INTERNATIONAL STANDARD

ISO/IEC 29192-2

First edition
2012-01-15

Information technology — Security techniques — Lightweight cryptography

Part 2: Block ciphers

Technologies de l'information — Techniques de sécurité — Cryptographie pour environnements contraints

Partie 2: Chiffrements par blocs

Reference number
ISO/IEC 29192-2:2012(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 29192-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Security techniques*.

ISO/IEC 29192 consists of the following parts, under the general title *Information technology — Security techniques — Lightweight cryptography*:

— *Part 1: General*

— *Part 2: Block ciphers*

— *Part 3: Stream ciphers*

— *Part 4: Mechanisms using asymmetric techniques*

Further parts may follow.

Introduction

This part of ISO/IEC 29192 specifies block ciphers suitable for lightweight cryptography, which are tailored for implementation in constrained environments.

ISO/IEC 29192-1 specifies the requirements for lightweight cryptography.

A block cipher maps blocks of n bits to blocks of n bits, under the control of a key of k bits.

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holder of these patent rights has assured ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of these patent rights is registered with ISO and IEC. Information may be obtained from:

Sony Corporation
System Technologies Laboratories
Attn Masanobu Katagi
Gotenyama Tec. 5-1-12 Kitashinagwa Shinagawa-ku
Tokyo
141-0001 Japan
Tel. +81-3-5448-3701
Fax +81-3-5448-6438
E-mail Masanobu.Katagi@jp.sony.com

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Security techniques — Lightweight cryptography

Part 2:
Block ciphers

1 Scope

This part of ISO/IEC 29192 specifies two block ciphers suitable for applications requiring lightweight cryptographic implementations:

— PRESENT: a lightweight block cipher with a block size of 64 bits and a key size of 80 or 128 bits;

— CLEFIA: a lightweight block cipher with a block size of 128 bits and a key size of 128, 192 or 256 bits.

2 Normative references

There are no normative references for this part of ISO/IEC 29192.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
block
string of bits of defined length

[ISO/IEC 18033-1]

3.2
block cipher
symmetric encipherment system with the property that the encryption algorithm operates on a block of plaintext, i.e. a string of bits of a defined length, to yield a block of ciphertext

[ISO/IEC 18033-1]

3.3
ciphertext
data which has been transformed to hide its information content

[ISO/IEC 9798-1]

3.4
key
sequence of symbols that controls the operation of a cryptographic transformation (e.g. encipherment, decipherment)

NOTE Adapted from ISO/IEC 11770-1.

3.5
***n*-bit block cipher**
block cipher with the property that plaintext blocks and ciphertext blocks are *n* bits in length

[ISO/IEC 10116]

3.6
plaintext
unenciphered information

NOTE Taken from ISO/IEC 9797-1:1999.

3.7
round key
sequence of symbols derived from the key using the key schedule, and used to control the transformation in each round of the block cipher

4 Symbols

0x A prefix for a binary string in hexadecimal notation

|| Concatenation of bit strings

$a \leftarrow b$ Updating a value of a by a value of b

$\oplus$ Bitwise exclusive-OR operation

5 Lightweight block cipher with a block size of 64 bits

5.1 PRESENT

5.1.1 PRESENT algorithm

The PRESENT algorithm [10] is a symmetric block cipher that can process data blocks of 64 bits, using a key of length 80 or 128 bits. The cipher is referred to as PRESENT-80 or PRESENT-128 when using an 80-bit or 128-bit key respectively.

5.1.2 PRESENT specific notations

$\boldsymbol{K_i} = \boldsymbol{k^i_{63} \ldots k^i_0}$ 64-bit round key that is used in round i

$\boldsymbol{k^i_b}$ bit b of round key $\boldsymbol{K_i}$

$\boldsymbol{K} = \boldsymbol{k_{79} \ldots k_0}$ 80-bit key register

$\boldsymbol{k_b}$ bit b of key register $\boldsymbol{K}$

STATE 64-bit internal state

b_i bit i of the current *STATE*

w_i 4-bit word where $0 \le i \le 15$

5.1.3 PRESENT encryption

The PRESENT block cipher consists of 31 'rounds', i.e. 31 applications of a sequence of simple transformations. A pseudocode description of the complete encryption algorithm is provided in Figure 1, where *STATE* denotes the internal state. The individual transformations used by the algorithm are defined in 5.1.5. Each round of the algorithm uses a distinct round key K_i ($1 \le i \le 31$), derived as specified in 5.1.6. Two consecutive rounds of the algorithm are shown for illustrative purposes in Figure 2.

```
generateRoundKeys()
for i = 1 to 31 do
    addRoundKey(STATE,K_i)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE,K_32)
```

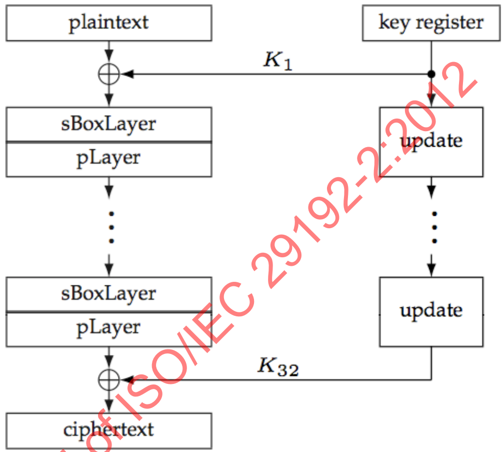

Figure 1 — The encryption procedure of PRESENT

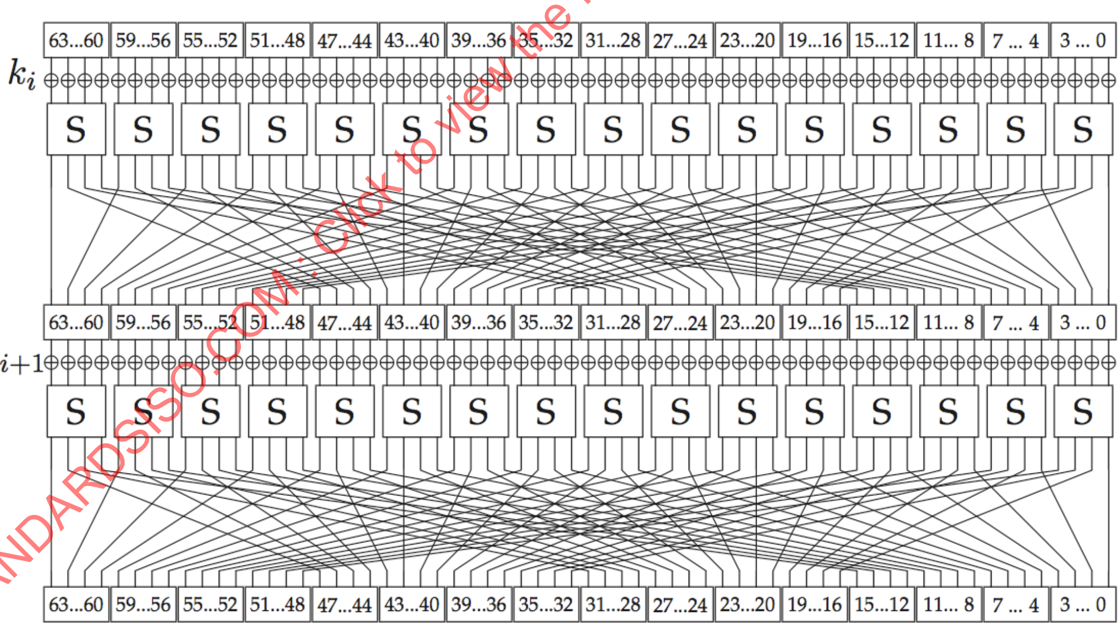

Figure 2 — Two rounds of PRESENT

5.1.4 PRESENT decryption

The complete PRESENT decryption algorithm is given in Figure 3. The individual transformations used by the algorithm are defined in 5.1.5. Each round of the algorithm uses a distinct round key K_i ($1 \le i \le 31$), derived as specified in 5.1.6.

```
generateRoundKeys()
addRoundKey(STATE,K32)
for i = 31 downto 1 do
   invpLayer(STATE)
   invsBoxLayer(STATE)

   addRoundKey(STATE,Ki)
end for
```

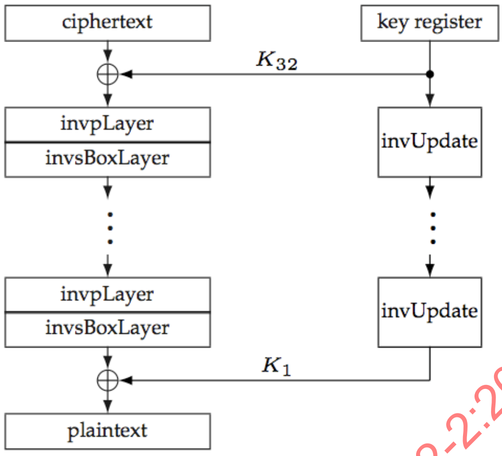

Figure 3 — The decryption procedure of PRESENT

5.1.5 PRESENT transformations

5.1.5.1 addRoundKey

Given round key $\boldsymbol{K_i} = \boldsymbol{k^i_{63}\ldots k^i_0}$ for $1 \le i \le 32$ and current *STATE* $b_{63}\ldots b_0$, **addRoundKey** consists of the operation for $0 \le j \le 63$, $b_j \leftarrow b_j \oplus k^i_j$.

5.1.5.2 sBoxLayer

The non-linear **sBoxLayer** of the encryption process of PRESENT uses a single 4-bit to 4-bit S-box S which is applied 16 times in parallel in each round. The S-box transforms the input *x* to an output $S(x)$ as given in hexadecimal notation in Table 1.

Table 1 — PRESENT S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
***S*(*x*)**	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

For **sBoxLayer** the current *STATE* $b_{63}\ldots b_0$ is considered as sixteen 4-bit words $w_{15}\ldots w_0$ where $w_i = b_{4*i+3} \,||\, b_{4*i+2} \,||\, b_{4*i+1} \,||\, b_{4*i}$ for $0 \le i \le 15$ and the output nibble $S(w_i)$ provides the updated state values as a concatenation $S(w_{15}) \,||\, S(w_{14}) \,||\, \ldots \,||\, S(w_0)$.

5.1.5.3 invsBoxLayer

The S-box used in the decryption procedure of PRESENT is the inverse of the 4-bit to 4-bit S-box S that is described in 5.1.5.2. The inverse S-box transforms the input x to an output $S^{-1}(x)$ as given in hexadecimal notation in Table 2.

Table 2 — PRESENT inverse S-box

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S^{-1}(x)$	5	E	F	8	C	1	2	D	B	4	6	3	0	7	9	A

5.1.5.4 pLayer

The bit permutation **pLayer** used in the encryption routine of PRESENT is given by Table 3. Bit i of *STATE* is moved to bit position $P(i)$.

Table 3 — PRESENT permutation layer pLayer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55

i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59

i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

5.1.5.5 invpLayer

The inverse permutation layer **invpLayer** used in the decryption routine of PRESENT is given by Table 4. Bit i of *STATE* is moved to bit position $P^{-1}(i)$.

Table 4 — PRESENT inverse permutation Layer invpLayer

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P^{-1}(i)$	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60

i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P^{-1}(i)$	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61

i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P^{-1}(i)$	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62

i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P^{-1}(i)$	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

5.1.6 PRESENT key schedule

5.1.6.1 PRESENT-80 and PRESENT-128

PRESENT can take keys of either 80 or 128 bits. In 5.1.6.2 the version with an 80-bit key (PRESENT-80) and in 5.1.6.3 the 128-bit version (PRESENT-128) is described.

