FNR : Arbitrary length small domain block cipher proposal

Sashank Dara, Scott Fluhrer
Cisco Systems Inc
email: {sadara,sfluhrer}@cisco.com

June 7, 2014

Abstract

We propose a practical flexible (or arbitrary) length small domain block cipher. FNR can cipher small domain data formats like IPv4, Port numbers, MAC Addresses, IPv6 address, any random short strings and numbers while preserving their input length.

In addition to the classic Feistel networks, Naor and Reingold propose usage of pair-wise independent permutation (PWIP) functions in first and last rounds of LR constructions to provide additional randomness and security. But their PWIP functions are based on Galois Fields. Representing GF(2^n) for different input lengths would be complicated for implementation. For this reason, the PWIP functions we propose are based on random N X N Invertible matrices.

In this paper we propose the specification of FNR mode of encryption. Its properties, limitations. We provide possible example applications of this block cipher for preserving formats of input types like IPv4 addresses, Credit card numbers. We provide reference implementation's experimental results and performance numbers in different setups. FNR should be used only when deterministic encryption is needed. It does not provide semantic security.

FNR denotes **F**lexible **N**aor and **R**eingold

Contents

List of Algorithms

List of Tables

List of Figures

1 Introduction

There is a compelling need for privacy of sensitive fields before data is shared with any cloud provider, semi-trusted vendors, partners etc. For example network telemetry data, transaction logs etc. are often required to be shared for benefiting from variety of Software as Service applications. Such sensitive data fields are of prescribed and arbitrary lengths. They are often part of well defined data formats like NetFlow, IPFIX etc. For example Port(16), IPv4(32), MAC (48) , IPv6 (128) etc.

While designing privacy for sensitive fields, it may be desirable to preserve the length of the inputs, in order to avoid any re-engineering of packet formats or database columns of existing systems. Traditional AES-128/256 encryption would encrypt plain text (of any smaller lengths) to result in a cipher text (a random string that is 128/256 bits). Expansion of cipher text length may be undesirable for said reasons. Also AES is fixed length cipher, input domains that are of smaller size need to be padded in order to perform encryption.

Small domain block ciphers are useful tool in designing privacy of sensitive data fields of smaller length (<128 bits). In addition to the classic Feistel networks, Naor and Reingold propose usage of *Pair-Wise Independent Permutation (PWIP)* functions in first and last rounds of LR constructions to provide additional randomness and security. We propose usage of invertible matrices to provide a neat and generic way to achieve pair-wise independence for any arbitrary length.

2 Design Goals

We intended to design a small domain block cipher that has below goals

Arbitrary length Input domains of variable lengths need to be supported. For example, a system that consists of NetFlow would have different domains like IPv4, Port, IPv6 etc. all are of different lengths.

Key Length A system might contain multiple domains of various lengths. If the key size is dependent on the input length, then managing key sizes of various lengths would be cumbersome. For this reasons key sizes should not depend on input length.

Secure building blocks The building blocks used for such design should be considered secure. For example techniques based on Feistel Networks of Luby Rackoff constructions, Substitution and Permutation Networks of AES are considered good blue prints for block cipher designs.

Leveraging hardware support Modern processors support AES at assembly level (say AES-NI of Intel and AMD). Such provisions should be leveraged for faster software implementations.

Supporting software platforms Due to the advances in cloud computing technology, privacy of smaller data fields may need to be implemented in variety of software platforms. For example browsers that run Java, JavaScript, thin clients based on REST interfaces etc. apart from ubiquitous C, CPP implementations. For this reason, variety of software platforms should be easily supportable.

Intellectual Property Free Either the building blocks that are used in the block cipher design or the block cipher itself should be free from any intellectual property rights.

3 Prior Art

Luby Rack off Constructions are considered seminal work in formalizing secure block cipher design [7]. They have been subjected to rigorous theoretical analysis and well laid security bounds are established.

