A SYMMETRIC-KEY ENCRYPTION ALGORITHM:

MULTI-S01[1]

An Integrity-Aware Block Encryption Based on Cryptographic Pseudorandom Number Generator

Specification Ver 1.2 (Revised on May 14, 2002)

HITACHI, LTD.

[1] MULTI-S01 stands for “MULTImedia encryption algorithm-S01”.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

Abstract

We propose a block-wise encryption algorithm MULTI-S01 that uses the PANAMA pseudorandom number generator (PRNG). Assuming the security of PANAMA , MULTI-S01 achieves high security of both data confidentiality and data integrity as well as high performance on software implementation and hardware implementation.

In terms of security, there are two characteristics: assuming a secure PRNG, we can guarantee information confidentiality and integrity. In addition, the attacker (without knowledge of secret key) cannot generally determine any part of actual output sequence of PRNG just by knowing known-plaintexts.

The MULTI-S01 algorithm also has two performance advantages over encryption algorithms based on block ciphers. Firstly, since PANAMA operates faster than a block cipher, the entire encryption can be done with less computational complexity with efficient $\mathbf{F}_{2^{64}}$ multiplication than when a block cipher is used. In proposed scheme, the encryption computationally costs constant overhead calculation and computation in proportion to message length. The latter substantially costs 64-bit random number generation and a multiplication in $\mathbf{F}_{2^{64}}$.

Secondly, pre-computation or parallel computation of a random sequence is easy to apply with our algorithm, so additional hardware resources can satisfy the demand to very high performance.

Employing PANAMA as a PRNG, software implementations of encryption and decryption on Alpha processor perform at approximately 17.8 and 18.0 cycle/byte, respectively. For algorithm and key initialization, approximately 34,000 clocks for encryption and decryption, are required.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

Contents

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

1. Preliminaries

1.1. Expressions of Variables and Symbols

In this document, we define following symbols for integer operation.

$\oplus$	Bitwise exclusive-or (XOR) operation
$\otimes$	Multiplication in finite field $\mathbf{F}2^{64}$
+	Arithmetic addition
×	Arithmetic multiplication
OR	Bitwise logical OR operation
NOT	Bit complement
$\mathrm{ROTL}_k(X)$	Left circular rotation in 32-bit register

$X^{(64)}$ refers to a 64-bit-long variable, called X. As for the subscripts, the number in the subscript means the index. We count the index from 1.

We define the field $\mathbf{F}_{2^{64}}$ as a quatient field $\mathbf{F}_2[X] / (f(X))$, where $f(X)$ is the following irreducible polynomial of degree 64:

$$f(X) = X^{64}+X^{4}+X^{3}+X+1.$$

The symbol $\lceil x \rceil$ denotes the ceiling function, the smallest integer that is larger or equal to x. For two binary strings $A^{(64)}$ and $B^{(64)}$, for instance, we use a notation $A||B$ to represent the concatenation of two bit strings A and B.

1.2. Byte Ordering in a Word (Endianess)

We define the rule of byte ordering in a 64-bit word as endianness. A key, plaintext, ciphertext, redundancy data, and initial value are given and represented as byte array data. If we convert a word to four bytes or convert four bytes to a word, we adopt big-endian. For instance, if we try to store data in a byte array X into a 64-bit word array Y, then the mathematical representation of Y is:

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

$$Y_i^{(64)} = \sum_{j=1,\ldots,8} X^{(8)}_{8(i-1)+j} 2^{(64-8j)}.$$

For visual comprehension, a 64-bit register $Y^{(64)}$ contains 8 bytes of data $X^{(8)}_i$ in the following order:

$Y^{(64)}$=[MSB]$X_1 \| X_2 \| X_3 \| X_4 \| X_5 \| X_6 \| X_7 \| X_8$[LSB],

where [MSB] and [LSB] represent the positions of the most significant byte and the least significant byte, respectively.

1.3. Operation in Finite Field $\mathbf{F}2^{64}$

MULTI-S01 uses operations in finite field $\mathbf{F}2^{64}$. In this section, we briefly introduce some fundamentals necessary for implementation.

Addition in is very simple, because it is identical to XOR operation. In this section, we substantially introduce multiplication in $\mathbf{F}2^{64}$. In this document, when multiplication in finite field $\mathbf{F}2^{64}$ is concerned, e.g., $A \otimes B$, we associate a 64-bit value A, to a polynomial whose degree is less than 64, and whose coefficients a_i are $a_i \in \{0,1\}$. Here we show how to determine the associated polynomials, $A(x)$ and $B(x)$.

