Ver.2

Cryptographic Techniques Specifications

CIPHERUNICORN-A

NEC Corporation

1 OVERVIEW 1
1.1 Purpose 1
1.2 Symbol definitions 1
1.3 Bit/byte/word ordering 1
2 DESIGN PRINCIPLE AND CRITERIA 2
2.1 Data randomizer 2
2.1.1 Feistel structure 2
2.1.2 Initial/final processing 3
2.2 Round function 3
2.2.1 Dual structure 3
2.2.2 Main stream 3
2.2.3 Temporary key generation mechanism 4
2.2.4 Operators 4
2.2.5 Operation units 4
2.3 Substitution tables 5
2.4 Key scheduler 5
3 ENCRYPTION ALGORITHM 6
3.1 Total structure 6
3.2 Data randomizer 7
3.2.1 Encryption 7
3.2.2 Decryption 9
3.3 F function 11
3.4 A3 function 13
3.5 Multiplication of constants 14
3.6 Tn function 15
3.7 Substitution tables 16
3.8 Key scheduler 19
3.9 MT function 25
REFERENCES 26

1 Overview

1.1 Purpose

These specifications describe the design principle, design criteria, and encryption algorithm of CIPHERUNICORN-A, a 128-bit block cipher.

1.2 Symbol definitions

These specifications make use of the following notation.

P	: 1 block of plaintext
C	: 1 block of ciphertext
IK_j	: 32-bit extended keys used in initial/final processing (j=0,1,...,7)
F^i	: F function of round i (i=0,1,...,15)
FKa^i,FKb^i	: 32-bit extended keys used in main stream of F^i (function keys)
SKa^i,SKb^i	: 32-bit extended keys used in temporary key generation mechanism of F^i (seed keys)
EK^i	: Group of four extended keys FKa^i, SKa^i, FKb^i, SKb^i used by F^i.
‖	: Data concatenation
$\wedge$	: Logical product
$\oplus$	: Exclusive OR (XOR)
$\boxplus$	: Addition (mod 2^{32})
$\boxminus$	: Subtraction (mod 2^{32})
$\otimes$	: Multiplication (mod 2^{32})
x>>n	: Right logical shift of x by n bits
x<<<n	: Left rotation of x by n bits

1.3 Bit/byte/word ordering

These specifications use big endian notation.

Q: 128-bit data (quad word)
D: 64-bit data (double word)
W: 32-bit data (word)
B: 8-bit data (byte)
E: 1-bit data (bit)

Given the above, the following holds.

$$
\begin{aligned}
Q &= D_0 \parallel D_1 \\
&= W_0 \parallel W_1 \parallel W_2 \parallel W_3 \\
&= B_0 \parallel B_1 \parallel B_2 \parallel \ldots\ldots \parallel B_{15} \\
&= E_0 \parallel E_1 \parallel E_2 \parallel \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \parallel E_{127}
\end{aligned}
$$

2 Design Principle and Criteria

Two methods that have been found to be effective in mounting attacks on block ciphers of any structure are linear cryptanalysis and differential cryptanalysis. These methods use shuffling bias in the data randomizer function to infer information on a key. Shuffling bias often originates in the base shuffling process. A structure in which shuffling bias cannot be detected in the base process is therefore desirable.

Against the above background, we decided to design CIPHERUNICORN-A so that shuffling bias does not appear in the round function, the base process of data shuffling. This was evaluated by statistically investigating the relationship between input and output.

In addition, to perform a uniform evaluation of encryption algorithms in the design process, we established a common evaluation scale in examining input and output with the encryption algorithm treated as a black box. We specified, in particular, the following items as constituting a state with no bias and sufficient shuffling, and we checked for this state using a statistical technique that we adopted for this purpose.

- A highly probable relationship between input and output bits does not exist.
- A highly probable relationship between output bits does not exist.
- A highly probable relationship between a change in input bits and a change in output bits does not exist.
- A highly probable relationship between a change in key bits and change in output bits does not exist.
- An output bit that has a high probability of being 0 or 1 does not exist.

Cipher input/output specifications are the same as those of the Advanced Encryption Standard (AES), that is, a block size of 128 bits and the capability of using a secret key length of 128, 192, or 256 bits. This cipher has been designed for high-speed operation on a 32-bit processor.

2.1 Data randomizer

2.1.1 Feistel structure

The Feistel structure has been adopted as the base structure of this cipher because of the following advantages.

- Encryption and decryption have the same structure
- Encryption and decryption can be performed at about the same speed
- No limitations are set on the structure of the round function
- The Feistel structure has been thoroughly analyzed

2.1.2 Initial/final processing

To prevent input to the 1st round function and input to the last round function from becoming known and making an attack easy to mount, initial and final processing have been added.

2.2 Round function

2.2.1 Dual structure

The round function adopts a dual structure that guarantees the security of one part of the structure if the other should be cracked. It consists of a main stream section and temporary key generation mechanism that input extended keys (function key and seed key, respectively). A temporary key is created by the temporary key generation mechanism and combined with the main stream.

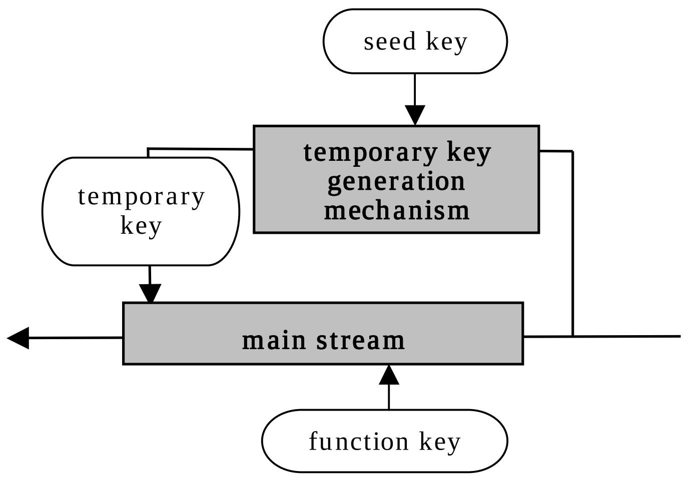

Figure 2.1 Dual structure of round function

2.2.2 Main stream

The structure of the main stream has the following properties.

