our round function from AES, the far more practical line of attack would be to do exhaustive search on its key space. Exhaustive search on AES is, by far, the best attack we know on FFX.

CLOSING REMARKS. Some people, on hearing FPE described, suspect that it must be some sort of cryptographic snake oil. Hopefully this note has made obvious that it is not. Once one understands what FPE is, its relation to other cryptographic primitives, and that there is no inherent insecurity associated to enciphering on a small message space, it should not be surprising that the cryptographic community has good solutions for this problem already well in hand. Whatever seems magical about FPE is, in the end, no more magical than what one sees across the landscape of contemporary cryptography.

Given FPE's utility, and given the maturity of techniques known for achieving it, standardization and widespread use of FPE would seem to be inevitable.

method	can encipher on	description	security
Cycle walking [8]	$\mathcal{X}$ where you have in hand a scheme that enciphers on $\mathcal{X}' \supseteq \mathcal{X}$ and membership in $\mathcal{X}$ is easily tested.	To encrypt $X \in \mathcal{X}$ repeatedly encipher it in $\mathcal{X}'$ until you land at a point in $\mathcal{X}$. Takes $\lvert\mathcal{X}'\rvert/\lvert\mathcal{X}\rvert$ expected steps, so $\mathcal{X}$ should not be too sparse in $\mathcal{X}'$. Classical construction (circa 1915) [65].	Preserves provable security of underlying FPE scheme.
Rank-then-encipher [5]	$\mathcal{X}$ where you have in hand a scheme that enciphers on $\mathcal{X}'$ and you can bijectively map between $\mathcal{X}$ and $\mathcal{X}'$.	To encrypt $X \in \mathcal{X}$ bijectively map it to a point in $\mathcal{X}'$, encrypt it in $\mathcal{X}'$, then bijectively map it back to $\mathcal{X}$. To be efficient, mapping from $\mathcal{X}$ to $\mathcal{X}'$ and back must be efficient.	Preserves provable security of underlying FPE scheme.

Figure 11: **Meta-techniques for FPE.** These methods turn an FPE scheme with one message space into an FPE with another, not vastly different one. The utility of these methods helps explain a focus on integer and string FPEs.

Acknowledgments

The author gratefully acknowledges the role of Voltage Security in rekindling my interest in the FPE problem, leading to publications [5, 26, 39], specification document [7], and of course this expository note. Particular thanks are due to Terence Spies, whose work in connection with FPE has enabled some of my own ideas to make that difficult journey into the real world.

Kind thanks also to Viet Tung Hoang, Terence Spies, and Till Stegers for helpful comments and corrections.

References

[1] R. Anderson and E. Biham. Two practical and provably secure block ciphers: BEAR and LION. *Fast Software Encryption* (FSE 1996), LNCS 1039, Springer, pp. 113–120, 1996.

[2] ANSI X9.19. Financial institution retail message authentication. American Bankers Association, 1986.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3), pp. 362–399, 2000. Earlier version in *CRYPTO 1994.*

[4] M. Bellare, K. Pietrzak, and P. Rogaway. Improved security analyses for CBC MACs. *Advances in Cryptology – CRYPTO 2005.* LNCS 3621, Springer, pp. 527–545, 2005.

[5] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. *Selected Areas in Cryptography* (SAC 2009), Springer, pp. 295–312, 2009.

[6] M. Bellare and P. Rogaway. On the construction of variable-input-length ciphers. *Fast Software Encryption* (FSE 1999), LNCS 1636, Springer, pp. 321–344, 1999.

[7] M. Bellare, P. Rogaway, and T. Spies. The FFX mode of operation for format-preserving encryption (Draft 1.1). February, 2010. Manuscript (standards proposal) submitted to NIST.

[8] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. *Topics in Cryptology – CT-RSA.* LNCS 2271, Springer, pp. 114–130, 2002.

[9] M. Brightwell and H. Smith. Using datatype-preserving encryption to enhance data warehouse security. *20th NISSC Proceedings*, pp. 141–149, 1997. Available at csrc.nist.gov/nissc/1997.

[10] D. Chakraborty and M. Nandi. An improved security bound for HCTR. *Fast Software Encryption* (FSE 2008), LNCS 5086, Springer, pp. 289–302, 2008.

[11] D. Chakraborty and P. Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. *IEEE Transactions on Information Theory*, 54(4), pp. 1683–1699, 2008.