Further variable input length block ciphers have been proposed in [3],[11]. These constructions require multiple application of original block cipher in order to make them arbitrary length block ciphers. This makes them computationally intensive and inefficient. Design of ciphers for arbitrary domains were also proposed in [4]. The *Prefix Cipher, Cycle Walking* mentioned in their work would be very expensive in practice. The *Generalized Feistel Network* approach mentioned in their work uses DES as PRF. RC5 has features for arbitrary domain lengths but it is patented. Elastic block cipher design has been proposed in [5] but they are not subjected to rigorous independent analysis.

Feistel Networks also form the foundational blocks for Format Preserving Encryption(FPE). FPE has been studied rigorously theoretically [2]. A white paper is available from Voltage Inc. [13] which has good overview. A very good synopsis is given by Rogaway [12]. Few modes of FPE have been recently proposed for NIST standardization [1].

Usage of *pair-wise* Independent Permutations in LR constructions was first proposed by Naor and Reingold [8]. While their techniques are based on performing operations in GF(2^n) we propose to operate on invertible matrices. This makes our scheme light weight and flexible enough to perform on any arbitrary input fields.

4 Definitions

4.1 Secrets

There are various secret keys used in FNR. Some of them are user supplied, some are internally generated for the usage of the algorithms.

Key A 128 bit long secret key, K, is needed. This is used internally by *Pseudo Random Function (PRF)* i.e AES algorithm. This is generated by a good entropy source or derived by using good key derivative function from a user supplied password.

Tweak A tweak, *T*, is like *salt* or *IV*. It should be nearly *n/2* bits length, where *n* is number of input bits. In practice, a small character string is supplied by the user, as tweak, which is then encoded as fixed length binary string appropriately.

A, B are two matrices. A is invertible binary matrix of *N X N* dimension. B is binary matrix of *1 X N* dimension. Where N denotes number of bits in the input. Both A,B should be uniformly distributed.

$$A_{n,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} a_{x,y} \in \{0,1\} \quad \forall x,y \in \{1 \cdots n\} \tag{1}$$

$$B_{1,n} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \end{pmatrix} b_{x,y} \in \{0,1\} \quad \forall x,y \in \{1 \cdots n\} \tag{2}$$

4.2 Inputs and Outputs

FNR, like any other block cipher, has two operations encryption and decryption. There are three inputs and an output for both of these operations. Typically the size of Plain text P is *n bits* such that n is in between 32 to 128 bits. Integer, Character strings are encoded as bit vectors before encryption.

4.2.1 Encryption

The inputs are Plain text *P*, that needs to be encrypted, a secret key *K*, a tweak *T*. The output of an encryption function is n bits of cipher text *C*.

4.2.2 Decryption

The inputs are cipher text *C*, secret key *K*, tweak *T*. The output is plain text *P*.

4.3 Notation

Our notation follows that of *Recommendation for Block Cipher Modes of Operations* [6]. A block cipher encryption that takes inputs key K , tweak T, numerical constants A,B and a numerical plain text X is denoted by FNR.Encrypt(K,T,X). The function returns a numerical string Y as output that is of same length as X. The decryption function is similarly denoted as FNR.Decrypt(K,T,Y) and it returns the plain text X .

Along with the Feistel constructions we use special functions called *Pair-wise Independent Permutations (PwIP)* to provide additional security. The function *PwIP* takes input a random numerical string X of n bits length. Along with predefined binary matrices A,B to output a uniformly distributed permutation of n bits H. So it can be represented as

FNR.PwIP(A,B,X) and output of which is H. An inverse function FNR.PwIP $^{-1}$(A,B,H) reverses such permutation and outputs X.