- Bijective if the temporary key is fixed
- Data is sufficiently shuffled in the main stream itself.

2.2.3 Temporary key generation mechanism

The structure of the temporary key generation mechanism has the following properties.

- The temporary key is output uniformly throughout its possible range.
- The structure is simpler than that of the main stream (considering the possibility of parallel processing).
- The structure differs from that of the main stream (difference in structure guarantees security).
- Size of temporary key is made shorter than that of seed key.
- Data is sufficiently shuffled in the temporary key generation mechanism itself.

Because the temporary key generation mechanism is simpler in structure than the main stream, an adversary is likely to mount an attack on this mechanism first. Even if the temporary key should become known, however, it is expected that the existence of multiple seed-key candidates will make it difficult to infer the secret key or function key from the seed key.

2.2.4 Operators

Considering a 32-bit processor to be the basic form of implementation for this cipher, we have adopted operators that can be processed at high speed on this kind of platform. We have also combined operations having different algebraic structures with the aim of making the cipher stronger.

2.2.5 Operation units

As a countermeasure to truncated differential attack, three types of operation units are used: 8, 32, and 64 bits.

2.3 Substitution tables

Four 8-bit input/output tables are used as a set of substitution tables. Each of these 8-bit input/output tables must satisfy the following conditions.

- Bijective
- Maximum differential probability of 2^{-6}
- Maximum linear probability of 2^{-6}
- An algebraic degree of 7
- Input/output polynomials of high degree and many terms
- Average number of diffusion bits (number of output bits changed due to change in one input bit) equal to 4.0
- No fixed points

The method adopted here to generate a substitution table that satisfies the above conditions is to use an inverse function over a Galois field (GF) of 2^8 in combination with an affine transformation.

An inverse function over a GF (2^8) is a bijective function with an algebraic degree of 7 known to have a maximum linear and differential probability of 2^{-6} (best case). The degree of its input/output polynomials is also high at 254. By incorporating an affine transformation, the number of terms in the input /output polynomials can be expected to increase.

In order to use a combination of four 8-bit input/output tables, moreover, a different irreducible polynomial was adopted for each table.

The following equation is used to generate a substitution table.

$$S(x) = \text{matrixA}\{ (x + c)^{-1} \bmod g\} + d$$

Here:

matrixA	: GF(2) 8×8 bijective matrix
c,d	: 8-bit constants (other than 0)
g	: 8th-degree irreducible polynomial

After selecting matrixA, c, d, and g by random numbers, a search is made for a substitution table that satisfies the above conditions.

2.4 Key scheduler

The structure of the key scheduler has the following properties.

- Mapping from the secret key to extended keys is injective.
- Each of the extended keys is affected by all information in the secret key.
- A highly probable relationship between the secret keys and extended keys or among the extended keys does not exist (secure against related-key attacks).
- The structure makes use of the constituent elements of the round function.

3 Encryption algorithm

3.1 Total structure

The CIPHERUNICORN-A has a Feistel structure that can use a data block length of 128 bits and a secret key length of 128, 192, or 256 bits. There are 16 rounds with addition/subtraction of extended keys performed as initial/final processing. The key scheduler has a modified Feistel structure for inputting the secret key. Here, after shuffling the secret key in dummy loops, extended keys are repeatedly extracted while shuffling.

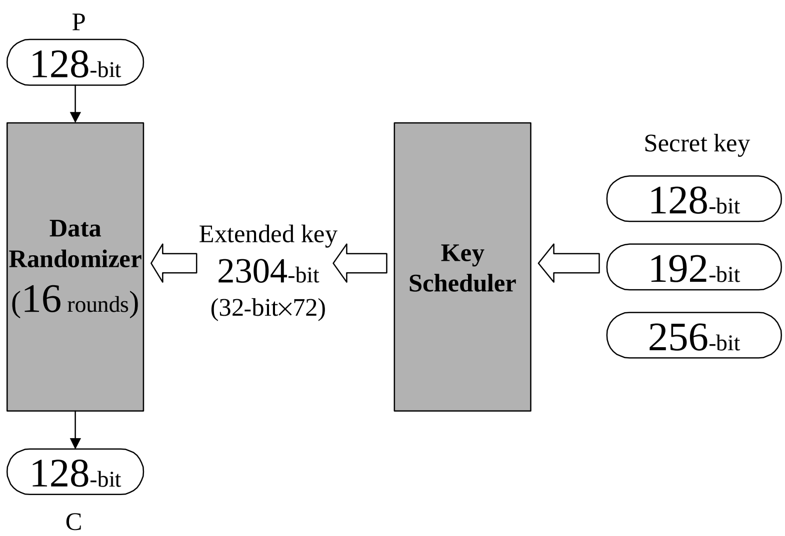

Figure 3.1 CIPHERUNICORN-A total structure

3.2 Data randomizer

3.2.1 Encryption

[Input] 1 block of plaintext: $P = P_0 \parallel P_1 \parallel P_2 \parallel P_3$ (128 bits)

Extended keys for F function: $EK^i = FKa^i \parallel SKa^i \parallel FKb^i \parallel SKb^i$ (128 bits: i=0,1,...,15)

Extended keys for initial/final processing: IK_j (32 bits: j=0,1,...,7)

[OutPut] 1 block of ciphertext: $C = C_0 \parallel C_1 \parallel C_2 \parallel C_3$ (128 bits)