- A bit value a_i={0,1} is defined as the ith most significant bit of A. Therefore, (a_1, a_2, a_3,...,a_{64})=A. Using a_i's, the associated polynomial is described as follows.
  $A(x) = a_1x^{63} + a_2x^{62} + a_3x^{61} + ... + a_{63}x + a_{64} = \Sigma_{i=1...64}\, a_i x^{(64-i)}$.
- Similarly, define b_i ={0,1} and associate following $B(x)$ to B.
  (b_1, b_2, b_3,...,b_{64})=B,
  $B(x) = b_1x^{63} + b_2x^{62} + b_3x^{61} + ... + b_{63}x + b_{64} = \Sigma_{i=1...64}\, b_i x^{(64-i)}$.

For the following calculation, we introduce two expressions. "$A(x)$ mod $P(x)$" denotes the modulo of $A(x)$ divided by $P(x)$. In this document, we set $P(x)=x^{64}+x^4+x^3+x+1$ and obtain the result of (modular) multiplication $C(x)$.

$$C(x) = A(x) \times B(x) \bmod P(x).$$

All the coefficients are in modulo 2, i.e., 0+0=0, 0+1=1, 1+1=0, 0×0=0, 0×1=0, and 1×1=1. Note that these operations are associative (the same result holds for swapping operands). Therefore, addition and multiplication correspond to XOR and AND operations, respectively. As the degree of $P(x)$ is 64, that of $C(x)$ is at most 63, i.e., the number of terms is at most 64. The 64-bit value corresponding to the obtained polynomial $C(x)$ is the result of multiplication.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

- Determine bit sequence c_i ={0,1} to be the coefficient of x^{64-i}, i.e., following relation is established.
  $C(x) = c_1x^{63} + c_2x^{62} + c_3x^{61} + ... + c_{63}x + c_{64} = \Sigma_{i=1...64}\, c_i x^{(64-i)}$.
- Determine the value of C, from c_i sequence as follows.
  $C=(c_1, c_2, c_3,...,c_{64})$.

Example

We demonstrate multiplication of *A* and *B*. Followings are hexadecimal expressions of A, and B.

A = 01234567 89abcdef,
B = fedcba98 76543210.

These binary expressions are

A = 0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111,
B = 1111 1110 1101 1100 1011 1010 1001 1000
0111 0110 0101 0100 0011 0010 0001 0000.

The associated polynomials are

$$A(x) = x^{56}+x^{53}+x^{49}+x^{48}+x^{46}+x^{42}+x^{40}+x^{38}+x^{37}+x^{34}+x^{33}+x^{32}+x^{31}+x^{27}+x^{24}$$
$$+x^{23}+x^{21}+x^{19}+x^{17}+x^{16}+x^{15}+x^{14}+x^{11}+x^{10}+x^{8}+x^{7}+x^{6}+x^{5}+x^{3}+x^{2}+x+1,$$
$$B(x) = x^{63}+x^{62}+x^{61}+x^{60}+x^{59}+x^{58}+x^{57}+x^{55}+x^{54}+x^{52}+x^{51}+x^{50}+x^{47}+x^{45}+x^{44}$$
$$+x^{43}+x^{41}+x^{39}+x^{36}+x^{35}+x^{30}+x^{29}+x^{28}+x^{26}+x^{25}+x^{22}+x^{20}+x^{18}+x^{13}+x^{12}+x^{9}+x^{4}.$$

Let $D(x) = A(x) \times B(x)$. Then $D(x)$ is

$$D(x)=A(x)x^{63}+A(x)x^{62}+A(x)x^{61}+A(x)x^{60}+A(x)x^{59}+A(x)x^{58}+A(x)x^{57}+A(x)x^{55}$$
$$+A(x)x^{54}+A(x)x^{52}+A(x)x^{51}+A(x)x^{50}+A(x)x^{47}+A(x)x^{45}+A(x)x^{44}+A(x)x^{43}$$
$$+A(x)x^{41}+A(x)x^{39}+A(x)x^{36}+A(x)x^{35}+A(x)x^{30}+A(x)x^{29}+A(x)x^{28}+A(x)x^{26}$$
$$+A(x)x^{25}+A(x)x^{22}+A(x)x^{20}+A(x)x^{18}+A(x)x^{13}+A(x)x^{12}+A(x)x^{9}+A(x)x^{4}$$
$$= x^{119}+x^{118}+x^{117}+x^{109}+x^{108}+x^{107}+x^{103}+x^{102}+x^{100}+x^{99}$$
$$+x^{94}+x^{93}+x^{91}+x^{87}+x^{83}+x^{78}+x^{76}+x^{71}+x^{69}+x^{68}$$
$$+x^{62}+x^{55}+x^{53}+x^{46}+x^{45}+x^{44}+x^{43}+x^{39}+x^{36}+x^{35}$$
$$+x^{29}+x^{27}+x^{23}+x^{22}+x^{19}+x^{12}+x^{7}+x^{6}+x^{5}+x^{4}.$$