5.1.6.2 80-bit key for PRESENT-80

The user-supplied key is stored in a key register $\boldsymbol{K}$ and represented as $k_{79}k_{78} \ldots k_0$. At round i the 64-bit round key $K_i = k^i_{63}k^i_{62} \ldots k^i_0$ consists of the 64 leftmost bits of the current contents of register $\boldsymbol{K}$. Thus at round i we have that:

$$K_i = k^i_{63}k^i_{62} \ldots k^i_0 = k_{79}k_{78} \ldots k_{16}.$$

After extracting the round key K_i, the key register $K = k_{79}k_{78} \ldots k_0$ is updated as follows.

1) $k_{79}k_{78} \ldots k_1k_0 \leftarrow k_{18}k_{17} \ldots k_{20}k_{19}$

2) $k_{79}k_{78}k_{77}k_{76} \leftarrow S[k_{79}k_{78}k_{77}k_{76}]$

3) $k_{19}k_{18}k_{17}k_{16}k_{15} \leftarrow k_{19}k_{18}k_{17}k_{16}k_{15} \oplus round\_counter$

In words, the key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the PRESENT S-box, and the $round\_counter$ value i is exclusive-ORed with bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ of K where the least significant bit of $round\_counter$ is on the right. The rounds are numbered from $1 \leq i \leq 31$ and $round\_counter = i$. Figure 4 depicts the key schedule for PRESENT-80 graphically.

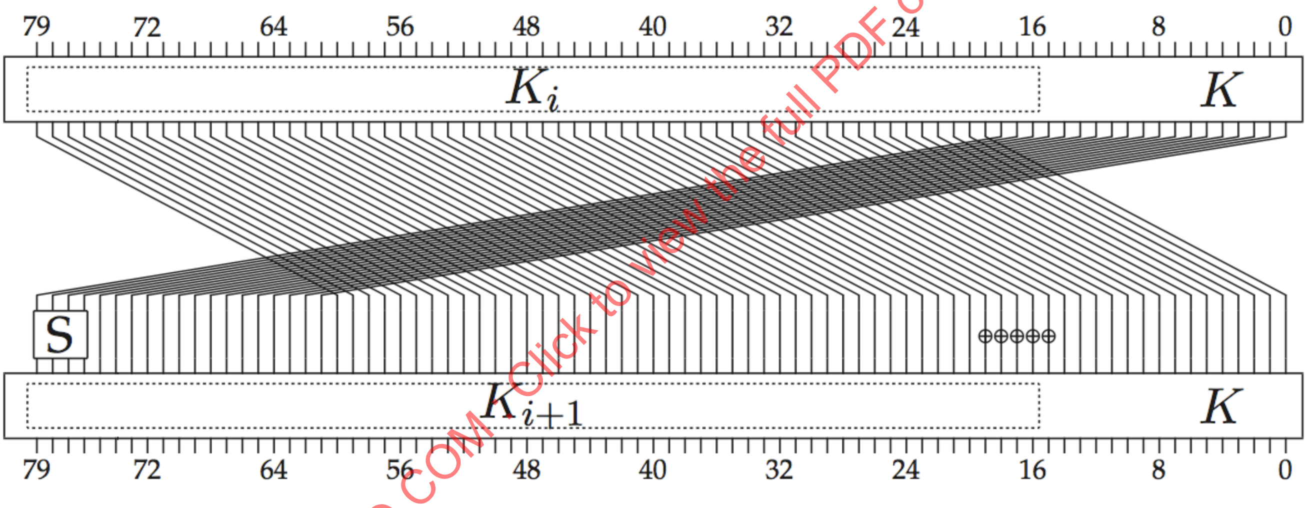

Figure 4 — PRESENT-80 key schedule

5.1.6.3 128-bit key for PRESENT-128

Similar to the 80-bit variant the user-supplied key is stored initially in a key register K and is represented as $k_{127}k_{126} \ldots k_0$. At round i the 64-bit round key $K_i = k^i_{63}k^i_{62} \ldots k^i_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = k^i_{63}k^i_{62} \ldots k^i_0 = k_{127}k_{126} \ldots k_{64}.$$

After extracting the round key K_i, the key register $K = k_{127}k_{126} \ldots k_0$ is updated as follows.

1) $k_{127}k_{126} \ldots k_1k_0 \leftarrow k_{66}k_{65} \ldots k_{68}k_{67}$

2) $k_{127}k_{126}k_{125}k_{124} \leftarrow S[k_{127}k_{126}k_{125}k_{124}]$

3) $k_{123}k_{122}k_{121}k_{120} \leftarrow S[k_{123}k_{122}k_{121}k_{120}]$

4) $k_{66}k_{65}k_{64}k_{63}k_{62} \leftarrow k_{66}k_{65}k_{64}k_{63}k_{62} \oplus round\_counter$

In words, the key register is rotated by 61 bit positions to the left, the left-most eight bits are passed through the PRESENT S-box, and the *round_counter* value i is exclusive-ORed with bits $k_{66}k_{65}k_{64}k_{63}k_{62}$ of K where the least significant bit of *round_counter* is on the right. The rounds are numbered from $1 \le i \le 31$ and *round_counter* $= i$. Figure 5 depicts the key schedule for PRESENT-128 graphically.

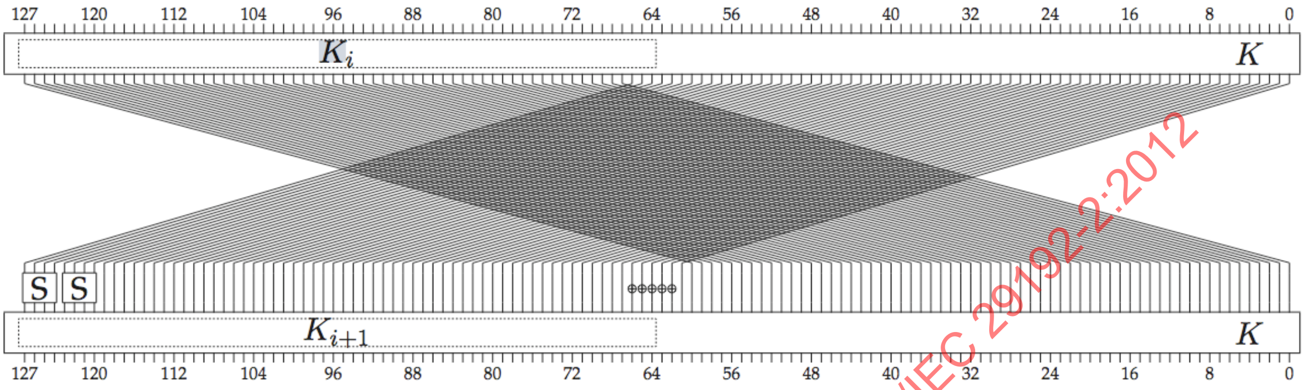

Figure 5 — PRESENT-128 key schedule

6 Lightweight block cipher with a block size of 128 bits

6.1 CLEFIA

6.1.1 CLEFIA algorithm

The CLEFIA algorithm [14] is a symmetric block cipher that can process data blocks of 128 bits using a cipher key of length 128, 192, or 256 bits. The number of rounds is 18, 22 and 26 for CLEFIA with 128-bit, 192-bit and 256-bit keys, respectively. The total number of round keys depends on the key length. The CLEFIA encryption and decryption functions require 36, 44 and 52 round keys for 128-bit, 192-bit and 256-bit keys, respectively.

6.1.2 CLEFIA specific notations

$a_{(b)}$ bit string of bit length b

$\{0,1\}^n$ A set of n-bit binary strings

$\bullet$ Multiplication in GF(2^n)

$<<< i$ i-bit left cyclic shift operation

$\tilde{}\ a$ Bitwise complement of bit string a

Σ^n n times operations of the DoubleSwap function Σ

6.1.3 CLEFIA encryption

The encryption process of CLEFIA is based on the 4-branch r-round generalized Feistel structure $GFN_{4,r}$. Let P, $C \in \{0,1\}^{128}$ be a plaintext and a ciphertext. Let P_i, $C_i \in \{0,1\}^{32}$ $(0 \leq i < 4)$ be divided plaintexts and ciphertexts where $P = P_0 \| P_1 \| P_2 \| P_3$ and $C = C_0 \| C_1 \| C_2 \| C_3$. Let WK_0, WK_1, WK_2, $WK_3 \in \{0,1\}^{32}$ be whitening keys and $RK_i \in \{0,1\}^{32}$ $(0 \leq i < 2r)$ be round keys provided by the key schedule. Then, r-round encryption function ENC_r is defined as follows:

ENC_r :

1) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow P_0 \| (P_1 \oplus WK_0) \| P_2 \| (P_3 \oplus WK_1)$

2) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow GFN_{4,r}(RK_0, ..., RK_{2r-1}, T_0, T_1, T_2, T_3)$

3) $C_0 \| C_1 \| C_2 \| C_3 \leftarrow T_0 \| (T_1 \oplus WK_2) \| T_2 \| (T_3 \oplus WK_3)$

6.1.4 CLEFIA decryption

The decryption function DEC_r is defined as follows:

DEC_r :

1) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow C_0 \| (C_1 \oplus WK_2) \| C_2 \| (C_3 \oplus WK_3)$

2) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow GFN_{4,r}^{-1}(RK_0, ..., RK_{2r-1}, T_0, T_1, T_2, T_3)$

3) $P_0 \| P_1 \| P_2 \| P_3 \leftarrow T_0 \| (T_1 \oplus WK_0) \| T_2 \| (T_3 \oplus WK_1)$

Figure 6 illustrates both ENC_r and DEC_r.

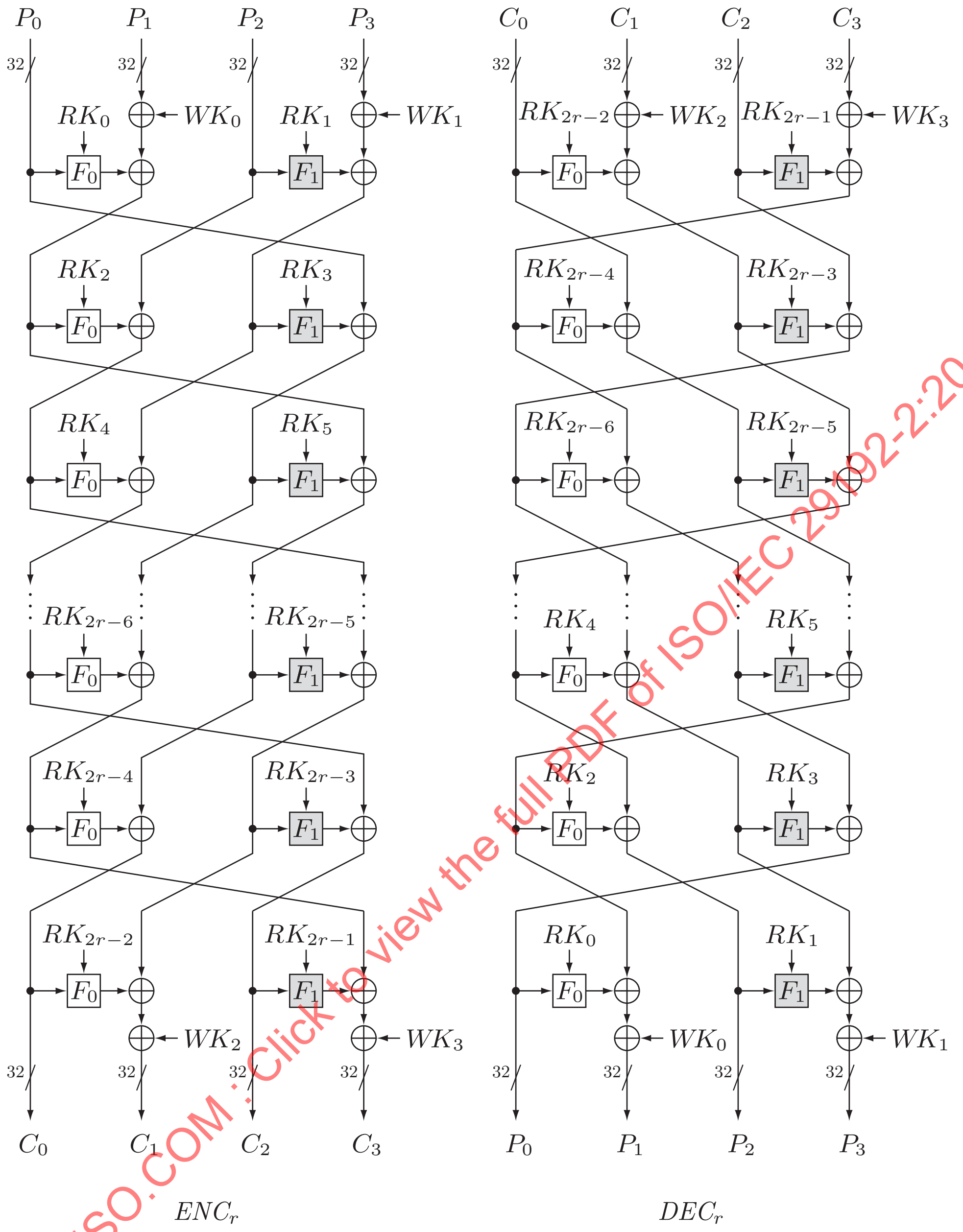

Figure 6 — The encryption procedure and the decryption procedure of CLEFIA

6.1.5 CLEFIA building blocks

6.1.5.1 GFN$_{d,r}$

The fundamental structure of CLEFIA is a generalized Feistel structure. This structure is employed in both a data processing part and a key schedule part.

