FNR : Arbitrary length small domain block cipher proposal

Sashank Dara and Scott Fluhrer

Cisco Systems, Inc,
170 West Tasman Drive, San Jose,
CA 95314
{sadara,sfluhrer}@cisco.com

Abstract. We propose a practical flexible (or arbitrary) length small domain block cipher, *FNR* encryption scheme. FNR denotes **F**lexible **N**aor and **R**eingold. It can cipher small domain data formats like IPv4, Port numbers, MAC Addresses, Credit card numbers, any random short strings while preserving their input length. In addition to the classic Feistel networks, Naor and Reingold propose usage of Pair-wise independent permutation (PwIP) functions based on Galois Field GF(2^n). Instead we propose usage of random $N \times N$ Invertible matrices in GF(2).

Keywords: Feistel Networks, Luby Rackoff, block ciphers, length preserving

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. Network telemetry data, transaction logs etc. are often required to be shared for benefiting from variety of *Software-as-Service* applications like security monitoring etc. Such sensitive data fields are 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 plaintext (of any smaller lengths) to result in a 128 bit ciphertext with the aid of padding. Expansion of ciphertext length may be undesirable for said reasons.

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 based on Galois Field $GF(2^n)$ in first and last rounds of LR constructions. It is proven to provide additional randomness and security. But $GF(2^n)$ representations for arbitrary lengths of inputs is difficult in practice. We propose usage of invertible matrices to provide a neat and generic way to achieve Pair-wise independence for any arbitrary length.

2 Prior Art

Luby Rackoff Constructions are considered seminal work in formalizing secure block cipher design [6]. 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],[10]. 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.

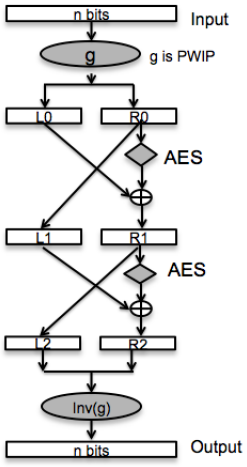


Fig. 1: Two Round NR

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 [11]. 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 [7] as shown in the figure 1. While their techniques are based on performing operations in GF(2^n) we propose to operate on invertible matrices. This makes our scheme flexible enough to perform on any arbitrary input fields.

2.1 Definitions

2.2 Secrets

There are various secret keys used in FNR.

1. **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.
2. **Tweak:** A tweak, T, is like *salt* or *IV*. In practice, A string is supplied by the user, as tweak, which is then encoded as fixed length binary string using some cryptographic hash function.
3. **A, B** are two matrices. A is invertible binary matrix of $N \times N$ dimension. B is binary vector of $1 \times N$ dimension. Where N denotes number of bits in the input. Both A,B should be uniformly distributed and randomly generated.

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

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

2.3 Pair-wise Independent Permutation (PwIP)

It is combinatorial construction to achieve a uniformly distributed permutation of given input. It has the property that for any two distinct inputs x, y, and any two distinct outputs x^1, y^1, the probability that x^1=PwIP(x) and y^1=PwIP(y) is uniform, that is, is $1/((2^n)*(2^n-1))$ independent of x, y, x^1, y^1.

Let the input X be a binary vector of n bits length, considered as 1 X N matrix, then $PwIP_{A,B}(X)$ 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.

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

$$PwIP_{A,B}(X) = (X \times A) \oplus B \text{ where A,B are defined in 1 and 2} \tag{4}$$

Inverse PwIP 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).