[Process] The system inputs one block of plaintext, adds (mod 2^{32}) extended keys as initial processing, shuffles data by a 16-round Feistel structure, subtracts (mod 2^{32}) extended keys as final processing, and outputs one block of ciphertext.

for i=0,…,3 do

{

$W^0_i = P_i \boxplus IK_i$

}

for i=0,…,14 do

{

$W^{i+1}_0 \parallel W^{i+1}_1 = W^i_2 \parallel W^i_3$

$W^{i+1}_2 \parallel W^{i+1}_3 = (W^i_0 \parallel W^i_1) \oplus F^i(W^i_2 \parallel W^i_3, EK^i)$

}

$W^{16}_2 \parallel W^{16}_3 = W^{15}_2 \parallel W^{15}_3$

$W^{16}_0 \parallel W^{16}_1 = (W^{15}_0 \parallel W^{15}_1) \oplus F^{15}(W^{15}_2 \parallel W^{15}_3, EK^{15})$

for i=0,…,3 do

{

$C_i = W^{16}_i \boxminus IK_{i+4}$

}

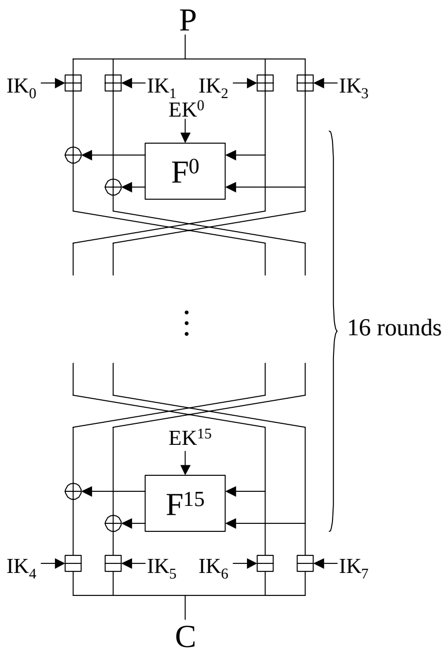

Figure 3.2 Data randomizer (encryption)

3.2.2 Decryption

[Input] 1 block of ciphertext: $C = C_0 \| C_1 \| C_2 \| C_3$ (128 bits)

Extended keys for F function: $EK^i = FKa^i \| SKa^i \| FKb^i \| SKb^i$

(128 bits: i=0,1,...,15)

Extended keys for initial/final processing: IK_j (32 bits: j=0,1,...,7)

[Output] 1 block of plaintext: $P = P_0 \| P_1 \| P_2 \| P_3$ (128 bits)

[Process] The system inputs one block of ciphertext, adds (mod 2^{32}) extended keys as initial processing, shuffles data by a 16-round Feistel structure, subtracts (mod 2^{32}) extended keys as final processing, and outputs one block of plaintext.

for i=0,…,3 do

{

$W^0_i = C_i \boxplus IK_{i+4}$

}

for i=0,…,14 do

{

$W^{i+1}_0 \| W^{i+1}_1 = W^i_2 \| W^i_3$

$W^{i+1}_2 \| W^{i+1}_3 = (W^i_0 \| W^i_1) \oplus F^{15-i}(W^i_2 \| W^i_3, EK^{15-i})$

}

$W^{16}_2 \| W^{16}_3 = W^{15}_2 \| W^{15}_3$

$W^{16}_0 \| W^{16}_1 = (W^{15}_0 \| W^{15}_1) \oplus F^0(W^{15}_2 \| W^{15}_3, EK^0)$

for i=0,…,3 do

{

$P_i = W^{16}_i \boxminus IK_i$

}

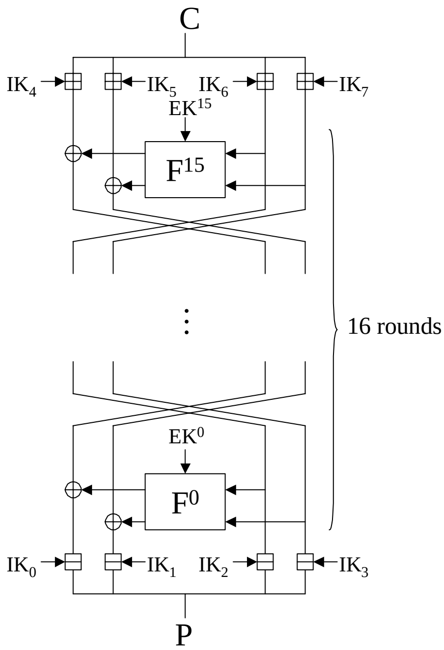

Figure 3.3 Data randomizer (decryption)

3.3 F function

[Input] Input data: $X = X_l \| X_r$ (64 bits)
Extended key for F^i function: $EK^i = FKa^i \| SKa^i \| FKb^i \| Skb^i$
(128 bits)

[Constants] Multiplication constants: Const0 =0x7e167289, Const1 =0xfe21464b

[Output] Output data: $Y = Y_l \| Y_r$ (64 bits)

[Process] An F function consists of a main stream section and a temporary key generation mechanism. In the main stream section, the function adds extended keys FKa^i and FKb^i to 64 bits of input data, executes the A3 function, performs multiplication of constants, and executes the Tn functions. In the temporary key generation mechanism, the function adds extended keys SKa^i and SKb^i to 64 bits of input data, performs multiplication of constants, passes result through the Tn functions, and produces a temporary key. This key is used to shuffle main stream data and generate 64 bits of output data.