CLEFIA uses a 4-branch and an 8-branch generalized Feistel network. The 4-branch generalized Feistel network is used in the data processing part and the key schedule for a 128-bit key. The 8-branch generalized Feistel network is applied in the key schedule for a 192-bit/256-bit key. We denote d-branch r-round generalized Feistel network employed in CLEFIA as $GFN_{d,r}$. $GFN_{d,r}$ uses two different 32-bit F-functions F_0 and F_1.

For d pairs of 32-bit input X_i and output Y_i $(0 \le i < d)$, and $dr/2$ 32-bit round keys RK_i $(0 \le i < dr/2)$, $GFN_{d,r}$ $(d = 4, 8)$ and the inverse function $GFN_{d,r}^{-1}$ $(d = 4)$ are defined as follows.

$GFN_{4,r}$:

1) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow X_0 \| X_1 \| X_2 \| X_3$

2) For $i = 0$ to $r - 1$ do the following:

 2.1) $T_1 \leftarrow T_1 \oplus F_0(RK_{2i}, T_0)$

 $T_3 \leftarrow T_3 \oplus F_1(RK_{2i+1}, T_2)$

 2.2) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow T_1 \| T_2 \| T_3 \| T_0$

3) $Y_0 \| Y_1 \| Y_2 \| Y_3 \leftarrow T_3 \| T_0 \| T_1 \| T_2$

$GFN_{8,r}$:

1) $T_0 \| T_1 \| ... \| T_7 \leftarrow X_0 \| X_1 \| ... \| X_7$

2) For $i = 0$ to $r - 1$ do the following:

 2.1) $T_1 \leftarrow T_1 \oplus F_0(RK_{4i}, T_0)$

 $T_3 \leftarrow T_3 \oplus F_1(RK_{4i+1}, T_2)$

 $T_5 \leftarrow T_5 \oplus F_0(RK_{4i+2}, T_4)$

 $T_7 \leftarrow T_7 \oplus F_1(RK_{4i+3}, T_6)$

 2.2) $T_0 \| T_1 \| ... \| T_6 \| T_7 \leftarrow T_1 \| T_2 \| ... \| T_7 \| T_0$

3) $Y_0 \| Y_1 \| ... \| Y_6 \| Y_7 \leftarrow T_7 \| T_0 \| ... \| T_5 \| T_6$

The inverse function $GFN_{4,r}^{-1}$ is obtained by changing the order of RK_i and the direction of word rotation at 2.2) and 3) in $GFN_{4,r}$.

$GFN_{4,r}^{-1}$:

1) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow X_0 \| X_1 \| X_2 \| X_3$

2) For $i = 0$ to $r - 1$ do the following:

 2.1) $T_1 \leftarrow T_1 \oplus F_0(RK_{2(r-i)-2}, T_0)$

 $T_3 \leftarrow T_3 \oplus F_1(RK_{2(r-i)-1}, T_2)$

 2.2) $T_0 \| T_1 \| T_2 \| T_3 \leftarrow T_3 \| T_0 \| T_1 \| T_2$

3) $Y_0 \| Y_1 \| Y_2 \| Y_3 \leftarrow T_1 \| T_2 \| T_3 \| T_0$

6.1.5.2 F-functions

Two F-functions F_0 and F_1 used in $GFN_{d,r}$ are defined as follows:

F_0: $(RK_{(32)}, x_{(32)}) \mapsto y_{(32)}$

1) $V \leftarrow RK \oplus x$

2) Let $V = V_0 \| V_1 \| V_2 \| V_3$, $V_i \in \{0,1\}^8$.

$V_0 \leftarrow S_0(V_0)$

$V_1 \leftarrow S_1(V_1)$

$V_2 \leftarrow S_0(V_2)$

$V_3 \leftarrow S_1(V_3)$

3) Let $y = y_0 \| y_1 \| y_2 \| y_3$, $y_i \in \{0,1\}^8$.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \leftarrow M_0 \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

F_1: $(RK_{(32)}, x_{(32)}) \mapsto y_{(32)}$

1) $V \leftarrow RK \oplus x$

2) Let $V = V_0 \| V_1 \| V_2 \| V_3$, $V_i \in \{0,1\}^8$.

$V_0 \leftarrow S_1(V_0)$

$V_1 \leftarrow S_0(V_1)$

$V_2 \leftarrow S_1(V_2)$

$V_3 \leftarrow S_0(V_3)$

3) Let $y = y_0 \| y_1 \| y_2 \| y_3$, $y_i \in \{0,1\}^8$.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \leftarrow M_1 \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

S_0 and S_1 are nonlinear 8-bit S-boxes, and M_0 and M_1 are 4x4 diffusion matrices described in the following clause. In each F-function two S-boxes and a matrix are used, but the S-boxes are used in a different order and the matrices differ. Figure 7 shows a graphical representation of the F-functions.

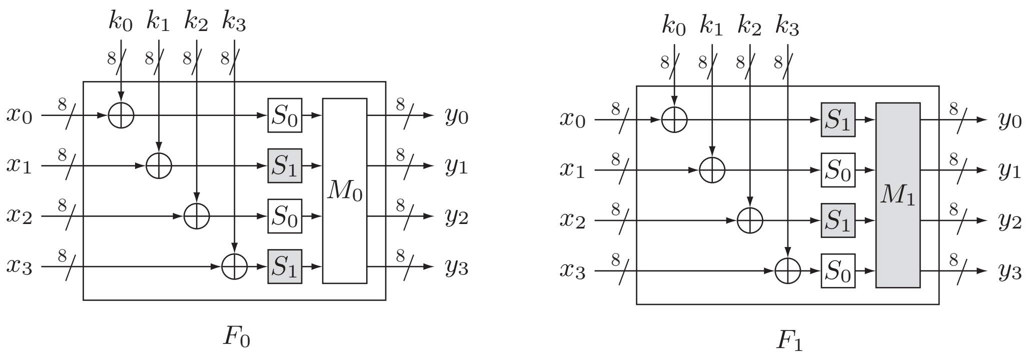

Figure 7 — F-functions

6.1.5.3 S-boxes

CLEFIA employs two different types of 8-bit S-boxes S_0 and S_1: S_0 is based on four 4-bit random S-boxes, and S_1 is based on the inverse function over GF(2^8).

Tables 5 and 6 show the output values of S_0 and S_1, respectively. In these tables all values are expressed in a hexadecimal notation. For an 8-bit input of an S-box, the upper 4 bits indicate a row and the lower 4 bits indicate a column. For example, if a value `0xab` is input, `0x7e` is output by S_0 because it is on the cross line of the row indexed by '`a.`' and the column indexed by '`.b`'.

Table 5 — S_0

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	57	49	d1	c6	2f	33	74	fb	95	6d	82	ea	0e	b0	a8	1c
1.	28	d0	4b	92	5c	ee	85	b1	c4	0a	76	3d	63	f9	17	af
2.	bf	a1	19	65	f7	7a	32	20	06	ce	e4	83	9d	5b	4c	d8
3.	42	5d	2e	e8	d4	9b	0f	13	3c	89	67	c0	71	aa	b6	f5
4.	a4	be	fd	8c	12	00	97	da	78	e1	cf	6b	39	43	55	26
5.	30	98	cc	dd	eb	54	b3	8f	4e	16	fa	22	a5	77	09	61
6.	d6	2a	53	37	45	c1	6c	ae	ef	70	08	99	8b	1d	f2	b4
7.	e9	c7	9f	4a	31	25	fe	7c	d3	a2	bd	56	14	88	60	0b
8.	cd	e2	34	50	9e	dc	11	05	2b	b7	a9	48	ff	66	8a	73
9.	03	75	86	f1	6a	a7	40	c2	b9	2c	db	1f	58	94	3e	ed
a.	fc	1b	a0	04	b8	8d	e6	59	62	93	35	7e	ca	21	df	47
b.	15	f3	ba	7f	a6	69	c8	4d	87	3b	9c	01	e0	de	24	52
c.	7b	0c	68	1e	80	b2	5a	e7	ad	d5	23	f4	46	3f	91	c9
d.	6e	84	72	bb	0d	18	d9	96	f0	5f	41	ac	27	c5	e3	3a
e.	81	6f	07	a3	79	f6	2d	38	1a	44	5e	b5	d2	ec	cb	90
f.	9a	36	e5	29	c3	4f	ab	64	51	f8	10	d7	bc	02	7d	8e

Table 6 — S_1

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	6c	da	c3	e9	4e	9d	0a	3d	b8	36	b4	38	13	34	0c	d9
1.	bf	74	94	8f	b7	9c	e5	dc	9e	07	49	4f	98	2c	b0	93
2.	12	eb	cd	b3	92	e7	41	60	e3	21	27	3b	e6	19	d2	0e
3.	91	11	c7	3f	2a	8e	a1	bc	2b	c8	c5	0f	5b	f3	87	8b
4.	fb	f5	de	20	c6	a7	84	ce	d8	65	51	c9	a4	ef	43	53
5.	25	5d	9b	31	e8	3e	0d	d7	80	ff	69	8a	ba	0b	73	5c
6.	6e	54	15	62	f6	35	30	52	a3	16	d3	28	32	fa	aa	5e
7.	cf	ea	ed	78	33	58	09	7b	63	c0	c1	46	1e	df	a9	99
8.	55	04	c4	86	39	77	82	ec	40	18	90	97	59	dd	83	1f
9.	9a	37	06	24	64	7c	a5	56	48	08	85	d0	61	26	ca	6f
a.	7e	6a	b6	71	a0	70	05	d1	45	8c	23	1c	f0	ee	89	ad
b.	7a	4b	c2	2f	db	5a	4d	76	67	17	2d	f4	cb	b1	4a	a8
c.	b5	22	47	3a	d5	10	4c	72	cc	00	f9	e0	fd	e2	fe	ae
d.	f8	5f	ab	f1	1b	42	81	d6	be	44	29	a6	57	b9	af	f2
e.	d4	75	66	bb	68	9f	50	02	01	3c	7f	8d	1a	88	bd	ac
f.	f7	e4	79	96	a2	fc	6d	b2	6b	03	e1	2e	7d	14	95	1d

a) S-box S_0

$S_0 : \{0,1\}^8 \rightarrow \{0,1\}^8 : x \mapsto y = S_0(x)$ is generated by combining four 4-bit S-boxes SS_0, SS_1, SS_2 and SS_3 in the following way. The values of these S-boxes are defined in Table 7.