[12] D. Chakraborty and P. Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. *Fast Software Encryption* (FSE 2006), LNCS 4047, Springer, pp. 293–309, 2006.

[13] P. Crowley. Mercy: A fast large block cipher for disk sector encryption. *Fast Software Encryption* (FSE 2000), LNCS 1978, Springer, pp. 49–63, 2000.

[14] A. Czumaj, P. Kanarek, M. Kutylowski and K. Lorys. Fast generation of random permutations via networks simulation. *Algorithmica*, 21(1), Springer, May 1998.

[15] R. Durstenfeld. Algorithm 235: Random permutation. *CACM*, 7(7), p. 420, July 1964.

[16] H. Feistel. Block cipher cryptographic system. US Patent #3,798,359. March 19, 1974.

[17] H. Feistel, W. Notz, and J. Smith. Some cryp-

tographic techniques for machine-to-machine data communications. *Proc. of the IEEE*, 63, pp. 1545–1554, 1975.

[18] R. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research,* 3rd ed.. Oliver & Boyd, pp. 26–27, 1938 (1948).

[19] S. Fluhrer. Cryptanalysis of the Mercy block cipher. *Fast Software Encryption* (FSE 2002), LNCS 2355, Springer, pp. 21–40, 2002.

[20] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci*, 28(2), pp. 270–299, 1984.

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

[22] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. *INDOCRYPT 2004*, LNCS 3348, Springer, pp. 315–327, 2004.

[23] S. Halevi. Invertible universal hashing and the TET encryption mode. Cryptology ePrint report 2007/014. May 24, 2007.

[24] S. Halevi and P. Rogaway. A parallelizable enciphering mode. *Topics in Cryptology – CT-RSA 2004*, LNCS 2964, Springer, pp. 292–304, 2004.

[25] S. Halevi and P. Rogaway. A tweakable enciphering mode. *Advances in Cryptology – CRYPTO 2003*, LNCS 2729, Springer pp. 482–499, 2003.

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

[27] IEEE P1619.2. Draft standard architecture for wide-block encryption for shared storage media. File EME2-for-p1619-2_v3. 2008.

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

[29] D. Knuth. *The Art of Computer Programming*, vol. 2, 3rd ed., pp. 145–146, 1969 (1998).

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

[31] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput*, 17(2), pp. 373–386, 1988.

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

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

[34] U. Mattsson. Format controlling encryption using datatype preserving encryption. Cryptology ePrint Report 2009/257.

[35] U. Maurer. A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators. *Advances in Cryptology – EUROCRYPT 1992*, LNCS 658, pp. 239–255, 1992.

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

[37] D. McGrew and S. Fluhrer. The security of the extended codebook (XCB) mode of operation. *Selected Areas of Cryptography* (SAC 2007), LNCS 4876, Springer, pp. 311–327, 2007.

[38] D. McGrew and J. Viega. Arbitrary block length (ABL) mode: security without data expansion. Submission to IEEE SISWG. April 15, 2004.

[39] B. Morris, P. Rogaway, and T. Stegers. How to encipher messages on a small domain: deterministic encryption and the Thorp shuffle. *Advances in Cryptology – CRYPTO 2009*, LNCS 5677, Springer, pp. 286–302, 2009.

[40] L. Moses and R. Oakford. *Tables of Random Permutations.* Stanford University Press, 1963.

[41] M. Nandi. Improving upon HCTR and matching attacks for Hash-Counter-Hash approach. Cryptology ePrint report 2008/090. Feb 28, 2008.

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

[43] M. Naor and O. Reingold. The NR mode of operation. Undated manuscript describing the mechanism of [42].

[44] National Bureau of Standards. FIPS 46. Data Encryption Standard. U.S. Dept. of Commerce, 1977.

[45] National Bureau of Standards. FIPS 74. Guidelines for Implementing and Using the NBS Data Encryption Standard. U.S. Dept. of Commerce, 1981.

[46] National Bureau of Standards. FIPS 113. Computer data authentication. U.S. Dept. of Commerce, 1985.

[47] National Institute of Standards. FIPS 197. Specification for the Advanced Encryption Standard. U.S. Dept. of Commerce, 2001.

[48] National Institute of Standards. NIST Special Pub. SP800-38D, Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Authored by M. Dworkin. 2007.

[49] J. Patarin. Security of random Feistel scheme with 5 or more rounds. Undated manuscript based on *CRYPTO 2004* paper of the same title.