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

Now we reduce the polynomial using $P(x)$. For simplest reduction, since $P(x)= x^{64}+x^4+x^3+x+1$, replace all terms with degrees more than 64 as following polynomials.

$$x^{64}=x^4+x^3+x+1,$$
$$x^{65}=x^5+x^4+x^2+x,$$
$$x^{66}=x^6+x^5+x^3+x^2,$$
$$\ldots$$
$$x^{124}= x^{64}+x^{63}+x^{61}+x^{60} = (x^4+x^3+x+1)+ x^{63}+x^{61}+x^{60}$$
$$= x^{63}+x^{61}+x^{60}+x^4+x^3+x+1,$$
$$x^{125}=x\times x^{124}=x^{64}+x^{62}+x^{61}+x^5+x^4+x^2+x = (x^4+x^3+x+1)+x^{62}+x^{61}+x^5+x^4+x^2+x$$
$$= x^{62}+x^{61}+x^5+x^3+x^2+1,$$
$$x^{126}=x\times x^{125}= x^{63}+x^{62}+x^6+ x^4+x^3+x.$$

Therefore we obtain the following polynomial as $C(x)$.

$$C(x)=x^{62}+x^{59}+x^{55}+x^{49}+x^{46}+x^{45}+x^{44}+x^{43}+x^{41}+x^{39}+x^{37}+x^{36}+x^{34}+x^{32}+x^{30}+x^{28}$$
$$+x^{27}+x^{26}+x^{24}+x^{23}+x^{20}+x^{18}+x^{17}+x^{16}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^7+x^5.$$

To obtain the result of multiplication, generate the bit sequence of coefficients.

```
c= 0100 1000 1000 0010 0111 1010 1011 0101
   0101 1101 1001 0111 0110 1111 1010 0000 (binary expression)
 = 48827ab5 5d976fa0 (hexadecimal expression) .
```

2. Algorithm Specification

2.1. Overall Structure and Interface

The MULTI-S01 cipher comes out two processes: encryption and decryption. Each process of encryption and decryption consists of a key stream generator and a mixing function. The mixing function for decryption is the inverse function of the mixing function for encryption.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

The key stream generator generates a key stream, *A*, *B* and *S* out of a secret key *K* and *Q*. The length of the key stream *B* depends on how long message is encrypted.

The encryption algorithm takes four inputs: the *m*-bit message *M*, 64-bit redundancy data *R*, secret key *K*, and initial vector *Q*. It outputs ciphertext *C*. These inputs are represented as byte-array data, i.e., $M^{(8)}_i$ (i = 1,..., $\lceil m / 8 \rceil$), $R^{(8)}_i$ (i = 1,…, 8), $K^{(8)}_i$ (i = 1,…, 32), and $Q^{(8)}_i$ (i = 1,…, 32). The length of the ciphertext is (64×($\lceil m / 64 \rceil$ +2)) bits stored in a byte array.

In corresponding decryption function, all four inputs, the *c*-bit ciphertext *C*, 64-bit redundancy data *R*, 256-bit secret key *K*, and 256-bit initial vector *Q* are given in byte-array data, i.e., $C^{(8)}_i$ (i = 1,..., $\lceil c / 8 \rceil$), $R^{(8)}_i$ (i = 1,…, 8), $K^{(8)}_i$ (i = 1,…, 32), and $Q^{(8)}_i$ (i = 1,…, 32). The decryption function outputs either the result of decryption *M'*, or an integrity error signal. If former is output, the result is stored in byte-array data. The encryption and decryption consist of 64-bit block-wise computations. Let $n = \lceil m / 64 \rceil$ +2 be the number of blocks. The key stream generator takes *K* and *Q* as inputs and generates a key stream $A^{(64)}$, *B* (64×*n* bits), and $S^{(64)}$. The mixing function for encryption takes *M*, *R*, *A*, *B* and *S* as inputs and generates *C*, while the mixing function for decryption takes *C*, *R*, *A*, *B* and *S* as inputs and outputs either the "*reject*" signal or the resultant data *M'*. Figure 1 is a block diagram of the overall structure on the MULTI-S01 cipher.

