INTERNATIONAL STANDARD

ISO/IEC 18033-3

Second edition
2010-12-15

Information technology — Security techniques — Encryption algorithms —

Part 3: Block ciphers

Technologies de l'information — Techniques de sécurité — Algorithmes de chiffrement

Partie 3: Chiffrement par blocs

Reference number
ISO/IEC 18033-3:2010(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.

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

ISO/IEC 18033-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 18033-3:2005), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 18033-3:2005/Cor.1:2006, ISO/IEC 18033-3:2005/Cor.2:2007 and ISO/IEC 18033-3:2005/Cor.3:2008.

ISO/IEC 18033 consists of the following parts, under the general title *Information technology — Security techniques — Encryption algorithms*:

— *Part 1: General*

— *Part 2: Asymmetric ciphers*

— *Part 3: Block ciphers*

— *Part 4: Stream ciphers*

Information technology — Security techniques — Encryption algorithms —

Part 3: **Block ciphers**

1 Scope

This part of ISO/IEC 18033 specifies block ciphers. A block cipher maps blocks of *n* bits to blocks of *n* bits, under the control of a key of *k* bits. A total of seven different block ciphers are defined. They are categorized in Table 1.

Table 1 — Block ciphers specified

Block length	Algorithm name	(see #)	Key length
64 bits	TDEA	(4.2)	128 or 192 bits
	MISTY1	(4.3)	128 bits
	CAST-128	(4.4)	
	HIGHT	(4.5)	
128 bits	AES	(5.2)	128, 192 or 256 bits
	Camellia	(5.3)	
	SEED	(5.4)	128 bits

The algorithms specified in this part of ISO/IEC 18033 have been assigned object identifiers in accordance with ISO/IEC 9834. The list of assigned object identifiers is given in Annex B. Any changes to the specification of the algorithms resulting in a change of functional behaviour will result in a change of the object identifier assigned to the algorithm.

2 Terms and definitions

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

2.1
block
string of bits of defined length

NOTE In this part of ISO/IEC 18033, the block length is either 64 or 128 bits.

[ISO/IEC 18033-1:2005]

2.2
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 18033-1:2005]

2.3
ciphertext
data which has been transformed to hide its information content

[ISO/IEC 9798-1:1997]

2.4
key
sequence of symbols that controls the operation of a cryptographic transformation (e.g. encipherment, decipherment)

NOTE In all the ciphers specified in this part of ISO/IEC 18033, keys consist of a sequence of bits.

[ISO/IEC 11770-1:1996]

2.5
***n*-bit block cipher**
block cipher with the property that plaintext blocks and ciphertext blocks are n bits in length

[ISO/IEC 10116:2006]

2.6
plaintext
unenciphered information

[ISO/IEC 9797-1:1999]

3 Symbols

n plaintext/ciphertext bit length for a block cipher

E_K encryption function with key K

D_K decryption function with key K

Nr the number of rounds for the AES algorithm, which is 10, 12 or 14 for the choices of key length 128, 192 or 256 bits respectively

$\oplus$ the bit-wise logical exclusive-OR operation on bit-strings, i.e., if A, B are strings of the same length then $A \oplus B$ is the string equal to the bit-wise logical exclusive-OR of A and B

$\otimes$ multiplication of two polynomials (each with degree < 4) modulo $x^4 + 1$

$\wedge$ the bit-wise logical AND operation on bit-strings, i.e., if A, B are strings of the same length then $A \wedge B$ is the string equal to the bit-wise logical AND of A and B

$\vee$ the bit-wise logical OR operation on bit-strings, i.e., if A, B are strings of the same length then $A \vee B$ is the string equal to the bit-wise logical OR of A and B

|| concatenation of bit strings

• finite field multiplication

$<<<_i$ the left circular rotation of the operand by i bits

$>>>_i$ the right circular rotation of the operand by i bits

$\overline{x}$ the bitwise complement of *x*

a mod *n* for integers a and n, (a mod n) denotes the (non-negative) remainder obtained when a is divided by n. Equivalently if $b = a$ mod n, then b is the unique integer satisfying:
(i) $0 \leq b < n$, and
(ii) $(b-a)$ is an integer multiple of n

$\boxplus$ addition in modular arithmetic, i.e., if *A*, *B* are *t*-bit strings then $A \boxplus B$ is defined to equal ($A+B \bmod 2^t$)

$\boxminus$ subtraction in modular arithmetic, i.e., if *A*, *B* are *t*-bit strings then $A \boxminus B$ is defined to equal ($A-B \bmod 2^t$)

4 64-bit block ciphers

4.1 Introduction

In this clause, four 64-bit block ciphers are specified; TDEA (or 'Triple DES') in 4.2, MISTY1 in 4.3, CAST-128 in 4.4, and HIGHT in 4.5.

Users authorized to access data that has been enciphered shall have the key that was used to encipher the data in order to decipher it. The algorithm for any cipher in this clause is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 128- (or 192-) bit key. Deciphering shall be accomplished using the same key as for enciphering.

4.2 TDEA

4.2.1 The Triple Data Encryption Algorithm

The Triple Data Encryption Algorithm (TDEA) is a symmetric cipher that can process data blocks of 64 bits, using cipher keys with length of 128 (or 192) bits, of which 112 (or 168) bits can be chosen arbitrarily, and the rest may be used for error detection. The TDEA is commonly known as Triple DES (Data Encryption Standard).

A TDEA encryption/decryption operation is a compound operation of DES encryption and decryption operations, where the DES algorithm is specified in Annex A. A TDEA key consists of three DES keys.

4.2.2 TDEA encryption/decryption

4.2.2.1 Encryption/decryption definitions

The TDEA is defined in terms of DES operations, where E_K is the DES encryption operation for the key *K* and D_K is the DES decryption operation for the key *K*.

4.2.2.2 TDEA encryption

The transformation of a 64-bit block *P* into a 64-bit block *C* is defined as follows:

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P))).$$

4.2.2.3 TDEA decryption

The transformation of a 64-bit block C into a 64-bit block P is defined as follows:

$$P = D_{K_1}(E_{K_2}(D_{K_3}(C))) .$$

4.2.3 TDEA keying options

This part of ISO/IEC 18033 specifies the following keying options for TDEA. The TDEA key comprises the triple (K_1, K_2, K_3).

1. Keying Option 1: K_1, K_2 and K_3 are different DES keys;
2. Keying Option 2: K_1 and K_2 are different DES keys and $K_3 = K_1$.

NOTE The option that $K_1 = K_2 = K_3$, the single-DES equivalent, is not recommended. Furthermore, the use of keying option 1 is preferred over keying option 2 since it provides additional security at the same performance level (see [3] for further details).

4.3 MISTY1

4.3.1 The MISTY1 algorithm

The MISTY1 algorithm is a symmetric block cipher that can process data blocks of 64 bits, using a cipher key with length of 128 bits.

4.3.2 MISTY1 encryption

The encryption operation is as shown in Figure 1. The transformation of a 64-bit block P into a 64-bit block C is defined as follows (KL, KO and KI are keys):

(1) $P = L_0 \| R_0$

$KL = KL_1 \| KL_2 \| \ldots \| KL_{10}$

$KO = KO_1 \| KO_2 \| \ldots \| KO_8$

$KI = KI_1 \| KI_2 \| \ldots \| KI_8$

(2) for i = 1, 3, …, 7 (increment in steps of 2 because the loop body consists of two rounds):

$R_i = \mathrm{FL}(L_{i-1}, KL_i)$

$L_i = \mathrm{FL}(R_{i-1}, KL_{i+1}) \oplus \mathrm{FO}(R_i, KO_i, KI_i)$

$L_{i+1} = R_i \oplus \mathrm{FO}(L_i, KO_{i+1}, KI_{i+1})$

$R_{i+1} = L_i$

for i = 9:

$R_i = \mathrm{FL}(L_{i-1}, KL_i)$

$L_i = \mathrm{FL}(R_{i-1}, KL_{i+1})$

(3) $C = L_9 \| R_9$

4.3.3 MISTY1 decryption

The decryption operation is as shown in Figure 2, and is identical in operation to encryption apart from the following two modifications.

(1) All FL functions are replaced by their inverse functions FL^{-1}.

(2) The order in which the subkeys are applied is reversed.

4.3.4 MISTY1 functions

4.3.4.1 MISTY1 function definitions

The MISTY1 algorithm uses a number of functions, namely S_7, S_9, FI, FO, FL and FL^{-1}, which are now defined.

4.3.4.2 Function FL

The FL function is used in encryption only and is shown in Figure 3. The FL function is defined as follows (*X* and *Y* are data, *KL* is a key):

(1) $X_{32} = X_L \,||\, X_R,\ KL_i = KL_{iL} \,||\, KL_{iR}$

(2) $Y_R = (X_L \wedge KL_{iL}) \oplus X_R$

(3) $Y_L = X_L \oplus (Y_R \vee KL_{iR})$

(4) $Y_{32} = Y_L \,||\, Y_R$

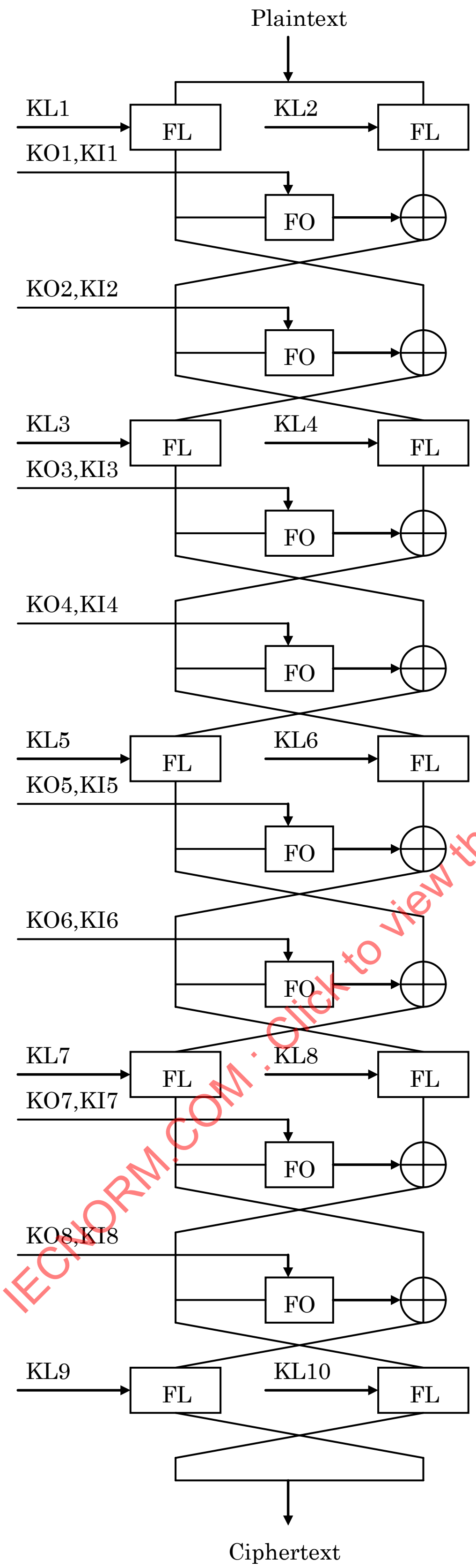

Figure 1 — The Encryption Procedure

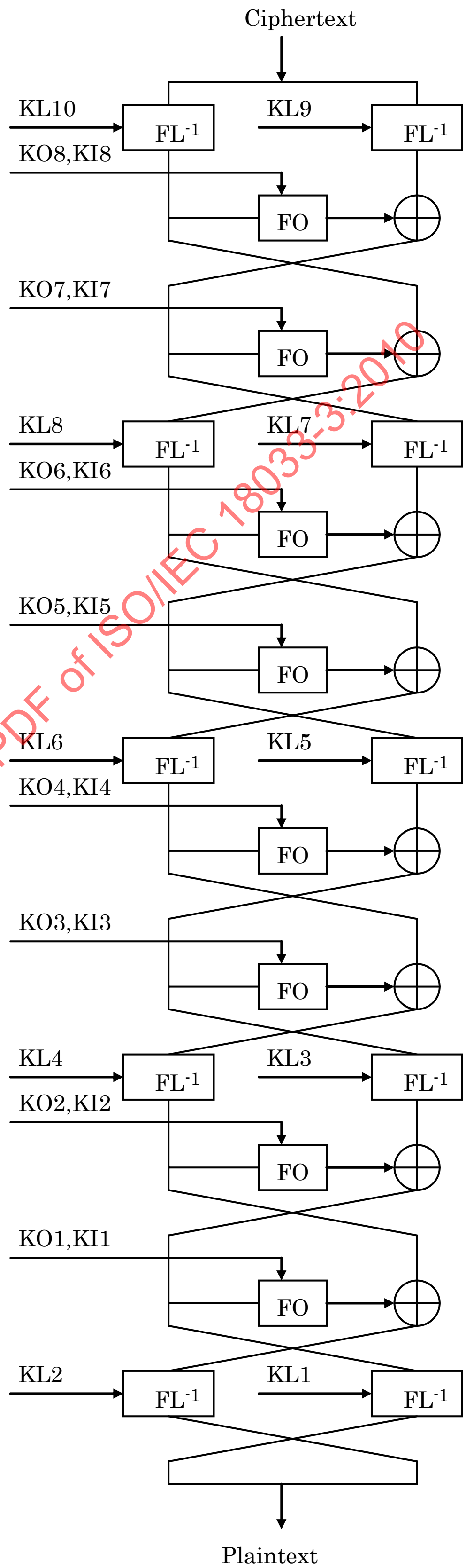

Figure 2 — The Decryption Procedure

4.3.4.3 Function FL^{-1}

The FL^{-1} function, which is the inverse to the FL function, is used in decryption only and is shown in Figure 4. The FL^{-1} function is defined as follows (X and Y are data, KL is a key):

(1) $Y_{32} = Y_L \,||\, Y_R$, $KL_i = KL_{iL} \,||\, KL_{iR}$

(2) $X_L = Y_L \oplus (Y_R \vee KL_{iR})$

(3) $X_R = (X_L \wedge KL_{iL}) \oplus Y_R$

(4) $X_{32} = X_L \,||\, X_R$

4.3.4.4 Function FO

The FO function is used in encryption and decryption, and is shown in Figure 5. The FO function is defined as follows (X and Y are data, KO and KI are keys):

(1) $X_{32} = L_0 \,||\, R_0$

$KO_i = KO_{i1} \,||\, KO_{i2} \,||\, KO_{i3} \,||\, KO_{i4}$, $KI_i = KI_{i1} \,||\, KI_{i2} \,||\, KI_{i3}$

(2) for j = 1 to 3 :

$R_j = \mathrm{FI}(L_{j-1} \oplus KO_{ij}, KI_{ij}) \oplus R_{j-1}$

$L_j = R_{j-1}$

(3) $Y_{32} = (L_3 \oplus KO_{i4}) \,||\, R_3$

4.3.4.5 Function FI

The FI function is used for encryption, decryption and the key schedule, and is shown in Figure 6, where Extnd is the operation zero-extended from 7 bits to 9 bits by the concatenation of two bits on the left side, and Trunc is the operation truncated by two bits on the left side. The FI function is defined as follows (X and Y are data, KI is a key):

(1) $X_{16} = L_0$ (9 bits) $||\, R_0$ (7 bits), $KI_{ij} = KI_{ijL} \,||\, KI_{ijR}$