4.4 Preliminaries

4.4.1 Pair-wise Independent Permutations (PwIP)

Pair-wise Independent Permutations (PwIP) are combinatorial constructions to achieve a uniformly distributed permutation of given input. A pairwise independent permutation is a permutation that has the property that, assuming a random (uniformly distributed key), that any two distinct inputs A, B, and any two distinct outputs X, Y, the probability that X=PWIP(A) and Y=PWIP(B) is uniform, that is, is $1/((2^n) * (2^{n-1})$ independent of A, B, X, Y.

The below function actually achieves stronger *Three-way Independent Permutation* criteria, for three distinct inputs A, B, C, and three distinct outputs X, Y, Z, the probability that X=PWIP(A), Y=PWIP(B), Z=PWIP(C) is independent of A, B, C, X, Y, Z

Let the input X be a binary vector of n bits length, considered as 1 X N matrix, then $f_{A,B}(X)$ as defined below gives a uniformly distributed permutation. The matrix operations $*, \oplus, \div$ are performed in GF(2). Also instead of bit-wise XOR operation, modular addition could be used too.

$$X_{1,n} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \end{pmatrix} x_{i,j} \in \{0,1\} \quad \forall i,j \in \{1 \cdots n\} \tag{3}$$

$$f_{A,B}(X) = (X \times A) \oplus B \tag{4}$$

The algorithm for the same is defined in Algorithm.1.

Algorithm 1: FNR PwIP Algorithm

```
Inputs : Matrix A, Matrix B, bitvector input
Output: bitvector output
Function PwIP( A, B, input, n) is
    /* bitvectors input and outputs are 1XN matrices */
    if ( (A is not Invertible) then return ⊥;
    else
    /* 1 X N multiplied by N X N results in 1 X N matrix */
        output = MATRIX_MULT (X, A ) ;
        output = MATRIX_ADD ( B, X) ;
    return output;
end
```

4.4.2 Inverse PwIP

The above uniformly distributed permutation can be un-permuted. The inverse of such a PwIP is defined as follows. *Note:* In case modular addition is used while performing *PwIP*, then Addition and Subtraction are same in Galois Field, GF(2).

$$f^{-1}_{A,B}(Y) = ((Y \oplus B) \times A^{-1}) \tag{5}$$

The algorithm for the inverse PwIP is defined in Algorithm.2.

Algorithm 2: FNR Inverse PwIP Algorithm

```
Inputs : Matrix A, Matrix B, bitvector input
Output: bitvector output
Function Inverse-PwIP( A, B, input, n) is
    /* bitvectors input and outputs are 1XN matrices      */
    if ( A is not invertible ) then  return ⊥;
    else
        C = MATRIX_INVERSE(A) ;
        output = MATRIX_ADD( B, input, n) ;
        output = MATRIX_MULT (output, C) ;
    return output;
end
```

4.5 Encryption

4.5.1 Overview

Input plain text is subjected to *PwIP* to get a uniformly distributed permutation of the same. This follows by a Feistel network of r rounds (the figure 1 shows only two rounds for simplicity). The number of rounds r is 7, detailed discussions is given in later sections 5, 7.1. The output of the Feistel network is subjected to $PwIP^{-1}$. The final output is then considered as cipher text.

4.5.2 Algorithm

The algorithm for the same is described in Algorithm.3

4.6 Decryption

4.6.1 Overview

The algorithm is very similar to encryption except that the processing is done in reverse way.

Algorithm 3: FNR Encryption Algorithm

```
Inputs : key k, char* tweak, bitvector plain, integer n
        /* n is max number of bits and even */
Output: bitvector cipher
        /* cipher and plain are of same bit length */
Function Encrypt(k, tweak, plain, n) is
begin
    /* Assumption: A, B Matrices are available as
       defined in 1, 2 */
    if $|plain| > n$ then return $\perp$;
    else if $|plain| < n$ then
    begin
        prepend '0' to plain such that $|plain| = n$;
    end
    /* perform uniform distributed permutation */
    bitvector d = pwip(A,B,plain,n);
    /* perform r rounds of Feistel network */
    while $i < r$ do
    begin
        left = d[0..n/2] ;
        right = d[n/2 .. n-1] ;
        left = right ;
        right = left $\otimes AES_{key}(i \parallel tweak \parallel right)$ ;
        d = left || right ;
        i++;
    end
    /* perform inverse of permutation */
    bitvector cipher = inverse-pwip(A,B,d,n);
    return cipher;
end
```

