Specification on a Block Cipher : Hierocrypt–3

Toshiba Corporation

May 2002

Contents

1 Design principle

We describe the design principle of Hierocrypt–3 in this section. We consider that the following points are important.

- Sufficient security against major attacks
- High performance on smartcards and middleware
- Efficiency in implementation
- Transparency of the design

To satisfy these conditions, we determined to use the nested SPN structure in the data randomizing part, and the Feistel structure in the key scheduling part.

1.1 Data randomizing part

We chose the SP network (SPN, for short) as a fundamental structure of the proposed ciphers.

The SPN structure has the following advantages.

- It has an established design recipe based on the coding theory.
- It does not have plain path unlike the Feistel structure.
- Apparent weak keys are difficult to occur compared with the Feistel structure.
- High speed encryption in a hardware implementation.

On the other hand, the disadvantages are as follows.

- The area tends to be bigger than the Feistel structure, as the designs of encryption and decryption are different.
- As the width of diffusion is as twice as the Feistel structure.

1.1.1 Nested SPN structure

The nested SPN structure is a hierarchical structure, where a higher-level S-box consists of the lower-level SP network. The branch numbers are hierarchically assured in the nested SP network. The calculational cost is cheaper than the original homogeneous SPN structure, because the width of diffusion is localized for the lower-level diffusion layer, and because the word number in MDS coding decreases in the higher-level diffusion layer. We propose to impose the following conditions in designing a nested SPN cipher [6, 5].

(i) The final round of SPN consists only of an S-box layer without the following diffusion layer in all levels.

(ii) Each diffusion layer is MDS in all levels.

(iii) The number of rounds is even in all levels except for the highest level.

(iv) Bit-wise key additions are located directly before all lowest-level S-box layers and directly after the final lowest-level S-box layer.

The condition (i) is introduced so that both encryption and decryption have the same structure [9, 3, 4]. The condition (ii) is derived from the fact that the branch number is assured by two consecutive rounds.

We chose following parameters for Hierocrypt–3.

(a) The lowest-level S-box is 8-bit

(b) Data randomizing part consists of the 2-level SPN structure (the higher-level and the lower-level SPN)

(c) The lower-level structure is 2-round SPN

(d) The diffusion layer sizes are four times as much as that of S-box in both levels

These conditions were determined by the following reasons. (a)8-bit is the upper-bound of S-box size where a table-lookup implementation is realistic. (b) SP networks with more than two layers are not efficient in calculational cost. (c) The minimum SP network where the active S-box number can be assured efficiently is two-round SPN (SPS). (d) The number of parallel S-boxes should be small to save the calculational cost.

1.1.2 Simplicity and flexibility of the nested SPN structure

The nested SPN structure is very simple and highly transparent. Its S-boxes and diffusion layers in each level can be designed independent to some extent, and it is easy to be flexible over the change of block cipher size.

1.2 Key scheduling part

In designing the key scheduling part, it is needed to avoid the decrease of effective key length because of a simple relations between extended keys as for the security. At the same time, we impose a condition that the calculational time for round key is shorter than that for data randomization, so that the initial delay is sufficiently short.

The key scheduling part consists of the intermediate key update part and the round key generation part, both parts have iterative round structures. As Hierocrypt–3 supports a key up to 256-bit length, intermediate key update part is designed so that its block size is 256-bit and its round function is bijective.

An SP network of 256-bit block is expensive in calculational cost. As the key scheduling part should operates faster than the data randomization part, we adopt the following constitution.

The encryption key is initially expanded to 256-bit by an appropriate padding for a 128- or 192-bit case. The 256-bit key data divides into two 128-bit halves. One half is iteratively updated by a bijective linear transformation. The other half is a 128-bit Feistel network where the "round key" is supplied from the former one.

The intermediate key update part for Hierocrypt–3 takes a round-trip type, where the update operations are reversed around the center of round structure. We call the rounds before the turning-back as plaintext side, and the rounds after the turning-back as ciphertext side. The round-trip structure gives a short initial delay even in the on-the-fly decryption.

The round key for the data randomizing part of Hierocrypt–3 is generated by concatenating four 64-bit data which are linear combinations of intermediate key bits. The linear combinations should be appropriately chosen so that weak keys do not appear, that is, there are no simple relations between the round keys.

Round-dependent constants are added to the linear transformation part of intermediate key update part, in order to avoid a periodical pattern to appear. Here the constants are made from the square roots of the small integers.

2 Design criteria

2.1 Items of design criteria

2.1.1 Security

The most fundamental security criterion is the key length, which express the security against the exhaustive key search (or the brute force search). Besides the key length, the security measures against the following attacks are important.

(1a) Differential cryptanalysis;

(1b) Linear cryptanalysis;

(1c) higher-order differential cryptanalysis;

(1d) Interpolation attack;

(1e) SQUARE-dedicated attack;

(1f) truncated differential attack;

(1g) impossible differential attack.

2.1.2 Performance

As for the performance in encryption and decryption, the following items are important.

(2a) speed of the data randomizing part;

(2b) key establishment time;

(2c) speed of the on-the-fly key scheduling.

2.1.3 Implementation efficiency

The following items are important for the implementation efficiency.

(3a) the object code is short;

(3b) required RAM is small;

(3c) required ROM is small.

2.2 Design of the components

2.2.1 s (lower-level S-box)

The S-box is the only nonlinear component. In designing the S-box, we consider the following points are most important.

(i) the maximum differential/linear probability;

(ii) algebraic order;

(iii) the number of terms in polynomial expressions;

(iv) nonexistence of a simple algebraic structure.

At first, we impose the S-box to take the minimum values for differential and linear probabilities: The condition can be satisfied by appropriate power functions over $\mathrm{GF}(2^8)$. But, if the power function is used as the S-box, the algorithm may be very weak against algebraic attacks such as the higher-order differential attack or the interpolation attack. Therefore, we insert bijective linear transformations both before input and after output of the power function.

$$s(x_{(8)}) = Add(Power(Perm(x_{(8)}))) ,$$
$$y_{(8)} = Perm(x_{(8)}) , \quad y_{i(1)} = x_{\pi[i](1)} ,$$

i	1	2	3	4	5	6	7	8
$\pi[i]$	3	7	5	8	6	2	4	1

$$Power : \mathrm{GF}(2^8) \rightarrow \mathrm{GF}(2^8) ,$$
$$Power\left(x_{(8)}\right) = {x_{(8)}}^{247} ,$$
$$Add\left(x_{(8)}\right) = x_{(8)} \oplus \mathtt{0x7} .$$

The power index 247 is chosen because it has the maximum value among the indices where the maximum differential/linear probability is 2^{-6}. The primitive polynomial for $\mathrm{GF}(2^8)$ is $z^8 + z^6 + z^5 + z + 1$.

The additional constant `0x7` has been chosen because the input-output Hamming weight distribution is near to that of the random function (the correlation coefficient is 0.09375).

2.2.2 mds_{L} (lower-level diffusion)

The design criteria for mds_{L} is as follows.

(i) maximum distance separable (MDS) map;

(ii) circulant;

(iii) the number of terms in a polynomial expression (bitwise) is maximum.

The criterion (i) assures that no less than 5 lower-level S-boxes s are active in the active higher-level S-box xs containing 8 lower-level S-boxes.

The criterion (ii) is for small implementation.

The criterion (iii) is for a high security against algebraic attacks such as the higher-order differential cryptanalysis and the interpolation attack. We consider

a combination function of the S-box and mds_{L}-function connecting to it, which has an 8-bit input and a 32-bit output. We chose the combination function whose weighted term sum is maximum under the constraint (i) and (ii). In the weighted term sum of function, a term of order n is counted as n.