(2) $R_1 = \mathrm{S}_9(L_0) \oplus \mathrm{Extnd}(R_0)$

(3) $L_1 = R_0$

(4) $R_2 = \mathrm{S7}(L_1) \oplus \mathrm{Trunc}(R_1) \oplus KI_{ijL}$

(5) $L_2 = R_1 \oplus KI_{ijR}$

(6) $R_3 = \mathrm{S}_9(L_2) \oplus \mathrm{Extnd}(R_2)$

(7) $L_3 = R_2$

(8) $Y_{16} = L_3 \,||\, R_3$

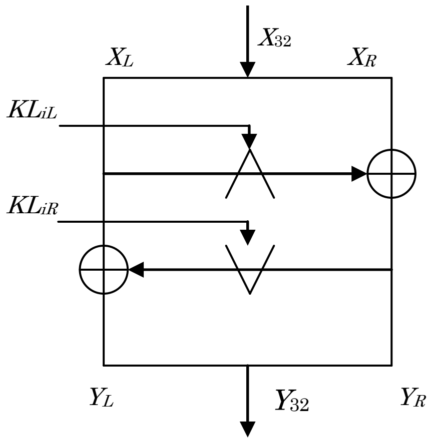

Figure 3 — The Function FL

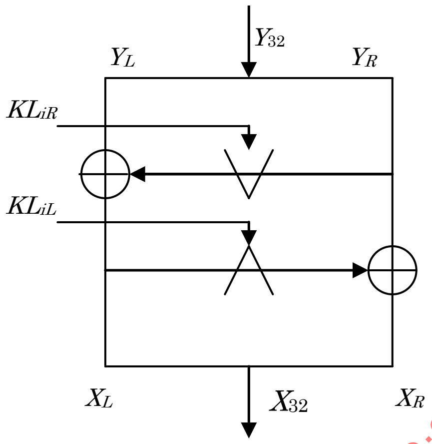

Figure 4 — The Function FL^{-1}

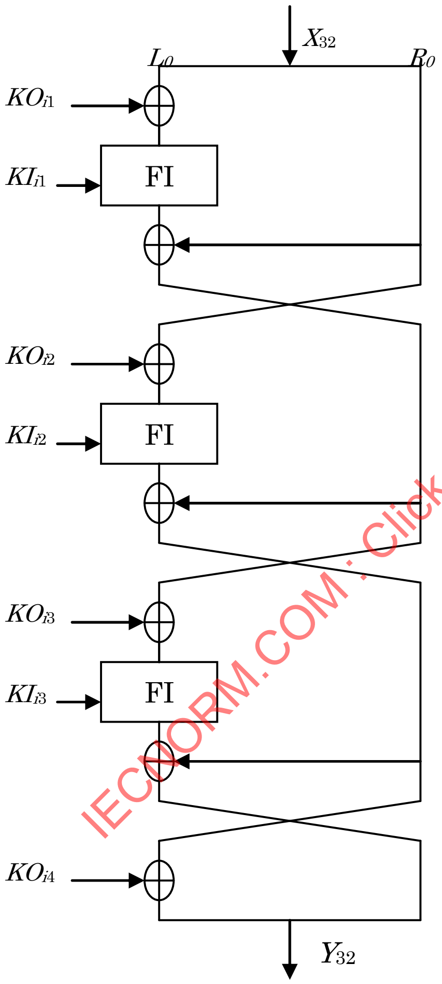

Figure 5 — The Function FO

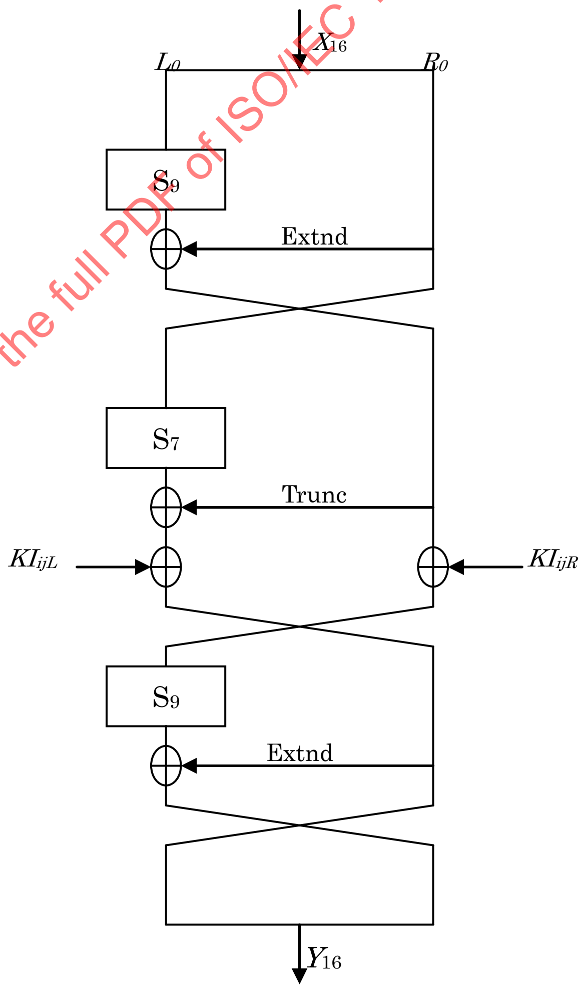

Figure 6 — The Function FI

4.3.4.6 Lookup Tables S_7 and S_9

S_7 is a bijective lookup table that accepts a 7-bit input and yields a 7-bit output. S_9 is a bijective lookup table that accepts a 9-bit input and yields a 9-bit output. Tables 2 and 3 define these lookup tables in a hexadecimal form. S_7 and S_9 can be also described in a simple algebraic form over GF(2) as shown in clause C.2.

For example, if the input to S_7 is {53}, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Table 2. This would result in S_7 having a value of {57}.

Table 2 — S_7

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	1b	32	33	5a	3b	10	17	54	5b	1a	72	73	6b	2c	66	49
1	1f	24	13	6c	37	2e	3f	4a	5d	0f	40	56	25	51	1c	04
2	0b	46	20	0d	7b	35	44	42	2b	1e	41	14	4b	79	15	6f
3	0e	55	09	36	74	0c	67	53	28	0a	7e	38	02	07	60	29
4	19	12	65	2f	30	39	08	68	5f	78	2a	4c	64	45	75	3d
5	59	48	03	57	7c	4f	62	3c	1d	21	5e	27	6a	70	4d	3a
6	01	6d	6e	63	18	77	23	05	26	76	00	31	2d	7a	7f	61
7	50	22	11	06	47	16	52	4e	71	3e	69	43	34	5c	58	7d

Table 3 — S_9

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	1c3	0cb	153	19f	1e3	0e9	0fb	035	181	0b9	117	1eb	133	009	02d	0d3
01	0c7	14a	037	07e	0eb	164	193	1d8	0a3	11e	055	02c	01d	1a2	163	118
02	14b	152	1d2	00f	02b	030	13a	0e5	111	138	18e	063	0e3	0c8	1f4	01b
03	001	09d	0f8	1a0	16d	1f3	01c	146	07d	0d1	082	1ea	183	12d	0f4	19e
04	1d3	0dd	1e2	128	1e0	0ec	059	091	011	12f	026	0dc	0b0	18c	10f	1f7
05	0e7	16c	0b6	0f9	0d8	151	101	14c	103	0b8	154	12b	1ae	017	071	00c
06	047	058	07f	1a4	134	129	084	15d	19d	1b2	1a3	048	07c	051	1ca	023
07	13d	1a7	165	03b	042	0da	192	0ce	0c1	06b	09f	1f1	12c	184	0fa	196
08	1e1	169	17d	031	180	10a	094	1da	186	13e	11c	060	175	1cf	067	119
09	065	068	099	150	008	007	17c	0b7	024	019	0de	127	0db	0e4	1a9	052
0a	109	090	19c	1c1	028	1b3	135	16a	176	0df	1e5	188	0c5	16e	1de	1b1
0b	0c3	1df	036	0ee	1ee	0f0	093	049	09a	1b6	069	081	125	00b	05e	0b4
0c	149	1c7	174	03e	13b	1b7	08e	1c6	0ae	010	095	1ef	04e	0f2	1fd	085
0d	0fd	0f6	0a0	16f	083	08a	156	09b	13c	107	167	098	1d0	1e9	003	1fe
0e	0bd	122	089	0d2	18f	012	033	06a	142	0ed	170	11b	0e2	14f	158	131
0f	147	05d	113	1cd	079	161	1a5	179	09e	1b4	0cc	022	132	01a	0e8	004
10	187	1ed	197	039	1bf	1d7	027	18b	0c6	09c	0d0	14e	06c	034	1f2	06e
11	0ca	025	0ba	191	0fe	013	106	02f	1ad	172	1db	0c0	10b	1d6	0f5	1ec
12	10d	076	114	1ab	075	10c	1e4	159	054	11f	04b	0c4	1be	0f7	029	0a4
13	00e	1f0	077	04d	17a	086	08b	0b3	171	0bf	10e	104	097	15b	160	168

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
14	0d7	0bb	066	1ce	0fc	092	1c5	06f	016	04a	0a1	139	0af	0f1	190	00a
15	1aa	143	17b	056	18d	166	0d4	1fb	14d	194	19a	087	1f8	123	0a7	1b8
16	141	03c	1f9	140	02a	155	11a	1a1	198	0d5	126	1af	061	12e	157	1dc
17	072	18a	0aa	096	115	0ef	045	07b	08d	145	053	05f	178	0b2	02e	020
18	1d5	03f	1c9	1e7	1ac	044	038	014	0b1	16b	0ab	0b5	05a	182	1c8	1d4
19	018	177	064	0cf	06d	100	199	130	15a	005	120	1bb	1bd	0e0	04f	0d6
1a	13f	1c4	12a	015	006	0ff	19b	0a6	043	088	050	15f	1e8	121	073	17e
1b	0bc	0c2	0c9	173	189	1f5	074	1cc	1e6	1a8	195	01f	041	00d	1ba	032
1c	03d	1d1	080	0a8	057	1b9	162	148	0d9	105	062	07a	021	1ff	112	108
1d	1c0	0a9	11d	1b0	1a6	0cd	0f3	05c	102	05b	1d9	144	1f6	0ad	0a5	03a
1e	1cb	136	17f	046	0e1	01e	1dd	0e6	137	1fa	185	08c	08f	040	1b5	0be
1f	078	000	0ac	110	15e	124	002	1bc	0a2	0ea	070	1fc	116	15c	04c	1c2

4.3.5 MISTY1 key schedule

The key scheduling part accepts a 128-bit key K and yields another 128-bit subkey K', as shown below. The figure of the key scheduling part is described in Figure 7.

The key scheduling operation is thus defined as follows.

(1) $K = K_1 \| K_2 \| K_3 \| K_4 \| K_5 \| K_6 \| K_7 \| K_8$

(2) for $i = 1$ to 7:

$K'_i = \mathrm{FI}(K_i, K_{i+1})$

(3) $K'_8 = \mathrm{FI}(K_8, K_1)$

(4) $K' = K'_1 \| K'_2 \| K'_3 \| K'_4 \| K'_5 \| K'_6 \| K'_7 \| K'_8$

(5) $KO_{i1} = K_i$, $KO_{i2} = K_{i+2}$, $KO_{i3} = K_{i+7}$, $KO_{i4} = K_{i+4}$, $KI_{i1} = K'_{i+5}$, $KI_{i2} = K'_{i+1}$, $KI_{i3} = K'_{i+3}$, ($i = 1,..,8$)

$KL_{iL} = K_{(i+1)/2}$ (odd i) or $K'_{(i/2)+2}$ (even i), $KL_{iR} = K'_{(i+1)/2+6}$ (odd i) or $K_{(i/2)+4}$ (even i) ($i = 1,..,10$)

NOTE When the value of a suffix is larger than 8, subtract by 8.

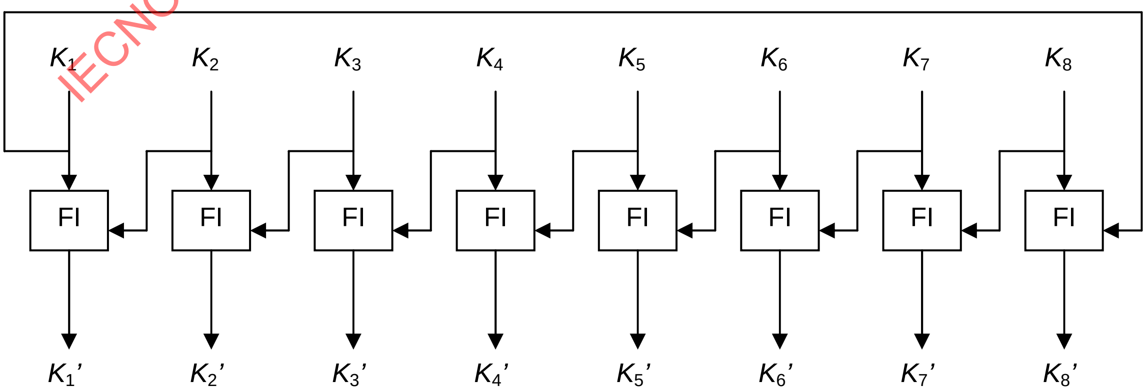

Figure 7 — MISTY1 Key Scheduling

4.4 CAST-128

4.4.1 The CAST-128 algorithm

The CAST-128 algorithm is a symmetric block cipher that can process data blocks of 64 bits, using a cipher key with length of 128 bits under 16 rounds.

NOTE The key length of the original version of CAST-128 is variable from 40 bits to 128 bits. This part of ISO/IEC 18033, however, specifies its use only with keys of 128 bits.

4.4.2 CAST-128 encryption

The transformation of a 64-bit block P into a 64-bit block C is defined as follows (Km and Kr are keys):

(1) $P = L_0 \parallel R_0$

(2) for i = 1 to 16:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, Km_i, Kr_i)$

(3) $C = R_{16} \parallel L_{16}$

4.4.3 CAST-128 decryption

The decryption operation is identical to the encryption operation given above, except that the rounds (and therefore the subkey pairs) are used in reverse order to compute (L_0, R_0) from (R_{16}, L_{16}).

4.4.4 CAST-128 functions

4.4.4.1 Pairs of Round Keys

CAST-128 uses a pair of subkeys per round: a 32-bit quantity Km is used as a "masking" key and a 5-bit quantity Kr is used as a "rotation" key.

4.4.4.2 f-functions

Three different round functions are used in the encryption and decryption operations, depending on the round number. The rounds are as follows (where "D" is the data input to the f function and "Ia" - "Id" are the most significant byte through least significant byte of I, respectively).

Type 1: $I = ((Km_i + D) \lll_{Kr_i})$

$\mathrm{f} = ((\mathrm{S1}[Ia] \oplus \mathrm{S2}[Ib]) - \mathrm{S3}[Ic]) + \mathrm{S4}[Id]$

Type 2: $I = ((Km_i \oplus D) \lll_{Kr_i})$

$\mathrm{f} = ((\mathrm{S1}[Ia] - \mathrm{S2}[Ib]) + \mathrm{S3}[Ic]) \oplus \mathrm{S4}[Id]$

Type 3: $I = ((Km_i - D) \lll_{Kr_i})$

$\mathrm{f} = ((\mathrm{S1}[Ia] + \mathrm{S2}[Ib]) \oplus \mathrm{S3}[Ic]) - \mathrm{S4}[Id]$

Rounds 1, 4, 7, 10, 13, and 16 use f function Type 1.

Rounds 2, 5, 8, 11, and 14 use f function Type 2.

Rounds 3, 6, 9, 12, and 15 use f function Type 3.

4.4.4.3 Substitution Boxes

4.4.4.3.1 The s-boxes S1 to S8

Eight substitution boxes are used: s-boxes S1, S2, S3, and S4 are round function s-boxes; S5, S6, S7, and S8 are key schedule s- boxes.