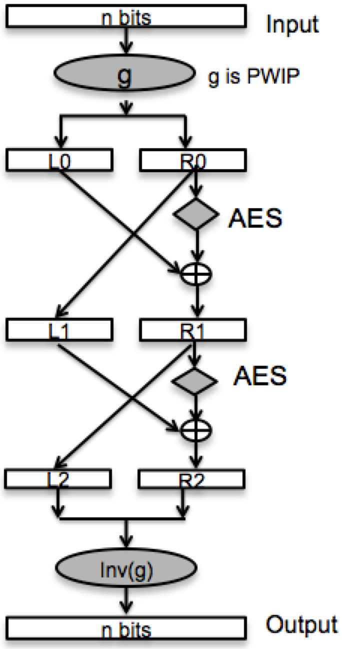

Figure 1: Two Round FNR

4.6.2 Algorithm

The algorithm for the same is described in Algorithm.4 . The differences with encryption algorithm can be observed as shown in line 13

5 Design Choices

Feistel Networks Luby Rackoff constructions based on Feistel Networks are subjected to rigorous theoretical analysis over many years. The principles underlying constructions of PRPs from a PRFs are widely accepted. Also their length preserving property makes them ideal choice for designing arbitrary length block ciphers.

Pairwise Independent Permutations Better security bounds were proven to be achievable by usage of *PwIP* in [8]. But their choice of performing operations in $GF(2^n)$ makes it difficult to define exact GF representations for arbitrary length ciphers. For this reason we propose performing GF operations in invertible matrices.

Round Functions The security of PRP constructed by a Feistel Network based scheme relies on security of underlying PRF (i.e round function) [7]. AES in ECB mode when performed only on a single block is considered a good PRF. Also using AES

Algorithm 4: FNR Decryption Algorithm

```
Inputs : key k, char* tweak, bitvector cipher, integer n
         /* n is max number of bits and even  */
Output: bitvector plain
         /* both cipher and plain are n bits  */
Function Decrypt(k, tweak, cipher, n) is
begin
    /* Assumption:  A,B Matrices are available
       as defined in 1, 2                              */
    if ($|cipher| \neq n$) then return $\perp$;
    /* perform pair wise permutation                   */
    bitvector d = pwip(A,B,cipher,n);
    /* perform r rounds of Feistel network             */
    while  $i < r$ do
    begin
        left = d[0..n/2] ;
        right = d[n/2 .. n-1] ;
        left = right ;
        right = left $\otimes AES_{rk_{(r-i)}}((r - i) \parallel tweak \parallel right)$
        d = left || right ;
        i++ ;
    end
    /* perform inverse of permutation                  */
    bitvector plain = inverse-pwip(A,B,d,n);
    return plain;
end
```

as round function one can leverage the underlying hardware support. Thus 2 of the design goals laid can be met by using AES.

Tweak Since the input domains are very small in nature, support of tweak would add additional randomness. Adding user supplied tweak string along with the iteration count is the simplest form tweak.

Rounds A minimum of 7 rounds are needed to mitigate adaptive chosen plain text and chosen cipher text attacks due to Patarin's proof [10]. More rounds may be preferred for input domains of smaller size.

6 Properties

6.1 Advantages

No length expansion The length of plain text and cipher text is same. No expansion in cipher text facilitates avoiding re-engineering of packet formats, database columns etc.

Range Preservation The encryption function results in the cipher which is in the same range of input values. This aides in designing format preservation of input domains.

