

Format Preserving Encryption

Terence Spies
Voltage Security, Inc.
(terence@voltage.com)

ABSTRACT

Encrypting Personally Identifiable Information (PII) in large databases has historically been difficult, because encrypting information typically implies expanding data and changing its format. Previous attempts to encrypt PII data like credit card numbers and Social Security Numbers without changing their format have used questionable cryptographic constructions. We examine the security model for this problem, extend a construction by Black and Rogaway, and propose practical constructions for encrypting credit card numbers and Social Security Numbers.

1. INTRODUCTION

Increased regulation, such as California SB1386 and the PCI Data Security Standard [9], and consumer awareness of privacy issues have motivated many businesses to investigate methods to encrypt Personally Identifiable Information (PII) and minimize the repercussions of losing data. One of the barriers to the adoption of effective encryption methods is the cost of modifying databases and applications to accommodate encrypted information. These costs are associated with two changes needed to accommodate classically encrypted data. First, privacy-critical information like credit card numbers or Social Security Numbers are often used as keys or indices in databases, so randomization of these fields by encrypting data may require significant schema changes. Second, applications may be written expecting data in a specific format; encryption will typically expand data and require a format change.

In addition to encryption of production databases, it may be desirable to produce test databases that resemble production databases. Often, the development and deployment process for large business applications involves testing an application against a production database before deploying it “live.” However, during this testing, PII that might be acceptably protected in a deployment environment now needs to be moved out into a probably less protected test environment. Encrypting the data would protect the data, but destroy the ability to test against that data, since the format would be changed, and, if the encryption was randomized, destroy referential integrity in the database.

It would be desirable to have an acceptably secure encryption process that would render a database of identifying information useless to an attacker without a key, but that would produce ciphertext in the same format as the plaintext. Certain compromises from the traditional semantic security model for

ciphers must be taken, because of the desired size of the ciphertext, and the fact that we might want to encrypt deterministically to preserve referential integrity in a database. However, we still want formal statements of the security of these methods. We term algorithms with this property Format Preserving Encryption (FPE) algorithms.

Historically, the best test for a cryptographic algorithm is the test of time. For this reason, we would like to have an FPE algorithm that is not new, or at least can be reduced (within some bound) to some existing, known cipher. The algorithms proposed in this paper have security proofs in the cryptographic literature, and are based on constructions that go back at least to 1986.

This paper discusses the history of the FPE problem, various solutions that have been proposed, the security model for FPE in general, and describes three methods that cover various FPE sub-cases.

We expand the range of encryptable plaintexts from the Black-Rogaway paper, examine the limits of these ciphers in the light of contributions from Patarin, and present practical constructions for encryption of two important data types, Social Security Numbers and credit card numbers. We also present performance figures for implementations of these ciphers.

2. HISTORY

The FPE problem goes back at least to 1997, when Smith and Brightwell [5] argued that an FPE algorithm would help secure databases and data warehouses:

Ciphertext (data in encrypted form) bears roughly the same resemblance to plaintext (data in its original form) as a hamburger does to a T-bone steak. A social security number, encrypted using the DES encryption algorithm, not only does not resemble a social security number but will likely not contain any numbers at all. A database field which was defined to hold a nine-character social security number (eleven, if you want to include the hyphens) would not be able to store the DES-encrypted version of the data. A Visual Basic program would not read it. A graphical interface would not display it. There would be nothing that you could do with the encrypted social security number unless you had made extensive provisions for changes in data

format throughout your application and physical database design.

They defined the problem and proposed an encryption method based on an existing block cipher (DES) in Cipher-Feedback Mode (CFB), which produces a series of data and key dependent offsets that can be used to encrypt each character of a formatted string by adding the offset. This method has the weakness that, using the same key, initial characters in the string will be encrypted identically, so, for example, credit card numbers that share a bank id can be identified. The authors propose a “rippling” scheme to induce dependencies among the characters in the string, but no argument for the security of this scheme is given.