4.4.4.3.2 S-Box S1

30fb40d4	9fa0ff0b	6beccd2f	3f258c7a	1e213f2f	9c004dd3	6003e540	cf9fc949
bfd4af27	88bbbdb5	e2034090	98d09675	6e63a0e0	15c361d2	c2e7661d	22d4ff8e
28683b6f	c07fd059	ff2379c8	775f50e2	43c340d3	df2f8656	887ca41a	a2d2bd2d
a1c9e0d6	346c4819	61b76d87	22540f2f	2abe32e1	aa54166b	22568e3a	a2d341d0
66db40c8	a784392f	004dff2f	2db9d2de	97943fac	4a97c1d8	527644b7	b5f437a7
b82cbaef	d751d159	6ff7f0ed	5a097a1f	827b68d0	90ecf52e	22b0c054	bc8e5935
4b6d2f7f	50bb64a2	d2664910	bee5812d	b7332290	e93b159f	b48ee411	4bff345d
fd45c240	ad31973f	c4f6d02e	55fc8165	d5b1caad	a1ac2dae	a2d4b76d	c19b0c50
882240f2	0c6e4f38	a4e4bfd7	4f5ba272	564c1d2f	c59c5319	b949e354	b04669fe
b1b6ab8a	c71358dd	6385c545	110f935d	57538ad5	6a390493	e63d37e0	2a54f6b3
3a787d5f	6276a0b5	19a6fcdf	7a42206a	29f9d4d5	f61b1891	bb72275e	aa508167
38901091	c6b505eb	84c7cb8c	2ad75a0f	874a1427	a2d1936b	2ad286af	aa56d291
d7894360	425c750d	93b39e26	187184c9	6c00b32d	73e2bb14	a0bebc3c	54623779
64459eab	3f328b82	7718cf82	59a2cea6	04ee002e	89fe78e6	3fab0950	325ff6c2
81383f05	6963c5c8	76cb5ad6	d49974c9	ca180dcf	380782d5	c7fa5cf6	8ac31511
35e79e13	47da91d0	f40f9086	a7e2419e	31366241	051ef495	aa573b04	4a805d8d
548300d0	00322a3c	bf64cddf	ba57a68e	75c6372b	50afd341	a7c13275	915a0bf5
6b54bfab	2b0b1426	ab4cc9d7	449ccd82	f7fbf265	ab85c5f3	1b55db94	aad4e324
cfa4bd3f	2deaa3e2	9e204d02	c8bd25ac	eadf55b3	d5bd9e98	e31231b2	2ad5ad6c
954329de	adbe4528	d8710f69	aa51c90f	aa786bf6	22513f1e	aa51a79b	2ad344cc
7b5a41f0	d37cfbad	1b069505	41ece491	b4c332e6	032268d4	c9600acc	ce387e6d
bf6bb16c	6a70fb78	0d03d9c9	d4df39de	e01063da	4736f464	5ad328d8	b347cc96
75bb0fc3	98511bfb	4ffbcc35	b58bcf6a	e11f0abc	bfc5fe4a	a70aec10	ac39570a
3f04442f	6188b153	e0397a2e	5727cb79	9ceb418f	1cacd68d	2ad37c96	0175cb9d
c69dff09	c75b65f0	d9db40d8	ec0e7779	4744ead4	b11c3274	dd24cb9e	7e1c54bd
f01144f9	d2240eb1	9675b3fd	a3ac3755	d47c27af	51c85f4d	56907596	a5bb15e6
580304f0	ca042cf1	011a37ea	8dbfaadb	35ba3e4a	3526ffa0	c37b4d09	bc306ed9
98a52666	5648f725	ff5e569d	0ced63d0	7c63b2cf	700b45e1	d5ea50f1	85a92872
af1fbda7	d4234870	a7870bf3	2d3b4d79	42e04198	0cd0ede7	26470db8	f881814c
474d6ad7	7c0c5e5c	d1231959	381b7298	f5d2f4db	ab838653	6e2f1e23	83719c9e
bd91e046	9a56456e	dc39200c	20c8c571	962bda1c	e1e696ff	b141ab08	7cca89b9
1a69e783	02cc4843	a2f7c579	429ef47d	427b169c	5ac9f049	dd8f0f00	5c8165bf

4.4.4.3.3 S-Box S2

1f201094	ef0ba75b	69e3cf7e	393f4380	fe61cf7a	eec5207a	55889c94	72fc0651
ada7ef79	4e1d7235	d55a63ce	de0436ba	99c430ef	5f0c0794	18dcdb7d	a1d6eff3
a0b52f7b	59e83605	ee15b094	e9ffd909	dc440086	ef944459	ba83ccb3	e0c3cdfb
d1da4181	3b092ab1	f997f1c1	a5e6cf7b	01420ddb	e4e7ef5b	25a1ff41	e180f806
1fc41080	179bee7a	d37ac6a9	fe5830a4	98de8b7f	77e83f4e	79929269	24fa9f7b
e113c85b	acc40083	d7503525	f7ea615f	62143154	0d554b63	5d681121	c866c359
3d63cf73	cee234c0	d4d87e87	5c672b21	071f6181	39f7627f	361e3084	e4eb573b
602f64a4	d63acd9c	1bbc4635	9e81032d	2701f50c	99847ab4	a0e3df79	ba6cf38c
10843094	2537a95e	f46f6ffe	a1ff3b1f	208cfb6a	8f458c74	d9e0a227	4ec73a34
fc884f69	3e4de8df	ef0e0088	3559648d	8a45388c	1d804366	721d9bfd	a58684bb
e8256333	844e8212	128d8098	fed33fb4	ce280ae1	27e19ba5	d5a6c252	e49754bd
c5d655dd	eb667064	77840b4d	a1b6a801	84db26a9	e0b56714	21f043b7	e5d05860
54f03084	066ff472	a31aa153	dadc4755	b5625dbf	68561be6	83ca6b94	2d6ed23b
eccf01db	a6d3d0ba	b6803d5c	af77a709	33b4a34c	397bc8d6	5ee22b95	5f0e5304
81ed6f61	20e74364	b45e1378	de18639b	881ca122	b96726d1	8049a7e8	22b7da7b
5e552d25	5272d237	79d2951c	c60d894c	488cb402	1ba4fe5b	a4b09f6b	1ca815cf
a20c3005	8871df63	b9de2fcb	0cc6c9e9	0beeff53	e3214517	b4542835	9f63293c
ee41e729	6e1d2d7c	50045286	1e6685f3	f33401c6	30a22c95	31a70850	60930f13
73f98417	a1269859	ec645c44	52c877a9	cdff33a6	a02b1741	7cbad9a2	2180036f
50d99c08	cb3f4861	c26bd765	64a3f6ab	80342676	25a75e7b	e4e6d1fc	20c710e6
cdf0b680	17844d3b	31eef84d	7e0824e4	2ccb49eb	846a3bae	8ff77888	ee5d60f6
7af75673	2fdd5cdb	a11631c1	30f66f43	b3faec54	157fd7fa	ef8579cc	d152de58
db2ffd5e	8f32ce19	306af97a	02f03ef8	99319ad5	c242fa0f	a7e3ebb0	c68e4906
b8da230c	80823028	dcdef3c8	d35fb171	088a1bc8	bec0c560	61a3c9e8	bca8f54d
c72feffa	22822e99	82c570b4	d8d94e89	8b1c34bc	301e16e6	273be979	b0ffeaa6
61d9b8c6	00b24869	b7ffce3f	08dc283b	43daf65a	f7e19798	7619b72f	8f1c9ba4
dc8637a0	16a7d3b1	9fc393b7	a7136eeb	c6bcc63e	1a513742	ef6828bc	520365d6
2d6a77ab	3527ed4b	821fd216	095c6e2e	db92f2fb	5eea29cb	145892f5	91584f7f
5483697b	2667a8cc	85196048	8c4bacea	833860d4	0d23e0f9	6c387e8a	0ae6d249
b284600c	d835731d	dcb1c647	ac4c56ea	3ebd81b3	230eabb0	6438bc87	f0b5b1fa
8f5ea2b3	fc184642	0a036b7a	4fb089bd	649da589	a345415e	5c038323	3e5d3bb9
43d79572	7e6dd07c	06dfdf1e	6c6cc4ef	7160a539	73bfbe70	83877605	4523ecf1

4.4.4.3.4 S-Box S3

8defc240	25fa5d9f	eb903dbf	e810c907	47607fff	369fe44b	8c1fc644	aececa90
beb1f9bf	eefbcaea	e8cf1950	51df07ae	920e8806	f0ad0548	e13c8d83	927010d5
11107d9f	07647db9	b2e3e4d4	3d4f285e	b9afa820	fade82e0	a067268b	8272792e
553fb2c0	489ae22b	d4ef9794	125e3fbc	21fffcee	825b1bfd	9255c5ed	1257a240
4e1a8302	bae07fff	528246e7	8e57140e	3373f7bf	8c9f8188	a6fc4ee8	c982b5a5
a8c01db7	579fc264	67094f31	f2bd3f5f	40fff7c1	1fb78dfc	8e6bd2c1	437be59b
99b03dbf	b5dbc64b	638dc0e6	55819d99	a197c81c	4a012d6e	c5884a28	ccc36f71

b843c213	6c0743f1	8309893c	0feddd5f	2f7fe850	d7c07f7e	02507fbf	5afb9a04
a747d2d0	1651192e	af70bf3e	58c31380	5f98302e	727cc3c4	0a0fb402	0f7fef82
8c96fdad	5d2c2aae	8ee99a49	50da88b8	8427f4a0	1eac5790	796fb449	8252dc15
efbd7d9b	a672597d	ada840d8	45f54504	fa5d7403	e83ec305	4f91751a	925669c2
23efe941	a903f12e	60270df2	0276e4b6	94fd6574	927985b2	8276dbcb	02778176
f8af918d	4e48f79e	8f616ddf	e29d840e	842f7d83	340ce5c8	96bbb682	93b4b148
ef303cab	984faf28	779faf9b	92dc560d	224d1e20	8437aa88	7d29dc96	2756d3dc
8b907cee	b51fd240	e7c07ce3	e566b4a1	c3e9615e	3cf8209d	6094d1e3	cd9ca341
5c76460e	00ea983b	d4d67881	fd47572c	f76cedd9	bda8229c	127dadaa	438a074e
1f97c090	081bdb8a	93a07ebe	b938ca15	97b03cff	3dc2c0f8	8d1ab2ec	64380e51
68cc7bfb	d90f2788	12490181	5de5ffd4	dd7ef86a	76a2e214	b9a40368	925d958f
4b39fffa	ba39aee9	a4ffd30b	faf7933b	6d498623	193cbcfa	27627545	825cf47a
61bd8ba0	d11e42d1	cead04f4	127ea392	10428db7	8272a972	9270c4a8	127de50b
285ba1c8	3c62f44f	35c0eaa5	e805d231	428929fb	b4fcdf82	4fb66a53	0e7dc15b
1f081fab	108618ae	fcfd086d	f9ff2889	694bcc11	236a5cae	12deca4d	2c3f8cc5
d2d02dfe	f8ef5896	e4cf52da	95155b67	494a488c	b9b6a80c	5c8f82bc	89d36b45
3a609437	ec00c9a9	44715253	0a874b49	d773bc40	7c34671c	02717ef6	4feb5536
a2d02fff	d2bf60c4	d43f03c0	50b4ef6d	07478cd1	006e1888	a2e53f55	b9e6d4bc
a2048016	97573833	d7207d67	de0f8f3d	72f87b33	abcc4f33	7688c55d	7b00a6b0
947b0001	570075d2	f9bb88f8	8942019e	4264a5ff	856302e0	72dbd92b	ee971b69
6ea22fde	5f08ae2b	af7a616d	e5c98767	cf1febd2	61efc8c2	f1ac2571	cc8239c2
67214cb8	b1e583d1	b7dc3e62	7f10bdce	f90a5c38	0ff0443d	606e6dc6	60543a49
5727c148	2be98a1d	8ab41738	20e1be24	af96da0f	68458425	99833be5	600d457d
282f9350	8334b362	d91d1120	2b6d8da0	642b1e31	9c305a00	52bce688	1b03588a
f7baefd5	4142ed9c	a4315c11	83323ec5	dfef4636	a133c501	e9d3531c	ee353783

4.4.4.3.5 S-Box S4

9db30420	1fb6e9de	a7be7bef	d273a298	4a4f7bdb	64ad8c57	85510443	fa020ed1
7e287aff	e60fb663	095f35a1	79ebf120	fd059d43	6497b7b1	f3641f63	241e4adf
28147f5f	4fa2b8cd	c9430040	0cc32220	fdd30b30	c0a5374f	1d2d00d9	24147b15
ee4d111a	0fca5167	71ff904c	2d195ffe	1a05645f	0c13fefe	081b08ca	05170121
80530100	e83e5efe	ac9af4f8	7fe72701	d2b8ee5f	06df4261	bb9e9b8a	7293ea25
ce84ffdf	f5718801	3dd64b04	a26f263b	7ed48400	547eebe6	446d4ca0	6cf3d6f5
2649abdf	aea0c7f5	36338cc1	503f7e93	d3772061	11b638e1	72500e03	f80eb2bb
abe0502e	ec8d77de	57971e81	e14f6746	c9335400	6920318f	081dbb99	ffc304a5
4d351805	7f3d5ce3	a6c866c6	5d5bcca9	daec6fea	9f926f91	9f46222f	3991467d
a5bf6d8e	1143c44f	43958302	d0214eeb	022083b8	3fb6180c	18f8931e	281658e6
26486e3e	8bd78a70	7477e4c1	b506e07c	f32d0a25	79098b02	e4eabb81	28123b23
69dead38	1574ca16	df871b62	211c40b7	a51a9ef9	0014377b	041e8ac8	09114003
bd59e4d2	e3d156d5	4fe876d5	2f91a340	557be8de	00eae4a7	0ce5c2ec	4db4bba6
e756bdff	dd3369ac	ec17b035	06572327	99afc8b0	56c8c391	6b65811c	5e146119
6e85cb75	be07c002	c2325577	893ff4ec	5bbfc92d	d0ec3b25	b7801ab7	8d6d3b24
20c763ef	c366a5fc	9c382880	0ace3205	aac9548a	eca1d7c7	041afa32	1d16625a
6701902c	9b757a54	31d477f7	9126b031	36cc6fdb	c70b8b46	d9e66a48	56e55a79
026a4ceb	52437eff	2f8f76b4	0df980a5	8674cde3	edda04eb	17a9be04	2c18f4df

b7747f9d	ab2af7b4	efc34d20	2e096b7c	1741a254	e5b6a035	213d42f6	2c1c7c26
61c2f50f	6552daf9	d2c231f8	25130f69	d8167fa2	0418f2c8	001a96a6	0d1526ab
63315c21	5e0a72ec	49bafefd	187908d9	8d0dbd86	311170a7	3e9b640c	cc3e10d7
d5cad3b6	0caec388	f73001e1	6c728aff	71eae2a1	1f9af36e	cfcbd12f	c1de8417
ac07be6b	cb44a1d8	8b9b0f56	013988c3	b1c52fca	b4be31cd	d8782806	12a3a4e2
6f7de532	58fd7eb6	d01ee900	24adffc2	f4990fc5	9711aac5	001d7b95	82e5e7d2
109873f6	6.13E+05	c32d9521	ada121ff	29908415	7fbb977f	af9eb3db	29c9ed2a
5ce2a465	a730f32c	d0aa3fe8	8a5cc091	d49e2ce7	0ce454a9	d60acd86	015f1919
77079103	dea03af6	78a8565e	dee356df	21f05cbe	8b75e387	b3c50651	b8a5c3ef
d8eeb6d2	e523be77	c2154529	2f69efdf	afe67afb	f470c4b2	f3e0eb5b	d6cc9876
39e4460c	1fda8538	1987832f	ca007367	a99144f8	296b299e	492fc295	9266beab
b5676e69	9bd3ddda	df7e052f	db25701c	1b5e51ee	f65324e6	6afce36c	0316cc04
8644213e	b7dc59d0	7965291f	ccd6fd43	41823979	932bcdf6	b657c34d	4edfd282
7ae5290c	3cb9536b	851e20fe	9833557e	13ecf0b0	d3ffb372	3f85c5c1	0aef7ed2