The lower-level diffusion mds_{L} satisfying the criteria (i)$\sim$(iii) is determined by the following procedure.

1. Make a circulant matrix with elements which satisfy the criterion (iii).
2. If the matrix is MDS, go to the following clause. Otherwise, return the preceding clause.
3. Calculate the total sum of weighted term sums for matrix elements for the matrix and its inverse. And add both total sums. If the result of the preceding clause is larger than the maximal value before, make it the new maximal value.
4. Return to the first clause.
5. Choose the one with the lowest calculational cost, from the upper candidates chosen by the term sum.

As the result of the above procedure, the following mds_{L} is obtained.

$$\begin{pmatrix} \mathtt{C4} & \mathtt{65} & \mathtt{C8} & \mathtt{8B} \\ \mathtt{8B} & \mathtt{C4} & \mathtt{65} & \mathtt{C8} \\ \mathtt{C8} & \mathtt{8B} & \mathtt{C4} & \mathtt{65} \\ \mathtt{65} & \mathtt{C8} & \mathtt{8B} & \mathtt{C4} \end{pmatrix}$$

Here, the matrix elements are expressed in hexadecimal.

2.2.3 MDS_{H} (higher-level diffusion)

In designing MDS_{H}, we consider the following items are important.

(i) MDS matrix

(ii) circulant

(iii) byte-wise multiple-path property

(iv) the number of connections between bytes should be as few as possible.

The condition (i) indicates that the matrix is an MDS map for four 32-bit word. Here, let $MDS(m, n)$ be an MDS map for m parallel n-bit words. As Proposition 2 of references [6, 5] shows, $MDS(32, 4)$ consists of eight parallel $MDS(4, 4)$. When all $MDS(4, 4)$ are the same, MDS_{H} is nothing but the combination of byte-wise XOR's and is expressed as 16×16 matrix.

The SQUARE-dedicated attack is efficient against reduced versions of the major SPN-type ciphers SQUARE and Rijndael. The attack is applicable to up to 6-round. We let multiple-path property be a property that there are no less than two byte-wise connections between any pair of higher-level S-boxes(xs) in two consecutive rounds.

The condition (iii) requires that MDS_{H} satisfies the multiple-path property for both forward and backward directions. The necessary and sufficient condition for multiple-path property of MDS_{H} is that all elements of $MDS(4, 4)$ are elements of { 3, 5, 6, 7, A, B, C, E }.

The condition (iv) is required for a small implementation in hardware.

The higher-level diffusion MDS_{H} satisfying the criteria (i)$\sim$(iv) is determined by the following procedure.

1. Make a circulant matrix by components of $\mathrm{GF}(2^4)$ which satisfy the multiple-path property.
2. If the matrix is MDS, go to the following clause. Otherwise, return the preceding clause.
3. Calculate the inverse matrix. If all elements satisfy the multiple-path property, go to the following clause. Otherwise, return to the preceding clause.
4. Let the matrix be a candidate for MDS_{H}.

Four candidates are obtained except for the freedom degrees of rotation and reverse by the above procedure. Considering hardware implementation, we chose the following one which has the fewest connections.

$$\begin{pmatrix} \mathsf{5} & \mathsf{5} & \mathsf{A} & \mathsf{E} \\ \mathsf{E} & \mathsf{5} & \mathsf{5} & \mathsf{A} \\ \mathsf{A} & \mathsf{E} & \mathsf{5} & \mathsf{5} \\ \mathsf{5} & \mathsf{A} & \mathsf{E} & \mathsf{5} \end{pmatrix}$$

Here, the matrix elements expressed in hexadecimal are the elements $\mathrm{GF}(2^4)$, where the primitive polynomial for $\mathrm{GF}(2^8)$ is $z^8 + z^6 + z^5 + z + 1$.

Inverse of this matrix is

$$\begin{pmatrix} \mathsf{B} & \mathsf{E} & \mathsf{E} & \mathsf{6} \\ \mathsf{6} & \mathsf{B} & \mathsf{E} & \mathsf{E} \\ \mathsf{E} & \mathsf{6} & \mathsf{B} & \mathsf{E} \\ \mathsf{E} & \mathsf{E} & \mathsf{6} & \mathsf{B} \end{pmatrix} .$$

2.2.4 $P^{(n)}$ (permutation for key scheduling)

$P^{(n)}$ is used as a diffusion layer of the key scheduling part. We impose the following conditions.

(i) The calculational time per round is shorter than that for the data randomizing part.

(ii) It has the scalability for the change of input-output data size.

(iii) It is highly diffusive (to make it difficult to infer the encryption key from the extended key).

(iv) It is a bijective function (to prevent the key degree of freedom from degenerating).

(v) MDS property is not required (we considered the tradeoff between speed and security).

By the above conditions, we chose the following linear transformation for $(4n)$-bit data.

$$Y_{(4n)} = P^{(n)}\left(X_{(4n)}\right) .$$

$$\begin{pmatrix} y_{1(n)} \\ y_{2(n)} \\ y_{3(n)} \\ y_{4(n)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{1(n)} \\ x_{2(n)} \\ x_{3(n)} \\ x_{4(n)} \end{pmatrix} .$$

This linear transformation is written as the combination of two involution-type linear transformations.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$$

3 Algorithm of Hierocrypt–3

3.1 Notations

An n-bit value is basically expressed with the subscript "(n)". For example, the value $X_{(n)}$ is an element of $\mathrm{GF}(2)^n$. A value expressed by a capital(s) describes an element of no less than 16 bits A value expressed by a small letter(s) describes an element of less than 16 bits.

We adopt the big endian convention. When a value $X_{(mn)}$ is expressed as a concatenation of m pieces of n-bit length, each piece is expressed with the subscript $i(n)$ $(i = 1, 2, \ldots, m)$, that is $X_{(mn)} = X_{1(n)} \| X_{2(n)} \| \cdots \| X_{m(n)}$. Furthermore, $X_{(mn)}$ and $X_{i(n)}$ are expressed as the following concatenations: $X_{(mn)} = x_{1(1)} \| x_{2(1)} \| \cdots \| x_{mn(1)}$, $X_{i(n)} = x_{ni-n+1(1)} \| x_{ni-n+2(1)} \| \cdots \| x_{ni-n+n(1)}$.

The following shows an example of concatenation expression of a 128-bit value $X_{(128)}$.

$$
\begin{aligned}
&X_{(128)} = X_{1(32)} \| X_{2(32)} \| X_{3(32)} \| X_{4(32)} \ , \\
&X_{i(32)} = x_{4i-4+1(8)} \| x_{4i-4+2(8)} \| \cdots \| x_{4i-4+4(8)} \ , \quad i = 1, 2, \ldots, 4, \\
&X_{j(8)} = x_{8j-8+1(1)} \| x_{8j-8+2(1)} \| \cdots \| x_{8j-8+8(1)} \ , \quad j = 1, 2, \ldots, 16.
\end{aligned}
$$

Note that the LSB of the value $X_{i(n)}$ (i=1,2,...,m) is $x_{in(1)}$, which is the in-th MSB of $X_{(mn)}$.

3.2 Structure of the algorithm

The structures of data randomization part and the key scheduling part are described in this section. Fundamental operations used there are described in the next section.