The first strong cryptographic attempt to solve a related problem was Black and Rogaway in 2002 [1]. In this paper, they attempt to find solutions to a closely related problem: how to find ciphers that will encrypt elements of a set of some size m into a set of the same size. Since all formatted strings fit into a finite set of some size, solutions to this problem can be used as the core of FPE algorithms. They propose three methods, which we examine below. Two of the methods, the Prefix method (which works for small sets) and the Cycle-Walking Cipher (which works for sets just smaller than the size of the block cipher) have strong security bounds.

The third method encrypts a much wider variety of data, using the Feistel construction first formally examined by Luby and Rackoff. [6]. The Feistel construction has the desirable property that ciphers built from it can be proven to reduce to some other cipher used as a round function, as long as the attacker only has access to some limited number of ciphertext/plaintext pairs. At the time of the Black-Rogaway paper, there was no proof of bounds strong enough to allow this construction to be used to encrypt data in the range of credit card numbers, which left an important use case unsolved.

Because the Luby-Rackoff construction is so important to the development of cryptographic algorithms, there has been a large body of work following that of Luby and Rackoff which attempts to give tighter security bounds. The basic Luby-Rackoff security problem is typically formulated as an attacker attempting to distinguish an instance of a Luby-Rackoff cipher from a random permutation, given the assumption that the Luby-Rackoff cipher is built from a pseudorandom function. This is important, as it is the weakest attack that can be mounted against a cipher. Any other attack would distinguish the cipher from a random permutation, so the strength of the cipher against this attack gives a lower bound on the security of the algorithm.

In the Luby-Rackoff model, security is measured by the number of ciphertext/plaintext pairs needed by an attacker of unbounded computational power to have a reasonable chance of distinguishing between the cipher and a random permutation. The bounds shown in these proofs establish a formula for the number of pairs based on the size of the plaintext and the number of rounds used.

At the time of the original Luby-Rackoff paper, bounds were only known for three and four rounds, and the number of pairs was essentially the square root of half the size of the plaintext space. While this bound was an important theoretical result, it would give a security bound of 16 bits for triple-DES (since it was

unknown if extra rounds would improve bounds.) Clearly there were better bounds to be found.

In the credit card case, this result would give a bound of 10,000 ciphertext/plaintext pairs needed, which is clearly too small for comfort. This bound was especially worrisome for the Black and Rogaway construction because running at three rounds, 10,000 ciphertext/plaintext pairs would give an attack that would leak information about other ciphertexts. In the paper, the authors speculate that more rounds would give more security, but left the problem to future work.

In 2004, Patarin [7] showed that if the Luby-Rackoff construction is run with a sufficient number of rounds (6, 7, or 8, depending on the security model), the number of ciphertext/plaintext rounds needed by an attacker with unbounded computational resources to establish a difference between a Luby-Rackoff cipher and a random permutation approaches the theoretical maximum, which is the square root of the size of the entire plaintext. In the credit card number case, the bound would now be 16 million ciphertext/plaintext pairs, for a theoretical attack by a computationally unbounded opponent (i.e., an attacker with infinite computing resources). Patarin also gives the best known realistic attack, which is still somewhat theoretic in that it requires more ciphertext/plaintext pairs than can be produced by a single key, and also at least 2^{64} computational steps.

In the light of the Patarin results, it seems worthwhile to re-examine the Luby-Rackoff construction for the FPE problem, as it gives an efficient, AES-based encryption algorithm with a strong proof of security within acceptable bounds.

3. The FPE Model

The archetypical FPE use case is the encryption of an internal database so that a limited number of applications and users (perhaps none) can recover the original values from the FPE encrypted values. In this setting, any security mechanism (FPE, traditional encryption, or other access-control mechanism) needs to be strong with the following limitations:

Constraint 1: The attacker knows the format and type of data in the database. We should assume that an attacker knows that they are looking for credit card numbers or social security numbers.