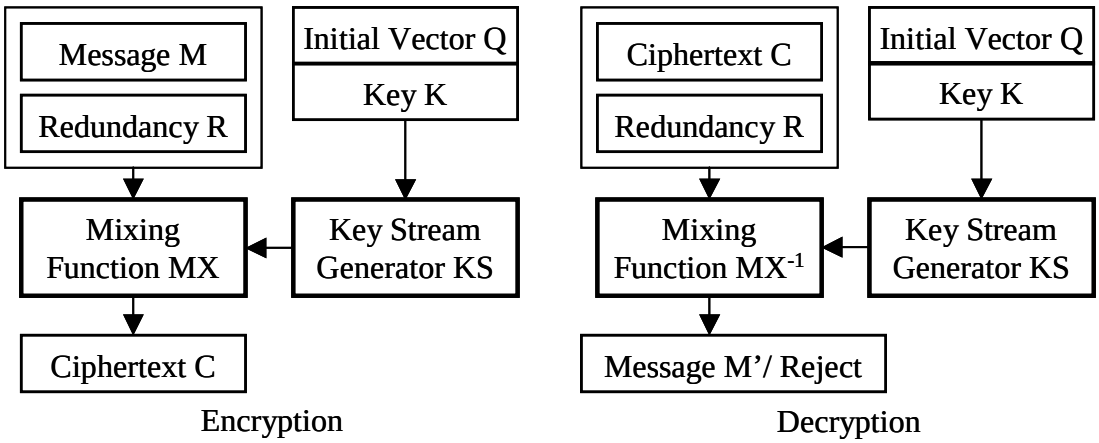

Figure 1: Overall block diagram of the MULTI-S01 cipher.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

2.2. Functionality

If a securely generated and shared key is used, an adversary without the secret key cannot know any information about a plaintext out of the ciphertext. In addition, if the redundancy data is shared securely (but not necessarily secret), this encryption scheme also provides security of detecting malicious alteration on a ciphertext. The security of detecting alteration, in other words, guarantees that the probability of successful malicious alteration is proven to be small enough.

2.3. Key Stream Generation

2.3.1. Key Stream Generator

The key stream generator prepares the necessary pseudorandom sequences A(64 bits), B(64×n bits), and S(64 bits) out of the input K and Q. The number of blocks n is defined as follows:

(Encryption) $n = \lceil m / 64 \rceil + 2$, where m is the number of bits in a message,
(Decryption) $n = \lceil c / 64 \rceil$, where c is the number of bits in a ciphertext.

For n block plaintext (namely the message data padded with random data and redundancy and stored in a 64-bit word array), an encryption uses a 64-bit non-zero pseudorandom number A, a (64×n)-bit pseudorandom sequence B, and 64-bit pseudorandom number S. The followings show how to generate A, B and S. We describe the key stream generator based on the PANAMA PRNG [2].

PANAMA PRNG

Algorithm

INPUT: 256-bit seed K,
256-bit initial vector Q.

OUTPUT: a 64-bit non-zero pseudorandom number A,
a (64× n)-bit pseudorandom sequence B,
a 64-bit pseudorandom number S.

(1) Initialize the PRNG with seed K , and initial vector Q.
(2) Generate a 64-bit non-zero pseudorandom number A with the PRNG.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

(3) Generate a (64× n)-bit pseudorandom sequence B.

(4) Generate a 64-bit pseudorandom sequence S.

As a default PRNG, we use the PANAMA PRNG.

2.3.2.Description of PANAMA

PANAMA was designed by J. Daemen and C. Clapp, and is a cryptographic module that can be used both as a hash function and as a stream cipher. It consists of 32-bit primitive operations. Now, we say “1 word” in the sense that it is 32-bit. In this paper, we use PANAMA as a PRNG, that generates the key stream of MULTI-S01.

PANAMA is a cryptographic module consisting of

a: a 17-word main register (*state*),

b: a 256-word additional register (*buffer*),

ρ: a non-linear transformation of a,

λ: a linear transformation of b.

The state is denoted by a and consists of 17 words a_0 to a_{16}. The buffer denoted by b is a linear feedback shift register with 32 stages; each stage consists of 8 words. An 8-word stage is denoted by b^j and its words by $b^j{}_i$. Both stages and words are indexed starting from 0. The index of a_i is always reduced to mod 17. Similarly the indices (i, j) of $b^j{}_i$ are reduced to mod 8 and mod 32, respectively.

PANAMA has three usage modes: Reset mode, Push mode, and Pull mode. When PANAMA operates to generate a pseudorandom sequence, the schedule of operation follows Table 1.

Table 1: The mode schedule of PANAMA key stream generator.

Time t (Round)	Mode	Input (256 bits)	Output (256 bits)
-34	**Reset**	—	—
-33	**Push**	K	—
-32	**Push**	Q	—
-31,. . . ,0	**Pull**	—	—
1, 2, ..., i, ...	**Pull**	—	Output Seq. Out_i

Reset mode

In Reset mode, the state and buffer are set to 0.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

Push mode

In Push mode, an 8-word input is applied and there is no output (see Figure 2).

Algorithm

INPUT: 8-word block p.

OUTPUT: no output.

(1) $x = b^{16}$.

(2) Update the buffer b ($b = \lambda(b) = \lambda(b, p)$. ($\lambda$ will be explained later.)).