3.2.1 Encryption

The T-round encryption of Hierocrypt–3 consists of $(T-1)$ operations of round function ρ, an operation of XS-function, and the final key addition (AK) .

$$
P_{(128)} \equiv X^{(0)}_{(128)} \overset{\rho}{\longmapsto} X^{(1)}_{(128)} \overset{\rho}{\longmapsto} \cdots \overset{\rho}{\longmapsto} X^{(T-1)}_{(128)} \overset{XS}{\longmapsto} X^{(T)}_{(128)} \overset{AK}{\longmapsto} C_{(128)}
$$

T is 6,7, or 8 for 128-, 192-, or 256-bit, respectively.

The 128-bit value $X^{(i)}_{(128)}$ is the output of the i-th operation of round function ρ $(i = 1, 2, \ldots, T-1)$. The plaintext $P_{(128)}$ is assigned to the 0-th value $X^{(0)}_{(128)}$.

The value $X^{(t)}_{(128)}$ is the output of the t-th operation of ρ-function for the input $X^{(t-1)}_{(128)}$ and the round key $K^{(t)}_{(256)}$.

$$
X^{(t)}_{(128)} = \rho(X^{(t-1)}_{(128)}, K^{(t)}_{(256)}), \quad t = 1, 2, \cdots, T-1.
$$

Similarly, $X^{(T)}_{(128)}$ is the output of XS-function for the input $X^{(T-1)}_{(128)}$ and the final key $K^{(T)}_{(256)}$.

$$
X^{(T)}_{(128)} = XS(X^{(T-1)}_{(128)}, K^{(T)}_{(256)}) \ .
$$

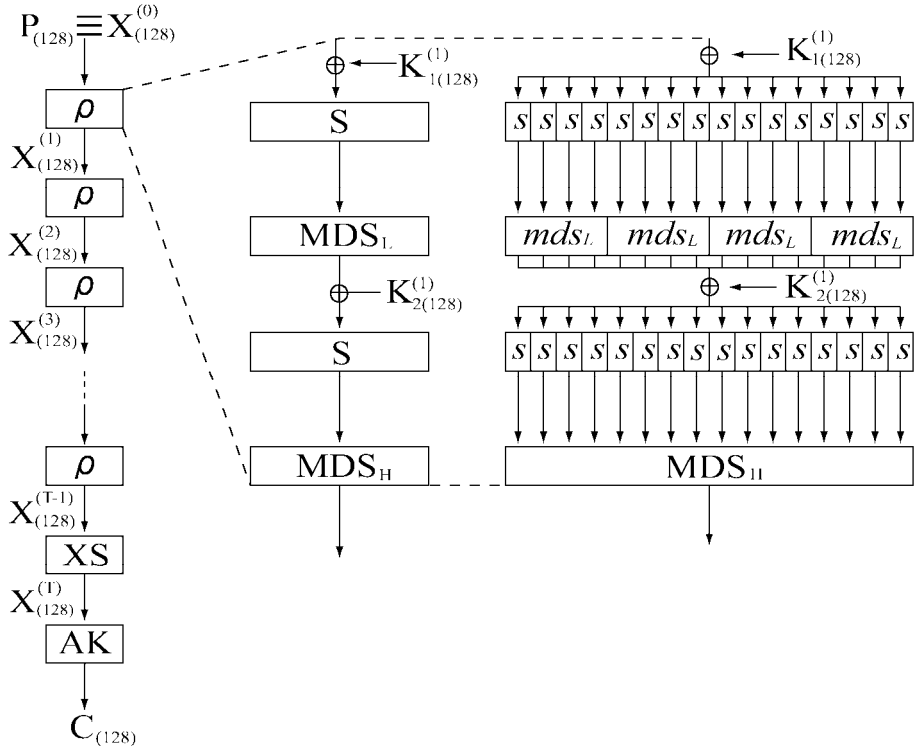


Figure 1: Structure of the algorithm

The ciphertext $C_{(128)}$ is given as the addition (XOR, exclusive or) between the T-th round output $X^{(T)}_{(128)}$ and the first half of the final key $K^{(T+1)}_{1(128)}$

$$C_{(128)} = X^{(T)}_{(128)} \oplus (K^{(T+1)}_{1(64)} \| K^{(T+1)}_{2(64)}) \ .$$

3.2.2 Decryption

The decryption of Hierocrypt–3 is the inverse of encryption, and consists of the final key addition, the inverse of XS-function (XS^{-1}), and $(T-1)$ inverse operations of round function (ρ^{-1}).

$$\begin{aligned}
&X^{(T)}_{(128)} = C_{(128)} \oplus (K^{(T+1)}_{1(64)} \| K^{(T+1)}_{2(64)}) \ , \\
&X^{(T-1)}_{(128)} = XS^{-1}(X^{(T)}_{(128)}, K^{(T)}_{(256)}) \ , \\
&X^{(t-1)}_{(128)} = \rho^{-1}(X^{(t)}_{(128)}, K^{(t)}_{(256)}) \ , \quad t = T-1, \ldots, 2, 1.
\end{aligned}$$

The plaintext $P_{(128)}$ is given as the final output $X^{(0)}_{(128)}$.

$$P_{(128)} = X^{(0)}_{(128)} \ .$$

3.2.3 Key scheduling

The main part of key scheduling consists of the intermediate key generation part and the round key generation part, preceded by the intermediate key initialization. The intermediate key part recursively generates intermediate key outputs $Z_{(256)}^{(t)}$ $(t = 1, 2, \ldots, T+1)$, and the round key generation part generates round keys $K_{(256)}^{(t)}$ $(t = 1, 2, \ldots, T+1)$ from the corresponding intermediate keys, as follows.

$$\begin{aligned} K_{(\mathsf{length})} &\overset{\mathsf{padding}}{\longmapsto} Z_{(256)}^{(-1)} \overset{\sigma_0}{\longmapsto} Z_{(256)}^{(0)} \overset{\sigma}{\longmapsto} Z_{(256)}^{(1)} \\ &\overset{\sigma}{\longmapsto} Z_{(256)}^{(2)} \overset{\sigma}{\longmapsto} \cdots \overset{\sigma}{\longmapsto} Z_{(256)}^{(t_{\mathsf{turn}})} \\ &\overset{\sigma^{-1}}{\longmapsto} Z_{(256)}^{(t_{\mathsf{turn}}+1)} \overset{\sigma^{-1}}{\longmapsto} \cdots \overset{\sigma^{-1}}{\longmapsto} Z_{(256)}^{(T+1)} \end{aligned}$$

The intermediate key $Z_{(128)}^{(t)}$ and the round key $K_{(128)}^{(t)}$ are divided into 4 pieces. $Z_{(128)}^{(t)}$ and $K_{(128)}^{(t)}$ are divided into 4 pieces.

$$\begin{aligned} Z_{(256)}^{(t)} &= Z_{1(64)}^{(t)} \| Z_{2(64)}^{(t)} \| Z_{3(64)}^{(t)} \| Z_{4(64)}^{(t)} \ , \\ K_{(256)}^{(t)} &= K_{1(64)}^{(t)} \| K_{2(64)}^{(t)} \| K_{3(64)}^{(t)} \| K_{4(64)}^{(t)} \ . \end{aligned}$$

To generate the intermediate keys, the σ-function is used for $1 \le t \le t_{\mathsf{turn}}$, and the σ^{-1}-function is used for $t_{\mathsf{turn}} + 1 \le t \le T + 1$, where $t_{\mathsf{turn}} = 4$ for the 128/192-bit key and where $t_{\mathsf{turn}} = 5$ for the 256-bit key. Under the recursion rule, the intermediate key values are symmetric with regard to the point $t = t_{\mathsf{turn}}$.