[50] J. Patarin. Security of random generic attacks on Feistel schemes. Cryptology ePrint report 2008/036. Earlier work in *ASIACRYPT 2001*.

[51] J. Patarin, V. Nachef, and C. Berbain. Generic attacks on unbalanced Feistel schemes with contracting functions. *ASIACRYPT 2006*. LNCS 4284, Springer, pp. 396–411, 2006.

[52] J. Patarin, V. Nachef, and C. Berbain. Generic attacks on unbalanced Feistel schemes with expanding functions. *ASIACRYPT 2007*. LNCS 4833, Springer, pp. 325–341, 2007.

[53] S. Patel, Z. Ramzan, and G. Sundaram. Efficient constructions of variable-input-length block ciphers. *Selected Areas in Cryptography* (SAC 2004), LNCS 3357, Springer, pp. 326–340, 2005.

[54] Payment Card Industry (PCI) Data Security Standard: Requirements and Security Assessment Procedures, version 1.2. October 2008.

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

[56] T. Ristenpart and P. Rogaway. How to enrich the message space of a cipher. *Fast Software Encryption* (FSE 2007), LNCS 4593, Springer, pp. 101-118, 2007.

[57] R. Rivest, M. Robshaw, and Y. L. Yin. The RC6 block cipher. Manuscript, August 20, 1998.

[58] S. Rudich. Limits on the provable consequences of one-way functions. Ph.D. Thesis, UC Berkeley, 1989.

[59] P. Sanders. Random permutations on distributed, external and hierarchical memory. *Information Processing Letter*, 67, pp. 305–309, 1998.

[60] P. Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. Cryptology ePrint report 2008/004. Extends [61].

[61] P. Sarkar. Improving upon the TET mode of operation. *Information Security and Cryptology* (ICISC 2007), LNCS 4817, Springer, pp. 180–192, 2007.

[62] P. Sarkar. Tweakable enciphering schemes using only the encryption function of a block cipher. Cryptology ePrint report 2009/216.

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

[64] R. Schroeppel. The Hasty Pudding cipher. Manuscript. AES candidate submitted to NIST. 1998.

[65] R. Schroeppel. Personal communication, approximately 2001.

[66] J. Smith. The design of Lucifer: a cryptographic device for data communications. IBM Research Report RC 3326, IBM T.J. Watson Research Center, Yorktown Heights, New York, USA. April 15, 1971.

[67] D. Socek, H. Kalva, S. Magliveras, O. Marques, D. Culibrk, and B. Furht. New approaches to encryption and steganography for digital videos. *Multimedia Systems*, 13, Springer, pp. 191–204, 2007.

[68] T. Spies. Feistel finite set encryption. NIST submission. February 2008. Evolved into [7].

[69] T. Spies. Personal communications, February 2009.

[70] T. Spies. Format preserving encryption. Manuscript, www.voltage.com/library.shtml. See also: Format preserving encryption: www.voltage.com. *Database and Network Journal*, Dec. 2008.

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

[72] P. Wang, D. Feng, and W. Wu. HCTR: A variable-input-length enciphering mode. *Information Security and Cryptology – CISC 05*, LNCS 3822, Springer, pp. 175–188, 2005.

[73] Y. Zheng, T. Matsumoto, and H. Imai. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. *Advances in Cryptology – CRYPTO '89*, LNCS 435, Springer, pp. 461–480, 1989.

A Formal Definition

Some of this section is taken from Bellare, Ristenpart, Rogaway, and Stegers [5] (which, in turn, evolved from an earlier version of this section).

SYNTAX. A scheme for *format-preserving encryption* (FPE) is a function $\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X} \cup \{\perp\}$ where the sets $\mathcal{K}$, $\mathcal{N}$, $\mathcal{T}$, and $\mathcal{X}$ are called the *key space*, *format space*, *tweak space*, and *domain*, respectively. All of these sets are nonempty and exclude $\perp$. We write $\mathcal{E}_K^{N,T}(X) = \mathcal{E}(K, N, T, X)$ for the encryption of X with respect to key K, format N, and tweak T. We require that whether or not $\mathcal{E}_K^{N,T}(X) = \perp$ depends only on N and X, not on K or T, and we let $\mathcal{X}_N = \{X \in \mathcal{X}\colon E_K^{N,T}(X) \in \mathcal{X} \text{ for all } (K,T) \in \mathcal{K} \times \mathcal{T}\}$ be the N-indexed *slice* of the domain. We demand that a point $X \in \mathcal{X}$ live in at least one slice, $X \in \mathcal{X}_N$ for some N (if X is in no slice it should not be included in $\mathcal{E}$'s domain). We demand that there be finitely many points in each slice, meaning that $\mathcal{X}_N$ is finite for all $N \in \mathcal{N}$. We require that $\mathcal{E}_K^{N,T}(\cdot)$ be a permutation on $\mathcal{X}_N$ for any $(K,T) \in \mathcal{K} \times \mathcal{T}$. Its inverse $\mathcal{D}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X} \cup \{\perp\}$ is defined by $\mathcal{D}_K^{N,T}(Y) = \mathcal{D}(K, N, T, Y) = X$ if $\mathcal{E}_K^{N,T}(X) = Y$.