4.4.4.3.6 S-Box S5

7ec90c04	2c6e74b9	9b0e66df	a6337911	b86a7fff	1dd358f5	44dd9d44	1731167f
08fbf1fa	e7f511cc	d2051b00	735aba00	2ab722d8	386381cb	acf6243a	69befd7a
e6a2e77f	f0c720cd	c4494816	ccf5c180	38851640	15b0a848	e68b18cb	4caadeff
5f480a01	0412b2aa	259814fc	41d0efe2	4e40b48d	248eb6fb	8dba1cfe	41a99b02
1a550a04	ba8f65cb	7251f4e7	95a51725	c106ecd7	97a5980a	c539b9aa	4d79fe6a
f2f3f763	68af8040	ed0c9e56	11b4958b	e1eb5a88	8709e6b0	d7e07156	4e29fea7
6366e52d	02d1c000	c4ac8e05	9377f571	0c05372a	578535f2	2261be02	d642a0c9
df13a280	74b55bd2	682199c0	d421e5ec	53fb3ce8	c8adedb3	28a87fc9	3d959981
5c1ff900	fe38d399	0c4eff0b	062407ea	aa2f4fb1	4fb96976	90c79505	b0a8a774
ef55a1ff	e59ca2c2	a6b62d27	e66a4263	df65001f	0ec50966	dfdd55bc	29de0655
911e739a	17af8975	32c7911c	89f89468	0d01e980	524755f4	03b63cc9	0cc844b2
bcf3f0aa	87ac36e9	e53a7426	01b3d82b	1a9e7449	64ee2d7e	cddbb1da	01c94910
b868bf80	0d26f3fd	9342ede7	04a5c284	636737b6	50f5b616	f24766e3	8eca36c1
136e05db	fef18391	fb887a37	d6e7f7d4	c7fb7dc9	3063fcdf	b6f589de	ec2941da
26e46695	b7566419	f654efc5	d08d58b7	48925401	c1bacb7f	e5ff550f	b6083049
5bb5d0e8	87d72e5a	ab6a6ee1	223a66ce	c62bf3cd	9e0885f9	68cb3e47	086c010f
a21de820	d18b69de	f3f65777	fa02c3f6	407edac3	cbb3d550	1793084d	b0d70eba
0ab378d5	d951fb0c	ded7da56	4124bbe4	94ca0b56	0f5755d1	e0e1e56e	6184b5be
580a249f	94f74bc0	e327888e	9f7b5561	c3dc0280	05687715	646c6bd7	44904db3
66b4f0a3	c0f1648a	697ed5af	49e92ff6	309e374f	2cb6356a	85808573	4991f840
76f0ae02	083be84d	28421c9a	44489406	736e4cb8	c1092910	8bc95fc6	7d869cf4
134f616f	2e77118d	b31b2be1	aa90b472	3ca5d717	7d161bba	9cad9010	af462ba2
9fe459d2	45d34559	d9f2da13	dbc65487	f3e4f94e	176d486f	097c13ea	631da5c7
445f7382	175683f4	cdc66a97	70be0288	b3cdcf72	6e5dd2f3	20936079	459b80a5
be60e2db	a9c23101	eba5315c	224e42f2	1c5c1572	f6721b2c	1ad2fff3	8c25404e
324ed72f	4067b7fd	0523138e	5ca3bc78	dc0fd66e	75922283	784d6b17	58ebb16e
44094f85	3f481d87	fcfeae7b	77b5ff76	8c2302bf	aaf47556	5f46b02a	2b092801
3d38f5f7	0ca81f36	52af4a8a	66d5e7c0	df3b0874	95055110	1b5ad7a8	f61ed5ad
6cf6e479	20758184	d0cefa65	88f7be58	4a046826	0ff6f8f3	a09c7f70	5346aba0
5ce96c28	e176eda3	6bac307f	376829d2	85360fa9	17e3fe2a	24b79767	f5a96b20
d6cd2595	68ff1ebf	7555442c	f19f06be	f9e0659a	eeb9491d	34010718	bb30cab8
e822fe15	88570983	750e6249	da627e55	5e76ffa8	b1534546	6d47de08	efe9e7d4

4.4.4.3.7 S-Box S6

f6fa8f9d	2cac6ce1	4ca34867	e2337f7c	95db08e7	016843b4	eced5cbc	325553ac
bf9f0960	dfa1e2ed	83f0579d	63ed86b9	1ab6a6b8	de5ebe39	f38ff732	8989b138
33f14961	c01937bd	f506c6da	e4625e7e	a308ea99	4e23e33c	79cbd7cc	48a14367
a3149619	fec94bd5	a114174a	eaa01866	a084db2d	09a8486f	a888614a	2900af98
01665991	e1992863	c8f30c60	2e78ef3c	d0d51932	cf0fec14	f7ca07d2	d0a82072
fd41197e	9305a6b0	e86be3da	74bed3cd	372da53c	4c7f4448	dab5d440	6dba0ec3
083919a7	9fbaeed9	49dbcfb0	4e670c53	5c3d9c01	64bdb941	2c0e636a	ba7dd9cd
ea6f7388	e70bc762	35f29adb	5c4cdd8d	f0d48d8c	b88153e2	08a19866	1ae2eac8
284caf89	aa928223	9334be53	3b3a21bf	16434be3	9aea3906	efe8c36e	f890cdd9
80226dae	c340a4a3	df7e9c09	a694a807	5b7c5ecc	221db3a6	9a69a02f	68818a54
ceb2296f	53c0843a	fe893655	25bfe68a	b4628abc	cf222ebf	25ac6f48	a9a99387
53bddb65	e76ffbe7	e967fd78	0ba93563	8e342bc1	e8a11be9	4980740d	c8087dfc
8de4bf99	a11101a0	7fd37975	da5a26c0	e81f994f	9528cd89	fd339fed	b87834bf
5f04456d	22258698	c9c4c83b	2dc156be	4f628daa	57f55ec5	e2220abe	d2916ebf
4ec75b95	24f2c3c0	42d15d99	cd0d7fa0	7b6e27ff	a8dc8af0	7345c106	f41e232f
35162386	e6ea8926	3333b094	157ec6f2	372b74af	692573e4	e9a9d848	f3160289
3a62ef1d	a787e238	f3a5f676	74364853	20951063	4576698d	b6fad407	592af950
36f73523	4cfb6e87	7da4cec0	6c152daa	cb0396a8	c50dfe5d	fcd707ab	0921c42f
89dff0bb	5fe2be78	448f4f33	754613c9	2b05d08d	48b9d585	dc049441	c8098f9b
7dede786	c39a3373	42410005	6a091751	0ef3c8a6	890072d6	28207682	a9a9f7be
bf32679d	d45b5b75	b353fd00	cbb0e358	830f220a	1f8fb214	d372cf08	cc3c4a13
8cf63166	061c87be	88c98f88	6062e397	47cf8e7a	b6c85283	3cc2acfb	3fc06976
4e8f0252	64d8314d	da3870e3	1e665459	c10908f0	513021a5	6c5b68b7	822f8aa0
3007cd3e	74719eef	dc872681	073340d4	7e432fd9	0c5ec241	8809286c	f592d891
08a930f6	957ef305	b7fbffbd	c266e96f	6fe4ac98	b173ecc0	bc60b42a	953498da
fba1ae12	2d4bd736	0f25faab	a4f3fceb	e2969123	257f0c3d	9348af49	361400bc
e8816f4a	3814f200	a3f94043	9c7a54c2	bc704f57	da41e7f9	c25ad33a	54f4a084
b17f5505	59357cbe	edbd15c8	7f97c5ab	ba5ac7b5	b6f6deaf	3a479c3a	5302da25
653d7e6a	54268d49	51a477ea	5017d55b	d7d25d88	44136c76	0404a8c8	b8e5a121
b81a928a	60ed5869	97c55b96	eaec991b	29935913	01fdb7f1	088e8dfa	9ab6f6f5
3b4cbf9f	4a5de3ab	e6051d35	a0e1d855	d36b4cf1	f544edeb	b0e93524	bebb8fbd
a2d762cf	49c92f54	38b5f331	7128a454	48392905	a65b1db8	851c97bd	d675cf2f

4.4.4.3.8 S-Box S7

85e04019	332bf567	662dbfff	cfc65693	2a8d7f6f	ab9bc912	de6008a1	2028da1f
0227bce7	4d642916	18fac300	50f18b82	2cb2cb11	b232e75c	4b3695f2	b28707de
a05fbcf6	cd4181e9	e150210c	e24ef1bd	b168c381	fde4e789	5c79b0d8	1e8bfd43
4d495001	38be4341	913cee1d	92a79c3f	089766be	baeeadf4	1286becf	b6eacb19
2660c200	7565bde4	64241f7a	8248dca9	c3b3ad66	28136086	0bd8dfa8	356d1cf2
107789be	b3b2e9ce	0502aa8f	0bc0351e	166bf52a	eb12ff82	e3486911	d34d7516
4e7b3aff	5f43671b	9cf6e037	4981ac83	334266ce	8c9341b7	d0d854c0	cb3a6c88
47bc2829	4725ba37	a66ad22b	7ad61f1e	0c5cbafa	4437f107	b6e79962	42d2d816
0a961288	e1a5c06e	13749e67	72fc081a	b1d139f7	f9583745	cf19df58	bec3f756
c06eba30	07211b24	45c28829	c95e317f	bc8ec511	38bc46e9	c6e6fa14	bae8584a
ad4ebc46	468f508b	7829435f	f124183b	821dba9f	aff60ff4	ea2c4e6d	16e39264
92544a8b	009b4fc3	aba68ced	9ac96f78	06a5b79a	b2856e6e	1aec3ca9	be838688
0e0804e9	55f1be56	e7e5363b	b3a1f25d	f7debb85	61fe033c	16746233	3c034c28
da6d0c74	79aac56c	3ce4e1ad	51f0c802	98f8f35a	1626a49f	eed82b29	1d382fe3
0c4fb99a	bb325778	3ec6d97b	6e77a6a9	cb658b5c	d45230c7	2bd1408b	60c03eb7
b9068d78	a33754f4	f430c87d	c8a71302	b96d8c32	ebd4e7be	be8b9d2d	7979fb06
e7225308	8b75cf77	11ef8da4	e083c858	8d6b786f	5a6317a6	fa5cf7a0	5dda0033

f28ebfb0	f5b9c310	a0eac280	08b9767a	a3d9d2b0	79d34217	021a718d	9ac6336a
2711fd60	438050e3	069908a8	3d7fedc4	826d2bef	4eeb8476	488dcf25	36c9d566
28e74e41	c2610aca	3d49a9cf	bae3b9df	b65f8de6	92aeaf64	3ac7d5e6	9ea80509
f22b017d	a4173f70	dd1e16c3	15e0d7f9	50b1b887	2b9f4fd5	625aba82	6a017962
2ec01b9c	15488aa9	d716e740	40055a2c	93d29a22	e32dbf9a	058745b9	3453dc1e
d699296e	496cff6f	1c9f4986	dfe2ed07	b87242d1	19de7eae	053e561a	15ad6f8c
66626c1c	7154c24c	ea082b2a	93eb2939	17dcb0f0	58d4f2ae	9ea294fb	52cf564c
9883fe66	2ec40581	763953c3	01d6692e	d3a0c108	a1e7160e	e4f2dfa6	693ed285
74904698	4c2b0edd	4f757656	5d393378	a132234f	3d321c5d	c3f5e194	4b269301
c79f022f	3c997e7e	5e4f9504	3ffafbbd	76f7ad0e	296693f4	3d1fce6f	c61e45be
d3b5ab34	f72bf9b7	1b0434c0	4e72b567	5592a33d	b5229301	cfd2a87f	60aeb767
1814386b	30bcc33d	38a0c07d	fd1606f2	c363519b	589dd390	5479f8e6	1cb8d647
97fd61a9	ea7759f4	2d57539d	569a58cf	e84e63ad	462e1b78	6580f87e	f3817914
91da55f4	40a230f3	d1988f35	b6e318d2	3ffa50bc	3d40f021	c3c0bdae	4958c24c
518f36b2	84b1d370	0fedce83	878ddada	f2a279c7	94e01be8	90716f4b	954b8aa3

4.4.4.3.9 S-Box S8

e216300d	bbddfffc	a7ebdabd	35648095	7789f8b7	e6c1121b	0e241600	052ce8b5
11a9cfb0	e5952f11	ece7990a	9386d174	2a42931c	76e38111	b12def3a	37ddddfc
de9adeb1	0a0cc32c	be197029	84a00940	bb243a0f	b4d137cf	b44e79f0	049eedfd
0b15a15d	480d3168	8bbbde5a	669ded42	c7ece831	3f8f95e7	72df191b	7580330d
94074251	5c7dcdfa	abbe6d63	aa402164	b301d40a	02e7d1ca	53571dae	7a3182a2
12a8ddec	fdaa335d	176f43e8	71fb46d4	38129022	ce949ad4	b84769ad	965bd862
82f3d055	66fb9767	15b80b4e	1d5b47a0	4cfde06f	c28ec4b8	57e8726e	647a78fc
99865d44	608bd593	6c200e03	39dc5ff6	5d0b00a3	ae63aff2	7e8bd632	70108c0c
bbd35049	2998df04	980cf42a	9b6df491	9e7edd53	06918548	58cb7e07	3b74ef2e
522fffb1	d24708cc	1c7e27cd	a4eb215b	3cf1d2e2	19b47a38	424f7618	35856039
9d17dee7	27eb35e6	c9aff67b	36baf5b8	09c467cd	c18910b1	e11dbf7b	06cd1af8
7170c608	2d5e3354	d4de495a	64c6d006	bcc0c62c	3dd00db3	708f8f34	77d51b42
264f620f	24b8d2bf	15c1b79e	46a52564	f8d7e54e	3e378160	7895cda5	859c15a5
e6459788	c37bc75f	db07ba0c	0676a3ab	7f229b1e	31842e7b	24259fd7	f8bef472
835ffcb8	6df4c1f2	96f5b195	fd0af0fc	b0fe134c	e2506d3d	4f9b12ea	f215f225
a223736f	9fb4c428	25d04979	34c713f8	c4618187	ea7a6e98	7cd16efc	1436876c
f1544107	bedeee14	56e9af27	a04aa441	3cf7c899	92ecbae6	dd67016d	151682eb
a842eedf	fdba60b4	f1907b75	20e3030f	24d8c29e	e139673b	efa63fb8	71873054
b6f2cf3b	9f326442	cb15a4cc	b01a4504	f1e47d8d	844a1be5	bae7dfdc	42cbda70
cd7dae0a	57e85b7a	d53f5af6	20cf4d8c	cea4d428	79d130a4	3486ebfb	33d3cddc
77853b53	37effcb5	c5068778	e580b3e6	4e68b8f4	c5c8b37e	0d809ea2	398feb7c
132a4f94	43b7950e	2fee7d1c	223613bd	dd06caa2	37df932b	c4248289	acf3ebc3
5715f6b7	ef3478dd	f267616f	c148cbe4	9052815e	5e410fab	b48a2465	2eda7fa4
e87b40e4	e98ea084	5889e9e1	efd390fc	dd07d35b	db485694	38d7e5b2	57720101
730edebc	5b643113	94917e4f	503c2fba	646f1282	7523d24a	e0779695	f9c17a8f
7a5b2121	d187b896	29263a4d	ba510cdf	81f47c9f	ad1163ed	ea7b5965	1a00726e
11403092	00da6d77	4a0cdd61	ad1f4603	605bdfb0	9eedc364	22ebe6a8	cee7d28a
a0e736a0	5564a6b9	10853209	c7eb8f37	2de705ca	8951570f	df09822b	bd691a6c
aa12e4f2	87451c0f	e0f6a27a	3ada4819	4cf1764f	0d771c2b	67cdb156	350d8384
5938fa0f	42399ef3	36997b07	0e84093d	4aa93e61	8360d87b	1fa98b0c	1149382c


```
e97625a5  0614d1b7  0e25244b  0c768347  589e8d82  0d2059d1  a466bb1e  f8da0a82
04f19130  ba6e4ec0  99265164  1ee7230d  50b2ad80  eaee6801  8db2a283  ea8bf59e
```

4.4.5 CAST-128 key schedule

Let the 128-bit key be x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF, where x0 represents the most significant byte and xF represents the least significant byte, and z0,...,zF be intermediate (temporary) bytes.

The subkeys are formed from the key x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF as follows.