1) $t_0 \leftarrow SS_0(x_0),\ t_1 \leftarrow SS_1(x_1),$ where $x = x_0 \,\|\, x_1,\ x_i \in \{0, 1\}^4$

2) $u_0 \leftarrow t_0 \oplus \texttt{0x2} \bullet t_1,\ u_1 \leftarrow \texttt{0x2} \bullet t_0 \oplus t_1$

3) $y_0 \leftarrow SS_2(u_0),\ y_1 \leftarrow SS_3(u_1),$ where $y = y_0 \,\|\, y_1,\ y_i \in \{0, 1\}^4$

The multiplication in $\texttt{0x2} \bullet t_i$ is performed in GF(2^4) defined by the lexicographically first primitive polynomial $z^4 + z + 1$. Figure 8 shows the construction of S_0.

Table 7 — SS_i ($0 \le i < 4$)

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$SS_0(x)$	e	6	c	a	8	7	2	f	b	1	4	0	5	9	d	3
$SS_1(x)$	6	4	0	d	2	b	a	3	9	c	e	f	8	7	5	1
$SS_2(x)$	b	8	5	e	a	6	4	c	f	7	2	3	1	0	d	9
$SS_3(x)$	a	2	6	d	3	4	5	e	0	7	8	9	b	f	c	1

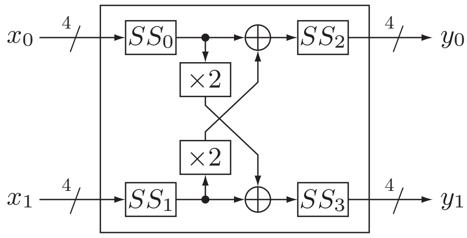

Figure 8 — S_0

b) S-box S_1

$S_1 : \{0,1\}^8 \rightarrow \{0,1\}^8 : x \mapsto y = S_1(x)$ is defined as follows:

$$y = \begin{cases} g((f(x))^{-1}) & \text{if} \quad f(x) \neq 0 \\ g(0) & \text{if} \quad f(x) = 0 \end{cases}$$

The inverse function is performed in GF(2^8) defined by a primitive polynomial $z^8 + z^4 + z^3 + z^2 + 1$ (= `0x11d`). f and g are affine transformations over GF(2), which are defined as follows.

$f : \{0,1\}^8 \rightarrow \{0,1\}^8 : x \mapsto y = f(x)$,

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$g : \{0,1\}^8 \rightarrow \{0,1\}^8 : x \mapsto y = g(x)$,

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Here, $x = x_0 \| x_1 \| x_2 \| x_3 \| x_4 \| x_5 \| x_6 \| x_7$ and $y = y_0 \| y_1 \| y_2 \| y_3 \| y_4 \| y_5 \| y_6 \| y_7$, $x_i, y_i \in \{0,1\}$. The constants in f and g can be represented as `0x1e` and `0x69`, respectively.

6.1.5.4 Diffusion matrices

The matrices M_0 and M_1 are defined as follows.

$$M_0 = \begin{pmatrix} 01 & 02 & 04 & 06 \\ 02 & 01 & 06 & 04 \\ 04 & 06 & 01 & 02 \\ 06 & 04 & 02 & 01 \end{pmatrix}, \quad M_1 = \begin{pmatrix} 01 & 08 & 02 & 0a \\ 08 & 01 & 0a & 02 \\ 02 & 0a & 01 & 08 \\ 0a & 02 & 08 & 01 \end{pmatrix}.$$

The multiplications of a matrix and a vector are performed in GF(2^8) defined by the lexicographically first primitive polynomial $z^8 + z^4 + z^3 + z^2 + 1$ (= `0x11d`).

6.1.6 CLEFIA key schedule

6.1.6.1 Overall structure

The key schedule of CLEFIA supports 128, 192 and 256-bit keys and outputs whitening keys WK_i $(0 \leq i < 4)$ and round keys RK_j $(0 \leq j < 2r)$ for the data processing part. Let K be the key and L be an intermediate key. The key schedule consists of the following two steps.

1) Generating L from K.

2) Expanding K and L (Generating WK_i and RK_j).

To generate L from K, the key schedule for a 128-bit key uses a 128-bit permutation $GFN_{4,12}$, while the key schedules for 192/256-bit keys use a 256-bit permutation $GFN_{8,10}$.

6.1.6.2 Key schedule for a 128-bit key

The 128-bit intermediate key L is generated in step 1 by applying $GFN_{4,12}$ which takes twenty-four 32-bit constant values $CON_i^{(128)}$ $(0 \leq i < 24)$ as round keys and $K = K_0 \| K_1 \| K_2 \| K_3$ as an input. Then K and L are used to generate WK_i $(0 \leq i < 4)$ and RK_j $(0 \leq j < 36)$ in steps 2 and 3. The thirty-six 32-bit constant values $CON_i^{(128)}$ $(24 \leq i < 60)$ used in step 3 are defined in 6.1.6.6. The DoubleSwap function Σ is defined in 6.1.6.5.

(Generating L from K)

1) $L \leftarrow GFN_{4,12}(CON_0^{(128)}, \ldots, CON_{23}^{(128)}, K_0, \ldots, K_3)$

(Expanding K and L)

2) $WK_0 \| WK_1 \| WK_2 \| WK_3 \leftarrow K$

3) For $i = 0$ to 8 do the following:

$T \leftarrow L \oplus (CON_{24+4i}^{(128)} \| CON_{24+4i+1}^{(128)} \| CON_{24+4i+2}^{(128)} \| CON_{24+4i+3}^{(128)})$

$L \leftarrow \Sigma(L)$

if i is odd: $T \leftarrow T \oplus K$

$RK_{4i} \| RK_{4i+1} \| RK_{4i+2} \| RK_{4i+3} \leftarrow T$

Table 8 shows the relationship between generated round keys and related data.

Table 8 — Expanding K and L (128-bit key)

$WK_0\ WK_1\ WK_2\ WK_3$	K
$RK_0\ RK_1\ RK_2\ RK_3$	$L \oplus (CON_{24}^{(128)} \parallel CON_{25}^{(128)} \parallel CON_{26}^{(128)} \parallel CON_{27}^{(128)})$
$RK_4\ RK_5\ RK_6\ RK_7$	$\Sigma(L) \oplus K \oplus (CON_{28}^{(128)} \parallel CON_{29}^{(128)} \parallel CON_{30}^{(128)} \parallel CON_{31}^{(128)})$
$RK_8\ RK_9\ RK_{10}\ RK_{11}$	$\Sigma^2(L) \oplus (CON_{32}^{(128)} \parallel CON_{33}^{(128)} \parallel CON_{34}^{(128)} \parallel CON_{35}^{(128)})$
$RK_{12}\ RK_{13}\ RK_{14}\ RK_{15}$	$\Sigma^3(L) \oplus K \oplus (CON_{36}^{(128)} \parallel CON_{37}^{(128)} \parallel CON_{38}^{(128)} \parallel CON_{39}^{(128)})$
$RK_{16}\ RK_{17}\ RK_{18}\ RK_{19}$	$\Sigma^4(L) \oplus (CON_{40}^{(128)} \parallel CON_{41}^{(128)} \parallel CON_{42}^{(128)} \parallel CON_{43}^{(128)})$
$RK_{20}\ RK_{21}\ RK_{22}\ RK_{23}$	$\Sigma^5(L) \oplus K \oplus (CON_{44}^{(128)} \parallel CON_{45}^{(128)} \parallel CON_{46}^{(128)} \parallel CON_{47}^{(128)})$
$RK_{24}\ RK_{25}\ RK_{26}\ RK_{27}$	$\Sigma^6(L) \oplus (CON_{48}^{(128)} \parallel CON_{49}^{(128)} \parallel CON_{50}^{(128)} \parallel CON_{51}^{(128)})$
$RK_{28}\ RK_{29}\ RK_{30}\ RK_{31}$	$\Sigma^7(L) \oplus K \oplus (CON_{52}^{(128)} \parallel CON_{53}^{(128)} \parallel CON_{54}^{(128)} \parallel CON_{55}^{(128)})$
$RK_{32}\ RK_{33}\ RK_{34}\ RK_{35}$	$\Sigma^8(L) \oplus (CON_{56}^{(128)} \parallel CON_{57}^{(128)} \parallel CON_{58}^{(128)} \parallel CON_{59}^{(128)})$

6.1.6.3 Key schedule for a 192-bit key

Two 128-bit values K_L and K_R are generated from a 192-bit key $K = K_0 \parallel K_1 \parallel K_2 \parallel K_3 \parallel K_4 \parallel K_5$, where $K_i \in \{0,1\}^{32}$. Then two 128-bit values L_L and L_R are generated by applying $GFN_{8,10}$ which takes $CON_i^{(192)}$ $(0 \leq i < 40)$ as round keys and $K_L \parallel K_R$ as a 256-bit input. Figure 9 shows the construction of $GFN_{8,10}$.

K_L, K_R and L_L, L_R are used to generate WK_i $(0 \leq i < 4)$ and RK_j $(0 \leq j < 44)$ in steps 4 and 5 below. In the latter part, forty-four 32-bit constant values $CON_i^{(192)}$ $(40 \leq i < 84)$ are used.

The following steps show the 192-bit/256-bit key schedule. For the 192-bit key schedule, the value of k is set as 192.

(Generating L_L, L_R from K_L, K_R for a k-bit key)

1) Set $k = 192$ or $k = 256$

2) If $k = 192$: $K_L \leftarrow K_0 \parallel K_1 \parallel K_2 \parallel K_3$, $K_R \leftarrow K_4 \parallel K_5 \parallel \sim K_0 \parallel \sim K_1$

   else if $k = 256$: $K_L \leftarrow K_0 \parallel K_1 \parallel K_2 \parallel K_3$, $K_R \leftarrow K_4 \parallel K_5 \parallel K_6 \parallel K_7$

3) Let $K_L = K_{L0} \parallel K_{L1} \parallel K_{L2} \parallel K_{L3}$, $K_R = K_{R0} \parallel K_{R1} \parallel K_{R2} \parallel K_{R3}$

   $L_L \parallel L_R \leftarrow GFN_{8,10}(CON_0^{(k)}, \ldots, CON_{39}^{(k)}, K_{L0}, \ldots, K_{L3}, K_{R0}, \ldots, K_{R3})$

(Expanding K_L, K_R and L_L, L_R for a k-bit key)

4) $WK_0 \parallel WK_1 \parallel WK_2 \parallel WK_3 \leftarrow K_L \oplus K_R$

5) For $i = 0$ to 10 (if $k = 192$), or 12 (if $k = 256$) do the following:

   If (i mod 4) = 0 or 1:

   $T \leftarrow L_L \oplus (CON_{40+4i}^{(k)} \parallel CON_{40+4i+1}^{(k)} \parallel CON_{40+4i+2}^{(k)} \parallel CON_{40+4i+3}^{(k)})$

   $L_L \leftarrow \Sigma(L_L)$

   if i is odd: $T \leftarrow T \oplus K_R$

else:

$T \leftarrow L_R \oplus (CON_{40+4i}^{(k)} \| CON_{40+4i+1}^{(k)} \| CON_{40+4i+2}^{(k)} \| CON_{40+4i+3}^{(k)})$