$WK^0_0 = SKa^i \boxplus X_r$
$WK^0_1 = SKb^i \boxplus X_l$
$WK^1_0 = WK^0_0 \otimes Const0$
$WK^1_1 = WK^0_1 \oplus T_0(WK^1_0)$
$WK^2_1 = WK^1_1 \otimes Const1$
$WK^2_0 = WK^1_0 \oplus T_0(WK^2_1)$
$WK^3_0 = WK^2_0 \otimes Const1$
$WK^3_1 = WK^2_1 \oplus T_0(WK^3_0)$
$WK^4_1 = WK^3_1 \otimes Const0$
$WK^4_0 = WK^3_0 \oplus T_0(WK^4_1)$
$WK^5_1 = WK^4_1 \oplus T_1(WK^4_0)$
$WK^5_0 = WK^4_0 \oplus T_1(WK^5_1)$
$WX^0_0 = FKa^i \boxplus X_l$
$WX^0_1 = FKb^i \boxplus X_r$
$WX^1_0 \| WX^1_1 = A3(WX^0_0 \| WX^0_1)$
$WX^2_0 = WX^1_0 \otimes Const0$
$WX^2_1 = WX^1_1 \oplus T_0(WX^2_0)$
$WX^3_1 = WX^2_1 \otimes Const1$
$WX^3_0 = WX^2_0 \oplus T_0(WX^3_1)$

$$WX^4{}_1 = WX^3{}_1 \oplus T_1(WX^3{}_0)$$
$$WX^4{}_0 = WX^3{}_0 \oplus T_1(WX^4{}_1)$$
$$WX^5{}_1 = WX^4{}_1 \oplus T_2(WX^4{}_0)$$
$$WX^5{}_0 = WX^4{}_0 \oplus T_2(WX^5{}_1)$$
$$WX^6{}_1 = WX^5{}_1 \oplus T_3(WX^5{}_0)$$
$$WX^6{}_0 = WX^5{}_0 \oplus T_3(WX^6{}_1)$$
$$k = (WK^5{}_1 >> 2) \wedge 0x3$$
$$WX^7{}_1 = WX^6{}_1 \oplus T_k(WX^6{}_0)$$
$$k = WK^5{}_1 \wedge 0x3$$
$$WX^7{}_0 = WX^6{}_0 \oplus T_k(WX^7{}_1)$$
$$Y_l = WX^7{}_0 \oplus WK^5{}_0$$
$$Y_r = WX^7{}_1 \oplus WK^5{}_0$$

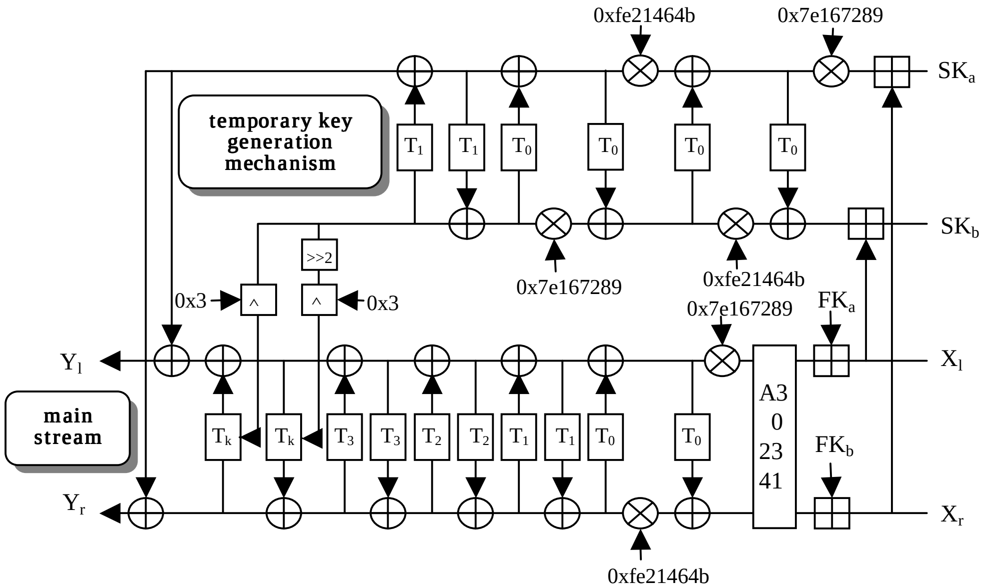

Figure 3.4 F function

3.4 A3 function

[Input] Input data: X (64 bits)

[Constants] Three constants: const0=0, const1=23, const2=41

Criteria for determining constants:

- One constant is zero (performance considerations).
- The number of output bits required in the inverse operation to determine 1 input bit becomes maximum (43).
- All 64 input bits appear in the lower 3 bytes of each 32-bit group in thedivided output (related to subsequent processing).

Six sets of constants were found as a result of performing an exhaustive search for constants that satisfy the above criteria. Of these, only one set featured constants that were both prime numbers, and it was this set that was selected.

[Output] Output data: Y (64 bits)

[Process] The function performs three left rotations on input data separately as specified by the constants, performs an exclusive OR on the resulting three sets of data, and outputs the result.

$$Y = (X \lll const0) \oplus (X \lll const1) \oplus (X \lll const2)$$

3.5 Multiplication of constants

[Input] Input data: X (32 bits)

[Constants] Two constants: Const0 = 0x7e167289, Const1 = 0xfe21464b

Criteria for determining constants:

- Odd number (to preserve bijection)
- Hamming weight is 16 (1/2 of number of bits in constant)
- All bits of input data X affect upper 8 bits of input to function T_0.
- When "shift + XOR" is substituted for multiplication and a differential of no more than 3 bits is given, the T_0 function's input (upper 8 bits) differential is not 0 or 0xff.

A search was performed for constants that satisfy the above criteria and four were found to exist. When using each of these constants in actual multiplication, the two constants indicated above were found to result in few high-probability relationships between input differential and output differential.