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

2.4 Feistel Networks

Feistel is symmetric structure to construct block ciphers. One round of Feistel is a $2n$ bit permutation δ, with an n bit round function as defined below

$$\delta_f(L,R) = (R, L \otimes f(R)) \qquad where |L| = |R| = n \tag{6}$$

An r round Feistel network is simply the composition of r one round Feistel structures, transforming r n-bit functions $f_1, f_2 ... f_r$ into a $2n$ bit permutation.

$$\delta_{f_1, f_2 .... f_r}(L,R) = \delta(f_1) \circ \delta(f_2) \circ ....\delta(f_r) \tag{7}$$

The security of PRP constructed by a Feistel Network based scheme relies on security of underlying PRF (i.e round function) [6]. The security guarantee depends on ensuring a different round function for each round. We propose using AES in ECB mode as the round function. Now to ensure the output is distinct in each round, we could use unique round key or by ensuring the inputs to the round function is distinct for each round. We achieve this by mixing a *round_const* for each round to the input to to PRF.

$$round\_const_r = \{\texttt{0x00, 0x03, 0x0c, 0x0f, 0x30, 0x33, 0x3c}\} \qquad where \quad r \in \{1,7\} \tag{8}$$

2.5 Encryption

The inputs to encryption algorithm are plaintext P, that needs to be encrypted, a secret key K, a tweak T and the matrices A,B. The output of an encryption function is n bits of ciphertext C.

Algorithm 1: FNR Encryption Algorithm

Inputs : key k, char* tweak,Matrix A, Matrix B, bitvector plain, integer n
`/* n is max number of bits and even */`
Output: bitvector cipher
`/* cipher and plain are of same bit length */`

```
1  Function Encrypt(k, tweak, plain, n) is
2  begin
3      if (|plain| ≠ n) then return ⊥;
4      bitvector d = PwIP(A,B,plain,n);
5      while  i < r do
6      begin
7          left = d[0..n/2] ;
8          right = d[n/2 .. n-1] ;
9          left = right ;
10         right = left ⊗AES_key(round_const_i ‖ tweak ‖ right) ;
11         d = left ‖ right ;
12         i++;
13     end
14     bitvector cipher =PwIP^(−1)(A,B,d,n);
15     return cipher;
16 end
```

Overview Input plaintext is subjected to *PwIP* to get a uniformly distributed permutation of the same. This follows by a Feistel network of $r = 7$ rounds. The output of the Feistel network is subjected to $PwIP^{-1}$. The final output is then considered as ciphertext. The algorithm for the same is described in Algorithm.1.

2.6 Decryption

The inputs to decryption algorithm are ciphertext C, secret key K, tweak T and the matrices A,B. The output is plaintext P.

Overview The algorithm is very similar to encryption except that the processing is done in reverse way. The algorithm for the same is described in Algorithm.2 . The differences with encryption algorithm can be observed as shown in line 11

3 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].

3.1 Round Functions

If assume that the AES output for any given input is uniformly distributed, that means that the AES output bits we actually use in the Feistel will be independent between even and odd rounds (even

Algorithm 2: FNR Decryption Algorithm

Inputs : key k, char* tweak,Matrix A, Matrix B, bitvector cipher, integer n
`/* n is max number of bits and even */`
Output: bitvector plain
`/* both cipher and plain are n bits */`

```
1  Function Decrypt(k, tweak, cipher, n) is
2  begin
3      if (|cipher| ≠ n) then return ⊥;
4      /* perform pair wise permutation */
5      bitvector d = PwIP(A,B,cipher,n);
6      while i < r do
7      begin
8          left = d[0..n/2] ;
9          right = d[n/2 .. n-1] ;
10         left = right ;
11         right = left ⊗AES_k(round_const_(r−i) ‖ tweak ‖ right)
12         d = left ‖ right ;
13         i++ ;
14     end
15     /* perform inverse of permutation */
16     bitvector plain = PwIP^(−1)(A,B,d,n);
17     return plain;
18 end
```

if the attacker could engineer a collision with probability 1; the fact that the collision probability between even and odd round is actually considerably smaller turns out to be irrelevant). As we add the round constants as defined in equation.8 as last byte to the input to AES

3.2 Round Count

A minimum of 7 rounds are needed to mitigate adaptive chosen plaintext and chosen ciphertext attacks due to Patarin's proof [9].