$L_R \leftarrow \Sigma(L_R)$

if i is odd: $T \leftarrow T \oplus K_L$

$RK_{4i} \| RK_{4i+1} \| RK_{4i+2} \| RK_{4i+3} \leftarrow T$

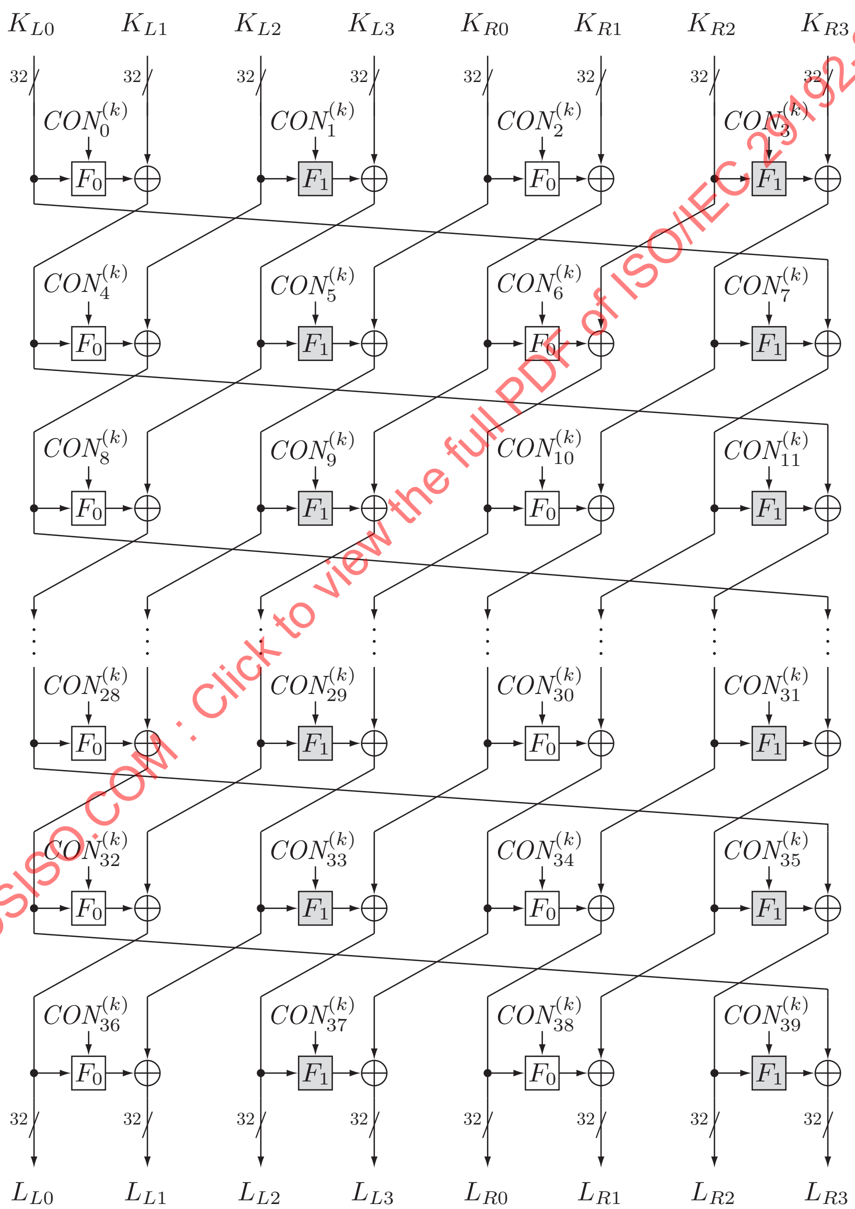

Figure 9 — Structure of $GFN_{8,10}$

Table 9 shows the relationship between generated round keys and related data.

Table 9 — Expanding K_L, K_R, L_L and L_R (192-bit key)

$WK_0\ WK_1\ WK_2\ WK_3$	$K_L \oplus K_R$
$RK_0\ RK_1\ RK_2\ RK_3$	$L_L \oplus (CON_{40}^{(192)} \Vert CON_{41}^{(192)} \Vert CON_{42}^{(192)} \Vert CON_{43}^{(192)})$
$RK_4\ RK_5\ RK_6\ RK_7$	$\Sigma(L_L) \oplus K_R \oplus (CON_{44}^{(192)} \Vert CON_{45}^{(192)} \Vert CON_{46}^{(192)} \Vert CON_{47}^{(192)})$
$RK_8\ RK_9\ RK_{10}\ RK_{11}$	$L_R \oplus (CON_{48}^{(192)} \Vert CON_{49}^{(192)} \Vert CON_{50}^{(192)} \Vert CON_{51}^{(192)})$
$RK_{12}\ RK_{13}\ RK_{14}\ RK_{15}$	$\Sigma(L_R) \oplus K_L \oplus (CON_{52}^{(192)} \Vert CON_{53}^{(192)} \Vert CON_{54}^{(192)} \Vert CON_{55}^{(192)})$
$RK_{16}\ RK_{17}\ RK_{18}\ RK_{19}$	$\Sigma^2(L_L) \oplus (CON_{56}^{(192)} \Vert CON_{57}^{(192)} \Vert CON_{58}^{(192)} \Vert CON_{59}^{(192)})$
$RK_{20}\ RK_{21}\ RK_{22}\ RK_{23}$	$\Sigma^3(L_L) \oplus K_R \oplus (CON_{60}^{(192)} \Vert CON_{61}^{(192)} \Vert CON_{62}^{(192)} \Vert CON_{63}^{(192)})$
$RK_{24}\ RK_{25}\ RK_{26}\ RK_{27}$	$\Sigma^2(L_R) \oplus (CON_{64}^{(192)} \Vert CON_{65}^{(192)} \Vert CON_{66}^{(192)} \Vert CON_{67}^{(192)})$
$RK_{28}\ RK_{29}\ RK_{30}\ RK_{31}$	$\Sigma^3(L_R) \oplus K_L \oplus (CON_{68}^{(192)} \Vert CON_{69}^{(192)} \Vert CON_{70}^{(192)} \Vert CON_{71}^{(192)})$
$RK_{32}\ RK_{33}\ RK_{34}\ RK_{35}$	$\Sigma^4(L_L) \oplus (CON_{72}^{(192)} \Vert CON_{73}^{(192)} \Vert CON_{74}^{(192)} \Vert CON_{75}^{(192)})$
$RK_{36}\ RK_{37}\ RK_{38}\ RK_{39}$	$\Sigma^5(L_L) \oplus K_R \oplus (CON_{76}^{(192)} \Vert CON_{77}^{(192)} \Vert CON_{78}^{(192)} \Vert CON_{79}^{(192)})$
$RK_{40}\ RK_{41}\ RK_{42}\ RK_{43}$	$\Sigma^4(L_R) \oplus (CON_{80}^{(192)} \Vert CON_{81}^{(192)} \Vert CON_{82}^{(192)} \Vert CON_{83}^{(192)})$

6.1.6.4 Key schedule for a 256-bit key

The key schedule for a 256-bit key is almost the same as that for 192-bit key, except for constant values, the required number of RK_i, and the initialization of K_R.

For a 256-bit key, the value of k is set as 256, and the steps are almost the same as in the 192-bit key case (see description in 6.1.6.3). The difference is that we use $CON_i^{(256)} (0 \le i < 40)$ as round keys to generate L_L and L_R, and then to generate $RK_j\ (0 \le j < 52)$, we use fifty-two 32-bit constant values $CON_i^{(256)} (40 \le i < 92)$.

Table 10 shows the relationship between generated round keys and related data.

Table 10 — Expanding K_L, K_R, L_L and L_R (256-bit key)

$WK_0\ WK_1\ WK_2\ WK_3$	$K_L \oplus K_R$
$RK_0\ RK_1\ RK_2\ RK_3$	$L_L \oplus (CON_{40}^{(256)} \Vert CON_{41}^{(256)} \Vert CON_{42}^{(256)} \Vert CON_{43}^{(256)})$
$RK_4\ RK_5\ RK_6\ RK_7$	$\Sigma(L_L) \oplus K_R \oplus (CON_{44}^{(256)} \Vert CON_{45}^{(256)} \Vert CON_{46}^{(256)} \Vert CON_{47}^{(256)})$
$RK_8\ RK_9\ RK_{10}\ RK_{11}$	$L_R \oplus (CON_{48}^{(256)} \Vert CON_{49}^{(256)} \Vert CON_{50}^{(256)} \Vert CON_{51}^{(256)})$
$RK_{12}\ RK_{13}\ RK_{14}\ RK_{15}$	$\Sigma(L_R) \oplus K_L \oplus (CON_{52}^{(256)} \Vert CON_{53}^{(256)} \Vert CON_{54}^{(256)} \Vert CON_{55}^{(256)})$
$RK_{16}\ RK_{17}\ RK_{18}\ RK_{19}$	$\Sigma^2(L_L) \oplus (CON_{56}^{(256)} \Vert CON_{57}^{(256)} \Vert CON_{58}^{(256)} \Vert CON_{59}^{(256)})$
$RK_{20}\ RK_{21}\ RK_{22}\ RK_{23}$	$\Sigma^3(L_L) \oplus K_R \oplus (CON_{60}^{(256)} \Vert CON_{61}^{(256)} \Vert CON_{62}^{(256)} \Vert CON_{63}^{(256)})$
$RK_{24}\ RK_{25}\ RK_{26}\ RK_{27}$	$\Sigma^2(L_R) \oplus (CON_{64}^{(256)} \Vert CON_{65}^{(256)} \Vert CON_{66}^{(256)} \Vert CON_{67}^{(256)})$
$RK_{28}\ RK_{29}\ RK_{30}\ RK_{31}$	$\Sigma^3(L_R) \oplus K_L \oplus (CON_{68}^{(256)} \Vert CON_{69}^{(256)} \Vert CON_{70}^{(256)} \Vert CON_{71}^{(256)})$
$RK_{32}\ RK_{33}\ RK_{34}\ RK_{35}$	$\Sigma^4(L_L) \oplus (CON_{72}^{(256)} \Vert CON_{73}^{(256)} \Vert CON_{74}^{(256)} \Vert CON_{75}^{(256)})$
$RK_{36}\ RK_{37}\ RK_{38}\ RK_{39}$	$\Sigma^5(L_L) \oplus K_R \oplus (CON_{76}^{(256)} \Vert CON_{77}^{(256)} \Vert CON_{78}^{(256)} \Vert CON_{79}^{(256)})$
$RK_{40}\ RK_{41}\ RK_{42}\ RK_{43}$	$\Sigma^4(L_R) \oplus (CON_{80}^{(256)} \Vert CON_{81}^{(256)} \Vert CON_{82}^{(256)} \Vert CON_{83}^{(256)})$
$RK_{44}\ RK_{45}\ RK_{46}\ RK_{47}$	$\Sigma^5(L_R) \oplus K_L \oplus (CON_{84}^{(256)} \Vert CON_{85}^{(256)} \Vert CON_{86}^{(256)} \Vert CON_{87}^{(256)})$
$RK_{48}\ RK_{49}\ RK_{50}\ RK_{51}$	$\Sigma^6(L_L) \oplus (CON_{88}^{(256)} \Vert CON_{89}^{(256)} \Vert CON_{90}^{(256)} \Vert CON_{91}^{(256)})$

6.1.6.5 DoubleSwap function

The DoubleSwap function $\Sigma : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ is defined as follows:

$$X_{(128)} \mapsto Y_{(128)}$$

$$Y = X[7\text{-}63] \parallel X[121\text{-}127] \parallel X[0\text{-}6] \parallel X[64\text{-}120],$$

where $X[a\text{-}b]$ denotes a bit string cut from the a-th bit to the b-th bit of X. Bit 0 is the most significant bit.

The DoubleSwap function is illustrated in Figure 10.