Arbitrary Length The design does not mandate any fixed input lengths. FNR is flexible for input domains that are ≥ 32 bits and ≤ 128 bits

Key Length The key length is not dependent on the input length and rather depends on underlying PRF (in this case AES-128/256).

6.2 Disadvantages

Performance The usage of matrices might add performance over head.

No Integrity FNR does not provide authentication and integrity.

Deterministic FNR does not provide any semantic security.

7 Security

Security of LR schemes under went rigorous analysis by the community over many years. Also usage of *pwip* is later proven to mitigate basic linear and differential cryptanalysis [14].

7.1 Round Count

The security measure of block ciphers is based on the probability with which an attacker can distinguish the cipher text from a random text. Although our *PwIP* is different from theirs, without loss of generality, detailed proof given in [8] holds good for FNR.

If r is round count, n is number of bits of input domain, m is number of queries an attacker needs to make, then the security measure for FNR, is defined as in Equation.6.

$$(r/2 * m^2/2^{(1-1/r)*n}) where\ r \geq 4 \tag{6}$$

It is to be noted that without the use of *pwip* functions the security measure of pure Feistel Networks due to Patarin's proof [9] is defined as in Equation.7

$$5 * (m^3)/(2^n) \tag{7}$$

So for example an input domain of 32 bits and round count of 7, it requires approximately 8757 pairs of plain text and cipher text. Where as without the use of *PwIP* functions attacker just needs around 950 pairs of plain text and cipher text.

A performance trade-off exists depending on the underlying implementation of AES. If AES execution of each round is based on implementation of a software library, performing more rounds may be costlier. Where as if AES execution is at assembly level instruction, as supported by few modern processors, the performance over head may be very negligible.

7.2 Brute force attacks

Luby Rackoff constructions provided a neat way to construct a secure PRP from a secure PRF [7]. The strength of such PRP is based on underlying PRF. In FNR mode the PRF being AES, the strength of FNR is as secure as AES. For this reason exhaustive search on key space of AES is best known attack on FNR.

7.3 Measures

Sound key management practices may mitigate attacks based on stealing keys. In situations where key management is not available, generating key from password based key derivative function like PBKDF2 is suggested than hard-coding the keys in the software code. Access control measures to the encryption/decryption oracles (devices that perform the operations) would mitigate attackers from learning the known plain and cipher text pairs.

8 Implementation And Performance

The software implementation of Feistel networks is pretty straightforward. Matrix operations are crucial to the overall performance of the implementation. It is also suggested

Key	Tweak
“0000000000000000”	“tweak-is-string”

Table 1: Secrets

that AES operations being supported at assembly level by processors can be leveraged for better performance.

The reference implementation has been benchmarked to measure the performance of the algorithms in Figure.2 and Figure.3. The graphs are plotted for both AES and AES-NI instructions as options for internal PRP. The X-axis represents the number of records encrypted and Y-axis represents the time taken in seconds. The benchmarking is performed an virtual machine that runs Ubuntu 12.4 with 8 GB RAM on an Intel Sandy Bridge Generation of Processor’s with 4 vCPU’s.