Constraint 2: The plaintext size will be relatively small, compared to typical cryptographic set sizes. A block cipher like AES operates on 128 bit blocks. Social security numbers are approximately 28 bits, and credit card numbers are approximately 58 bits long.

This means that any mechanism (FPE or other) must protect against attacker access to the encrypt/decrypt function; otherwise, an attacker can simply start asking for decryptions of arbitrary data and potentially get a dangerous amount of PII data.

In the case of FPE, two other important limitations are put on the algorithm:

Constraint 3: Data cannot be expanded. When an FPE algorithm encrypts an N-digit number, it must output an N-digit number.

Constraint 4: Data must be encrypted deterministically. Data is typically being encrypted in a database, and it is highly desirable to preserve the ability to use a column as a key or index, which requires that multiple instances of a given data item encrypt to an identical data item (when using the same key).

Note that Constraint 4 also can be seen as a consequence of Constraint 3. Since no additional data can be stored in the data field (otherwise expansion would result), there is no space for randomness needed for initialization vectors or other randomizers. Location within the database, or other fields could be used as randomness, but then the ability to decrypt is conditioned on the availability of that data. If data is decrypted within an application external to the database, this data may not be available.

Within these constraints, we need an algorithm that will encrypt data by permuting strings in a given format to different strings in the same format, will not leak information to an attacker that has access to a large number of ciphertexts, and will survive the exposure of a reasonable number of plaintext/ciphertext pairs. The algorithms that meet this standard are built to survive only in the FPE setting, and are not generally recommended for building into communication applications, or other settings where the attacker can be assumed to have access to the encrypt/decrypt function or access to unlimited ciphertext/plaintext pairs.

The compensating factor that makes it possible to construct a secure cipher within these constraints is that, while the attacker knows that they are looking for a specific kind of data (a Social Security Number, for instance), they must find one particular number that matches the other personal data in the database. This means that we can essentially permute the set of numbers, and measure the success of the attacker by how successful they can be at guessing a specific number given a permuted number and some other defined data.

The practical security goal here is to turn a currently passive attack (stealing database data through an application, log file, or backup tape) into one that requires active subversion of a trusted encryption/decryption function, and that will keep data safe even if a substantial number of encryptions and decryptions escape. In section 4, we examine each FPE method and detail the security bounds that are shown for each method.

3.1 Test Data Model

The model of encrypting data for testing in a lab or for development access is slightly different than the full FPE model. In this case, the encryption process may not need to be arbitrarily reversible, but instead is used to permute the data, then the key is thrown away. In this case, any possible encryption/decryption oracle is destroyed, and the best that an attacker can do, short of finding a way to recorelate the data with plaintext by breaking into internal databases, is to examine the ciphertext.

The FPE methods below are very strong in this model. Currently, many applications use hash functions truncated to output formatted data. This preserves the referential integrity requirement, but creates the possibility of internal collisions (where two data items encrypt to the same value), possibly leading

to difficult to detect bugs. Since all the FPE algorithms are permutations, they guarantee that data items will not collide when encrypted. This is an especially desirable feature for small data items like Social Security Numbers where the probability of a collision is non-negligible.

4. FPE Methods

This section details three practical FPE methods, all derived from the three methods in the Black and Rogaway paper [1]. We give performance measurements for these methods, and offer a modification and security bound improvement for the Feistel construction based on Patarin’s results. The next section details the practical FPE schemes for the specific credit card number and Social Security Number cases.

Each of the three methods handles a different input set size, either for security or performance reasons. The following table summarizes the practical and secure bounds for each of the methods, in terms of overall set size and also in terms of the number of decimal digits of a formatted string. Set sizes and formats that are not covered by this table may be covered by application of a compound method specified in section 5. The values in the table are approximate, and actual bounds are dependent on the security level required by the specific application. The table attempts to be conservative in terms of security model. The description of the individual algorithms specifies how set size interacts with security and performance.

| Method | Set Sizes (bits) | Decimal Digits |
|---------------|------------------|----------------|
| Prefix | 1-20 | 1-6 |
| Cycle-walking | 50-63 | 16-19 |
| Feistel-Cycle | 40-240 | 12-80 |

4.1 The Prefix method

The Prefix method is very simple, but only works on small datasets. The method works by essentially “writing down” a random permutation in memory, and using that permutation to encrypt data.