$$Z_{(256)}^{(t)} = Z_{(256)}^{(2t_{\mathsf{turn}}-t)} \ , \quad t_{\mathsf{turn}} + 1 \le t \le T + 1 \ .$$

3.2.4 Round-dependent constants

To prevent periodic patterns from appearing in the intermediate key generation, and to improve resistance against the related key attack, we introduce round-dependent constants additions to the intermediate key generation part. The round-dependent constants have been made by combining two from the four 32-bit values which are given as binary expansions of irrational numbers.

$$\begin{aligned} H_0 &= \mathtt{0x5A827999} = trunc(\sqrt{2}/4), \\ H_1 &= \mathtt{0x6ED9EBA1} = trunc(\sqrt{3}/4), \\ H_2 &= \mathtt{0x8F1BBCDC} = trunc(\sqrt{5}/4), \\ H_3 &= \mathtt{0xCA62C1D6} = trunc(\sqrt{10}/4), \end{aligned}$$

Where, $trunc(x) = \lfloor 2^{32} x \rfloor$.

$$\begin{aligned} G_0(0) &= H_3 \| H_0 \ , \quad G_0(1) = H_2 \| H_1 \ , \\ G_0(2) &= H_1 \| H_3 \ , \quad G_0(3) = H_0 \| H_2 \ , \\ G_0(4) &= H_2 \| H_3 \ , \quad G_0(5) = H_1 \| H_0 \ . \end{aligned}$$

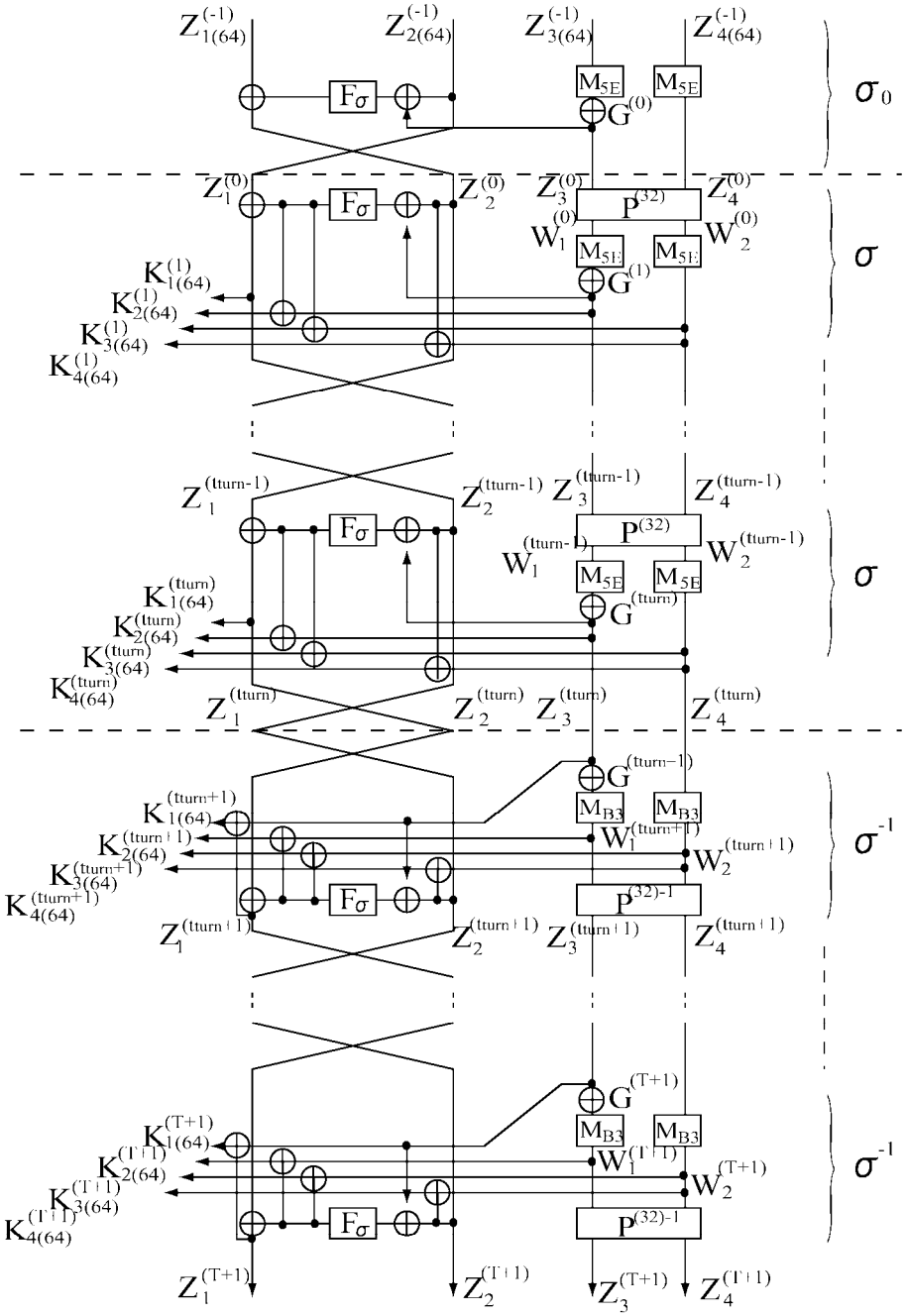


Figure 2: Key scheduling

3.2.5 Preprocessing

The intermediate key $Z^{(-1)}_{(256)}$ is made of the encryption key $K_{(\mathsf{length})}$ (length = 128, 192, 256) with an initial operation. The intermediate key $Z^{(0)}_{(256)}$ is derived from $Z^{(-1)}_{(256)}$ through the pre-whitening operation σ_0. The padding operation is done when length = 192, 256 where padded values are concatenations of the above mentioned 32-bit constants H_i. The padding operations are described as follows.

[128-bit key]

$$K_{1(64)} \| K_{2(64)} = K_{(128)} \ ,$$
$$Z^{(-1)}_{1(64)} = K_{1(64)} \ , \quad Z^{(-1)}_{2(64)} = K_{2(64)} \ , \quad Z^{(-1)}_{3(64)} = K_{1(64)} \ , \quad Z^{(-1)}_{4(64)} = H_3 \| H_2 \ .$$

[192-bit key]

$$K_{1(64)} \| K_{2(64)} \| K_{3(64)} = K_{(192)} \ ,$$
$$Z^{(-1)}_{1(64)} = K_{1(64)} \ , \quad Z^{(-1)}_{2(64)} = K_{2(64)} \ , \quad Z^{(-1)}_{3(64)} = K_{3(64)} \ , \quad Z^{(-1)}_{4(64)} = H_2 \| H_3 \ .$$

[256-bit key]

$$K_{1(64)} \| K_{2(64)} \| K_{3(64)} \| K_{4(64)} = K_{(256)} \ ,$$
$$Z^{(-1)}_{1(64)} = K_{1(64)} \ , \quad Z^{(-1)}_{2(64)} = K_{2(64)} \ , \quad Z^{(-1)}_{3(64)} = K_{3(64)} \ , \quad Z^{(-1)}_{4(64)} = K_{4(64)} \ .$$

[Pre-whitening] (σ_0-function)
The pre-whitening is done, before iterative operation by the σ-function. The pre-whitening operation σ_0 is made from σ by removing $P^{(32)}$.