(3) Update the state a ($a = \rho(a) = \rho(a, p, x)$. ($\rho$ will be explained later.)).

Note that ρ uses the old positions of buffer b.

Pull mode

In Pull mode, there is no input and an 8-word random sequence is delivered for each round. We use Pull mode to confuse the buffer and the state again (see Figure 2.) and to generate the key stream.

Algorithm

INPUT: number of rounds n.

OUTPUT: pseudorandom number K (8-word).

Iterate the following operation for $k = 0$ to n.

(1) Output $K = (a_9, \ldots, a_{16})$.

(2) $x = b^{16}$, $q = b^4$.

(3) Update the buffer b ($b = \lambda(b) = \lambda(b, p)$).

(4) Update the state a ($a = \rho(a) = \rho(a, p, x)$).

Note that the state-update function ρ uses old x value as an input as well as Push mode.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

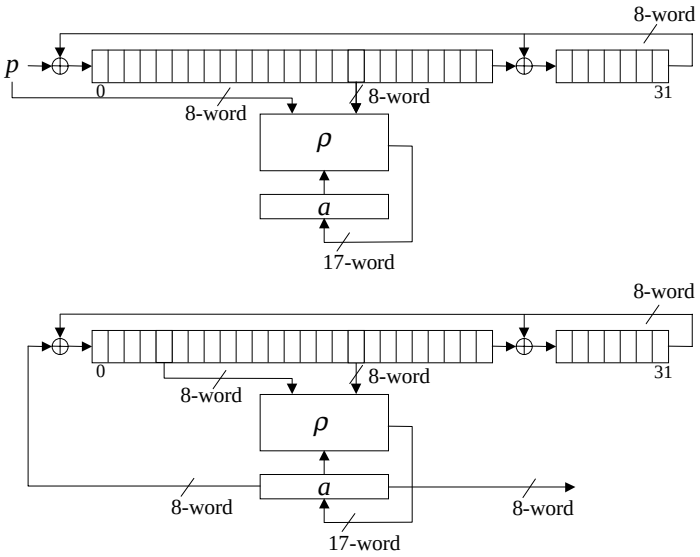

Figure 2: **Push** (above) and **Pull** (below) modes of PANAMA.

Updating of the buffer *b*

The buffer is a variation of Linear-feedback shift register with a word-oriented structure. The buffer is updated as follows (see Figure 3).

Algorithm

We denote the updated buffer d (i.e., $d = \lambda(b)$). λ is described as follows.

$d^j = b^{j-1}$ if $j \neq 0, 25$,

$d^0 = b^{31} \oplus q$,

$d^{25}_i = b^{24}_i \oplus b^{31}_{i+2}$, for $0 \leq i < 8$.

In Push mode, q is the input block p. In Pull mode, it is part of the state a given by

$q_i = a_{i+1}$ for $0 \leq i < 8$.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

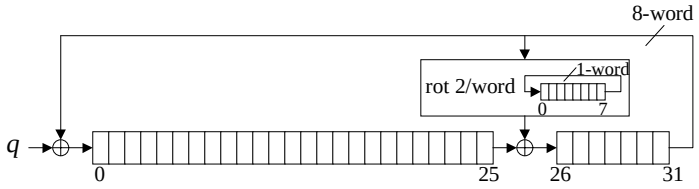

Figure 3: The buffer updating transformation λ.

Non-linear transformation ρ

The transformation of state a is composed of four transformations γ, π, θ, and σ (see Figure 4).

$\rho(a) = \sigma(\theta(\pi(\gamma(a))))$.

γ is a non-linear transformation defined by:

$\gamma(a_i) = a_i \oplus (a_{i+1} \text{ OR } (\text{NOT } a_{i+2}))$, $0 \le i < 17$.

π consists of a permutation of word positions and 32-bit left rotations. Let ROT $_k$ denote a rotation over k positions from LSB to MSB. Then we have:

$\pi(a_i) = \text{ROT}_{i(i+1)/2 \bmod 32}(a_{7i})$, for $0 \le i < 17$.

θ is a linear transformation defined by:

$c_i = a_i \oplus a_{i+1} \oplus a_{i+4}$ for $0 \le i < 17$.

The transformation σ corresponds to the bit-wise addition of buffer and input words. It is given by:

$\sigma(a_0) = a_0 \oplus 1$,

$\sigma(a_{i+1}) = a_{i+1} \oplus l_i$, $0 \le i < 8$,

$\sigma(a_{i+9}) = a_{i+9} \oplus b^{16}{}_i$, $0 \le i < 8$.