4.1.1 Prefix Method Description:

To encrypt data with the Prefix method, we first construct a table which stores a permutation over the full plaintext set, then simply look up the ciphertext value using the plaintext. This means that encryptions and decryptions are very fast. Table set up is more expensive, but acceptable for small set sizes.

To set up the table, we pick an underlying cipher, typically AES or 3DES. We then encrypt the values 0...N-1, where N is the size of the input set. We record the input number and the encrypted value, then sort by the encrypted value. For example, say that we want to encrypt data in the set of a single digit (0-9). We encrypt

ten values with AES, then sort, creating a table that looks like the following:

| Digit | AES encryption of digit, sorted |
|-------|----------------------------------|
| 7 | 12d76795b5e818b38be9813260ab0c5f |
| 3 | 203c3c515ae6101c4858fe07ecb78ec0 |
| 1 | 25dabcc8862842c228a2d7ac5058b780 |
| 2 | 416f3563827406dab2ef1246393fed32 |
| 6 | 45bd0fb7d45bd276d499ad4b8cd52e55 |
| 9 | 4c9c6ecbdf1ab60ef0b31d753fa594c6 |
| 4 | 4e20e4d8c4195df66a3cc9fe60f5b98f |
| 0 | 5c3f46a3cf7a8da378eb95f546a20ab2 |
| 8 | 98bc8588c55900703ef11fa80447e32c |
| 5 | d99851ff58a9bf03d717ff6601639795 |

We can then throw away the encrypted values, leaving a permutation over the set size. In this example, the encryption of 2 would be 1. Decryption can be performed by simply finding the index of the encrypted value. For example, a ciphertext of 7 would correspond to the plaintext of 0. We can build these tables for any size set up to just under the size of the underlying block cipher. (If the input set was the same size as the block cipher, we could just directly use the block cipher.)

4.1.2 Prefix Cipher Tweaking

Because rekeying the Prefix Cipher requires rebuilding the table, it may be desirable in certain circumstances to be able to apply a “tweak” to the cipher, which will alter the encryption function without requiring a rekey. The security requirements for tweaks, and some methods for building tweaked ciphers, are given in [8].

The basic method for applying a tweak T to a cipher E (of size N) that is shown secure in [8] is to perform the following operation, which encrypts a plaintext P using a key K.

$$C = E((E(P, K) + T) \bmod N, K)$$

Normally, this construction is seen to be inefficient, because it requires two invocations of the underlying cipher. In the case of the Prefix Cipher, the cost of encryption is only a lookup, so the construction is very fast, requiring only two memory lookups and an addition operation. We will use this tweaking construction to construct a Social Security Number encryption operation later.

4.1.3 Prefix Method Security

Black and Rogaway [1] give a straightforward proof that breaking this construction reduces to breaking the underlying cipher. They conclude that finding a difference between this kind of Prefix permutation and a random permutation would require finding a difference between the underlying cipher and a random permutation.

4.1.4 Prefix Method Performance

The Prefix method can be thought of as having a very slow key setup time (building the permutation in memory) and a very fast encrypt and decrypt time (doing the single lookup in the table.) The performance question comes down to the time required to build the table, and the memory required to hold the table.

One useful optimization is to initially encrypt the set elements, but only record the initial 32 bits in the table. The table sort can then be done on these 32 bit elements, and if two elements are equivalent, re-encrypt and compare the entire encrypted value. This optimization makes the intermediate table smaller, and lowers the amount of copying required during the sort.