interface $Z^{(0)}_{(256)} = \sigma_0(Z^{(-1)}_{(256)}, G^{(0)}_{(64)})$
definition
$$Z^{(0)}_{3(64)} = M_{5E}(Z^{(-1)}_{3(64)}) \oplus G^{(0)}_{(64)} \ ,$$
$$Z^{(0)}_{4(64)} = M_{5E}(Z^{(-1)}_{4(64)}) \ ,$$
$$Z^{(0)}_{1(64)} = Z^{(-1)}_{2(64)} \ ,$$
$$Z^{(0)}_{2(64)} = Z^{(-1)}_{1(64)} \oplus F_\sigma(Z^{(-1)}_{2(64)} \oplus Z^{(0)}_{3(64)}) \ .$$

As the round-dependent constant $G^{(0)}_{(64)}$, the following 64-bit concatenated value is used.

$$G^{(0)}_{(64)} = G_0(5) = H_1 \| H_0 \ .$$

3.2.6 Intermediate key update (σ-function)

The intermediate key $Z^{(t)}_{(256)}$ is generated by the operation σ up to $t = t_{\mathsf{turn}}$, and afterwards by the inverse operation σ^{-1}. The sequence of intermediate

Table 1: Key schedule for 128-bit key (6 rounds)

round key	t	operation	$G^{(t)}_{(64)}$
–	-1 (PAD)	–	$H_3 \| H_2$
–	0 (PW)	σ_0	$G_0(5)$
$K^{(1)}_{(256)}$	1	σ	$G_0(0)$
$K^{(2)}_{(256)}$	2	σ	$G_0(1)$
$K^{(3)}_{(256)}$	3	σ	$G_0(2)$
$K^{(4)}_{(256)}$	4	σ	$G_0(3)$
$K^{(5)}_{(256)}$	5	σ^{-1}	$G_0(3)$
$K^{(6)}_{(256)}$	6	σ^{-1}	$G_0(2)$
$K^{(7)}_{(256)}$	7	σ^{-1}	$G_0(1)$

Table 2: Key schedule for 192-bit key (7 rounds)

round key	t	operation	$G^{(t)}_{(64)}$
–	-1 (PAD)	–	$H_2 \| H_3$
–	0 (PW)	σ_0	$G_0(5)$
$K^{(1)}_{(256)}$	1	σ	$G_0(1)$
$K^{(2)}_{(256)}$	2	σ	$G_0(0)$
$K^{(3)}_{(256)}$	3	σ	$G_0(3)$
$K^{(4)}_{(256)}$	4	σ	$G_0(2)$
$K^{(5)}_{(256)}$	5	σ^{-1}	$G_0(2)$
$K^{(6)}_{(256)}$	6	σ^{-1}	$G_0(3)$
$K^{(7)}_{(256)}$	7	σ^{-1}	$G_0(0)$
$K^{(8)}_{(256)}$	8	σ^{-1}	$G_0(1)$

keys is symmetric with respect to the point $t = t_{\mathsf{turn}}$ for this round-trip-type scheduling.

$$Z^{(t)}_{(256)} = Z^{(2t_{\mathsf{turn}}-t)}_{(256)} \ , \quad t_{\mathsf{turn}} \le t \le T+1 \ .$$

We call the region: $(1 \le t \le t_{\mathsf{turn}})$ as the plaintext side, and the other region: $(t_{\mathsf{turn}}+1 \le t \le T+1)$ as the ciphertext side, corresponding to the position in the data randomizing part.

Table 3: Key schedule for 256-bit key (8 rounds)

round key	t	operation	$G^{(t)}_{(64)}$
–	-1 (PAD)	–	–
–	0 (PW)	σ_0	$G_0(5)$
$K^{(1)}_{(256)}$	1	σ	$G_0(4)$
$K^{(2)}_{(256)}$	2	σ	$G_0(0)$
$K^{(3)}_{(256)}$	3	σ	$G_0(2)$
$K^{(4)}_{(256)}$	4	σ	$G_0(1)$
$K^{(5)}_{(256)}$	5	σ	$G_0(3)$
$K^{(6)}_{(256)}$	6	σ^{-1}	$G_0(3)$
$K^{(7)}_{(256)}$	7	σ^{-1}	$G_0(1)$
$K^{(8)}_{(256)}$	8	σ^{-1}	$G_0(2)$
$K^{(9)}_{(256)}$	9	σ^{-1}	$G_0(0)$

[Iterative update of intermediate key(plaintext side)] $(1 \leq t \leq t_{\mathsf{turn}})$

interface $Z^{(t)}_{(256)} = \sigma(Z^{(t-1)}_{(256)}\ ,\ \ G^{(t)}_{(64)})$
definition
$W^{(t-1)}_{1(64)} \| W^{(t-1)}_{2(64)} = P^{(32)}(Z^{(t-1)}_{3(64)} \| Z^{(t-1)}_{4(64)})$,
$Z^{(t)}_{3(64)} = M_{5E}(W^{(t-1)}_{1(64)}) \oplus G^{(t)}_{(64)}$,
$Z^{(t)}_{4(64)} = M_{5E}(W^{(t-1)}_{2(64)})$,
$Z^{(t)}_{1(64)} = Z^{(t-1)}_{2(64)}$,
$Z^{(t)}_{2(64)} = Z^{(t-1)}_{1(64)} \oplus F_\sigma(Z^{(t-1)}_{2(64)} \oplus Z^{(t)}_{3(64)})$.

[Iterative update of intermediate key(ciphertext side)] $(t_{\mathsf{turn}}+1 \leq t \leq T+1)$

interface $Z^{(t)}_{(256)} = \sigma^{-1}(Z^{(t-1)}_{(256)}, G^{(t)}_{(64)})$
definition
$Z^{(t)}_{1(64)} = Z^{(t-1)}_{2(64)} \oplus F_\sigma(Z^{(t-1)}_{1(64)} \oplus Z^{(t-1)}_{3(64)})$,
$Z^{(t)}_{2(64)} = Z^{(t-1)}_{1(64)}$,
$W^{(t)}_{1(64)} = M_{B3}(Z^{(t-1)}_{3(64)} \oplus G^{(t)}_{(64)})$,
$W^{(t)}_{2(64)} = M_{B3}(Z^{(t-1)}_{4(64)})$,
$Z^{(t)}_{3(64)} \| Z^{(t)}_{4(64)} = {P^{(32)}}^{-1}(W^{(t)}_{1(64)} \| W^{(t)}_{2(64)})$.

3.2.7 Round key generation

The different rules are applied to generate a round key from the corresponding intermediate key for the plaintext side and the ciphertext side.

[Round key generation(plaintext side)] $(1 \leq t \leq t_{\mathsf{turn}})$

$$
\begin{aligned}
V_{(64)}^{(t)} &= F_\sigma(Z_{2(64)}^{(t-1)} \oplus Z_{3(64)}^{(t)}) \ , \\
K_{1(64)}^{(t)} &= Z_{1(64)}^{(t-1)} \oplus V_{(64)}^{(t)} \ , \\
K_{2(64)}^{(t)} &= Z_{3(64)}^{(t)} \oplus V_{(64)}^{(t)} \ , \\
K_{3(64)}^{(t)} &= Z_{4(64)}^{(t)} \oplus V_{(64)}^{(t)} \ , \\
K_{4(64)}^{(t)} &= Z_{2(64)}^{(t-1)} \oplus Z_{4(64)}^{(t)} \ .
\end{aligned}
$$

[Round key generation(ciphertext side)] $(t_{\mathsf{turn}} + 1 \leq t \leq T + 1)$

$$
\begin{aligned}
V_{(64)}^{(t)} &= F_\sigma(Z_{1(64)}^{(t-1)} \oplus Z_{3(64)}^{(t-1)}) \ , \\
K_{1(64)}^{(t)} &= Z_{1(64)}^{(t)} \oplus Z_{3(64)}^{(t-1)} \ , \\
K_{2(64)}^{(t)} &= W_{1(64)}^{(t)} \oplus V_{(64)}^{(t)} \ , \\
K_{3(64)}^{(t)} &= W_{2(64)}^{(t)} \oplus V_{(64)}^{(t)} \ , \\
K_{4(64)}^{(t)} &= Z_{1(64)}^{(t-1)} \oplus W_{2(64)}^{(t)} \ .
\end{aligned}
$$

3.3 Fundamental operations

In this section, we explain fundamental operations using in the encryption algorithm explained in a previous section. Fig 3 summarize relation among fundamental operations.