In Push mode l is the input words p, and in Pull mode l is b^4.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

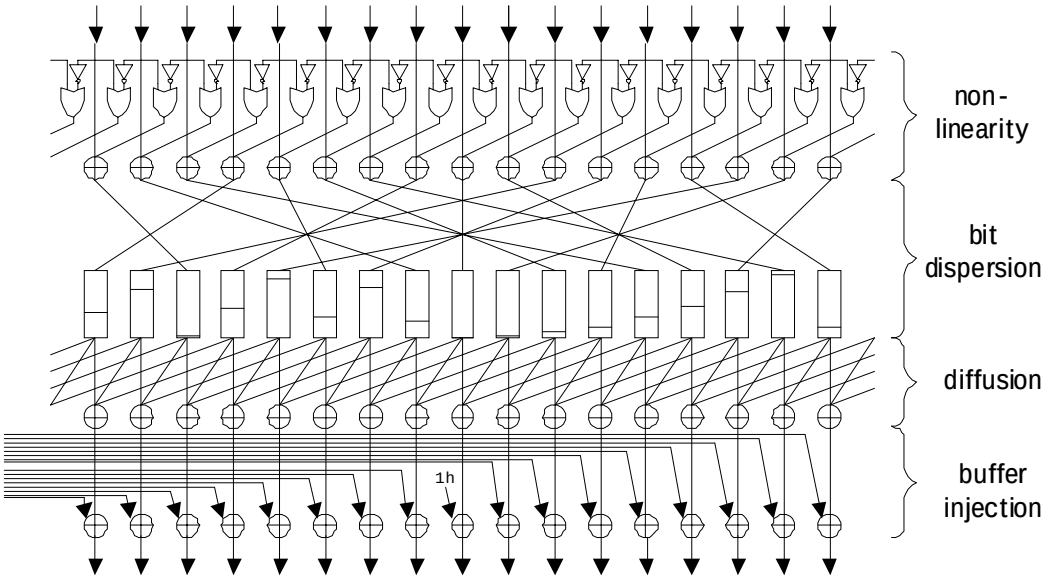

Figure 4: Non-linear transformation ρ.

2.4. Generating *A*, *B*, and *S* using **PANAMA**

In this section, we define the way to generate *A*, *B*, and *S* using PANAMA PRNG. Let *n* be the number of blocks (64 bits) in *B*. Use of the initial value *Q* is mentioned afterwards.

Sequence Generation

Input:	Secret key *K* (256 bits)
	Initial value *Q* (256 bits)
Output:	Key stream *A* (64 bits), *B* (64n bits), and *S* (64 bits)
Step 1:	Initialize PANAMA and input *K* and *Q*,
Step 2:	Let *A* be the first 64 bit output,
Step 3:	If *A* = 0, then do Step 2 again,
Step 4:	Let *B* be the following 64*n* bit output sequence,
Step 5:	Let *S* be the following 64 bit output.

Use and Remark of Initial Value *Q*

In MULTI-S01 cipher, the initial value of Panama is used for these purposes.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

PARALLEL COMPUTATION: Generation of key stream is assigned to plural processors according to the initial value.
RESYNCHRONIZATION: Update the initial value Q for Panama so that the decryptor can recover synchronization.
OPERATIONS FOR VERY LONG MESSAGES: We mention about it at Chapter 3.

The initial value Q must not be secret nor randomly chosen. However it must be shared between the encryption and the decryption. For other remark, whenever the encryption begins, the encryption must use the *flesh* key stream that has never appeared before. In other words, one may not use the same pair of (K,Q) for different combination of message and redundancy.

2.5. Mixing Function for Encryption

The mixing function for encryption consists of two parts: data padding and data mixing. Data padding generates the P $(64\times n)$-bit intermediate value, out of M, S and R. Data mixing generates C out of P, A, and B.

Data padding takes m-bit message, 64-bit redundancy data and secret padding data. Former two data are given as a byte sequence $M^{(8)}{}_i$, $(i = 1,..., \lceil m / 8 \rceil)$, and redundancy data $R^{(8)}{}_i$ $(i = 1,...,8)$. These are used to generate the intermediate value stored as a 64-bit word array $P^{(64)}{}_i$ $(i = 1,..., n)$. The 64-bit word array $P^{(64)}$ is assigned as follows:

$$P^{(64)}{}_i = \sum_{j=1,\ldots,8} M^{(8)}{}_{8(i-1)+j}\, 2^{64-8j} \quad (i = 1,..., n-2),$$

$$P^{(64)}{}_{n-1} = S,$$

$$P^{(64)}{}_n = \sum_{j=1,\ldots,8} R^{(8)}{}_j\, 2^{64-8j}.$$

Similarly, the initial feedback value $F^{(64)}{}_0$ is set as follows:

$$F^{(64)}{}_0 = 0\,.$$

In data mixing, ciphertext $C^{(64)}{}_i$ $(i = 1,..., n)$ is generated out of P_i, A, and B_i with the following equations:

$$F^{(64)}{}_i = P^{(64)}{}_i \oplus B^{(64)}{}_i\,,$$
$$C^{(64)}{}_i = (F^{(64)}{}_i \otimes A^{(64)}) \oplus F^{(64)}{}_{i-1} \quad (i = 1,..., n).$$

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

The ciphertext $C^{(8)}$ is the result of converting the 64-bit word array $C^{(64)}$ to the byte array. The encryption is depicted in Figure 5.