The following table shows the performance of the prefix cipher, using AES-256 as the base cipher, at various set sizes on a 2.34 Ghz Pentium IV with 1 GB memory running Microsoft Windows XP Professional. Since encryption and decryption is trivial (requires a single memory lookup), we only show the time required to build the key table.

The table shows the size of the plaintext in bits, the number of decimal digits that can be encoded by the cipher, and the time to build the table in milliseconds.

| Bits | Decimal Digits | Table Build Time |
|------|----------------|------------------|
| 10 | 3 | 0.476 |
| 14 | 4 | 5.5 |
| 17 | 5 | 62 |
| 20 | 6 | 760 |

4.2 Cycle-walking Method

The Cycle-walking construction, like the Prefix method, is quite simple, but works on a limited class of sets. The Cycle-walking construction works by encrypting the plaintext with an existing block cipher (AES or 3DES) repeatedly until the cipher output falls in the acceptable output range. To encrypt a ciphertext C in the range 0..N, using some base cipher E, we perform the following operation:

$$T = E(C)$$

$$\text{while}(T > N) T = E(T)$$

The larger the difference is between the size of the output of the base cipher and the size of the desired output set, the longer this operation will take to terminate. For this construction to be practical, N needs to be some small number of bits shorter than the output of the cipher.

4.2.1 Cycle-walking Method Security

[1] contains the proof that the Cycle-walking cipher does not degrade the security of the underlying cipher. It also contains the proof that the method will terminate in all cases, though it is possible that there are rare cases where a large number of cycles will be required.

4.2.2 Cycle-walking Method Performance

The following table shows Cycle-Walking performance using 3DES for a variety of set sizes close to 64 bits. The timings were taken on a 2.34 Ghz Pentium IV with 1GB memory running Windows XP Professional. The table shows the bit size of the input, the number of encoded decimal digits, and the number of encryption operations per second. The timings were taken by generating a random decimal value in the range shown, packing it into binary form, and encrypting using 3DES in the Cycle-walking mode.

| Bits | Digits | Enc/Second |
|------|--------|------------|
| 54 | 16 | 500 |
| 57 | 17 | 5000 |
| 60 | 18 | 43000 |
| 64 | 19 | 150000 |

As can be seen from the table, the Cycle-walking method yields a usable cipher for values in the size range of standard credit card numbers, which range from 16-19 digits long.

4.3 Feistel + Cycle Method

This method is more complex than the Prefix or Cycle-walking constructions, but allows for encryption over a wide variety of set sizes with good performance. We adapt the Black-Rogaway construction to use a standard “textbook” Feistel network, which lets us get close to desired set size, and use the Cycle-walking technique to get the output into the output set. Black and Rogaway use a more modified Feistel at three rounds. We use the unmodified Luby-Rackoff Feistel construction with two different round function constructions with a minimum of eight rounds.

4.3.1 Feistel + Cycle Description:

The Feistel + Cycle construction has two main parts. First, the Feistel network that is closest to the size of the plaintext is constructed. This is then used to encrypt the data. The cycle-walking technique is then used to insure that the ciphertext is in the appropriate range. Because the underlying cipher is built to be very close to the desired size of the ciphertext (within less than 2 bits) so very few cycles are required to get the ciphertext into the required size.

The Feistel construction for a given bit length n consists of multiple rounds of the following construction using some Pseudo-Random Function (PRF) f , which takes $n/2$ bits and outputs $n/2$ bits. It operates on the plaintext, which is divided into a left and right half, called L and R . The round function produces a new L and R , which is either output or fed back into the next round. The round function performs the following operation to calculate the new L and R values:

$$R' = L \text{ XOR } f(R)$$

$$L' = R$$

To construct a Feistel-Cycle cipher $FC_{N, F, x}$ to encrypt a value P from $0 \dots N$, using a base pseudo-random function F and x rounds, we do the following:

Define the Feistel network:

1. Find the smallest W s.t. $2^{2W} > \log_2(N)$. This is the width of the Feistel network we will use.
2. Define $F'(x) = \text{trunc}(F(x), w)$
3. $\text{Round}(R, L) = L \text{ XOR } F'(R)$
4. $\text{Feistel}_{N, F, x}(P)$ is then computed by:
 1. Find R, L s.t. $P = R * 2^W + L$
 2. Repeat x times: { $T = \text{Round}(R, L), L = R, R = T$ }
 3. Output $R * 2^W + L$

$FC_{N, F, x}(P)$ is then computed as:

1. $C = \text{Feistel}_{N, F, x}(P)$
2. while($C > N$) { $C = \text{Feistel}_{N, F, x}(C)$ }

To fully specify a Feistel-Cycle cipher for a specific instance, we now need to just decide two things. First, how many rounds to run. The security bounds that we need are comfortably achieved with 8 rounds for most datasets. Second, what do we use for F ? The security bounds reduce the problem of distinguishing the whole cipher from a random permutation to distinguishing F from a random function. Hence, we need a strong random function of $n/2$ bits for a wide variety of n .

The problem of building PRFs has been widely studied, and there are two methods that give acceptable PRFs for use here. We can't simply use AES, since it is a pseudo-random permutation (PRP), not a PRF, and it is too wide. We need a random function that is exactly the same width as the left element so that L can be XOR'ed with the output. If n is small enough, simply taking the first $n/2$ bits of AES actually gives a strong PRF. If n is larger, we can take the output of two instances of AES XORed together. [2] details the proof that these constructions yield strong PRFs.

There is a long tradition of using truncated PRPs as strong PRFs. For example, the IETF RFC on a PRF for Kerberos [10] uses this construction.

4.3.2 Feistel + Cycle Security

We can measure security of this cipher in three different ways. First is the resistance to attack by the best possible computationally unbounded attacker. Second is the resistance to attack by brute force. Third is the best known attack that distinguishes Feistel networks from random permutations.

Attacker 1: The Optimal Unbounded Attacker

For any Feistel network, we can measure the amount of entropy in the function, estimate how much entropy a plaintext/ciphertext pair gives away, and then derive how many plaintext/ciphertext pairs are required for any possible attacker. This is the strongest possible security model. The result of this analysis [5] is that the best possible unbounded attacker will require $\sqrt{2n}$ ciphertext/plaintext pairs, and Patarin [7] shows that 6 rounds of a Feistel network is sufficient to get to this upper bound. This means that any theoretic adversary attacking a 6 or more round Feistel network operating on 56 bits, which is the set required for credit card numbers, will require a minimum of 16 million plaintext/ciphertext pairs. The same cipher operating over the Social Security Number space, which is about 28 bits, will require a minimum of 16,384 pairs. Since this number is small, we consider hybrid techniques in the next section to handle this case. Note that for Feistel networks of over 6 rounds, these bounds are for a hypothesized optimal attacker, and no practical attack comes close to these bounds.

The unbounded attack model is most interesting in that it provides a strong lower bound. We know, through the proof, that no attack can be constructed that is more efficient. The currently known attacks against Feistel networks are far worse than these bounds.

Attacker 2: The Brute Force Attacker

The closest match to the unbounded attacker is an attacker that attempts to enumerate all possible round functions that are consistent with the plaintext/ciphertext pairs it has seen. This attacker would require $2^{\log_2(r)+\sqrt{r}}$ pairs, and would need to perform a very large number of computations. We can see the difficulty of the attack by examining the computations required to attack a small six bit wide Feistel network using eight rounds.

A six bit wide Feistel network will use three bit PRFs. There are 8^8 or 2^{24} possible functions of this type. With eight rounds, the brute force attacker will need to keep track of 2^{192} possible Feistel networks, and eliminate them as they become inconsistent with the ciphertext/plaintext pairs. It is possible that there is some optimization that would make this attack practical, but it appears that even a trivially small Feistel requires an currently impractical amount of computational power.