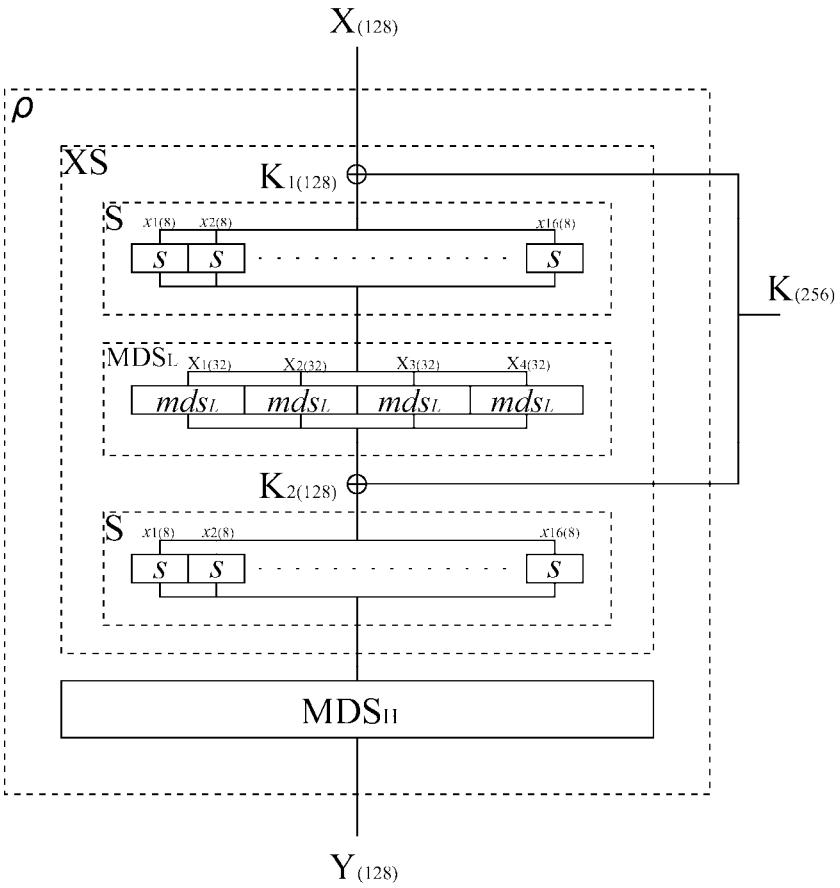


Figure 3: Summary of functions

3.3.1 Round function ρ

The ρ-function, which is the round function of the data randomization part, is a composite function of the XS-function and the MDS_{L}-function, where the input data are the 128-bit value $X_{(128)}$ and the 256-bit value $K_{(256)}$.

$$\rho(X_{(128)}, K_{(256)}) = MDS_{\mathsf{H}}(XS(X_{(128)}, K_{(256)})) \ .$$

3.3.2 XS-function

XS-function is a composite function of the S-function, 128-bit key addition, and MDS_{L}-function.

interface $Y_{(128)} = XS\left(X_{(128)}, K_{(256)}\right)$
definition
$K_{1(128)} \| K_{2(128)} = K_{(256)}$,
$XS\left(X_{(128)}, K_{(256)}\right) = S\left(MDS_{\mathsf{L}}\left(S\left(X_{(128)} \oplus K_{1(128)}\right)\right) \oplus K_{2(128)}\right)$.

3.3.3 S-function

The S-functions consists of sixteen operations of s-function for 8-bit subdata of the 128-bit input data.

interface $Y_{(128)} = S\left(X_{(128)}\right)$
definition
$x_{1(8)} \| x_{2(8)} \| \cdots \| x_{16(8)} = X_{(128)}$,
$Y_{(128)} = s(x_{1(8)}) \| s(x_{2(8)}) \| \cdots \| s(x_{16(8)})$.

3.3.4 s-function

The s-function is a nonlinear transformation for 8-bit input/output value, which is given as the following table where all numbers are represented in hexadecimal.

interface $y_{(8)} = s(x_{(8)})$
definition
$(s(\mathtt{00})\ s(\mathtt{01})\ s(\mathtt{02})\ \ldots\ s(\mathtt{0F})\ s(\mathtt{10})\ s(\mathtt{11})\ \ldots\ s(\mathtt{FF})) =$

```
( 07 FC 55 70 98 8E 84 4E BC 75 CE 18 02 E9 5D 80
  1C 60 78 42 9D 2E F5 E8 C6 7A 2F A4 B2 5F 19 87
  0B 9B 9C D3 C3 77 3D 6F B9 2D 4D F7 8C A7 AC 17
  3C 5A 41 C9 29 ED DE 27 69 30 72 A8 95 3E F9 D8
  21 8B 44 D7 11 0D 48 FD 6A 01 57 E5 BD 85 EC 1E
  37 9F B5 9A 7C 09 F1 B1 94 81 82 08 FB C0 51 0F
  61 7F 1A 56 96 13 C1 67 99 03 5E B6 CA FA 9E DF
  D6 83 CC A2 12 23 B7 65 D0 39 7D 3B D5 B0 AF 1F
  06 C8 34 C5 1B 79 4B 66 BF 88 4A C4 EF 58 3F 0A
  2C 73 D1 F8 6B E6 20 B8 22 43 B3 33 E7 F0 71 7E
  52 89 47 63 0E 6D E3 BE 59 64 EE F6 38 5C F4 5B
  49 D4 E0 F3 BB 54 26 2B 00 86 90 FF FE A6 7B 05
  AD 68 A1 10 EB C7 E2 F2 46 8A 6C 14 6E CF 35 45
  50 D2 92 74 93 E1 DA AE A9 53 E4 40 CD BA 97 A3
  91 31 25 76 36 32 28 3A 24 4C DB D9 8D DC 62 2A
  EA 15 DD C2 A5 0C 04 1D 8F CB B4 4F 16 AB AA A0 ) .
```

3.3.5 MDS_{L}-function

The MDS_{L}-function consists of four operations of mds_{L}-function for 32-bit subdata of the 128-bit input data.

interface $Y_{(128)} = MDS_{\mathsf{L}}\left(X_{(128)}\right)$
definition
$X_{1(32)} \| X_{2(32)} \| X_{3(32)} \| X_{4(32)} = X_{(128)}$,
$Y_{(128)} = mds_{\mathsf{L}}(X_{1(32)}) \| mds_{\mathsf{L}}(X_{2(32)}) \| mds_{\mathsf{L}}(X_{3(32)}) \| mds_{\mathsf{L}}(X_{4(32)})$.