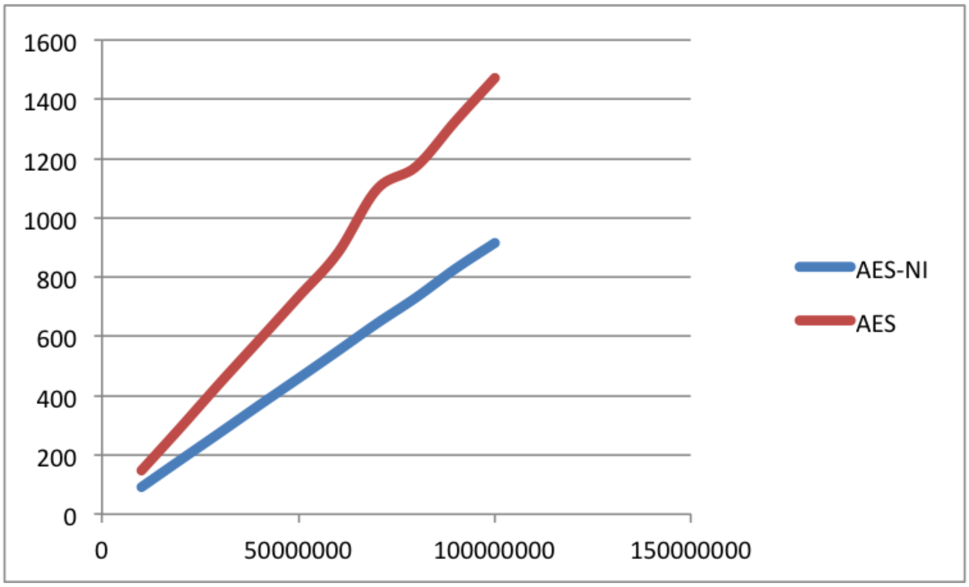

Figure 2: Performance of IPv4 Addresses

9 Test Vectors

Note the inputs key and tweak are character strings in Table.1. Tweak is an arbitrary length string which is expanded into a fixed length form. Note that even though same secrets mentioned Table.1 are used, the results might vary due to the choice of A,B Matrices used in PWIP function.

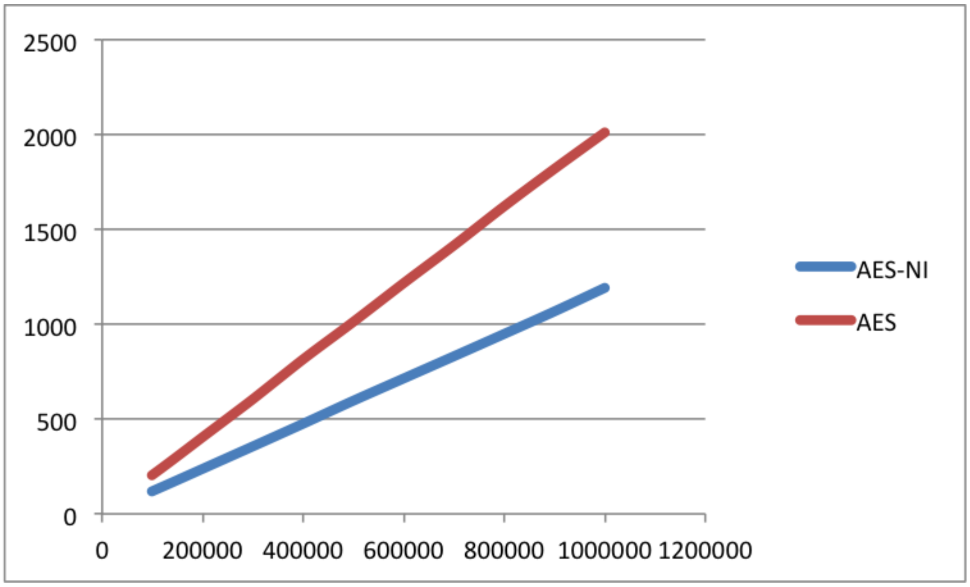

Figure 3: Performance of Credit Card Numbers

9.1 IPv4 addresses

The test vectors for various IPv4 Addresses are given in Table.2 . Each IPv4 is ranked as 32 bit integer before it is encrypted, the resultant cipher text is a 32 bit integer which is de-ranked into a dotted notation.

9.2 Credit card numbers

The test vectors for various credit card numbers are given in Table.3. Each CC number is ranked as 15 digit number by dropping the LUHN_CHECKSUM. The ranked integer is then encrypted to get a cipher text that is again 15 digit number. Such integer is de-ranked by appending a LUHN_CHECKSUM at the end into a valid Credit card number.