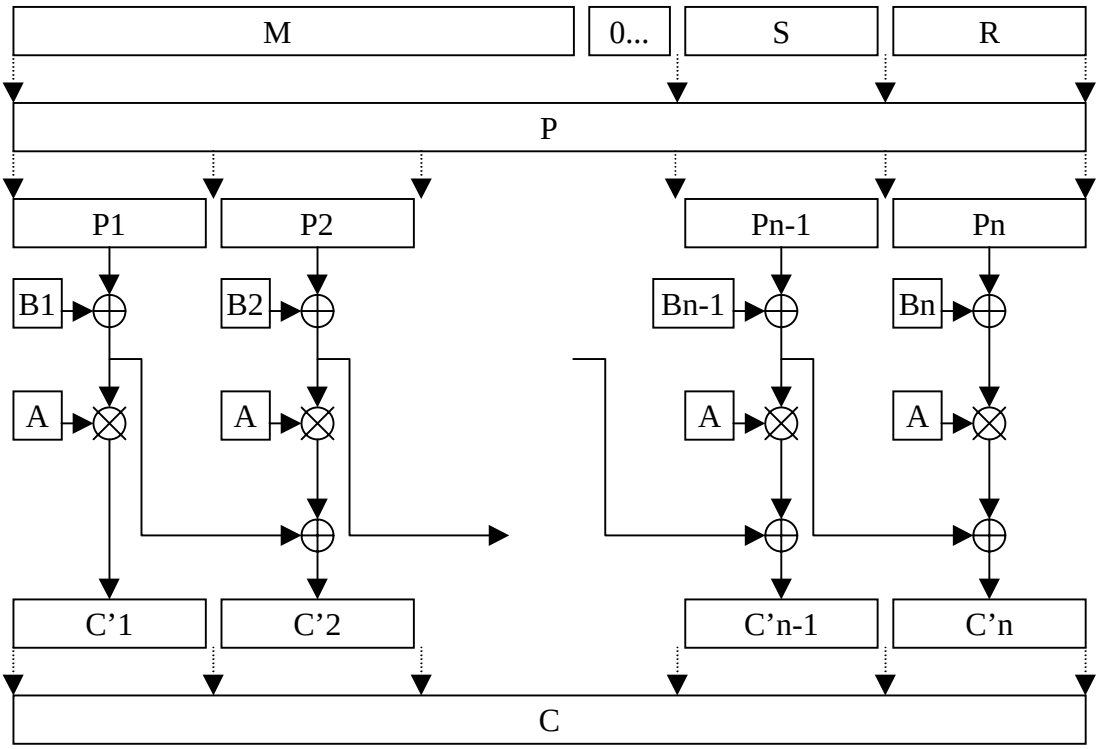

Figure 5: Mixing function for encryption.

2.6. Mixing Function for Decryption

The mixing function for decryption consists of two parts: data unmixing and integrity checking. Data unmixing generates an intermediate value *P'* out of *C'*, *A*, and *B*. Integrity checking outputs a "*reject*" signal or message *M'* according to check of data *P'*, *S*, and $R^{(8)}{}_i$.

The unmixing part initially converts the byte data array $C'^{(8)}{}_i$ to a 64-bit word array $C'^{(64)}{}_i$ (i =1,..., n , where n is the size of the array $C'^{(64)}$), respectively, and generates $P'^{(64)}{}_i$ out of $A^{(64)}$, $B^{(64)}{}_i$, and $C'^{(64)}{}_i$ with the following equations:

$$F'^{(64)}{}_0 = 0,$$
$$F'^{(64)}{}_i = (C'^{(64)}{}_i \oplus F'^{(64)}{}_{i-1}) \otimes (A^{(64)})^{-1} \ (i = 1,..., n),$$

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

$$P'^{(64)}_i = F'^{(64)}_i \oplus B^{(64)}_i \ (\ i = 1,..., n).$$

At the integrity checking part, $P'^{(64)}_i$ (i =1,..., n) is divided as follows:

$$M'^{(64)}_i = P'^{(64)}_i \ (\ i = 1,..., n-2),$$
$$S'^{(64)} = P'^{(64)}_{n-1},$$
$$R'^{(64)} = P'^{(64)}_n.$$

$R^{(64)}$ denotes the result of converting $R^{(8)}$ to a 64-bit value. Only if both $S^{(64)}=S'^{(64)}$ and $R^{(64)}=R'^{(64)}$ hold, the integrity checking part outputs the byte array of the message, the result of converting $M'^{(64)}_i$. Otherwise, this part outputs a "*reject*" signal. Figure 6 shows decryption.