[Output] Output data: Y (32 bits)

[Process] Input data is multiplied by a multiplication constant and the result is output.

$$Y = X \otimes Constn \qquad n=0,1$$

3.6 Tn function

[Input] Input data: $X = X_0 \| X_1 \| X_2 \| X_3$ (32 bits)

Input number: n (n=0,1,2,3)

[Output] Output data: Y (32 bits)

[Process] The function divides input data into four bytes and treats the byte corresponding to the input number as the input value to a substitution table. There are four 8-bit-input/8-bit-output substitution tables denoted as S_0, S_1, S_2, and S_3 in order from high to low bytes.

For the T_k function, k is given as input and takes on a value of 0, 1, 2, or 3.

$$Y = S_0(X_n) \| S_1(X_n) \| S_2(X_n) \| S_3(X_n)$$

$$X = X_0 \| X_1 \| X_2 \| X_3$$
$$Y = Y_0 \| Y_1 \| Y_2 \| Y_3$$

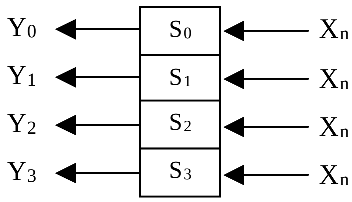

Figure 3.5 Tn function

3.7 Substitution tables

[Input] Input data: X (8 bits)

[Output] Output data: Y (8 bits)

[Process] Data in substitution table S_n at position corresponding to input data is output.

$$Y = S_n(X) \qquad n=0,1,2,3$$

The equation for generating each of the four substitution tables is as follows.

$$S_n(x) = \mathrm{matrixA}\{ (x + c)^{-1} \bmod g\} + d$$

Table 3.1 Substitution table parameters

S_n	matrixA	c	g	d
S_0	{0x23, 0x4e, 0x9c, 0xb1, 0x49, 0xd8, 0xc6, 0xe4}	233	0x11d	28
S_1	{0x7e, 0x2a, 0xef, 0x52, 0x34, 0xa2, 0x70, 0xd7}	26	0x165	171
S_2	{0x32, 0x04, 0x8f, 0x83, 0x89, 0x67, 0xcf, 0x3b}	43	0x14d	155
S_3	{0x34, 0x20, 0xba, 0xd0, 0x66, 0xd7, 0xb2, 0xa8}	200	0x171	47

Here, matrixA = {0x23, 0x4e, 0x9c, 0xb1, 0x49, 0xd8, 0xc6, 0xe4}of S_0 indicates the GF(2) 8×8 matrix shown below.

$$\mathrm{matrixA} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Note that irreducible polynomial g = 0x11d = $100011101_{(2)}$ of S_0 is the following polynomial.

$$g = x^8 + x^4 + x^3 + x^2 + 1$$

Table 3.2 Substitution table S_0

$S_0(0)=149$, $S_0(1)=111$, ..., $S_0(255)=92$

149	111	237	155	21	85	108	76	236	75	193	84	22	138	89	5
51	145	13	153	148	163	86	59	204	175	91	117	126	70	144	10
248	146	201	0	97	208	23	214	147	234	66	65	226	57	210	224
172	40	154	87	178	235	135	220	110	121	96	8	9	53	241	105
143	169	182	139	112	16	183	67	233	39	197	74	166	218	231	242
161	159	192	37	177	228	47	119	14	18	244	56	3	195	239	219
33	167	26	180	54	61	58	222	4	30	191	34	107	249	142	150
95	42	124	25	232	181	120	93	5	68	6	48	129	41	104	73
188	165	212	160	250	141	123	216	94	238	81	202	7	122	196	17
207	102	184	189	243	72	206	12	200	225	164	176	247	1	2	254
71	185	229	187	251	137	69	168	50	24	171	173	158	221	127	27
252	114	152	82	209	38	203	128	215	213	36	174	134	179	90	118
80	246	253	125	29	44	15	227	98	205	255	77	198	194	133	130
79	103	78	49	19	140	109	211	223	63	64	151	62	217	170	83
136	45	115	199	20	46	190	240	132	28	162	230	131	106	32	88
157	31	43	156	113	186	35	101	52	60	11	100	116	245	99	92

Table 3.3 Substitution table S_1

$S_1(0)=174$, $S_1(1)=255$, ..., $S_1(255)=53$

174	255	161	109	254	40	95	67	33	124	133	58	224	238	129	56
137	57	169	87	221	220	163	84	14	239	171	138	74	192	66	104
8	250	43	115	126	88	212	103	62	82	143	4	117	226	28	155
65	156	139	183	235	125	217	116	111	237	157	68	160	184	213	172
170	132	73	2	1	232	92	249	136	106	175	5	9	140	38	191
50	251	85	12	27	48	46	52	145	78	168	159	100	188	16	227
26	198	244	205	178	72	142	162	51	246	241	128	194	177	122	20
144	49	83	166	247	225	11	7	102	242	185	18	150	165	121	98
93	197	70	151	75	118	202	216	108	207	15	112	99	35	101	69
86	61	79	110	13	218	149	6	134	29	36	131	181	154	180	230
77	193	164	17	211	3	209	105	94	206	44	19	60	123	10	31
130	195	76	208	54	252	219	203	199	39	189	80	167	90	32	30
233	64	245	182	120	231	127	47	22	135	55	114	234	41	21	81
173	223	23	253	153	25	45	248	97	179	186	119	200	146	187	210
0	228	24	190	141	236	63	201	96	113	240	147	229	91	107	214
89	59	152	215	176	204	243	148	42	158	71	34	222	37	196	53