$z0z1z2z3 = x0x1x2x3 \oplus S5[xD] \oplus S6[xF] \oplus S7[xC] \oplus S8[xE] \oplus S7[x8]$

$z4z5z6z7 = x8x9xAxB \oplus S5[z0] \oplus S6[z2] \oplus S7[z1] \oplus S8[z3] \oplus S8[xA]$

$z8z9zAzB = xCxDxExF \oplus S5[z7] \oplus S6[z6] \oplus S7[z5] \oplus S8[z4] \oplus S5[x9]$

$zCzDzEzF = x4x5x6x7 \oplus S5[zA] \oplus S6[z9] \oplus S7[zB] \oplus S8[z8] \oplus S6[xB]$

$K1 = S5[z8] \oplus S6[z9] \oplus S7[z7] \oplus S8[z6] \oplus S5[z2]$

$K2 = S5[zA] \oplus S6[zB] \oplus S7[z5] \oplus S8[z4] \oplus S6[z6]$

$K3 = S5[zC] \oplus S6[zD] \oplus S7[z3] \oplus S8[z2] \oplus S7[z9]$

$K4 = S5[zE] \oplus S6[zF] \oplus S7[z1] \oplus S8[z0] \oplus S8[zC]$

$x0x1x2x3 = z8z9zAzB \oplus S5[z5] \oplus S6[z7] \oplus S7[z4] \oplus S8[z6] \oplus S7[z0]$

$x4x5x6x7 = z0z1z2z3 \oplus S5[x0] \oplus S6[x2] \oplus S7[x1] \oplus S8[x3] \oplus S8[z2]$

$x8x9xAxB = z4z5z6z7 \oplus S5[x7] \oplus S6[x6] \oplus S7[x5] \oplus S8[x4] \oplus S5[z1]$

$xCxDxExF = zCzDzEzF \oplus S5[xA] \oplus S6[x9] \oplus S7[xB] \oplus S8[x8] \oplus S6[z3]$

$K5 = S5[x3] \oplus S6[x2] \oplus S7[xC] \oplus S8[xD] \oplus S5[x8]$

$K6 = S5[x1] \oplus S6[x0] \oplus S7[xE] \oplus S8[xF] \oplus S6[xD]$

$K7 = S5[x7] \oplus S6[x6] \oplus S7[x8] \oplus S8[x9] \oplus S7[x3]$

$K8 = S5[x5] \oplus S6[x4] \oplus S7[xA] \oplus S8[xB] \oplus S8[x7]$

$z0z1z2z3 = x0x1x2x3 \oplus S5[xD] \oplus S6[xF] \oplus S7[xC] \oplus S8[xE] \oplus S7[x8]$

$z4z5z6z7 = x8x9xAxB \oplus S5[z0] \oplus S6[z2] \oplus S7[z1] \oplus S8[z3] \oplus S8[xA]$

$z8z9zAzB = xCxDxExF \oplus S5[z7] \oplus S6[z6] \oplus S7[z5] \oplus S8[z4] \oplus S5[x9]$

$zCzDzEzF = x4x5x6x7 \oplus S5[zA] \oplus S6[z9] \oplus S7[zB] \oplus S8[z8] \oplus S6[xB]$

$K9 = S5[z3] \oplus S6[z2] \oplus S7[zC] \oplus S8[zD] \oplus S5[z9]$

$K10 = S5[z1] \oplus S6[z0] \oplus S7[zE] \oplus S8[zF] \oplus S6[zC]$

K11 = S5[z7] ⊕ S6[z6] ⊕ S7[z8] ⊕ S8[z9] ⊕ S7[z2]

K12 = S5[z5] ⊕ S6[z4] ⊕ S7[zA] ⊕ S8[zB] ⊕ S8[z6]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]

x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]

x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]

xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K13 = S5[x8] ⊕ S6[x9] ⊕ S7[x7] ⊕ S8[x6] ⊕ S5[x3]

K14 = S5[xA] ⊕ S6[xB] ⊕ S7[x5] ⊕ S8[x4] ⊕ S6[x7]

K15 = S5[xC] ⊕ S6[xD] ⊕ S7[x3] ⊕ S8[x2] ⊕ S7[x8]

K16 = S5[xE] ⊕ S6[xF] ⊕ S7[x1] ⊕ S8[x0] ⊕ S8[xD]

NOTE The remaining half is identical to what is given above, carrying on from the last created x0..xF to generate keys K17 - K32.

z0z1z2z3 = x0x1x2x3 ⊕ S5[xD] ⊕ S6[xF] ⊕ S7[xC] ⊕ S8[xE] ⊕ S7[x8]

z4z5z6z7 = x8x9xAxB ⊕ S5[z0] ⊕ S6[z2] ⊕ S7[z1] ⊕ S8[z3] ⊕ S8[xA]

z8z9zAzB = xCxDxExF ⊕ S5[z7] ⊕ S6[z6] ⊕ S7[z5] ⊕ S8[z4] ⊕ S5[x9]

zCzDzEzF = x4x5x6x7 ⊕ S5[zA] ⊕ S6[z9] ⊕ S7[zB] ⊕ S8[z8] ⊕ S6[xB]

K17 = S5[z8] ⊕ S6[z9] ⊕ S7[z7] ⊕ S8[z6] ⊕ S5[z2]

K18 = S5[zA] ⊕ S6[zB] ⊕ S7[z5] ⊕ S8[z4] ⊕ S6[z6]

K19 = S5[zC] ⊕ S6[zD] ⊕ S7[z3] ⊕ S8[z2] ⊕ S7[z9]

K20 = S5[zE] ⊕ S6[zF] ⊕ S7[z1] ⊕ S8[z0] ⊕ S8[zC]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]

x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]

x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]

xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K21 = S5[x3] ⊕ S6[x2] ⊕ S7[xC] ⊕ S8[xD] ⊕ S5[x8]

K22 = S5[x1] ⊕ S6[x0] ⊕ S7[xE] ⊕ S8[xF] ⊕ S6[xD]

K23 = S5[x7] ⊕ S6[x6] ⊕ S7[x8] ⊕ S8[x9] ⊕ S7[x3]

K24 = S5[x5] ⊕ S6[x4] ⊕ S7[xA] ⊕ S8[xB] ⊕ S8[x7]

$z0z1z2z3 = x0x1x2x3 \oplus S5[xD] \oplus S6[xF] \oplus S7[xC] \oplus S8[xE] \oplus S7[x8]$

$z4z5z6z7 = x8x9xAxB \oplus S5[z0] \oplus S6[z2] \oplus S7[z1] \oplus S8[z3] \oplus S8[xA]$

$z8z9zAzB = xCxDxExF \oplus S5[z7] \oplus S6[z6] \oplus S7[z5] \oplus S8[z4] \oplus S5[x9]$

$zCzDzEzF = x4x5x6x7 \oplus S5[zA] \oplus S6[z9] \oplus S7[zB] \oplus S8[z8] \oplus S6[xB]$

$K25 = S5[z3] \oplus S6[z2] \oplus S7[zC] \oplus S8[zD] \oplus S5[z9]$

$K26 = S5[z1] \oplus S6[z0] \oplus S7[zE] \oplus S8[zF] \oplus S6[zC]$

$K27 = S5[z7] \oplus S6[z6] \oplus S7[z8] \oplus S8[z9] \oplus S7[z2]$

$K28 = S5[z5] \oplus S6[z4] \oplus S7[zA] \oplus S8[zB] \oplus S8[z6]$

$x0x1x2x3 = z8z9zAzB \oplus S5[z5] \oplus S6[z7] \oplus S7[z4] \oplus S8[z6] \oplus S7[z0]$

$x4x5x6x7 = z0z1z2z3 \oplus S5[x0] \oplus S6[x2] \oplus S7[x1] \oplus S8[x3] \oplus S8[z2]$

$x8x9xAxB = z4z5z6z7 \oplus S5[x7] \oplus S6[x6] \oplus S7[x5] \oplus S8[x4] \oplus S5[z1]$

$xCxDxExF = zCzDzEzF \oplus S5[xA] \oplus S6[x9] \oplus S7[xB] \oplus S8[x8] \oplus S6[z3]$

$K29 = S5[x8] \oplus S6[x9] \oplus S7[x7] \oplus S8[x6] \oplus S5[x3]$

$K30 = S5[xA] \oplus S6[xB] \oplus S7[x5] \oplus S8[x4] \oplus S6[x7]$

$K31 = S5[xC] \oplus S6[xD] \oplus S7[x3] \oplus S8[x2] \oplus S7[x8]$

$K32 = S5[xE] \oplus S6[xF] \oplus S7[x1] \oplus S8[x0] \oplus S8[xD]$

The following step is used to create masking subkeys and rotation subkeys.

Let Km_1, …, Km_{16} be 32-bit masking subkeys (one per round). Let Kr_1, …, Kr_{16} be 32-bit rotation subkeys (one per round); only the least significant 5 bits are used in each round.

for i = 1 to 16:

$Km_i = K_i$

$Kr_i = K_{16+i}$

This completes the CAST-128 key schedule.

4.5 HIGHT

4.5.1 The HIGHT algorithm

The HIGHT algorithm is a symmetric block cipher that can process data blocks of 64 bits, using a cipher key with length of 128 bits.

4.5.2 HIGHT encryption

The encryption operation is as shown in Figure 8. The transformation of a 64-bit block P into a 64-bit block C is defined as follows (WKi and SKi are whitening key bytes and subkey bytes respectively defined in 4.5.5):

(1) $P = P_7 \| P_6 \| P_5 \| P_4 \| P_3 \| P_2 \| P_1 \| P_0$ (P_i are plaintext bytes)

(2) $X_{0,0} = P_0 \boxplus WK_0$, $X_{0,1} = P_1$,

$X_{0,2} = P_2 \oplus WK_1$, $X_{0,3} = P_3$,

$X_{0,4} = P_4 \boxplus WK_2$, $X_{0,5} = P_5$,

$X_{0,6} = P_6 \oplus WK_3$, $X_{0,7} = P_7$.

(3) for i = 0 to 30:

$X_{i+1,0} = X_{i,7} \oplus (F_0(X_{i,6}) \boxplus SK_{4i+3})$, $X_{i+1,1} = X_{i,0}$,

$X_{i+1,2} = X_{i,1} \boxplus (F_1(X_{i,0}) \oplus SK_{4i})$, $X_{i+1,3} = X_{i,2}$,

$X_{i+1,4} = X_{i,3} \oplus (F_0(X_{i,2}) \boxplus SK_{4i+1})$, $X_{i+1,5} = X_{i,4}$,

$X_{i+1,6} = X_{i,5} \boxplus (F_1(X_{i,4}) \oplus SK_{4i+2})$, $X_{i+1,7} = X_{i,6}$.

for i = 31:

$X_{i+1,0} = X_{i,0}$, $X_{i+1,1} = X_{i,1} \boxplus (F_1(X_{i,0}) \oplus SK_{124})$,

$X_{i+1,2} = X_{i,2}$, $X_{i+1,3} = X_{i,3} \oplus (F_0(X_{i,2}) \boxplus SK_{125})$,

$X_{i+1,4} = X_{i,4}$, $X_{i+1,5} = X_{i,5} \boxplus (F_1(X_{i,4}) \oplus SK_{126})$,

$X_{i+1,6} = X_{i,6}$, $X_{i+1,7} = X_{i,7} \oplus (F_0(X_{i,6}) \boxplus SK_{127})$.

(4) $C_0 = X_{32,0} \boxplus WK_4$, $C_1 = X_{32,1}$,

$C_2 = X_{32,2} \oplus WK_5$, $C_3 = X_{32,3}$,

$C_4 = X_{32,4} \boxplus WK_6$, $C_5 = X_{32,5}$,

$C_6 = X_{32,6} \oplus WK_7$, $C_7 = X_{32,7}$.

(5) $C = C_7 \| C_6 \| C_5 \| C_4 \| C_3 \| C_2 \| C_1 \| C_0$ (C_i are ciphertext bytes)

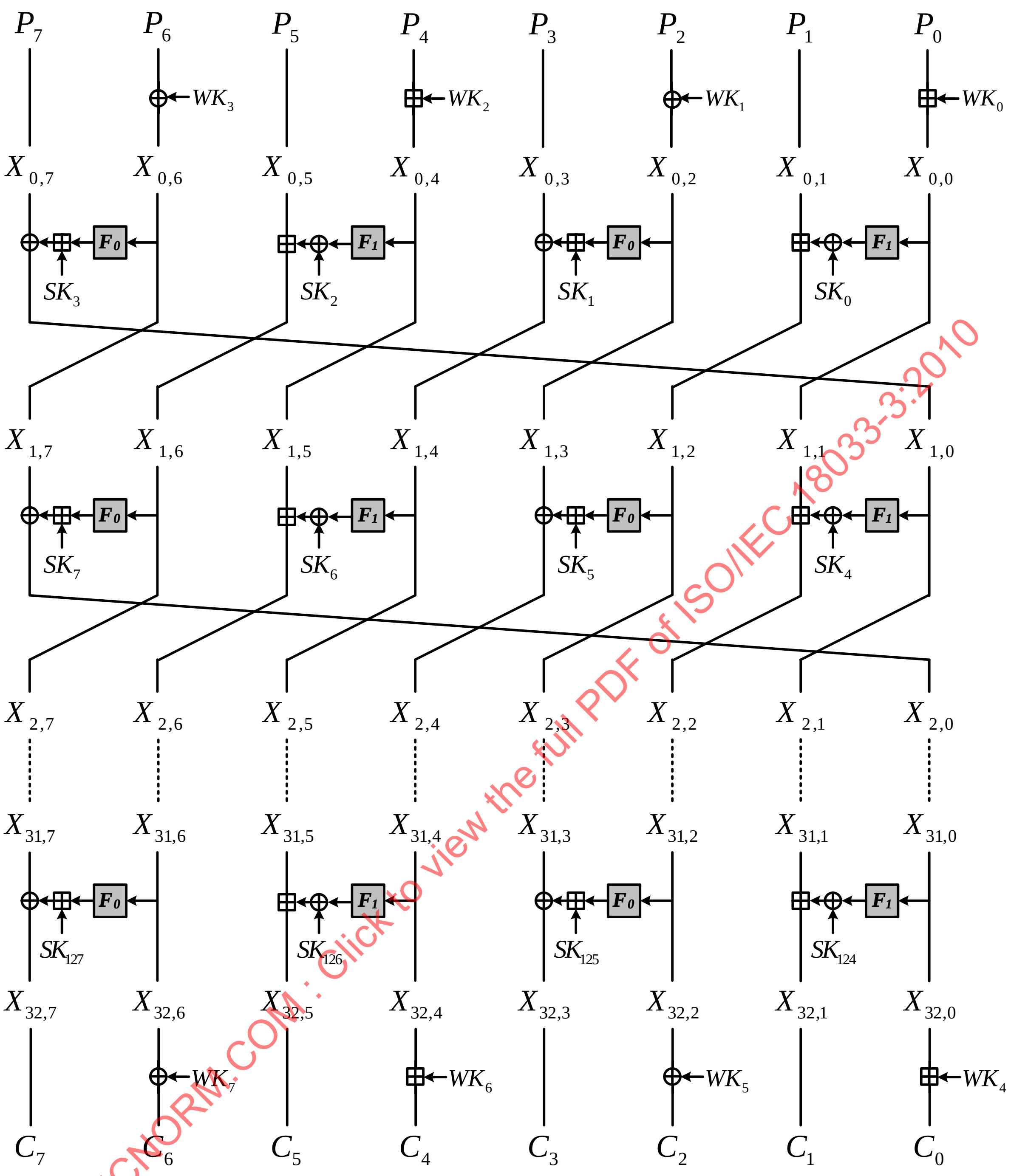

Figure 8 — Encryption procedure of HIGHT

4.5.3 HIGHT decryption

The decryption operation is identical in operation to encryption apart from the following two modifications.

(1) All ⊞ operations are replaced by ⊟ operations except for the ⊞ operations connecting SK_i and outputs of F_0.