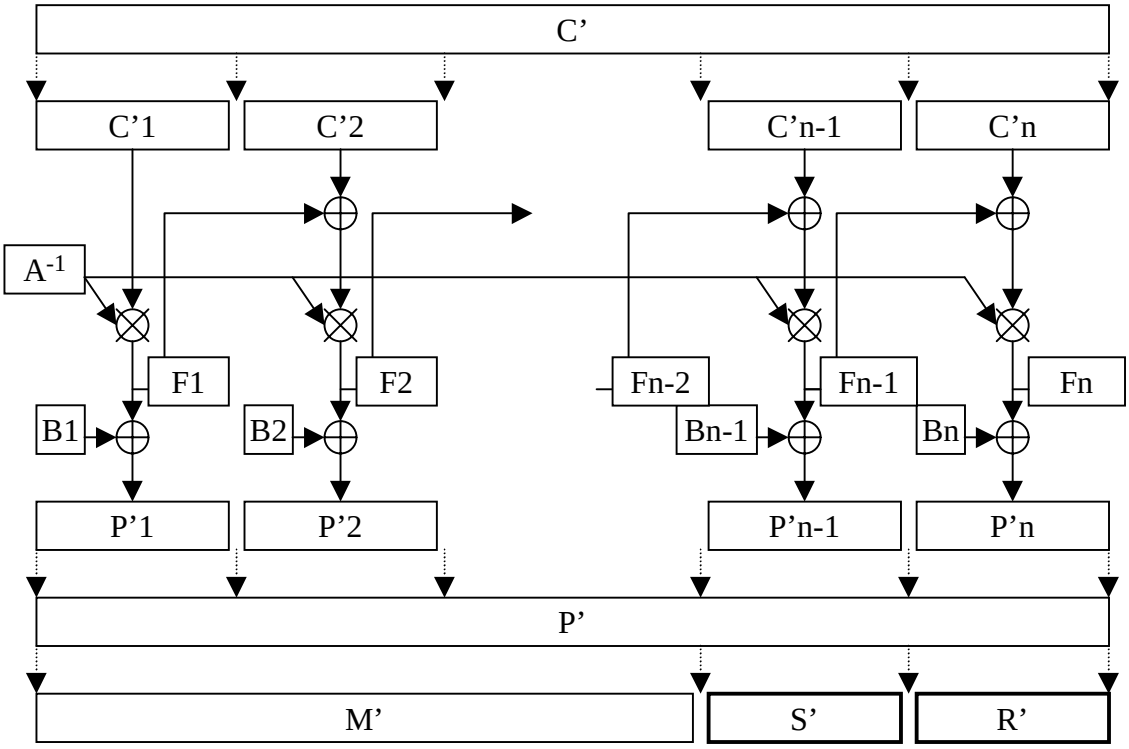

Figure 6: Mixing function for decryption.

To generate A^{-1} , the multiplicative inverse of A, one can use the equation $A^{-1} = A^q$, where $q = 2^{64-2}$ in Finite field $\mathbf{F}2^{64}$. For more efficient algorithms, Euclid's algorithm and almost inverse algorithm [1] are also usable.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.

3. Operations for Large Message

For messages larger than 2^{32} blocks, one divides the long message into large blocks of (2^{38}–128) bits in length and encrypts each large block individually. Note that for each large block, one must use different initial value for PANAMA key stream generation so that different key streams are generated for each large block.

Let **M-S01**(K ,Q ,R ,M) be the encryption function with K, Q, R, and M. If a long message μ is encrypted, one divides into 2^{38}–128-bit blocks, μ_1 , μ_2 , μ_3 ,..., μ_k . The ciphertext C for K is

$$C = \mathbf{M\text{-}S01}(K,0,0,\mu_1)\| \mathbf{M\text{-}S01}(K,1,1,\mu_2)\| ...\| \mathbf{M\text{-}S01}(K,k,k,\mu_k)$$

References

[1] Schroeppel, R., Orman, H., O'Malley, S., Spatscheck, O., "Fast Key Exchange with Elliptic Curve Systems," *Advances in Cryptology—CRYPTO'95, LNCS Vol. 963, Springer-Verlag*, 1995.

[2] Daemen, J., Clapp, C., "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption, 5th International Workshop, FSE'98 Proceedings, LNCS Vol. 1372, Springer-Verlag*, 1998.

Copyright © 2000, 2001, 2002 Hitachi, Ltd. All Rights Reserved.