Table 3.4 Substitution table S_2

$S_2(0)=37$, $S_2(1)=34$, ..., $S_2(255)=124$

37	34	162	132	134	220	91	143	41	45	229	247	98	178	68	56
212	97	70	15	58	72	216	208	14	96	214	217	133	179	28	154
120	123	83	100	235	3	230	160	193	245	164	155	255	175	79	148
227	219	23	95	111	11	87	104	163	203	189	29	156	173	211	64
157	53	196	89	81	4	84	16	192	74	13	181	20	184	57	183
90	119	93	207	38	131	94	60	116	1	213	122	5	101	144	117
75	46	8	172	170	152	231	210	66	54	10	187	128	204	12	102
243	115	137	147	159	233	59	221	253	112	165	198	105	222	234	153
43	201	121	180	86	205	225	242	182	55	63	232	254	44	9	21
136	65	114	31	40	49	0	36	169	22	249	35	62	17	174	248
158	151	24	50	176	108	67	127	150	18	2	168	194	171	195	145
99	25	80	224	33	200	197	118	161	61	142	77	190	209	48	139
238	206	42	125	239	237	52	223	88	167	26	130	76	191	7	71
215	27	126	6	251	51	241	129	135	246	244	146	32	177	73	82
226	110	78	186	240	141	166	69	107	85	103	149	250	109	202	19
113	140	138	39	185	228	106	47	252	199	188	92	218	30	236	124

Table 3.5 Substitution table S_3

$S_3(0)=24$, $S_3(1)=252$, ..., $S_3(255)=34$

24	252	144	121	17	42	77	127	2	35	173	21	129	58	105	113
112	229	185	189	76	204	209	87	5	96	82	99	133	140	66	64
192	107	194	220	16	68	183	171	219	51	92	13	152	86	135	123
98	174	103	156	157	59	145	155	158	8	231	132	83	49	23	32
85	69	251	36	233	238	222	149	37	248	26	18	125	11	137	253
79	52	56	95	241	187	44	167	124	102	227	115	212	142	154	93
247	211	33	28	67	10	147	225	215	210	246	160	131	73	65	57
1	182	180	199	207	126	216	224	61	81	202	196	146	188	119	128
50	30	91	161	89	12	195	74	235	223	226	172	245	7	218	159
242	217	208	38	163	45	39	4	62	136	104	179	88	197	6	0
141	190	243	214	109	162	60	165	198	228	221	164	106	101	203	236
143	48	110	80	176	78	234	181	97	84	20	70	29	168	27	72
71	90	255	19	254	114	25	230	47	43	100	178	40	41	249	186
150	205	184	201	139	75	54	22	63	244	108	175	46	169	240	153
151	116	122	232	166	117	14	94	111	206	237	177	200	31	170	120
213	53	148	15	55	239	3	191	134	250	193	9	130	118	138	34

3.8 Key scheduler

[Input] Secret key: $M = M_0 \| M_1 \| \ldots\ldots \| M_{LINE-1}$ (LINE=4(128 bits),
LINE=6(192 bits),
LINE=8(256 bits))

[Output] Extended keys for F function: EK^i (i=0,1,…,15) (128 bits × 16 rounds)
Extended keys for initial/final processing: IK_j (j=0,1, …,7) (32 bits × 8)

[Process] The key scheduler consists of multiple MT functions.
In its basic structure, the secret key is first passed through a dummy loop of MT functions three times where the number of rounds of MT functions in each loop depends on the length of the secret key. After this, the secret key is passed through 16 rounds of MT functions 9 times. Each of these passes generates 8 32-bit extended keys. There are 4, 6, and 8 rounds of MT functions in each dummy loop for secret-key lengths of 128, 192, and 256 bits, respectively. These are not interchangeable.
Figure 3.12 shows how these generated extended keys are used.

```
cnt = 0
n = 16+2;
W_i=M_i (i = 0,…,LINE-1)
for i = 0,..,2 do
{
    for j = 0,..,LINE-1 do
    {
        W_j ‖ W_(j+1)%LINE = MT(W_j ‖ W_(j+1)%LINE)
    }
}
for i = 0,..,(16+2)/2-1 do
{
    for j = i*16 ,…, i*16+8-1 do
    {
        W_j%LINE ‖ W_(j+1)%LINE = MT(W_j%LINE ‖ W_(j+1)%LINE)
    }
    for j = i*16+8 ,…, i*16+16-1 do
    {
```

```
            W_{j%LINE} ‖ W_{(j+1)%LINE}  =  MT(W_{j%LINE} ‖ W_{(j+1)%LINE})
            WK[cnt++]  =  W_{(j+1)%LINE}
        }
    }

    IK0  =  WK[0    ];
    IK1  =  WK[n    ];
    IK2  =  WK[n*2];
    IK3  =  WK[n*3];
    IK4  =  WK[n    -1];
    IK5  =  WK[n*2-1];
    IK6  =  WK[n*3-1];
    IK7  =  WK[n*4-1];

    for   i  =  0,…,  16-1  do
    {
        FKa^i  =  WK[        1+i];
        SKa^i  =  WK[n    +1+i];
        FKb^i  =  WK[n*2+1+i];
        SKb^i  =  WK[n*3+1+i];
    }
```

$M(128\text{-bit}) = M_0 \| M_1 \| M_2 \| M_3 \ (M_n\text{:32-bit})$

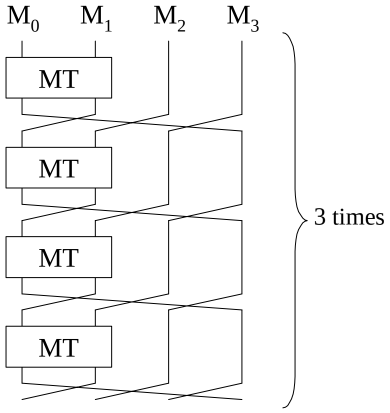

Figure 3.6 Key scheduler dummy loop (128-bit secret key)

$M(192\text{-bit}) = M_0 \| M_1 \| M_2 \| M_3 \| M_4 \| M_5 \ (M_n\text{:32-bit})$