3.3.6 mds_{L}-function

The mds_{L}-function is a linear transformation which is represented by 4×4 matrix multiplication where all matrix and vector elements are regarded as elements of $\mathrm{GF}(2^8)$.

interface $Y_{(32)} = mds_{\mathsf{L}}(X_{(32)})$
definition
$x_{1(8)} \| x_{2(8)} \| x_{3(8)} \| x_{4(8)} = X_{(32)}$,
$Y_{(32)} = y_{1(8)} \| y_{2(8)} \| y_{3(8)} \| y_{4(8)}$,

$$\begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \end{pmatrix} = \begin{pmatrix} \mathtt{C4} & \mathtt{65} & \mathtt{C8} & \mathtt{8B} \\ \mathtt{8B} & \mathtt{C4} & \mathtt{65} & \mathtt{C8} \\ \mathtt{C8} & \mathtt{8B} & \mathtt{C4} & \mathtt{65} \\ \mathtt{65} & \mathtt{C8} & \mathtt{8B} & \mathtt{C4} \end{pmatrix} \begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \end{pmatrix} .$$

The inverse function of mds_{L}-function, mds_{L}^{-1}-function is given by following definition.

interface $X_{(32)} = mds_{\mathsf{L}}^{-1}(Y_{(32)})$
definition
$y_{1(8)} \| y_{2(8)} \| y_{3(8)} \| y_{4(8)} = Y_{(32)}$,
$X_{(32)} = x_{1(8)} \| x_{2(8)} \| x_{3(8)} \| x_{4(8)}$,

$$\begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \end{pmatrix} = \begin{pmatrix} \mathtt{82} & \mathtt{C4} & \mathtt{34} & \mathtt{F6} \\ \mathtt{F6} & \mathtt{82} & \mathtt{C4} & \mathtt{34} \\ \mathtt{34} & \mathtt{F6} & \mathtt{82} & \mathtt{C4} \\ \mathtt{C4} & \mathtt{34} & \mathtt{F6} & \mathtt{82} \end{pmatrix} \begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \end{pmatrix} .$$

Here, 8-bit data $x_{(8)}$ and the matrix element a (in hexadecimal) are regarded as elements of $\mathrm{GF}(2^8)$ related as follows.

$$x_{(8)} \Leftrightarrow \sum_{i=1}^{8} x_{i(1)} z^{8-i} ,$$
$$a = \sum_{i=0}^{7} a_i 2^i \Leftrightarrow \sum_{i=0}^{7} a_i z^i .$$

The following polynomial $p(x)$ is used as the primitive polynomial for the Galois field $\mathrm{GF}(2^8)$.

$$p(z) = z^8 + z^6 + z^5 + z + 1 .$$

3.3.7 MDS_{H}-function

The MDS_{H}-function is a linear transformation consisting of exclusive or's between 8-bit subdata $x_{i(8)}$ ($\in \mathrm{GF}(2)^8$; $i = 1, 2, \ldots, 16$), which is represented by the following matrix form.

interface $Y_{(128)} = MDS_{\mathsf{H}}(X_{(128)})$
definition
$x_{1(8)} \| x_{2(8)} \| \cdots \| x_{16(8)} = X_{(128)}$,
$Y_{(128)} = y_{1(8)} \| y_{2(8)} \| \cdots \| y_{16(8)}$,

$$\begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \\ y_{5(8)} \\ y_{6(8)} \\ y_{7(8)} \\ y_{8(8)} \\ y_{9(8)} \\ y_{10(8)} \\ y_{11(8)} \\ y_{12(8)} \\ y_{13(8)} \\ y_{14(8)} \\ y_{15(8)} \\ y_{16(8)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \\ x_{5(8)} \\ x_{6(8)} \\ x_{7(8)} \\ x_{8(8)} \\ x_{9(8)} \\ x_{10(8)} \\ x_{11(8)} \\ x_{12(8)} \\ x_{13(8)} \\ x_{14(8)} \\ x_{15(8)} \\ x_{16(8)} \end{pmatrix}.$$

The inverse function of MDS_{H}-function, MDS_{H}^{-1}-function is given by following definition.

interface $X_{(128)} = MDS_{\mathsf{H}}^{-1}(Y_{(128)})$
definition
$y_{1(8)} \| y_{2(8)} \| \cdots \| y_{16(8)} = Y_{(128)}$,
$X_{(128)} = x_{1(8)} \| x_{2(8)} \| \cdots \| x_{16(8)}$,

$$\begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \\ x_{5(8)} \\ x_{6(8)} \\ x_{7(8)} \\ x_{8(8)} \\ x_{9(8)} \\ x_{10(8)} \\ x_{11(8)} \\ x_{12(8)} \\ x_{13(8)} \\ x_{14(8)} \\ x_{15(8)} \\ x_{16(8)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \\ y_{5(8)} \\ y_{6(8)} \\ y_{7(8)} \\ y_{8(8)} \\ y_{9(8)} \\ y_{10(8)} \\ y_{11(8)} \\ y_{12(8)} \\ y_{13(8)} \\ y_{14(8)} \\ y_{15(8)} \\ y_{16(8)} \end{pmatrix}.$$

3.3.8 $P^{(n)}$-function

The $P^{(n)}$ function consists of the a linear transformation for the input $X_{(4n)}$ which is a concatenation of four n-bit values $x_{i(n)}$ $(i = 1, 2, 3, 4)$ where each

element is regarded as an element of $\mathrm{GF}(2)^n$.

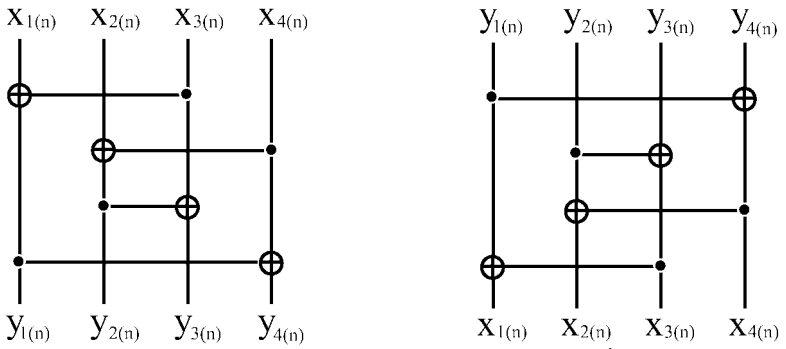
interface $Y_{(4n)} = P^{(n)}\left(X_{(4n)}\right)$
definition
$x_{1(n)} \| x_{2(n)} \| x_{3(n)} \| x_{4(n)} = X_{(4n)}$,
$Y_{(4n)} = y_{1(n)} \| y_{2(n)} \| y_{3(n)} \| y_{4(n)}$,

$$\begin{pmatrix} y_{1(n)} \\ y_{2(n)} \\ y_{3(n)} \\ y_{4(n)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{1(n)} \\ x_{2(n)} \\ x_{3(n)} \\ x_{4(n)} \end{pmatrix} .$$

The inverse function of $P^{(n)}$, ${P^{(n)}}^{-1}$, is given by the following equation.

interface $X_{(4n)} = {P^{(n)}}^{-1}\left(Y_{(4n)}\right)$
definition
$y_{1(n)} \| y_{2(n)} \| y_{3(n)} \| y_{4(n)} = Y_{(4n)}$,
$X_{(4n)} = x_{1(n)} \| x_{2(n)} \| x_{3(n)} \| x_{4(n)}$,

$$\begin{pmatrix} x_{1(n)} \\ x_{2(n)} \\ x_{3(n)} \\ x_{4(n)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_{1(n)} \\ y_{2(n)} \\ y_{3(n)} \\ y_{4(n)} \end{pmatrix} .$$