In summary, an FPE scheme enciphers the points in each of the (finite) slices $\mathcal{X}_N$ that collectively comprise its domain $\mathcal{X}$.

A practical FPE scheme $\mathcal{E}\colon \mathcal{K}\times\mathcal{N}\times\mathcal{T}\times\mathcal{X} \to \mathcal{X}\cup\{\perp\}$ must be realizable by efficient algorithms: an algorithm $\mathcal{E}$ to encrypt, an algorithm $\mathcal{D}$ to decrypt, and an algorithm to sample uniformly from the key space $\mathcal{K}$. Thus $\mathcal{K}$, $\mathcal{N}$, $\mathcal{T}$, and $\mathcal{X}$ should consist of strings or points easily encoded as strings, and the algorithms for E and D should return $\perp$ when presented a point outside of we $\mathcal{K}\times\mathcal{N}\times\mathcal{T}\times\mathcal{X}$. We do not draw any distinction between an integer element of $\mathcal{X}$, say, and a string that encodes such a point.

Sometimes the format space and/or the tweak space are *trivial*, meaning that $|\mathcal{N}| = 1$ or $|\mathcal{T}| = 1$, respectively. In such cases the format or tweak is irrelevant and we omit mention of it, writing, for example, $E_K^N(X)$, $E_K^T(X)$, or $E_K(X)$ for the encryption under a scheme with trivial tweak space, format space, or both.

THE FORMAT OF A POINT. Let $\mathcal{E}\colon \mathcal{K}\times\mathcal{N}\times\mathcal{T}\times\mathcal{X} \to \mathcal{X}\cup\{\perp\}$ be an FPE scheme. Then we can speak of $X \in \mathcal{X}$ as having *format* N if $X \in \mathcal{X}_N$. One could associate to $\mathcal{E}$ a *format function* $\phi\colon \mathcal{X} \to \mathcal{P}(\mathcal{N}) \setminus \{\emptyset\}$ that maps each $X \in \mathcal{X}$ to its possible formats; formally, $\phi(X) = \{N \in \mathcal{N}\colon X \in \mathcal{X}_N\}$.

With the definition above, a point may have multiple formats. Often this will not be the case: each $X \in \mathcal{X}$ will belong to exactly one $\mathcal{X}_N$. In that case we can regard the format function as mapping $\phi\colon \mathcal{X} \to \mathcal{N}$ and interpret $\phi(X)$ as *the* format of X. FPE is somewhat simpler to understand for such *unique-format* FPEs: you can examine an X and know the slice $\mathcal{X}_{\phi(X)}$ on which you mean to encipher it. For a unique format FPE it is fine to write $\mathcal{E}_K^T(X)$ instead of $\mathcal{E}_K^{N,T}(X)$ since N is determined by X.

One interpretation of format function ϕ is to capture the partial information about the plaintext that the encryption is *allowed* to leak. Ciphertexts are expected to be indistinguishable from one another only when they are distinct and of a common format.

SPECIFICATIONS. An FPE problem, as needed by some application, will specify the desired collection of slices, $\{\mathcal{X}_N\}_{N\in\mathcal{N}}$. It will also specify the desired tweak space $\mathcal{T}$. Typically it is easy to support whatever tweak space one wants, but it may be quite hard to support a given collection of slices $\{\mathcal{X}_N\}_{N\in\mathcal{N}}$. Indeed it may be hard to accommodate a single slice, depending on what it is. We therefore call the collection of slices $\{\mathcal{X}_N\}_{N\in\mathcal{N}}$ the *specification* for an FPE scheme. We will write $\mathcal{X} = \{\mathcal{X}_N\}_{N\in\mathcal{N}}$ for a specification, only slightly abusing notation because the domain $\mathcal{X}$ *is* the union of slices in $\{\mathcal{X}_N\}_{N\in\mathcal{N}}$. The question confronting the cryptographer is *how to design an FPE scheme with a given specification*. We now provide some example possibilities.