Attacker 3: The Best Known Attacker

Patarin showed the best known realizable attack against Feistel networks [7]. These attacks are realizable in principle, but require more ciphertext/plaintext pairs than can be produced by a single key, and require significant computational resources. The Patarin attacks are also pure distinguishing attacks that produce a single

bit, which is the guess as to if the plaintext/ciphertext pairs are produced by a random permutation or by a Feistel network.

To distinguish an $2n$ bit wide Feistel network of 8 or more rounds from a random permutation, the Patarin attacks require $2^{(r-6)n}$ plaintext/ciphertext pairs, and $2^{(r-4)n}$ computations. So, for the credit card case with 8 rounds, it would require 2^{56} plaintext/ciphertext pairs, and 2^{112} computational steps. This means the attacker would need every single possible plaintext/ciphertext pair, and would need to perform more computations than would be needed to break an 80 bit key. Increasing the round count beyond 8 makes these attacks even more difficult.

4.3.3 Feistel + Cycle Performance

The Feistel + Cycle construction’s performance is dependent upon the number of rounds used and the specific PRF that is used in the round function. For any plaintext that is smaller than the block size of the PRF, the performance is essentially $i*r*cost(PRf)$, where r is the round count and i is the average number of times the cipher cycles to get an acceptable output. For the truncated AES PRF, performance is very close to $i*r*cost(AES) + cost(AES$ key setup). For the summing AES PRF, performance is close to $2*i*r*cost(AES) + 2*cost(AES$ key setup).

Measurement of the Feistel+Cycle method was done on a 2.34 Ghz Pentium IV with 1GB of memory running Windows XP Professional, using AES as the base for the internal PRF. The cipher was measured for a range of round counts ranging from 8 to 128, using both the truncation and additive PRF constructions.

| Rounds | PRF | Enc/Second |
|--------|-------|------------|
| 128 | Trunc | 7000 |
| 32 | Trunc | 10500 |
| 8 | Trunc | 18000 |
| 8 | Add | 8100 |

The range of rounds was measured to examine the viability of running large numbers of rounds to defeat any possibility of the worst-case brute force attack working for small inputs.

5. Specific and Compound Methods

It is possible to use these methods in combination to yield FPE techniques that will allow for granting access to only selected digits of a number (for example, limiting a customer service application to the last four digits of a Social Security Number, while granting another application full access.) These methods can be used to fill the gap between where the Prefix method is not practical and where the Feistel + Cycle construction may not be considered fully secure.

5.1 Compound for Social Security Numbers

The Social Security Number case is interesting because this is a common type of PII, and it is small enough that it does not fall into the obvious bounds for the known three provable FPE methods. This case is valuable enough that we supply a method that gives reasonable security bounds for the entire number, and also gives the valuable property of being able to either decrypt the entire number or just the last four digits, which are often used in customer service applications for confirmation purposes.

The construction works in the following way. We use as the basis for the construction two ciphers:

$\text{Pre5}(P, T, K)$: A Prefix cipher encrypting all five digit decimal values using key K , and an arbitrary tweak value T

$\text{FC9}(P, K)$: A Feistel-cycle cipher encrypting all nine digit decimal values using key K .

$H(P)$: A hash function from four digit decimal values to an arbitrary bit string.

Construct two keys, $K1$ and $K2$. Divide the Social Security Number into the initial five digits L and the last four digits R . The encrypted number is then computed by:

$$E = \text{FC9}(\text{Pre5}(L, H(R), K1), K2)$$

Now an entity can be given access to just the last four digits by giving them just $K2$. They can use $K2$ to undo the Feistel encryption only. An entity with access to $K1$ and $K2$ can decrypt the entire number by first undoing the Feistel encryption with $K2$, then undoing the Prefix encryption with $K1$ and the decrypted last four digits.