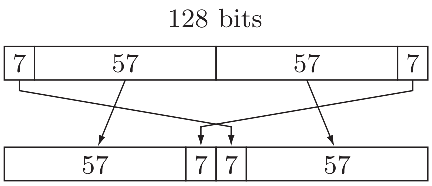

Figure 10 — DoubleSwap Function Σ

6.1.6.6 Constant values

32-bit constant values $CON_i^{(k)}$ are used in the key schedule algorithm. We need 60, 84 and 92 constant values for 128, 192 and 256-bit keys, respectively. Let $P_{(16)}$ = `0xb7e1` $(= (e-2)\,2^{16})$ and $Q_{(16)}$ = `0x243f` $(= (\pi - 3)\,2^{16})$, where e is the base of the natural logarithm (2.71828...) and π is the circle ratio (3.14159...). $CON_i^{(k)}$, for k = 128,192, 256 are generated in the following way (See Table 11 for the repetition numbers $l^{(k)}$ and the initial values $IV^{(k)}$).

1) $T_0^{(k)} \leftarrow IV^{(k)}$

2) For $i = 0$ to $l^{(k)} - 1$ do the following:

   2.1) $CON_{2i}^{(k)} \leftarrow (T_i^{(k)} \oplus P) \parallel (\tilde{}T_i^{(k)} <<< 1)$

   2.2) $CON_{2i+1}^{(k)} \leftarrow (\tilde{}T_i^{(k)} \oplus Q) \parallel (T_i^{(k)} <<< 8)$

   2.3) $T_{i+1}^{(k)} \leftarrow T_i^{(k)} \bullet$ `0x0002`$^{-1}$

In step 2.3, the multiplication is performed in the field GF(2^{16}) defined by a primitive polynomial $z^{16} + z^{15} + z^{13} + z^{11} + z^5 + z^4 + 1$ (= `0x1a831`)[1]) . `0x0002`$^{-1}$ is a constant denoting the multiplicative inverse of a finite field element z (= `0x0002`).

Table 11 — Required numbers of constant values

k	# of $CON_i^{(k)}$	$l^{(k)}$	$IV^{(k)}$
128	60	30	`0x428a` $(= (\sqrt[3]{2} - 1)\cdot 2^{16})$
192	84	42	`0x7137` $(= (\sqrt[3]{3} - 1)\cdot 2^{16})$
256	92	46	`0xb5c0` $(= (\sqrt[3]{5} - 1)\cdot 2^{16})$

Tables 12 - 14 show the values of $T_i^{(k)}$ and Tables 15 - 17 show the values of $CON_i^{(k)}$.

1) The lower 16-bit value is defined as `0xa831` = $(\sqrt[3]{101} - 4)\cdot 2^{16}$. '101' is the smallest prime number satisfying the primitive polynomial condition in this form.

Table 12 — $T_i^{(128)}$

i	0	1	2	3	4	5	6	7
$T_i^{(128)}$	428a	2145	c4ba	625d	e536	729b	ed55	a2b2
i	8	9	10	11	12	13	14	15
$T_i^{(128)}$	5159	fcb4	7e5a	3f2d	cb8e	65c7	e6fb	a765
i	16	17	18	19	20	21	22	23
$T_i^{(128)}$	87aa	43d5	f5f2	7af9	e964	74b2	3a59	c934
i	24	25	26	27	28	29		
$T_i^{(192)}$	649a	324d	cd3e	669f	e757	a7b3		

Table 13 — $T_i^{(192)}$

i	0	1	2	3	4	5	6	7
$T_i^{(192)}$	7137	ec83	a259	8534	429a	214d	c4be	625f
i	8	9	10	11	12	13	14	15
$T_i^{(192)}$	e537	a683	8759	97b4	4bda	25ed	c6ee	6377
i	16	17	18	19	20	21	22	23
$T_i^{(192)}$	e5a3	a6c9	877c	43be	21df	c4f7	b663	8f29
i	24	25	26	27	28	29	30	31
$T_i^{(192)}$	938c	49c6	24e3	c669	b72c	5b96	2dcb	c2fd
i	32	33	34	35	36	37	38	39
$T_i^{(192)}$	b566	5ab3	f941	a8b8	545c	2a2e	1517	de93
i	40	41						
$T_i^{(192)}$	bb51	89b0						

Table 14 — $T_i^{(256)}$

i	0	1	2	3	4	5	6	7
$T_i^{(256)}$	b5c0	5ae0	2d70	16b8	0b5c	05ae	02d7	d573
i	8	9	10	11	12	13	14	15
$T_i^{(256)}$	bea1	8b48	45a4	22d2	1169	dcac	6e56	372b
i	16	17	18	19	20	21	22	23
$T_i^{(256)}$	cf8d	b3de	59ef	f8ef	a86f	802f	940f	9e1f
i	24	25	26	27	28	29	30	31
$T_i^{(256)}$	9b17	9993	98d1	9870	4c38	261c	130e	0987
i	32	33	34	35	36	37	38	39
$T_i^{(256)}$	d0db	bc75	8a22	4511	f690	7b48	3da4	1ed2
i	40	41	42	43	44	45		
$T_i^{(256)}$	0f69	d3ac	69d6	34eb	ce6d	b32e		

Table 15 — $CON_i^{(128)}$

i	0	1	2	3
$CON_i^{(128)}$	f56b7aeb	994a8a42	96a4bd75	fa854521
i	4	5	6	7
$CON_i^{(128)}$	735b768a	1f7abac4	d5bc3b45	b99d5d62
i	8	9	10	11
$CON_i^{(128)}$	52d73592	3ef636e5	c57a1ac9	a95b9b72
i	12	13	14	15
$CON_i^{(128)}$	5ab42554	369555ed	1553ba9a	7972b2a2
i	16	17	18	19
$CON_i^{(128)}$	e6b85d4d	8a995951	4b550696	2774b4fc
i	20	21	22	23
$CON_i^{(128)}$	c9bb034b	a59a5a7e	88cc81a5	e4ed2d3f
i	24	25	26	27
$CON_i^{(128)}$	7c6f68e2	104e8ecb	d2263471	be07c765
i	28	29	30	31
$CON_i^{(128)}$	511a3208	3d3bfbe6	1084b134	7ca565a7
i	32	33	34	35
$CON_i^{(128)}$	304bf0aa	5c6aaa87	f4347855	9815d543
i	36	37	38	39
$CON_i^{(128)}$	4213141a	2e32f2f5	cd180a0d	a139f97a
i	40	41	42	43
$CON_i^{(128)}$	5e852d36	32a464e9	c353169b	af72b274
i	44	45	46	47
$CON_i^{(128)}$	8db88b4d	e199593a	7ed56d96	12f434c9
i	48	49	50	51
$CON_i^{(128)}$	d37b36cb	bf5a9a64	85ac9b65	e98d4d32
i	52	53	54	55
$CON_i^{(128)}$	7adf6582	16fe3ecd	d17e32c1	bd5f9f66
i	56	57	58	59
$CON_i^{(128)}$	50b63150	3c9757e7	1052b098	7c73b3a7

Table 16 — $CON_i^{(192)}$

i	0	1	2	3
$CON_i^{(192)}$	c6d61d91	aaf73771	5b6226f8	374383ec
i	4	5	6	7
$CON_i^{(192)}$	15b8bb4c	799959a2	32d5f596	5ef43485
i	8	9	10	11
$CON_i^{(192)}$	f57b7acb	995a9a42	96acbd65	fa8d4d21
i	12	13	14	15
$CON_i^{(192)}$	735f7682	1f7ebec4	d5be3b41	b99f5f62
i	16	17	18	19
$CON_i^{(192)}$	52d63590	3ef737e5	1162b2f8	7d4383a6
i	20	21	22	23
$CON_i^{(192)}$	30b8f14c	5c995987	2055d096	4c74b497
i	24	25	26	27
$CON_i^{(192)}$	fc3b684b	901ada4b	920cb425	fe2ded25
i	28	29	30	31
$CON_i^{(192)}$	710f7222	1d2eeec6	d4963911	b8b77763
i	32	33	34	35
$CON_i^{(192)}$	524234b8	3e63a3e5	1128b26c	7d09c9a6
i	36	37	38	39
$CON_i^{(192)}$	309df106	5cbc7c87	f45f7883	987ebe43
i	40	41	42	43
$CON_i^{(192)}$	963ebc41	fa1fdf21	73167610	1f37f7c4
i	44	45	46	47
$CON_i^{(192)}$	01829338	6da363b6	38c8e1ac	54e9298f
i	48	49	50	51
$CON_i^{(192)}$	246dd8e6	484c8c93	fe276c73	9206c649
i	52	53	54	55
$CON_i^{(192)}$	9302b639	ff23e324	7188732c	1da969c6
i	56	57	58	59
$CON_i^{(192)}$	00cd91a6	6cec2cb7	ec7748d3	8056965b
i	60	61	62	63
$CON_i^{(192)}$	9a2aa469	f60bcb2d	751c7a04	193dfdc2
i	64	65	66	67
$CON_i^{(192)}$	02879532	6ea666b5	ed524a99	8173b35a
i	68	69	70	71
$CON_i^{(192)}$	4ea00d7c	228141f9	1f59ae8e	7378b8a8
i	72	73	74	75
$CON_i^{(192)}$	e3bd5747	8f9c5c54	9dcfaba3	f1ee2e2a
i	76	77	78	79
$CON_i^{(192)}$	a2f6d5d1	ced71715	697242d8	055393de
i	80	81	82	83
$CON_i^{(192)}$	0cb0895c	609151bb	3e51ec9e	5270b089

Table 17 — $CON_i^{(256)}$

i	0	1	2	3
$CON_i^{(256)}$	0221947e	6e00c0b5	ed014a3f	8120e05a
i	4	5	6	7
$CON_i^{(256)}$	9a91a51f	f6b0702d	a159d28f	cd78b816
i	8	9	10	11
$CON_i^{(256)}$	bcbde947	d09c5c0b	b24ff4a3	de6eae05
i	12	13	14	15
$CON_i^{(256)}$	b536fa51	d917d702	62925518	0eb373d5
i	16	17	18	19
$CON_i^{(256)}$	094082bc	6561a1be	3ca9e96e	5088488b
i	20	21	22	23
$CON_i^{(256)}$	f24574b7	9e64a445	9533ba5b	f912d222
i	24	25	26	27
$CON_i^{(256)}$	a688dd2d	caa96911	6b4d46a6	076cacdc
i	28	29	30	31
$CON_i^{(256)}$	d9b72353	b596566e	80ca91a9	eceb2b37
i	32	33	34	35
$CON_i^{(256)}$	786c60e4	144d8dcf	043f9842	681edeb3
i	36	37	38	39
$CON_i^{(256)}$	ee0e4c21	822fef59	4f0e0e20	232feff8
i	40	41	42	43
$CON_i^{(256)}$	1f8eaf20	73af6fa8	37ceffa0	5bef2f80
i	44	45	46	47
$CON_i^{(256)}$	23eed7e0	4fcf0f94	29fec3c0	45df1f9e
i	48	49	50	51
$CON_i^{(256)}$	2cf6c9d0	40d7179b	2e72ccd8	42539399
i	52	53	54	55
$CON_i^{(256)}$	2f30ce5c	4311d198	2f91cf1e	43b07098
i	56	57	58	59
$CON_i^{(256)}$	fbd9678f	97f8384c	91fdb3c7	fddc1c26
i	60	61	62	63
$CON_i^{(256)}$	a4efd9e3	c8ce0e13	be66ecf1	d2478709
i	64	65	66	67
$CON_i^{(256)}$	673a5e48	0b1bdbd0	0b948714	67b575bc
i	68	69	70	71
$CON_i^{(256)}$	3dc3ebba	51e2228a	f2f075dd	9ed11145
i	72	73	74	75
$CON_i^{(256)}$	417112de	2d5090f6	cca9096f	a088487b
i	76	77	78	79
$CON_i^{(256)}$	8a4584b7	e664a43d	a933c25b	c512d21e
i	80	81	82	83
$CON_i^{(256)}$	b888e12d	d4a9690f	644d58a6	086cacd3
i	84	85	86	87
$CON_i^{(256)}$	de372c53	b216d669	830a9629	ef2beb34
i	88	89	90	91
$CON_i^{(256)}$	798c6324	15ad6dce	04cf99a2	68ee2eb3