EXAMPLES. **(1)** AES-128 can be regarded as an FPE with a single slice, $\{0,1\}^{128}$. The key space is $\mathcal{K} = \{0,1\}^{128}$ and the format space and tweak space are trivial. **(2)** To encipher 16-digit decimal numbers, take $\mathcal{X} = \{0,1,\ldots,9\}^{16}$ and just the one slice. **(3)** To encipher 512-byte disk sectors using an 8-byte sector index as the tweak, let $\mathcal{X} = \{0,1\}^{4096}$, $\mathcal{T} = \{0,1\}^{64}$, and just the one slice. **(4)** To encipher PANs of 12–19 digits with a proper Luhn checksum, the ciphertext having the same length as the plaintext, the specification could be $\mathcal{X} = \{\mathcal{X}_N\}_{N\in\mathcal{N}}$ where $\mathcal{N} = \{12,13,\ldots,18,19\}$ and $\mathcal{X}_N$ is the set of all strings $X \in \{0,1,\ldots,9\}^N$ satisfying the predicate $\mathsf{LuhnOK}(X)$. Here $|\mathcal{X}_N| = 10^{N-1}$. **(5)** One nice FPE scheme has slices that are $\{0,1\}^N$ for each $N \geq 0$. It allows length-preserving encryption of any binary string. **(6)** One can imagine doing FPE on rather unusual spaces. For example, slice $\mathcal{X}_N$ could encode all N-vertex graphs, or $\mathcal{X}_N$ could be all valid C programs on N bytes. Designing an efficient FPE for the last example might well be impossible.

INTEGER FPES. All of the examples just given were unique-format FPEs. The following example is not. The slices are $\mathcal{X}_N = \mathbb{Z}_N$ for $N \in \mathcal{N} \subseteq \mathbb{N}$. This allows enciphering natural numbers with respect to any permitted modulus N. We call such schemes *integer* FPEs.

STRING FPES. Here is another useful FPE, this one a unique-format FPE. The slices are $\mathcal{X}_N = \Sigma^N$ for $N \in \mathcal{N} \subseteq \mathbb{N}$ and Σ an arbitrary (finite, nonempty) alphabet. This allows enciphering strings of permitted lengths over desired alphabets. We call such schemes *string* FPEs.

SECURITY. Let $\mathcal{E}\colon \mathcal{K}\times\mathcal{N}\times\mathcal{T}\times\mathcal{X} \to \mathcal{X}\cup\{\perp\}$ be an FPE scheme. Let A be an algorithm with access to two oracles. Then we define A's *advantage* (or, more specifically, its *strong PRP advantage*) $\mathbf{Adv}_{\mathcal{E}}^{\pm\text{prp}}(A)$ as the real number

$$\Pr[A^{\mathcal{E}_K(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[A^{\pi(\cdot,\cdot,\cdot),\pi^{-1}(\cdot,\cdot,\cdot)} \Rightarrow 1]$$

where the oracles that superscript A have the following behavior. In the experiment underlying the first addend we choose a random $K \xleftarrow{\$} \mathcal{K}$ and then provide the adversary oracles for $\mathcal{E}_K(\cdot,\cdot,\cdot)$ and $\mathcal{D}_K(\cdot,\cdot,\cdot)$. In the experiment underlying the second addend we let π be chosen uniformly among all functions $\pi(N,T,X)$ that, for each format $N \in \mathcal{N}$ and each tweak $T \in \mathcal{T}$, $\pi(N,T,\cdot)$ is a permutation on $\mathcal{X}_N$. We let π^{-1} be this function's inverse. Extend both maps to return $\perp$ on strings outside of $\mathcal{N}\times\mathcal{T}\times\mathcal{X}$. Now provide the adversary with the pair oracles π, π^{-1}.

The definition just given generalizes prior (strong-PRP) notions for the security of blockciphers, wide-blocksize blockciphers, VIL (variable-input length) enciphering scheme, and schemes that encrypt on "unusual" spaces of strings. The definition may seem a bit abstract, but it may help to bring out the essential commonality of all the problems just named.