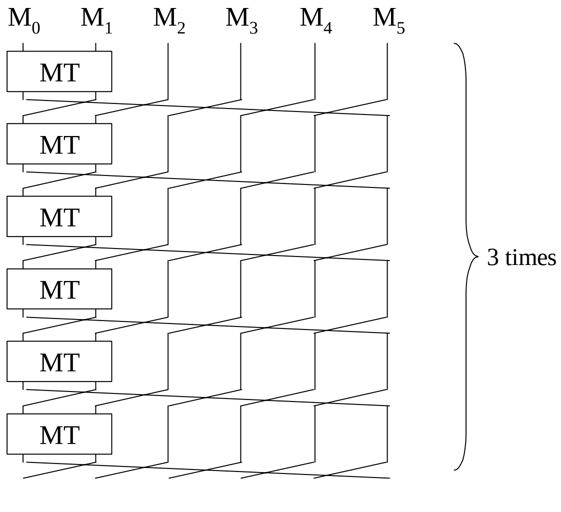

Figure 3.7 Key scheduler dummy loop (192-bit secret key)

$M(256\text{-bit}) = M_0 \| M_1 \| M_2 \| M_3 \| M_4 \| M_5 \| M_6 \| M_7$ (M_n:32-bit)

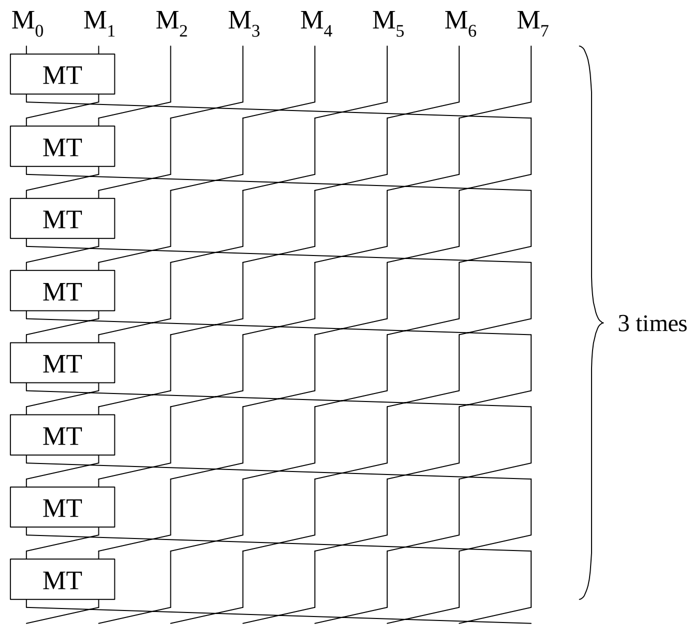

Figure 3.8 Key scheduler dummy loop (256-bit secret key)

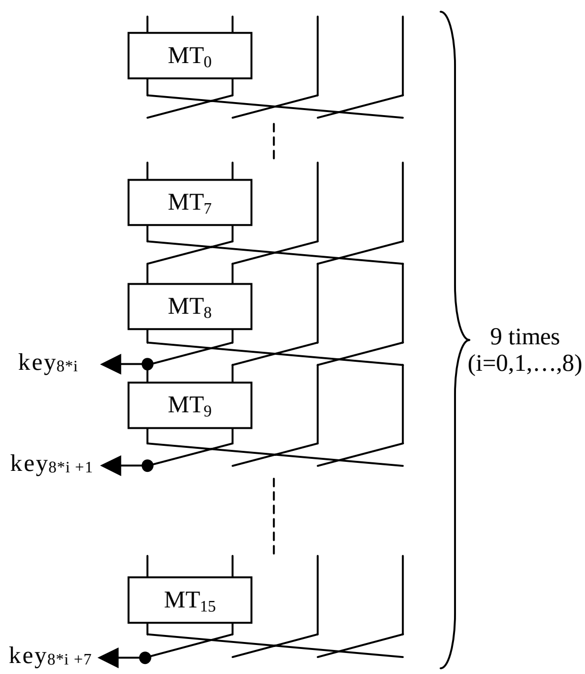

Figure 3.9 Key scheduler extraction of extended keys (128-bit secret key)

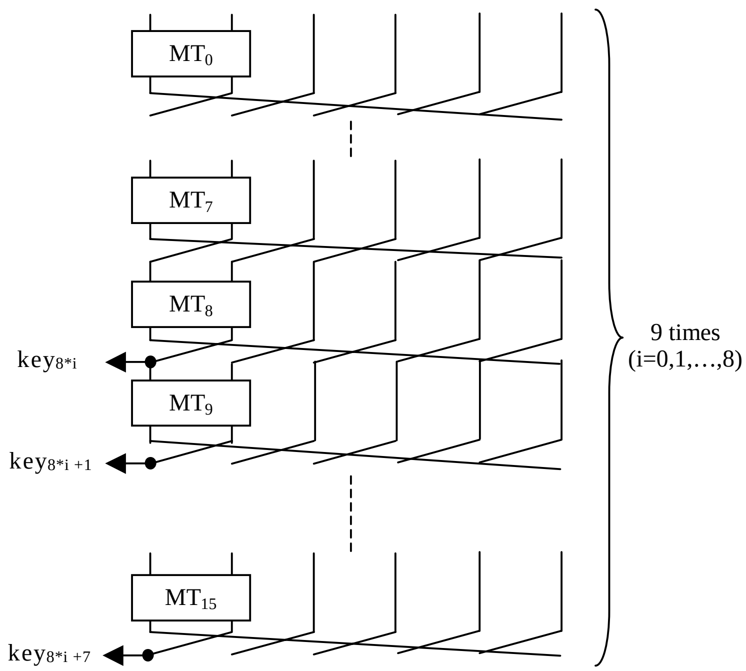

Figure 3.10 Key scheduler extraction of extended keys (192-bit secret key)