Annex A
(normative)

Object identifiers

This annex lists the object identifiers assigned to algorithms specified in this part of ISO/IEC 29192.

```
--
-- ISO/IEC 29192-2 ASN.1 Module
--
LightweightCryptography-2 {
iso(1) standard(0) lightweight-cryptography(29192) part2(2)
asn1-module(0) algorithm-object-identifiers(0)}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
-- EXPORTS All; --
-- IMPORTS None; --
OID ::= OBJECT IDENTIFIER -- Alias
-- Synonyms --
is29192-2 OID ::= {iso(1) standard(0) lightweight-cryptography(29192)
part2(2)}
id-lbc64 OID ::= {is29192-2 cipher-64-bit(1)}
id-lbc128 OID ::= {is29192-2 cipher-128-bit(2)}
-- Assignments --
id-bc64-present OID ::= {id-lbc64 present(1)}
id-bc128-clefia OID ::= {id-lbc128 clefia(1)}
LightweightCryptographyIdentifier ::= SEQUENCE {
algorithm ALGORITHM.&id({BlockAlgorithms}),
parameters ALGORITHM.&Type({BlockAlgorithms}{@algorithm}) OPTIONAL
}
BlockAlgorithms ALGORITHM ::= {
{ OID id-bc64-present PARMS KeyLengthID } |
{ OID id-bc128-clefia PARMS KeyLengthID },
... -- Expect additional algorithms --
}
KeyLength ::= INTEGER
KeyLengthID ::= CHOICE {
int KeyLength,
oid OID
}
```

```
-- Cryptographic algorithm identification --
ALGORITHM ::= CLASS {
&id OBJECT IDENTIFIER UNIQUE,
&Type OPTIONAL
}
WITH SYNTAX {OID &id [PARMS &Type] }
END -- LightweightCryptography-2 --
```

Annex B
(informative)

Test vectors

This annex provides test vectors for PRESENT and CLEFIA for each key length in hexadecimal notation.

B.1 PRESENT test vectors

B.1.1 PRESENT-80

	Plaintext	**0123456789abcdef**		
	Key	**0123456789abcdef**	**0123**	
	Ciphertext	**f8dd50531d973bde**		
Round	**Round Key Value**	**After addRoundKey**	**After sLayer**	**After pLayer**
1	0123456789abcdef	0000000000000000	cccccccccccccccc	ffffffff00000000
2	1024602468acf135	efdb9fdb68acf135	1278e278a3f425b0	19a22a346eeaa266
3	8f37a2048c048d14	96958830e2ee2f72	eae033bc161162d6	e302a14bee4d0eb2
4	e3c4d1e6f4409181	00c670ad1a0d9f33	cc4adcf75fc7e2bb	de6beff8135f0bd3
5	62345c789a3cde8a	bc5fb3808963d559	84028b3c3eab700e	8d71414916f90698
6	92460c468b8f1345	1f374d0f9d7615dd	52bd97c2e7da5077	3ab096eb65d3bc6b
7	f37a3248c188d172	c9caa4a3a45b6d19	4e4ff9fbf908a75e	5fd9fa875b8d1fc6
8	0c4d1e6f46491832	5394e4e81dc407f4	0be919135749cd29	741d20ec61425fd5
9	0345c189a3cde8cd	7758e165c28fb718	dd0315a046328d53	c20cc4c61271dc27
10	f460c068b831347d	366c04aeaa40e85a	baa4c9f1ff9c130f	eef11ad1e2c587ed
11	f7a33e8c180d1703	1952245dfac890ee	5e0669072f43ec11	444cd96c59d88553
12	a4d1fef467d18304	e09d27983e090657	1ce76de3b1ceca0d	66bd7e393b9495c1
13	345c149a3fde8cfc	52e16aa3044a193d	0615affbc99f5eb7	0ff6569d4f17377b
14	360c068b829347fd	39fa5016cd847086	be2f0c5a4739dc3a	d51d56ccf163927a
15	fa33e6c180d17055	2f2eb00d71b2e22f	62618cc7d5861662	0ea0a7d6e11711c8
16	fd1fff467cd8301d	f3bf58909dcf21d5	2b8203ece7426570	638003eed6da4446
17	85c15fa3ffe8cf93	e6415c4d29328bd5	1a9504976eb63870	626415d241fab32a
18	a0c070b82bf47ff5	c2a4656a6a0eccdf	46f9a0afafc14472	3be0e16e6bc33152
19	833e54180e170577	b8deb57665d43425	837180daa079b960	8b9c222261aa723c
20	21ffd067ca8301cb	aa63f245ab2973f7	ffab2690f86edb2d	f2ddc4b9fcb6d28d
21	ac15c43ffa0cf95a	5ec8008606ba2bd7	0143cc3aca8f687d	0df52c9b135a5213
22	9c073582b887ff4b	91f21919abddad58	e5265e5ef877f703	85c8dfbcb5bd4abd
23	a3e57380e6b0571b	262dac3c530d1da6	6a67f4b40bc757fa	4a63bd3efa571a5e
24	dffd347cae701cdd	959e894254270683	e0e13e96096dca3b	a65da538ad271a53
25	815c7bffa68f95c2	2701dec70ba88f91	6dc5714dc8f332e5	61e2fba3883e5d39
26	9073702b8f7ff4dd	f1918b880741a9e4	25e53833cd95fe19	24ed70dcab0c5b7b
27	fe57120e6e0571e2	daba62d2c5092a99	7f8fa67640ce6fee	7837d7bfdf1fd204
28	0fd37fcae241cdcd	77e4a8753d5e1fc9	dd19f3d0b701524e	da81ca4b0cc5fed8
29	f5c781fa6ff95c46	2f464bb1633ca29e	629a9885abb4f6e1	3eea811ed0ee2969
30	47373eb8f03f4df1	79ddbfa620d16498	de7782fa6c75a9e3	cb4ef2f277abb235
31	b57108e6e7d71e08	7e3ffa14907cac3d	d1b22f59ecd4f4b7	a5ea86fd3c8be72b
32	5d37d6ae211cdcf5	f8dd50531d973bde		

B.1.2 PRESENT-128

	Plaintext	**0123456789abcdef**		
	Key	**0011223344556677**	**8899aabbccddeeff**	
	Ciphertext	**88728500054418de**		

Round	**Round Key Value**	**After addRoundKey**	**After sLayer**	**After pLayer**
1	0011223344556677	01326754cdfeab98	c5b6ad094721f8e3	ad0ed4ca386b6559
2	25133557799bbddf	881de19d41f0d886	335715e7952c733a	02913758d32ffdce
3	29004488cd115599	2b9173d01e3ea857	68e5db7c51b1f30d	6d29bb89a62c1efd
4	63944cd55de66ef7	0ebdf75cfbca700a	c1872d04284fdccf	a45f953f18915419
5	29a4011223344557	8dfb942d3ba5114e	3728e967b8f05591	1ce24b2ceba0c5af
6	178e5133557799ba	0b6c1a1fbed75c15	c8a45f52817d0450	e4909e3625200e72
7	0ca69004488cd114	e8360e326dacdf66	13bac1b6a7f472aa	3aa3097873efe668
8	e35e3944cd55de67	d9fd303cbeba380f	7e27bcb4818fb3c2	4ebad512fa1d9a5c
9	19329a4011223346	57884f52eb3fa91a	0d33920618b2fe5f	486d410f353d78ab
10	488d78e51335577b	00e039ea26082fd0	cc1cbe1f6ac3627c	dd61d5ab0dde2b12
11	d364ca69004488cf	0e051fc20d9aa3dd	c1c05246c7effb77	a0bcabfb057f485f
12	252235e3944cd55f	859e9e1891339d00	30e1e153e5bbe7cc	28bb2acfa9bc9774
13	1d4d9329a4011220	35f6b9e60dbd8554	b02a8e1ac7873009	9da104d0b5588259
14	c99488d78e513356	54358c073b09b10f	09b034cdb8ce85c2	63fa073628916984
15	4975364ca690044b	2a8f317a8e016dcf	6f32b5df31c5a742	4b28c736f98d6fd4
16	ab2652235e3944ce	e00e9515a7b42b1a	1cc1e050fd89685f	68f56acb088992d3
17	7525d4d9329a4015	1dd0be123a13d2c6	577c8156bf5b764a	18d1f36e61dde6f8
18	faac99488d78e517	e27d6a26eca503ef	16d7af6a14f0cb12	2d2c76685f25b4a6
19	17d4975364ca6904	3af8e13b3befdda2	bf2315b8b81277f6	c3c2440ff29fdeae
20	e8eab2652235e390	2b28f66ad0aa3d3e	68632aaf7cffb7b1	477aa1f4bfbe11bf
21	5c5f525d4d9329a1	1b25f3a9f22d381e	58602bfe2667b351	4708a3722ffc861f
22	6ba3aac99488d78b	2cab09bbbb745194	64f8ce8888d905e9	3ff3ec26a4022035
23	a5717d4975364ca3	9a82916fd1346c96	ef36e5a275b9a4ea	ca3bdcc6fbab64f0
24	a5ae8eab2652235b	6f95526dddf947ab	a2e006a7772e9df8	a21f25d6e7f201ce
25	d195c5f525d4d934	738ae023c226d8fa	db3f1c6b466a732f	d51196e9737ff90d
26	e696ba3aac99488b	33872cd3dfe6b186	bb3d647b721a853a	d1191e84ebd3f3a6
27	ad465717d4975362	7c5f49933f44a0c4	d4029eebb299fc49	8fbdc60e17c889b9
28	989a5ae8eab26524	17279ce6fd7aec9d	5d6de41a27df14e7	5932fc7729d3d279
29	e1b5195c5f525d4a	b887e52b76818f33	833d1068da3532bb	91c31290626f78bb
30	0662696ba3aac993	97a17bfbc1c5b128	edf5d82845408563	ed08f8e6a2037845
31	d886d465717d4972	358e2c83d37e3137	b031643b7bd1b5bd	816b0ca5abcab3ff
32	091989a5ae8eab21	88728500054418de		

B.2 CLEFIA test vectors

B.2.1 CLEFIA with a 128-bit key

key	ffeeddcc bbaa9988 77665544 33221100
plaintext	00010203 04050607 08090a0b 0c0d0e0f
ciphertext	de2bf2fd 9b74aacd f1298555 459494fd

L	8f89a61b 9db9d0f3 93e65627 da0d027e

$WK_{0,1,2,3}$	ffeeddcc bbaa9988 77665544 33221100
$RK_{0,1,2,3}$	f3e6cef9 8df75e38 41c06256 640ac51b
$RK_{4,5,6,7}$	6a27e20a 5a791b90 e8c528dc 00336ea3
$RK_{8,9,10,11}$	59cd17c4 28565583 312a37cc c08abd77
$RK_{12,13,14,15}$	7e8e7eec 8be7e949 d3f463d6 a0aad6aa
$RK_{16,17,18,19}$	e75eb039 0d657eb9 018002e2 9117d009
$RK_{20,21,22,23}$	9f98d11e babee8cf b0369efa d3aaef0d
$RK_{24,25,26,27}$	3438f93b f9cea4a0 68df9029 b869b4a7
$RK_{28,29,30,31}$	24d6406d e74bc550 41c28193 16de4795
$RK_{32,33,34,35}$	a34a20f5 33265d14 b19d0554 5142f434

plaintext		00010203	04050607	08090a0b	0c0d0e0f
initial whitening key			ffeeddcc		bbaa9988
after whitening		00010203	fbebdbcb	08090a0b	b7a79787
Round 1	input	00010203	fbebdbcb	08090a0b	b7a79787
	F-function	F_0		F_1	
	input	00010203		08090a0b	
	round key	f3e6cef9		8df75e38	
	after key add	f3e7ccfa		85fe5433	
	after S	290246e1		777de8e8	
	after M	547a3193		abf12070	
Round 2	input	af91ea58	08090a0b	1c56b7f7	00010203
	F-function	F_0		F_1	
	input	af91ea58		1c56b7f7	
	round key	41c06256		640ac51b	
	after key add	ee51880e		785c72ec	
	after S	cb5d2b0c		63a5edd2	
	after M	f51cebb3		82dfe347	
Round 3	input	fd15e1b8	1c56b7f7	82dee144	af91ea58
	F-function	F_0		F_1	
	input	fd15e1b8		82dee144	
	round key	6a27e20a		5a791b90	
	after key add	973203b2		d8a7fad4	
	after S	c2c7c6c2		be59e10d	
	after M	d8dfd8de		e15ea81c	
Round 4	input	c4896f29	82dee144	4ecf4244	fd15e1b8
	F-function	F_0		F_1	
	input	c4896f29		4ecf4244	
	round key	e8c528dc		00336ea3	
	after key add	2c4c47f5		4efc2ce7	
	after S	9da4dafc		43bce638	
	after M	b5b28e96		b65c519a	