$P^{(n)}$-function ${P^{(n)}}^{-1}$-function

Figure 4: $P^{(n)}$-function and ${P^{(n)}}^{-1}$-function

3.3.9 M_{5E}-function

The M_{5E}-function consists of a concatenation of two 32-bit linear transformations, where each 8-bit subdata is regarded as an element of $\mathrm{GF}(2)^8$.

interface $Y_{(64)} = M_{5E}\left(X_{(64)}\right)$
definition
$x_{1(8)} \| x_{2(8)} \| \cdots \| x_{8(8)} = X_{(64)}$,
$Y_{(64)} = y_{1(8)} \| y_{2(8)} \| \cdots \| y_{8(8)}$,

$$\begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \end{pmatrix} ,$$

$$\begin{pmatrix} y_{5(8)} \\ y_{6(8)} \\ y_{7(8)} \\ y_{8(8)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_{5(8)} \\ x_{6(8)} \\ x_{7(8)} \\ x_{8(8)} \end{pmatrix} .$$

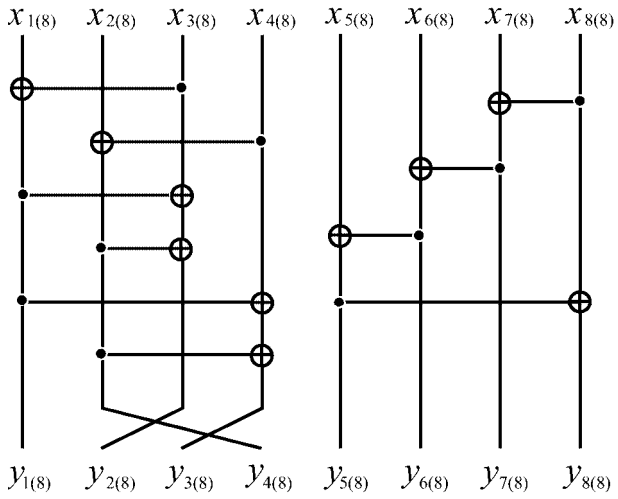


Figure 5: M_{5E}-function

3.3.10 M_{B3}-function

The M_{B3}-function consists of a concatenation of two 32-bit linear transformations, where each 8-bit subdata is regarded as an element of $\mathrm{GF}(2)^8$.

interface $Y_{(64)} = M_{\mathsf{B3}}\left(X_{(64)}\right)$
definition
$x_{1(8)} \| x_{2(8)} \| \cdots \| x_{8(8)} = X_{(64)}$,
$Y_{(64)} = y_{1(8)} \| y_{2(8)} \| \cdots \| y_{8(8)}$,

$$\begin{pmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \end{pmatrix} ,$$

$$\begin{pmatrix} y_{5(8)} \\ y_{6(8)} \\ y_{7(8)} \\ y_{8(8)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{5(8)} \\ x_{6(8)} \\ x_{7(8)} \\ x_{8(8)} \end{pmatrix} .$$

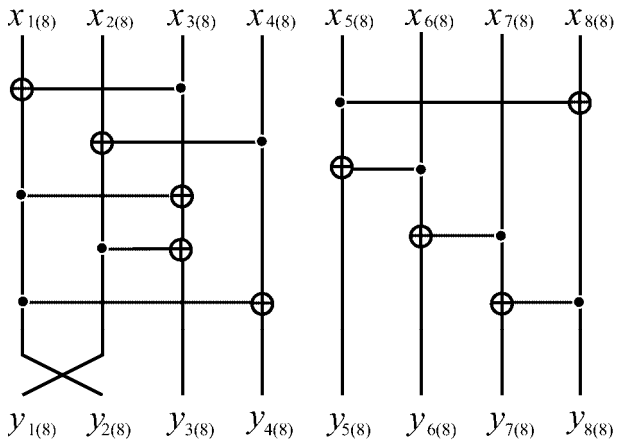


Figure 6: M_{B3}-function

3.3.11 F_σ-function

The F_σ-function is a nonlinear function for 64-bit input/output value, which consists of the s-functions and the $P^{(16)}$-functions.

interface $Y_{(64)} = F_\sigma\left(X_{(64)}\right)$
definition
$x_{1(8)} \| x_{2(8)} \| \cdots \| x_{8(8)} = X_{(64)}$,
$Y_{(64)} = P^{(16)}\left(s(x_{1(8)}) \| s(x_{2(8)}) \| \cdots \| s(x_{8(8)})\right)$.

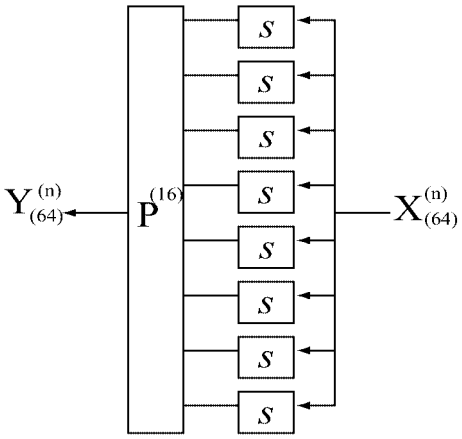


Figure 7: F_σ-function

References

[1] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology, Vol. 4, No. 1, pp. 3–72, 1991.

[2] M. Matsui. Linear cryptanalysis method for DES cipher. In Eurocrypt'93, LNCS 765, pp. 386–397. Springer Verlag, 1994.

[3] J. Daemen, L.R. Knudsen, and V. Rijmen. The block cipher SQUARE. In Fast Software Encryption (4), LNCS 1267, pp. 149–165, 1997.

[4] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 2000.

[5] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura. Specification and assessment of the cipher Hierocrypt. Technical Report of IEICE(Japan) ISEC2000-7, 2000.

[6] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura. The block cipher Hierocrypt. In Selected Areas in Cryptography 2000, LNCS 2012, pp.72–88, 2000.

[7] S. Hong, S. Lee, J. Lim, J. Sung, and D. Cheon. "Provable security against differential and linear cryptanalysis for the SPN structure". In Fast Software Encryption 2000, LNCS 1978, Springer-Verlag, pp. 273–283, 2000.

[8] L.R. Knudsen and T.A. Berson. "Truncated differentials of SAFER". In Fast Software Encryption (5), pp. 15–25, LNCS 1039, 1996.

[9] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. DcWin. The cipher shark. In Fast Software Encryption(3), LNCS 1039, pp. 99–112, 1996.

[10] S. Lucks. "Attacking seven rounds of Rijndael under 192-bit and 256-bit keys". In The third AES Conference, 2000.

[11] M. Matsui. Cryptanalysis of a reduced version of the block cipher E2. Fast Software Encryption'99, Vol. LNCS 1636, , 1999.

[12] M. Matsui, "New Block Encryption Algorithm MISTY", Fast Software Encryption, 4th International Workshop Proceeding, LNCS **1267**, Springer-Verlag, 1997, pp.54-68.

[13] H. Muratani, K. Ohkuma, F. Sano, M. Motoyama, and S. Kawamura. Proposition of a 64-bit version of Hierocrypt. Technical Report of IPSJ(Japan) CSEC11-8, Vol. 11, No. 8, 2000.

[14] K. Ohkuma, H. Muratani, F. Sano, M. Motoyama, and S. Kawamura. A revised nested SPN cipher. Technical Report of IPSJ(Japan) CSEC11-7, Vol. 11, No. 7, 2000.