10 Conclusions

In this paper we proposed a flexible and practical arbitrary block domain cipher. We provide the reference implementation's performance results, test vectors. Also we provided examples of how to preserve formats of few data types like IPv4 addresses and Credit card numbers. Our work is flexible variant of Naor and Reingold's work. To be precise we propose usage of Triple Wise Independent Functions. We recommend using this block cipher for domain sizes 32 bits to 128 bits. Different techniques need to be used to achieve good security for inputs less than 32 bits or greater than 128 bits

Plain Text		**Cipher Text**	
Raw(Dotted)	*Ranked(Integer)*	*Raw(Integer)*	*De-ranked(Dotted)*
192.168.1.0	3232235776	2676870780	159.141.206.124
192.168.1.1	3232235777	2129658955	126.240.4.75
192.168.1.2	3232235778	3505438271	208.240.190.63
192.168.1.3	3232235779	3073749301	183.53.177.53
192.168.1.4	3232235780	2962433103	176.147.36.79
192.168.1.5	3232235781	1726684009	102.235.27.105
192.168.1.6	3232235782	344899540	20.142.191.212
192.168.1.7	3232235783	2172459699	129.125.26.179
192.168.1.8	3232235784	257448048	15.88.88.112
192.168.1.9	3232235785	1699298390	101.73.60.86

Table 2: Test Vectors for IPv4 Addresses

Plain Text		**Cipher Text**	
Raw	*Ranked*	*Raw*	*De-ranked*
4556584414106354	455658441410635	975846115884519	9758461158845197
4486224784662570	448622478466257	716640796278824	7166407962788248
4929883910358398	492988391035839	665162088006340	6651620880063403
4929880239524890	492988023952489	932731766659682	9327317666596825
4916550835157636	491655083515763	949857941349711	9498579413497119
4486508454953164	448650845495316	297883963574671	2978839635746717
4929374350260880	492937435026088	233324444317587	2333244443175870
4486811141966098	448681114196609	595721797141331	5957217971413317
4916855430912917	491685543091291	173658071436224	1736580714362241
4024007179621627	402400717962162	756355301132583	7563553011325836

Table 3: Test Vectors for Credit Card numbers

11 Acknowledgments

We sincerely thank Dr. David McGrew, Anthony Grieco, Dr.Zulfikar Ramzan, for their crucial suggestions, improvements in our work. The reference implementation is written by Scott Fluhrer and demo applications were written by Kaushal Bhandankar.

References

[1] 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*.

[2] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Selected Areas in Cryptography*, pages 295–312. Springer, 2009.

[3] Mihir Bellare and Phillip Rogaway. On the construction of variable-input-length ciphers. In *Fast Software Encryption*, pages 231–244. Springer, 1999.

[4] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *Topics in CryptologyCT-RSA 2002*, pages 114–130. Springer, 2002.

[5] Debra Lee Cook. *Elastic block ciphers*. PhD thesis, Columbia University, 2006.

[6] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001.

[7] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[8] Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Lubyrackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.

[9] Jacques Patarin. Improved security bounds for pseudorandom permutations. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 142–150. ACM, 1997.

[10] Jacques Patarin. Luby-rackoff: 7 rounds are enough for 2 n (1- ε) security. In *Advances in Cryptology-CRYPTO 2003*, pages 513–529. Springer, 2003.

[11] Sarvar Patel, Zulfikar Ramzan, and Ganapathy S Sundaram. Efficient constructions of variable-input-length block ciphers. In *Selected Areas in Cryptography*, pages 326–340. Springer, 2005.

[12] Phillip Rogaway and Davis Tweet. Format-preserving encryption. 2010.

[13] Terence Spies. Format preserving encryption. *Unpublished white paper, www. voltage. com Database and Network Journal (December 2008), Format preserving encryption: www. voltage. com*, 2008.

[14] Serge Vaudenay. Decorrelation: a theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, 2003.