The initial encryption with the Prefix Cipher insures that a theoretical attacker that uses the smaller bounds of the Feistel cipher run on a small plaintext space will only recover (in the worst case) the last four digits of the number and encrypted versions of the first five digits.

5.2 Methods for Credit Card Numbers

Credit card numbers are an interesting case for FPE, as they carry a check digit at the end of the number that is calculated with the Luhn algorithm, which is a modified sum of the digits. This digit does not need to be encrypted, since it can simply be recalculated when the remaining digits are decrypted.

There are three methods to encrypt with this checksum:

Method 1 – Transparent Encryption

The goal here is to encrypt a credit card number so that an encrypted number is indistinguishable from a plaintext number. It maintains a valid checksum. This is useful because untrusted applications that require a valid checksum will continue to function unmodified. The risk here is that applications, since they

cannot distinguish, may accidentally use an encrypted number as a plaintext number.

To do this, encrypt all the digits except the last using an FPE algorithm. After encryption, add a checksum digit with a valid checksum of the encrypted digits. This has the advantage that, to any application that tests the checksum, encrypted CCNs look identical to valid CCNs. To decrypt, FPE decrypt all the digits except for the last, then regenerate the checksum digits on the plaintext digits.

Method 2 – Checksum marking

In some situations, it is valuable to encrypt so that the format is preserved, but that there is some indication that a number is encrypted. By inducing a systematic offset in the checksum, a program can reliably tell that a number is encrypted, even though the format remains the same.

Encrypt the number but exclude the final digit, and replace the checksum digit with the checksum value plus 1. This insures that the checksum digit is invalid, and gives a reliable method to distinguish an encrypted CCN from a plaintext CCN.

Method 3 – Checksum encoding

The constant checksum offset can also be used to encode a small amount of extra information in the checksum digit. This can be used to select from a set of keys, adding diversity to an existing key management scheme.

Generate nine keys, encrypt all the digits (but the final) with one of these keys, and replace the checksum digits with the checksum plus a key identifying value from 1 to 9. This insures that the checksum is invalid, and gives a method to change keys and record what key is used to encrypt a specific value without adding any data to the database. This gives an additional key selection value, but must be supplemented with other sources to identify the key, more than 9 keys will be used.

6. REFERENCES

- [1] J. Black and P. Rogaway. *Ciphers with Arbitrary Finite Domains*. RSA Data Security Conference, Cryptographer's Track (RSA CT '02), Lecture Notes in Computer Science, vol. 2271, pp. 114-130, Springer, 2002.
- [2] M. Bellare and R. Impagliazzo. *A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP / PRF conversion*. Manuscript, 1999.
- [3] S. Lucks. *The Sum of PRPs Is a Secure PRF*. Lecture Notes in Computer Science", volume 1807, 2000
- [4] H. E. Smith and M. Brightwell. *Using Datatype-Preserving Encryption to Enhance Data Warehouse Security*. NIST 20th National Information Systems Security Conference, pp.141, 1997.
- [5] M. Naor and O. Reingold, *On the construction of pseudo-random permutations: Luby-Rackoff revisited*, J. of Cryptology, vol 12, 1999, pp. 29-66. Extended abstract in: Proc. 29th Ann. ACM Symp. on Theory of Computing, 1997, pp. 189-199

- [6] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.
- [7] J. Patarin. Security of Random Feistel Schemes With 5 or More Rounds. In proceedings of Crypto 2004, The 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004.
- [8] M. Liskov, R. Rivest, D. Wagner. Tweakable Block Ciphers. Proceedings of Crypto 2002, The 22nd Annual International Cryptology Conference, Santa Barbara, CA, Aug 15-19, 2002.
- [9] PCI Security Standards Council, Payment Card Industry (PCI) Data Security Standard, <https://www.pcisecuritystandards.org/tech/index.htm>, September 2006
- [10] N. Williams, Internet Engineering Task Force RFC 4402 (<http://www.rfc-archive.org/getrfc.php?rfc=4402>)