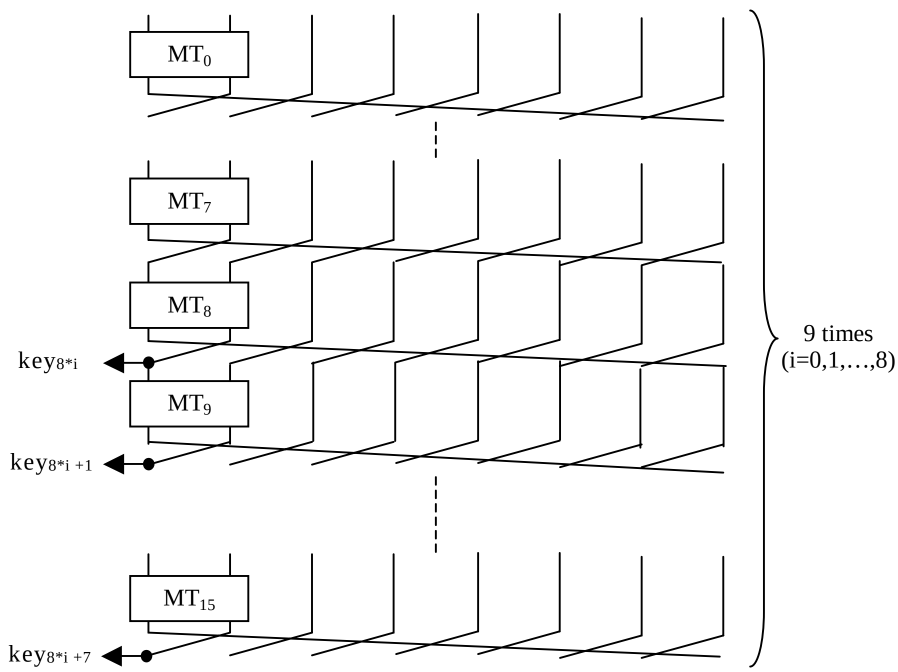

Figure 3.11 Key scheduler extraction of extended keys (256-bit secret key)

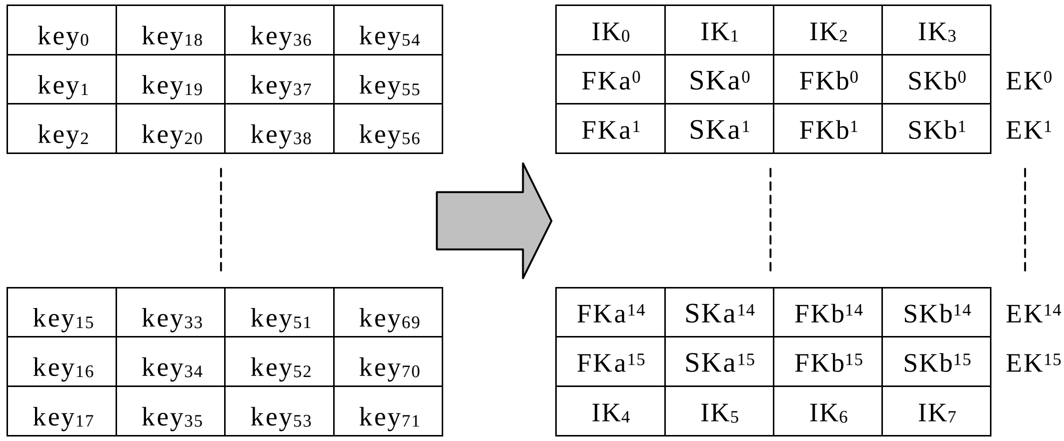

Figure 3.12 Extended key correspondence

3.9 MT function

[Input] Input data: $X = X_0 \| X_1$ (64 bits)

[Constant] Multiplication constant: Const = 0x01010101

Criteria for determining constant:

- Odd number (to preserve bijection)
- 32-bit input data are accumulated in the upper 8 bits (T_0 function input) of multiplication output.
- Hamming weight is minimum among all constants satisfying the above two conditions.

The constant indicated above satisfies the above criteria and was thus selected.

[Output] Output data: $Y = Y_0 \| Y_1$ (64 bits)

[Process] The function multiples the upper 32 bits of input data by the constant Const and inputs the result into the T_0 function. It then performs an exclusive OR between the output of this T_0 function and the lower 32 bits of input data and outputs the result.

$$Y_0 = X_0 \otimes \text{Const}$$
$$Y_1 = X_1 \oplus T_0(Y_0)$$

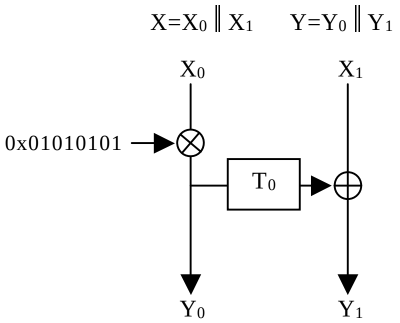

Figure 3.13 MTfunction

References

[1] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," EUROCRYPT'93, LNCS765, pp.386-397, Springer-Verlag, 1994.

[2] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems (Extended Abstract)," proceeding of CRYPTO'90, pp.2-21, 1990.

[3] T. Jakobsen and L.R. Knudsen, "The Interpolation Attack on Block Ciphers," FSE'97, LNCS1267, pp.28-40, Springer-Verlag, 1997.

[4] L.R. Knudsen and T.A. Berson, "Truncated Differentials of SAFER," FSE'96, LNCS1039, pp.15-25, Springer-Verlag, 1996.

[5] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," EUROCRYPT'93, LNCS765, pp.398-409, Springer-Verlag, 1994.

[6] E. Biham, "On Matsui's Linear Cryptanalysis," EUROCRYPT'94, LNCS950, pp.341-355, Springer-Verlag, 1994.