(2) The order in which the keys WK_i and SK_i are applied is reversed.

4.5.4 HIGHT functions

4.5.4.1 The functions F_0 and F_1

The HIGHT algorithm uses two functions, namely, F_0 and F_1 which are now defined.

4.5.4.2 Function F_0

The F_0 function is used for encryption and decryption. The function F_0 is defined as follows:

$$F_0(x) = (x{<}{<}{<}_1) \oplus (x{<}{<}{<}_2) \oplus (x{<}{<}{<}_7)$$

4.5.4.3 Function F_1

The F_1 function is used for encryption and decryption. The function F_1 is defined as follows:

$$F_1(x) = (x{<}{<}{<}_3) \oplus (x{<}{<}{<}_4) \oplus (x{<}{<}{<}_6)$$

4.5.5 HIGHT key schedule

The key scheduling part accepts a 128-bit master key $K = K_{15} \| K_{14} \| \cdots \| K_0$ and yields 8 whitening key bytes WK_i and 128 subkey bytes SK_i, as shown below.

The generation of whitening keys is defined as follows.

for $i = 0, 1, 2, 3$:

$$WK_i = K_{i+12}$$

for $i = 4, 5, 6, 7$:

$$WK_i = K_{i-4}$$

The 128 subkeys are used for encryption and decryption, 4 subkeys per round. The generation of subkeys is defined as follows.

(1) $s_0 = 0, s_1 = 1, s_2 = 0, s_3 = 1, s_4 = 1, s_5 = 0, s_6 = 1$

$$\delta_0 = s_6 \| s_5 \| s_4 \| s_3 \| s_2 \| s_1 \| s_0$$

(2) for $i = 1$ to 127:

$$s_{i+6} = s_{i+2} \oplus s_{i-1}$$

$$\delta_i = s_{i+6} \| s_{i+5} \| s_{i+4} \| s_{i+3} \| s_{i+2} \| s_{i+1} \| s_i$$

(3) for $i = 0$ to 7:

for $j = 0$ to 7:

$$SK_{16 \cdot i + j} = K_{j-i \bmod 8} \boxplus \delta_{16 \cdot i + j}$$

for $j = 0$ to 7:

$$SK_{16 \cdot i + j + 8} = K_{(j-i \bmod 8)+8} \boxplus \delta_{16 \cdot i + j + 8}$$

5 128-bit block ciphers

5.1 Introduction

In this clause, three 128-bit block ciphers are specified; AES in 5.2, Camellia in 5.3, and SEED in 5.4.

5.2 AES

5.2.1 The AES algorithm

The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192 and 256 bits. The AES algorithm is also known as the Rijndael algorithm. The AES algorithm may be used with the three different key lengths indicated above, and therefore these different options are referred to as "AES-128", "AES-192" and "AES-256".

In the AES algorithm, the length of the input and output block is 128 bits (4 words). The length of the cipher key K is 128, 192 or 256 bits. The number of rounds, $\boldsymbol{Nr}$, is 10, 12 or 14, depending on the key length, described in Table 4.

Table 4 — Number of rounds in AES

Key length	Number of rounds ($\boldsymbol{Nr}$)
AES-128	10
AES-192	12
AES-256	14

For both encryption and decryption, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a round key to the State. These transformations (and their inverses) are described in 5.2.4.

The encryption and decryption operations are described in 5.2.2 and 5.2.3, respectively, while the key schedule is described in 5.2.5.

5.2.2 AES encryption

The AES algorithm consists of a sequence of operations performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes, each containing 4 bytes. In the State array denoted by the symbol s, each individual byte has two indices, with its row number r in the range $0 \le r < 4$ and its column number c in the range $0 \le c < 4$. The State is denoted by $S = (s_{r,c})$.

At the start of encryption process, the 16 bytes of the State are initialised with plaintext bytes p_i, from top to bottom and from left to right as illustrated in Figure 9.

p_0	p_4	p_8	p_{12}		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
p_1	p_5	p_9	p_{13}		$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
p_2	p_6	p_{10}	p_{14}	→	$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
p_3	p_7	p_{11}	p_{15}		$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Figure 9 — Initialisation of the State

After an initial round key addition, the State is transformed by implementing a round function ***Nr*** times, with the final round differing slightly from the first ***Nr*** -1 rounds. The final contents of the State are then sent to the output as ciphertext.

The complete encryption operation can be described as follows:

(1) S = `AddRoundKey`(P, W_0)

(2) for i = 1 to ***Nr*** –1:

S = `SubBytes`(S)

S = `ShiftRows`(S)

S = `MixColumns`(S)

S = `AddRoundKey`(S, W_i)

(3) S = `SubBytes`(S), S = `ShiftRows`(S)

(4) C = `AddRoundKey`(S, W_{Nr})

The individual transformations – `SubBytes()`, `ShiftRows()`, `MixColumns()` and `AddRoundKey()` – process the State and are described in 5.2.4. All ***Nr*** rounds are identical with the exception of the final round, which does not include the `MixColumns()` transformation. In the above operation, the array W_i contains the round keys described in 5.2.5.

5.2.3 AES decryption

All transformations used in the encryption operations are invertible. An implementation of the decryption operation that maintains the sequence of transformations used in the encryption operation, replacing the transformations with their inverses, follows.

The complete decryption operation can be described as follows:

(1) S = `AddRoundKey`(C, W_{Nr})

(2) for i = ***Nr*** –1 down to 1:

S = `ShiftRows`$^{-1}$(S)

S = `SubBytes`$^{-1}$(S)

S = `AddRoundKey`(S, W_i)

S = `MixColumns`$^{-1}$(S)

(3) S = `ShiftRows`$^{-1}$(S)

S = `SubBytes`$^{-1}$(S)

(4) P = `AddRoundKey`(S, W_0)

The individual transformations – `SubBytes`$^{-1}$`()`, `ShiftRows`$^{-1}$`()`and `MixColumns`$^{-1}$`()`– process the State and are described in 5.2.4. All ***Nr*** rounds are identical with the exception of the final round in the decryption process, which does not include the `MixColumns`$^{-1}$`()` transformation.

The computation of the round keys W_i is described in 5.2.5.

5.2.4 AES transformations

5.2.4.1 Defined transformations for AES

The AES algorithm uses a number of transformations, namely `SubBytes()`, `SubBytes`$^{-1}$`()`, `ShiftRows()`, `ShiftRows`$^{-1}$`()`, `MixColumns()`, `MixColumns`$^{-1}$`()` and `AddRoundKey()`, which are now defined.

5.2.4.2 `SubBytes()` transformation

The `SubBytes()` transformation substitutes each individual State byte $s_{i,j}$ by a new value $s'_{i,j}$ using a substitution table (S-box), which is invertible.

Figure 10 illustrates the effect of the `SubBytes()` transformation on the State.

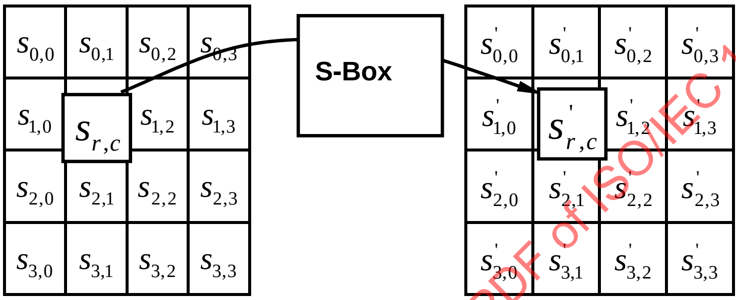

Figure 10 — `SubBytes()` applies the S-box to each byte of the State

The S-box used in the `SubBytes()` transformation is presented in hexadecimal form in Table 5.

Table 5 — AES S-box

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

For example, if $s_{1,1}$ = {53}, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Table 5. This would result in $s'_{1,1}$ having a value of {ed}.

5.2.4.3 `SubBytes`$^{-1}$`()` transformation

`SubBytes`$^{-1}$`()` is the inverse of the `SubBytes()` transformation, in which the inverse S-box is applied to each byte of the State. This is obtained by applying the inverse of the transformation described in 5.2.4.2.

The inverse S-box used in the `SubBytes`$^{-1}$`()` transformation is presented in Table 6.

Table 6 — AES Inverse S-box

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

5.2.4.4 `ShiftRows()` transformation

In the `ShiftRows()` transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, Row 0, is not shifted.

Specifically, the `ShiftRows()` transformation proceeds as follows:

$$S'_{r,c} = S_{r,(c+r) \bmod 4} \quad \text{for } 0 < r < 4 \quad \text{and} \quad 0 \leq c < 4,$$

where r is the row number.

This has the effect of moving bytes to the left (i.e., lower values of c in a given row), while the leftmost bytes wrap around to the rightmost positions of the row (i.e., higher values of c in a given row).

Figure 11 illustrates the `ShiftRows()` transformation, which the bytes are rotated cyclically to the left.

5.2.4.5 `ShiftRows`$^{-1}$`()` transformation

`ShiftRows`$^{-1}$`()` is the inverse of the `ShiftRows()` transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, Row 0, is not shifted. The bottom three rows are cyclically shifted by 4 – r bytes, where r is the row number.

Specifically, the `ShiftRows`$^{-1}$`()` transformation proceeds as follows:

$$S'_{r,(c+r) \bmod 4} = S_{r,c} \quad \text{for } 0 < r < 4 \quad \text{and} \quad 0 \leq c < 4$$

Figure 12 illustrates the `ShiftRows`$^{-1}$`()` transformation.

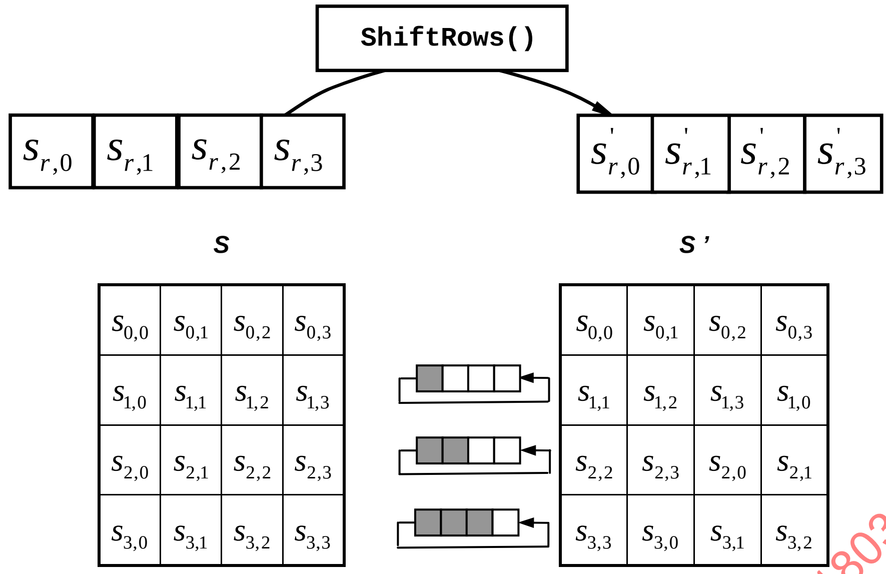

Figure 11 — `ShiftRows()` cyclically shifts the last three rows in the State

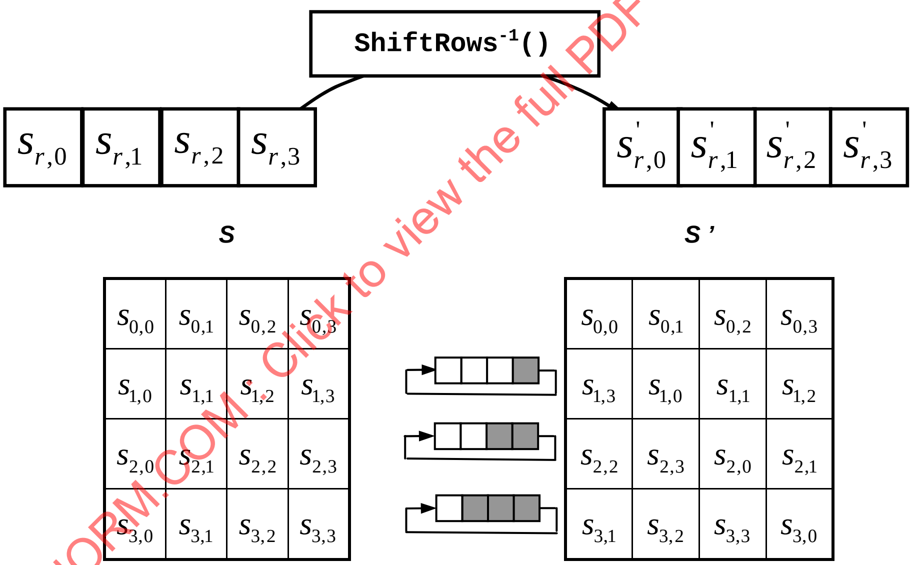

Figure 12 — `ShiftRows`$^{-1}$`()` cyclically shifts the last three rows in the State

5.2.4.6 `MixColumns()` transformation

The `MixColumns()` transformation operates on the State column-by-column. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by $a(x)$ = $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. This can be written as a matrix multiplication.

$$s'(x) = a(x) \otimes s(x): \quad \begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < 4.$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \end{aligned}$$

The $\oplus$ operator in these expressions denotes addition in GF(2^8), which corresponds to bitwise XOR. The multiplications are performed modulo the irreducible polynomial of the field. In the case of the AES algorithm the polynomial of $x^8 + x^4 + x^3 + x + 1$ is used.

Figure 13 illustrates the `MixColumns()` transformation.

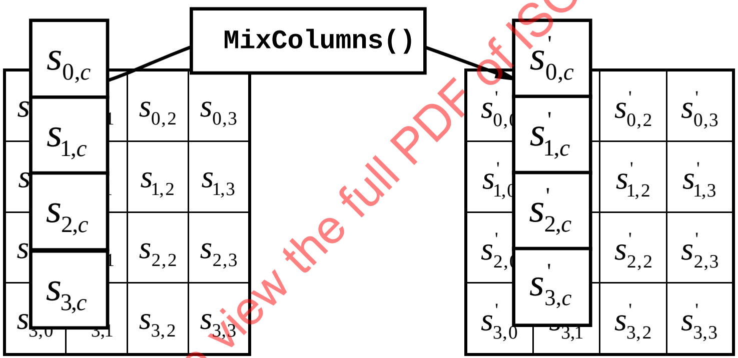

Figure 13 — `MixColumns()` operates on the State column-by-column

5.2.4.7 `MixColumns`$^{-1}$`()` transformation

`MixColumns`$^{-1}$`()` is the inverse of the `MixColumns()` transformation. `MixColumns`$^{-1}$`()` operates on the State column-by-column. This transformation can be represented as a matrix multiplication, where each byte is interpreted as an element in the finite field GF(2^8):

$$s'(x) = a^{-1}(x) \otimes s(x): \quad \begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < 4.$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$\begin{aligned} s'_{0,c} &= (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s'_{2,c} &= (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{aligned}$$

5.2.4.8 AddRoundKey( ) transformation

In the `AddRoundKey( )` transformation, a round key is added to the State by a simple bitwise XOR operation. Each round key consists of 4 words (128 bits) from the key schedule (described in 5.2.5). Those 4 words are each added into the columns of the State, such that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{(4*i+c)}] \quad \text{for } 0 \le c < 4,$$

where $w_{(4*i+c)}$ are the c-th key schedule words of i-th round key $W_i = [w_{(4*i)}, w_{(4*i+1)}, w_{(4*i+2)}, w_{(4*i+3)}]$ and i is a value in the range $0 \le i \le \boldsymbol{Nr}$. In the encryption operation, the initial round key addition occurs when $i = 0$, prior to the first application of the round function. The application of the `AddRoundKey( )` transformation to the ***Nr*** rounds of the encryption occurs when $1 \le i \le \boldsymbol{Nr}$.

The action of this transformation is illustrated in Figure 14. The byte address within words of the key schedule is described in 5.2.5.