The security measure of block ciphers is based on the probability with which an attacker can distinguish the ciphertext from a random text. Although our *PwIP* is different from theirs, without loss of generality, detailed proof given in [7] 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.9.

$$(r/2 * m^2/2^{(1-1/r)*n}) where\ r \geq 4 \tag{9}$$

It is to be noted that without the use of *PwIP* functions the security measure of pure Feistel Networks due to Patarin's proof [8] is defined as in Equation.10

$$5 * (m^3)/(2^n) \tag{10}$$

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

4 Implementation

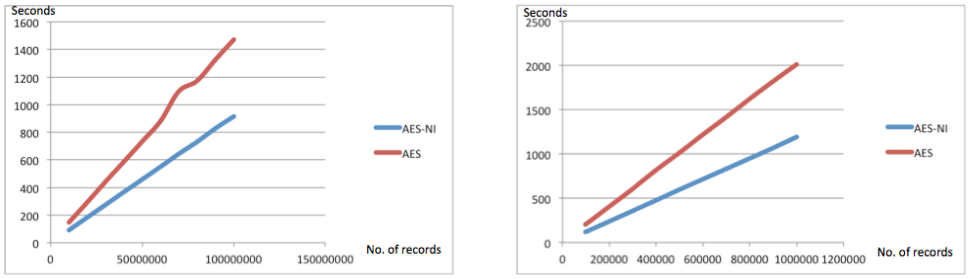
4.1 Feistel Network

Our reference implementation is slightly different from most implementations of LR, in that we don't divide the block into two separate halves; instead, we use the even bits as one half and the odd bits as other half, and we don't swap them; instead, we alternate between rounds which half we use as the input to our random function, and which half we XOR the output of the random function into. Since we have an odd number of rounds ($r = 7$), this all works out.

Nits: if the block we're encrypting has an odd number of bits, this is strictly speaking an unbalanced Feistel (if unbalanced only by a single bit). In addition, if we're encrypting a single bit, this really isn't a Feistel at all (because one half is empty).

4.2 Performance

The performance of the algorithms have been benchmarked in Figure.2. The graphs are plotted for both AES and AES-NI instructions as options for internal PRP. 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. The source code is available under LGPLv2[12].



(a) IPv4 Addresses (b) Credit Card Numbers

Fig. 2: Performance of FNR

4.3 Test Vectors

For generating the below test vectors, the key used is *'0000000000000000'* and tweak used is *'tweak-is-string'*. Tweak is an arbitrary length string which is expanded into a fixed length form. Note that even though same secrets are used, the results might vary due to the choice of A,B Matrices used in PwIP function.

The test vectors for various IPv4 Addresses, Credit Card numbers are given in Table.1 and Table.2. Each IPv4 is ranked as 32 bit integer before it is encrypted, the resultant ciphertext is a 32 bit integer which is de-ranked into a dotted notation. Each CC number is ranked as 15 digit number by dropping the LUHN_CHECKSUM. The ranked integer is then encrypted to get a ciphertext 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.

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

Table 1: 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

Table 2: Test Vectors for Credit Card numbers

5 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. We recommend using this block cipher for domain sizes 32 bits to 128 bits. FNR does not provide authentication and integrity. FNR does not provide any semantic security when used in ECB mode (like all other deterministic modes)

6 Acknowledgments

We sincerely thank our colleagues Dr. David McGrew, Anthony Grieco, Dr.Zulfikar Ramzan, for their crucial suggestions, improvements in our work. Also we acknowledge exhaustive comments, corrections and suggestions from Dr. Praveen Gauravaram (Tata Consultancy Services Innovation Labs), Dr. Kapali Viswanathan (Hewlett-Packard), Dr. Saugat Majumdar (Aruba Networks).

The reference implementation is written by Scott Fluhrer and demo applications were written by Kaushal Bhandankar. Many bug fixes, patches, code review performed by our colleagues Olve Maudal, Anand Verma, Mohit Kumar and Abhishek Singh.

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. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.
7. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Lubyrackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.
8. 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.
9. 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.
10. 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.
11. Phillip Rogaway and Davis Tweet. Format-preserving encryption. 2010.
12. Scott Fluhrer Sashank Dara. Reference Implementation of FNR. `https://github.com/sashank/libfnr`, 2014.
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.