Round 5	input	376c6fd2 4ecf4244	4b49b022 c4896f29
	F-function	F_0	F_1
	input	376c6fd2	4b49b022
	round key	59cd17c4	28565583
	after key add	6ea17816	631fe5a1
	after S	f26ad3e5	62af9f1b
	after M	29f08afd	be01d127
Round 6	input	673fc8b9 4b49b022	7a88be0e 376c6fd2
	F-function	F_0	F_1
	input	673fc8b9	7a88be0e
	round key	312a37cc	c08abd77
	after key add	5615ff75	ba020379
	after S	b39c8e58	2dd1e9a2
	after M	5999a79e	0429b329
Round 7	input	12d017bc 7a88be0e	3345dcfb 673fc8b9
	F-function	F_0	F_1
	input	12d017bc	3345dcfb
	round key	7e8e7eec	8be7e949
	after key add	6c5e6950	b8a235b2
	after S	8b737025	67a08eba
	after M	6ed11b09	dfd3cd32
Round 8	input	1459a507 3345dcfb	b8ec058b 12d017bc
	F-function	F_0	F_1
	input	1459a507	b8ec058b
	round key	d3f463d6	a0aad6aa
	after key add	c7adc6d1	1846d321
	after S	e7ee5a5f	9e97f1a1
	after M	8c9d011c	93684eec
Round 9	input	bfd8dde7 b8ec058b	81b85950 1459a507
	F-function	F_0	F_1
	input	bfd8dde7	81b85950
	round key	e75eb039	0d657eb9
	after key add	58866dde	8cdd27e9
	after S	4e821daf	59c56044
	after M	e6d6501e	6d5839b4
Round 10	input	5e3a5595 81b85950	79019cb3 bfd8dde7
	F-function	F_0	F_1
	input	5e3a5595	79019cb3
	round key	018002e2	9117d009
	after key add	5fba5777	e8164cba
	after S	612d8f7b	0185a49c
	after M	3a1b0e97	b9b479c8
Round 11	input	bba357c7 79019cb3	066ca42f 5e3a5595
	F-function	F_0	F_1
	input	bba357c7	066ca42f
	round key	9f98d11e	babee8cf
	after key add	243b86d9	bcd24ce0
	after S	f70f1144	cb72a481
	after M	28974052	4a6700b1
Round 12	input	5196dce1 066ca42f	145d5524 bba357c7
	F-function	F_0	F_1
	input	5196dce1	145d5524
	round key	b0369efa	d3aaef0d
	after key add	e1a0421b	c7f7ba29
	after S	6f7efd4f	72642dce
	after M	ffb5db32	907d3820

Round 13	input	f9d97f1d	145d5524	2bde6fe7	5196dce1
	F-function	F_0		F_1	
	input	f9d97f1d		2bde6fe7	
	round key	3438f93b		f9cea4a0	
	after key add	cde18626		d210cb47	
	after S	3f751141		ab28e0da	
	after M	0a744c28		1c3e38a3	
Round 14	input	1e29190c	2bde6fe7	4da8e442	f9d97f1d
	F-function	F_0		F_1	
	input	1e29190c		4da8e442	
	round key	68df9029		b869b4a7	
	after key add	76f68925		f5c150e5	
	after S	fe6db7e7		fc0c25f6	
	after M	aaa2c803		c4315b8d	
Round 15	input	817ca7e4	4da8e442	3de82490	1e29190c
	F-function	F_0		F_1	
	input	817ca7e4		3de82490	
	round key	24d6406d		e74bc550	
	after key add	a5aae789		daa3e1c0	
	after S	8d233818		2904757b	
	after M	7bd4cced		eac2f0fb	
Round 16	input	367c28af	3de82490	f4ebe9f7	817ca7e4
	F-function	F_0		F_1	
	input	367c28af		f4ebe9f7	
	round key	41c28193		16de4795	
	after key add	77bea93c		e235ae62	
	after S	7c4a935b		669b8953	
	after M	598e6940		c119609f	
Round 17	input	64664dd0	f4ebe9f7	4065c77b	367c28af
	F-function	F_0		F_1	
	input	64664dd0		4065c77b	
	round key	a34a20f5		33265d14	
	after key add	c72c6d25		73439a6f	
	after S	e7e61de7		788c85b4	
	after M	2ac01b0a		c755adfa	
Round 18	input	de2bf2fd	4065c77b	f1298555	64664dd0
	F-function	F_0		F_1	
	input	de2bf2fd		f1298555	
	round key	b19d0554		5142f434	
	after key add	6fb6f7a9		a06b7161	
	after S	b44d648c		7e99ea2a	
	after M	ac7738f2		12d0c82d	
output		de2bf2fd	ec12ff89	f1298555	76b685fd
final whitening key			77665544		33221100
after whitening		de2bf2fd	9b74aacd	f1298555	459494fd
ciphertext		de2bf2fd	9b74aacd	f1298555	459494fd

B.2.2 CLEFIA with a 192-bit key

key	ffeeddcc bbaa9988 77665544 33221100 f0e0d0c0 b0a09080
plaintext	00010203 04050607 08090a0b 0c0d0e0f
ciphertext	e2482f64 9f028dc4 80dda184 fde181ad

L_L	db05415a 800082db 7cb8186c d788c5f3
L_R	1ca9b2e1 b4606829 c92dd35e 2258a432
$WK_{0,1,2,3}$	0f0e0d0c 0b0a0908 77777777 77777777
$RK_{0,1,2,3}$	4d3bfd1b 7a1f5dfa 0fae6e7c c8bf3237
$RK_{4,5,6,7}$	73c2eeb8 dd429ec5 e220b3af c9135e73
$RK_{8,9,10,11}$	38c46a07 fc2ce4ba 370abf2d b05e627b
$RK_{12,13,14,15}$	38351b2f 74bd6e1e 1b7c7dce 92cfc98e
$RK_{16,17,18,19}$	509b31a6 4c5ad53c 6fc2ba33 e1e5c878
$RK_{20,21,22,23}$	419a74b9 1dd79e0e 240a33d2 9dabfd09
$RK_{24,25,26,27}$	6e3ff82a 74ac3ffd b9696e2e cc0b3a38
$RK_{28,29,30,31}$	ed785cbd 9c077c13 04978d83 2ec058ba
$RK_{32,33,34,35}$	4bbd5f6a 31fe8de8 b76da574 3a6fa8e7
$RK_{36,37,38,39}$	521213ce 4f1f59d8 c13624f6 ee91f6a4
$RK_{40,41,42,43}$	17f68fde f6c360a9 6288bc72 c0ad856b

plaintext		00010203	04050607	08090a0b	0c0d0e0f
initial whitening key			0f0e0d0c		0b0a0908
after whitening		00010203	0b0b0b0b	08090a0b	07070707
Round 1	input	00010203	0b0b0b0b	08090a0b	07070707
	F-function	F_0		F_1	
	input	00010203		08090a0b	
	round key	4d3bfd1b		7a1f5dfa	
	after key add	4d3aff18		721657f1	
	after S	43c58e9e		ed85d736	
	after M	b5021a3b		c397f62b	
Round 2	input	be091130	08090a0b	c490f12c	00010203
	F-function	F_0		F_1	
	input	be091130		c490f12c	
	round key	0fae6e7c		c8bf3237	
	after key add	b1a77f4c		0c2fc31b	
	after S	f3d10ba4		13d83a3d	
	after M	9fba69c1		6683cae3	
Round 3	input	97b363ca	c490f12c	6682c8e0	be091130
	F-function	F_0		F_1	
	input	97b363ca		6682c8e0	
	round key	73c2eeb8		dd429ec5	
	after key add	e4718d72		bbc05625	
	after S	79ea66ed		f47b0d7a	
	after M	61c21ea5		120e06e2	
Round 4	input	a552ef89	6682c8e0	ac0717d2	97b363ca
	F-function	F_0		F_1	
	input	a552ef89		ac0717d2	
	round key	e220b3af		c9135e73	
	after key add	47725c26		651449a1	
	after S	daeda541		355c651b	
	after M	28a43c63		cb1ab573	
Round 5	input	4e26f483	ac0717d2	5ca9d6b9	a552ef89
	F-function	F_0		F_1	
	input	4e26f483		5ca9d6b9	
	round key	38c46a07		fc2ce4ba	
	after key add	76e29e84		a0853203	
	after S	fe663e39		7edcc7c6	
	after M	5ce7dafe		ac7f4e3e	

Round 6	input	f0e0cd2c	5ca9d6b9	092da1b7	4e26f483
	F-function	F_0		F_1	
	input	f0e0cd2c		092da1b7	
	round key	370abf2d		b05e627b	
	after key add	c7ea7201		b973c3cc	
	after S	e77f9fda		174a3a46	
	after M	b9869270		8fc7e089	
Round 7	input	e52f44c9	092da1b7	c1e1140a	f0e0cd2c
	F-function	F_0		F_1	
	input	e52f44c9		c1e1140a	
	round key	38351b2f		74bd6e1e	
	after key add	dd1a5fe6		b55c7a14	
	after S	c5496150		5aa5c15c	
	after M	33d8590f		e62eb913	
Round 8	input	3af5f8b8	c1e1140a	16ce743f	e52f44c9
	F-function	F_0		F_1	
	input	3af5f8b8		16ce743f	
	round key	1b7c7dce		92cfc98e	
	after key add	21898576		8401bdb1	
	after S	a118dc09		3949b1f3	
	after M	f091202d		04f9e827	
Round 9	input	31703427	16ce743f	e1d6acee	3af5f8b8
	F-function	F_0		F_1	
	input	31703427		e1d6acee	
	round key	509b31a6		4c5ad53c	
	after key add	61eb0581		ad8c79d2	
	after S	2a8d3304		eeffc072	
	after M	f9639a90		8bebfe3d	
Round 10	input	efadeeaf	e1d6acee	b11e0685	31703427
	F-function	F_0		F_1	
	input	efadeeaf		b11e0685	
	round key	6fc2ba33		e1e5c878	
	after key add	806f549c		50fbcefd	
	after S	cd5eeb61		25d7fe02	
	after M	a100e35b		26a4e16d	
Round 11	input	40d64fb5	b11e0685	17d4d54a	efadeeaf
	F-function	F_0		F_1	
	input	40d64fb5		17d4d54a	
	round key	419a74b9		1dd79e0e	
	after key add	014c3b0c		0a034b44	
	after S	49a4c013		b4c6c912	
	after M	51c0208f		f1a2c339	
Round 12	input	e0de260a	17d4d54a	1e0f2d96	40d64fb5
	F-function	F_0		F_1	
	input	e0de260a		1e0f2d96	
	round key	240a33d2		9dabfd09	
	after key add	c4d415d8		83a4d09f	
	after S	801beebe		86b8f8ed	
	after M	8a9aef34		3e451646	