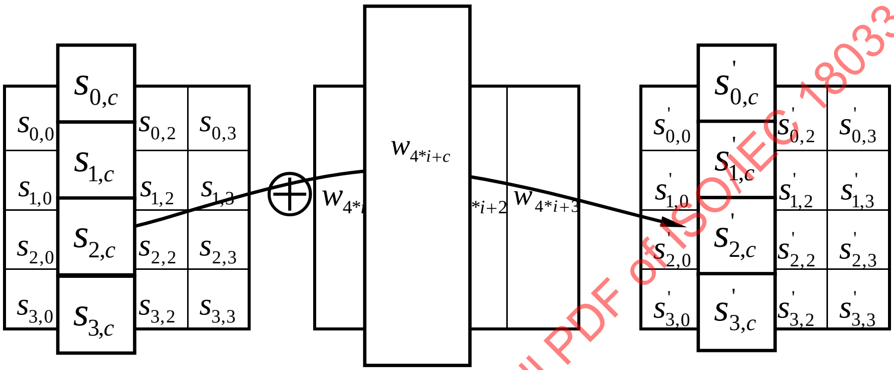

Figure 14 — AddRoundKey( ) XORs each column of the State with a word from the key schedule

5.2.5 AES key schedule

The AES algorithm takes the cipher key K and performs a key expansion routine to generate a key schedule. The key expansion generates a total of $4(\boldsymbol{Nr} + 1)$ words: the algorithm requires an initial set of 4 words, and each of the ***Nr*** rounds requires 4 words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted w_j, with j in the range $0 \le j < 4(\boldsymbol{Nr} + 1)$.

The complete key expansion operation for AES-128 and AES-192 can be described as follows:

(1) $[w_0, w_1, w_2, w_3] = K$ and $\boldsymbol{Nk} = 4$ for AES-128

$[w_0, w_1, w_2, w_3, w_4, w_5] = K$ and $\boldsymbol{Nk} = 6$ for AES-192

(2) for $j = \boldsymbol{Nk}$ to $4(\boldsymbol{Nr} + 1) - 1$:

if (j mod $\boldsymbol{Nk} = 0$) then

$w_j = w_{j-Nk} \oplus$ `SubBytes*(ShiftColumn(`w_{j-1}`))` $\oplus R^{i/Nk}{}_C$

else

$w_j = w_{j-Nk} \oplus w_{j-1}$

(3) $W_i = [w_{(4*i)}, w_{(4*i+1)}, w_{(4*i+2)}, w_{(4*i+3)}]$ for $0 \le i \le \boldsymbol{Nr}$.

In the above operation, the first round key W_0, used in the initial key addition, is directly filled with the 4 words of the secret key K. The 32-bit columns w_j of the remaining round keys are derived recursively. The `SubBytes*()` transformation substitutes the bytes of a single column in the same way as the `SubBytes()` transformation described in 5.2.4.2. The `shiftcolumn()` transformation is an upward cyclic shift over one byte position. The constants R^i_C are fixed 4-byte columns defined as $(\texttt{02}^{i-1}, \texttt{00}, \texttt{00}, \texttt{00})^T$ with {02} representing the element x in GF(2^8) (using the same irreducible polynomial $x^8 + x^4 + x^3 + x + 1$).

The complete key expansion operation for AES-256 can be described as follows:

(1) $[w_0, w_1, w_2, w_3] = K_0, [w_4, w_5, w_6, w_7] = K_1$, $\boldsymbol{Nk} = 8$

(2) for $j = \boldsymbol{Nk}$ to $4(\boldsymbol{Nr} + 1) - 1$:

if (j mod $\boldsymbol{Nk} = 0$) then

$$w_j = w_{j-Nk} \oplus \texttt{SubBytes*}(\texttt{ShiftColumn}(w_{j-1})) \oplus R^{i/Nk}_C$$

else if (j mod $\boldsymbol{Nk} = 4$) then

$$w_j = w_{j-Nk} \oplus \texttt{SubBytes*}(w_{j-1})$$

else

$$w_j = w_{j-Nk} \oplus w_{j-1}$$

(3) $W_i = [w_{(4*i)}, w_{(4*i+1)}, w_{(4*i+2)}, w_{(4*i+3)}]$ for $0 \le i \le \boldsymbol{Nr}$.

In the above operation, K_0 and K_1 represent the first and the second half of 256-bit cipher key K respectively.

5.3 Camellia

5.3.1 The Camellia algorithm

The Camellia algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192 and 256 bits. This interface is the same as the AES algorithm's.

5.3.2 Camellia encryption

5.3.2.1 128-bit key

The encryption process for 128-bit keys operates over 18 rounds and is shown in Figure 15. The transformation of a 128-bit block P into a 128-bit block C is defined as follows (L and R are variables with 64-bit length, and kw, k and kl are round keys with 64-bit length):

(1) $L_0 \,||\, R_0 = P \oplus (kw_1 \,||\, kw_2)$

(2) for i = 1 to 18:

$L_i = \mathrm{F}(L_{i-1}, k_i) \oplus R_{i-1}$

$R_i = L_{i-1}$

if (i = 6 or 12) then

$L_i = \mathrm{FL}(L_i, kl_{i/3-1})$

$R_i = \mathrm{FL}^{-1}(R_i, kl_{i/3})$

(3) $C = (R_{18} \oplus kw_3) \,||\, (L_{18} \oplus kw_4)$

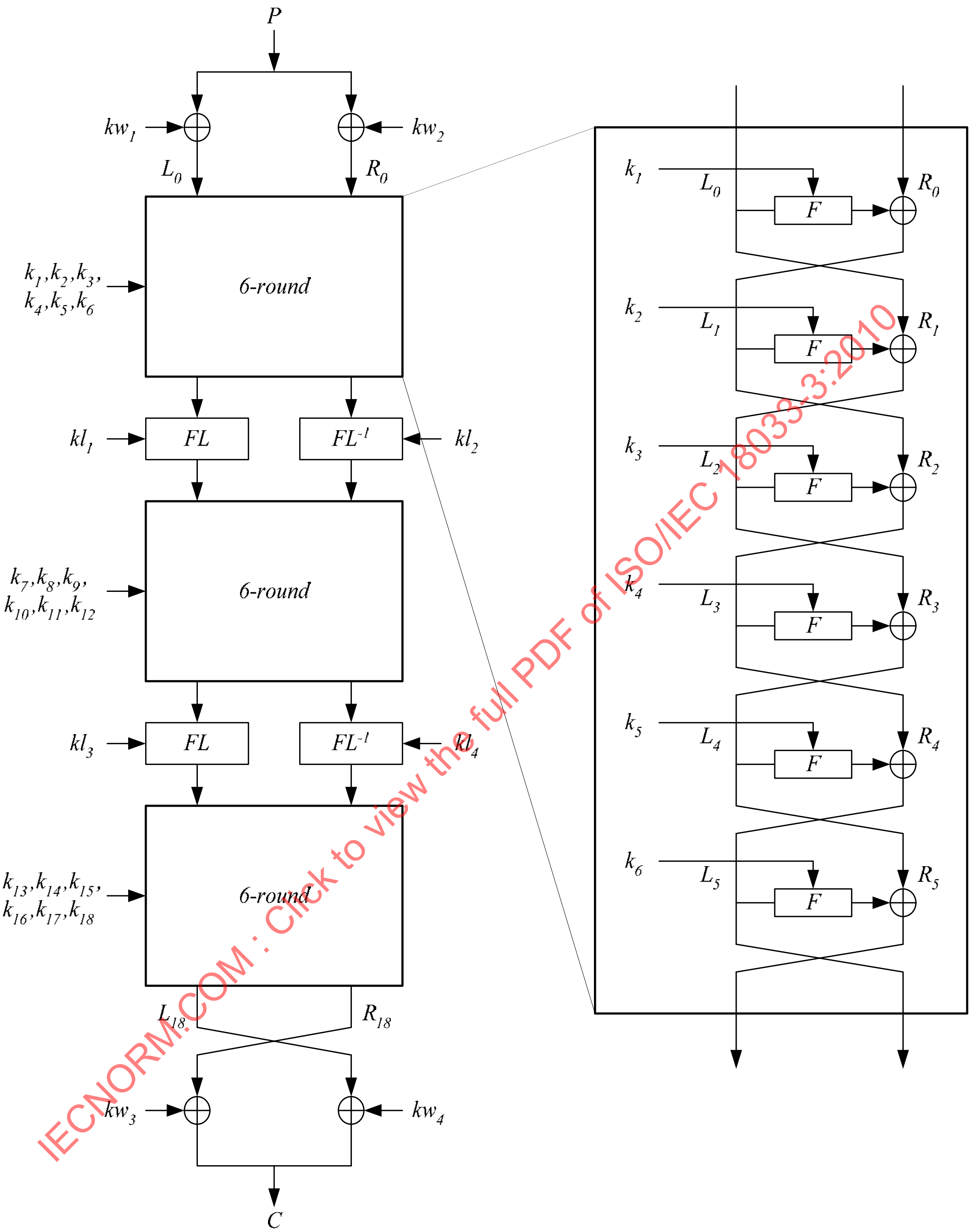

Figure 15 — Encryption procedure of Camellia for 128-bit key

5.3.2.2 192-bit and 256-bit key

The encryption process for 192-bit or 256-bit keys operates over 24 rounds and is shown in Figure 16. The transformation of a 128-bit block P into a 128-bit block C is defined as follows (L and R are variables with 64-bit length, and kw, k and kl are round keys with 64-bit length):

(1) $L_0 \| R_0 = P \oplus (kw_1 \| kw_2)$

(2) for i = 1 to 24:

$L_i = \mathrm{F}(L_{i-1}, k_i) \oplus R_{i-1}$

$R_i = L_{i-1}$

if (i = 6 or 12 or 18) then

$L_i = \mathrm{FL}(L_i, kl_{i/3-1})$

$R_i = \mathrm{FL}^{-1}(R_i, kl_{i/3})$

(3) $C = (R_{24} \oplus kw_3) \| (L_{24} \oplus kw_4)$

5.3.3 Camellia decryption

5.3.3.1 128-bit key

The decryption process for 128-bit keys is shown in Figure 17, and is identical in operation to encryption apart from the position and ordering of the round keys, which are reversed.

The decryption operation is thus defined as follows.

(1) $R_{18} \| L_{18} = C \oplus (kw_3 \| kw_4)$

(2) for i = 18 down to 1:

$R_{i-1} = \mathrm{F}(R_i, k_i) \oplus L_i$

$L_{i-1} = R_i$

if (i = 13 or 7) then

$R_{i-1} = \mathrm{FL}(R_{i-1}, kl_{(i-1)/3})$

$L_{i-1} = \mathrm{FL}^{-1}(L_{i-1}, kl_{(i-1)/3-1})$

(3) $P = (L_0 \oplus kw_1) \| (R_0 \oplus kw_2)$

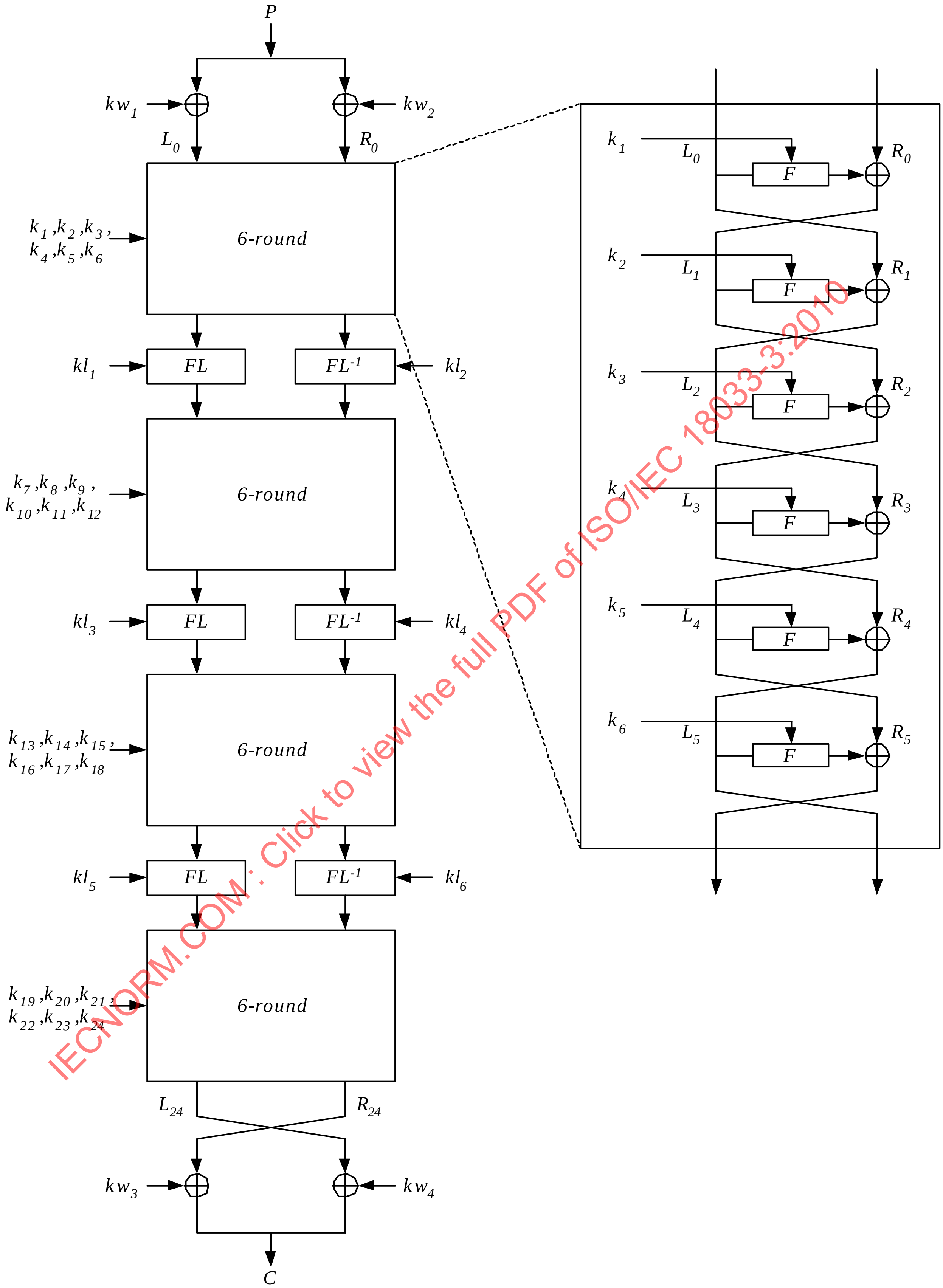

Figure 16 — Encryption procedure of Camellia for 192-bit and 256-bit keys

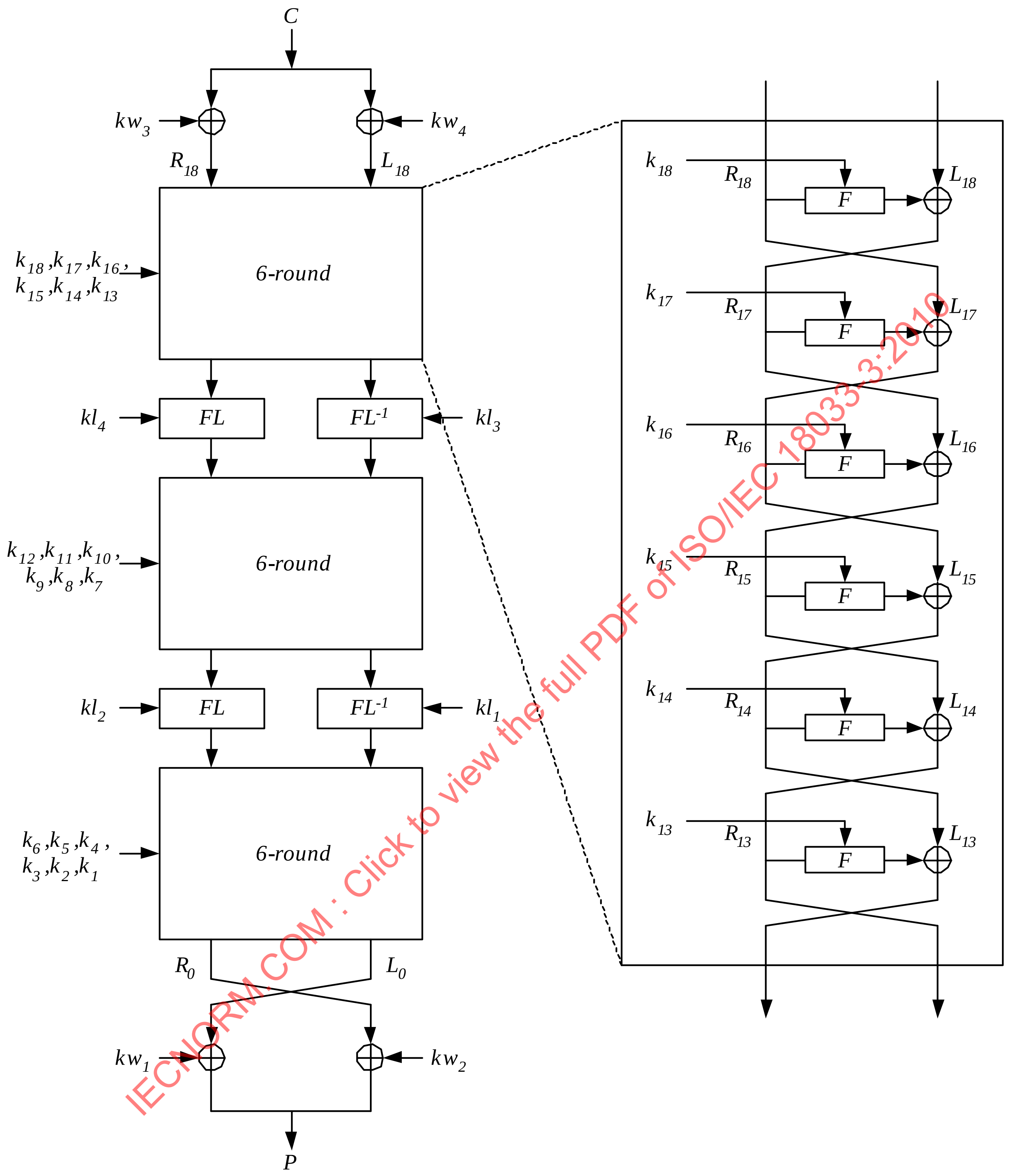

Figure 17 — Decryption procedure of Camellia for 128-bit key

5.3.3.2 192-bit and 256-bit key

The decryption process for 192-bit or 256-bit keys is shown in Figure 18 and is identical in operation to encryption apart from the position and ordering of the round keys, which are reversed.

The decryption operation is thus defined as follows.

(1) $R_{24} \| L_{24} = C \oplus (kw_3 \| kw_4)$

(2) for $i = 24$ down to 1:

$R_{i-1} = \mathrm{F}(R_i, k_i) \oplus L_i$

$L_{i-1} = R_i$

if (i = 19 or 13 or 7) then

$R_{i-1} = \mathrm{FL}(R_{i-1}, kl_{(i-1)/3})$

$L_{i-1} = \mathrm{FL}^{-1}(L_{i-1}, kl_{(i-1)/3-1})$

(3) $P = (L_0 \oplus kw_1) \| (R_0 \oplus kw_2)$

5.3.4 Camellia functions

5.3.4.1 Defined functions

The Camellia algorithm uses a number of functions, namely F, FL, FL^{-1} and S-boxes which are now defined.

5.3.4.2 F-function

The F-function is shown in Figure 19. It comprises a bitwise XOR, followed by an application of 8 parallel 8x8-bit S-boxes, followed by a diffusion layer (the P-function). The variables, x_j, y_j, z_j, z'_j, are 8 bits wide, and the variables, L_i, k_i, L'_i, are 64 bits wide. The 64-bit input L_i is first bitwise XORed with a 64-bit round key k_i, and is then partitioned into eight 8-bit segments y_j such that

$$y_1 \| y_2 \| y_3 \| y_4 \| y_5 \| y_6 \| y_7 \| y_8 = L_i \oplus k_i ,$$

where

$$L_i = x_1 \| x_2 \| x_3 \| x_4 \| x_5 \| x_6 \| x_7 \| x_8.$$

Each y_j is then passed through an 8x8-bit S-box s_t to give eight 8-bit segments z_j, where

$$z_1 = s_1[y_1],\ z_2 = s_2[y_2],\ z_3 = s_3[y_3],\ z_4 = s_4[y_4],\ z_5 = s_2[y_5],\ z_6 = s_3[y_6],\ z_7 = s_4[y_7],\ z_8 = s_1[y_8].$$

The eight 8-bit segments z_j are then acted on by the P-function, which is a diffusion layer which outputs eight 8-bit segments z'_j, where

$$z'_1 = z_1 \oplus z_3 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8, \qquad z'_5 = z_1 \oplus z_2 \oplus z_6 \oplus z_7 \oplus z_8,$$

$$z'_2 = z_1 \oplus z_2 \oplus z_4 \oplus z_5 \oplus z_7 \oplus z_8, \qquad z'_6 = z_2 \oplus z_3 \oplus z_5 \oplus z_7 \oplus z_8,$$

$$z'_3 = z_1 \oplus z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_8, \qquad z'_7 = z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_8,$$

$$z'_4 = z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7, \qquad z'_8 = z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7.$$

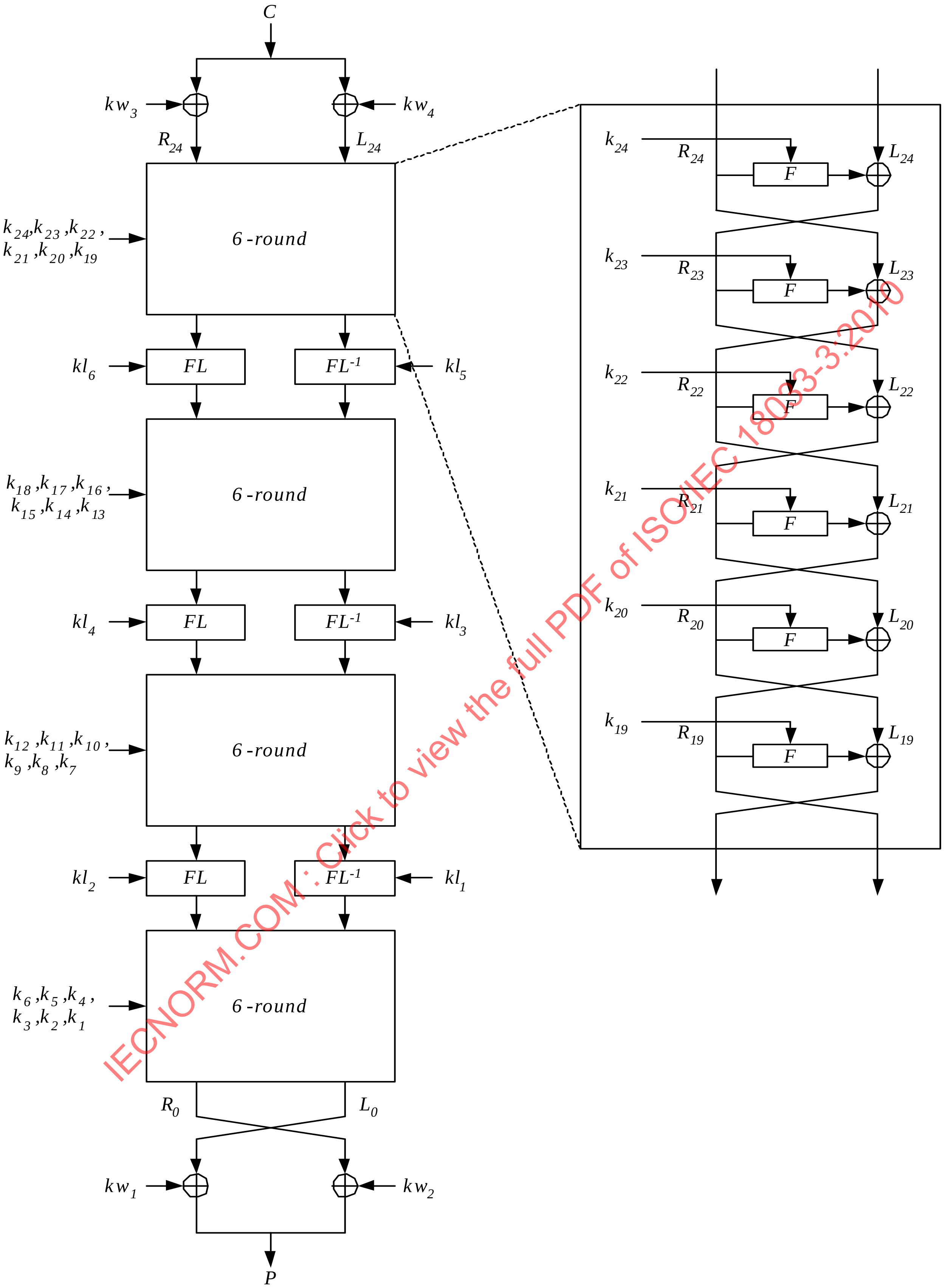

Figure 18 — Decryption procedure of Camellia for 192-bit and 256-bit keys

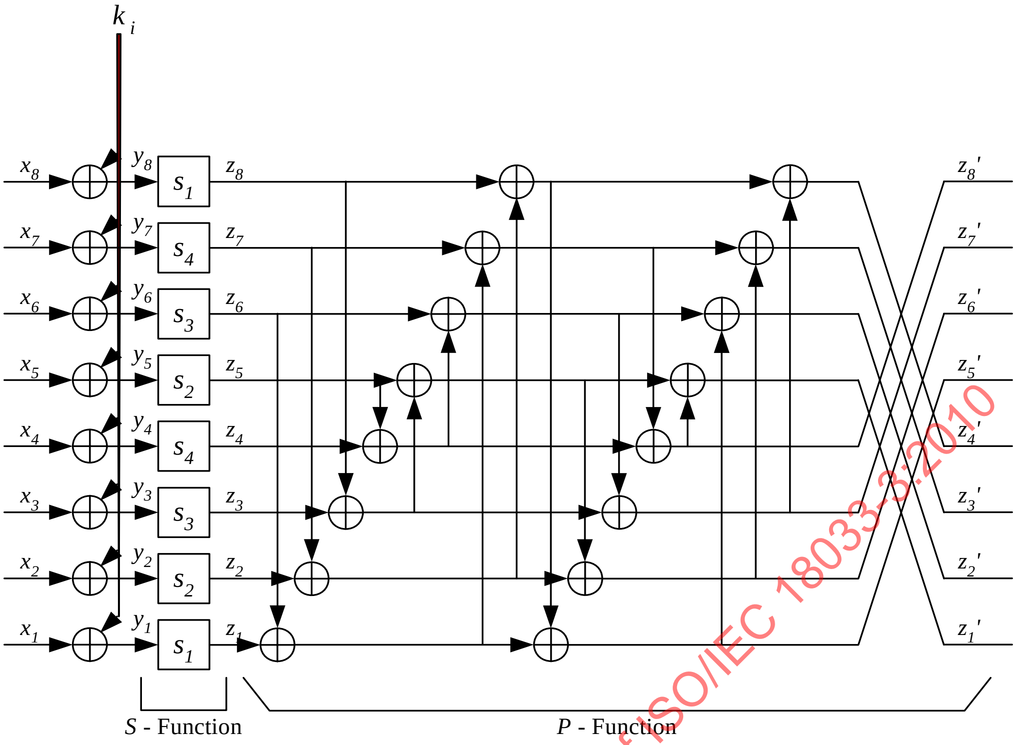

Figure 19 — F-function

The P-function can alternatively be represented in matrix-vector form as

$$\begin{pmatrix} z_8 \\ z_7 \\ \vdots \\ z_1 \end{pmatrix} \mapsto \begin{pmatrix} z'_8 \\ z'_7 \\ \vdots \\ z'_1 \end{pmatrix} = P \begin{pmatrix} z_8 \\ z_7 \\ \vdots \\ z_1 \end{pmatrix},$$

where

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

The 64-bit output of the F-function L'_i is then constructed by concatenating the 8-bit z'_j, where

$$L'_i = z'_1 || z'_2 || z'_3 || z'_4 || z'_5 || z'_6 || z'_7 || z'_8.$$

5.3.4.3 FL-function

The FL-function is shown in Figure 20. The FL-function is defined as follows (X and Y are 64-bit data, kl is a 64-bit round key; X_L, X_R, Y_L, Y_R, kl_{iL}, kl_{iR} are 32-bit wide):

(1) $X_L \| X_R = X$, $kl_{iL} \| kl_{iR} = kl_i$

(2) $Y_R = ((X_L \wedge kl_{iL}) \lll_1) \oplus X_R$, $Y_L = (Y_R \vee kl_{iR}) \oplus X_L$

(3) $Y = Y_L \| Y_R$

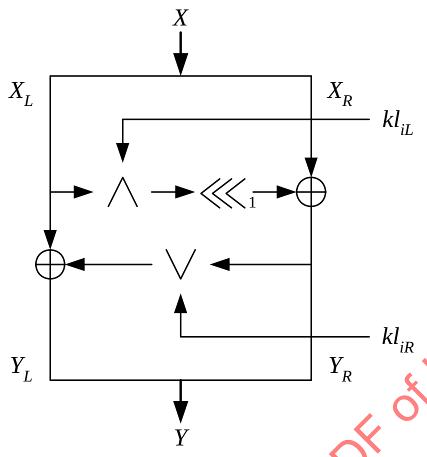

Figure 20 — FL-function

5.3.4.4 FL^{-1}-function

The FL^{-1}-function is shown in Figure 21. The FL^{-1}-function is defined as follows (X and Y are 64-bit data, kl is a 64-bit round key; X_L, X_R, Y_L, Y_R, kl_{iL}, kl_{iR} are 32-bit wide):

(1) $Y_L \| Y_R = Y$, $kl_{iL} \| kl_{iR} = kl_i$

(2) $X_L = (Y_R \vee kl_{iR}) \oplus Y_L$, $X_R = ((X_L \wedge kl_{iL}) \lll_1) \oplus Y_R$

(3) $X = X_L \| X_R$

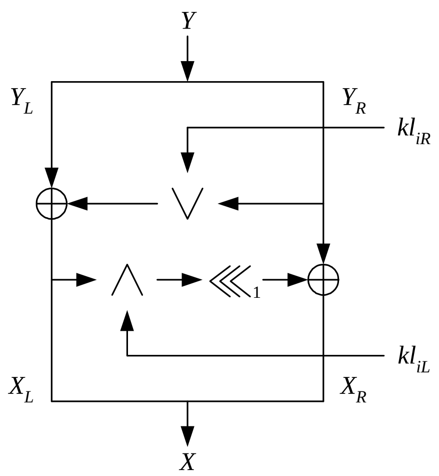

Figure 21 — FL^{-1}-function

5.3.4.5 S-boxes

5.3.4.5.1 The S-boxes s_1 to s_4

S-boxes s_1, s_2, s_3 and s_4 are given in the following subclauses. These S-boxes accept 8-bit inputs and yield 8-bit outputs, respectively. They can be also described in a simple algebraic form. The algebraic form of s_1 is shown in C.3.

5.3.4.5.2 S-box s_1

s_1 is given as follows:

$$s_1: \quad y = \mathbf{h}(\mathbf{g}(\mathbf{f}(\text{c5} \oplus x))) \oplus \text{6e}$$

where operations **f**, **g** and **h** take 8-bit inputs $a = a_1 \| a_2 \| a_3 \| a_4 \| a_5 \| a_6 \| a_7 \| a_8$ and output 8-bit values $b = b_1 \| b_2 \| b_3 \| b_4 \| b_5 \| b_6 \| b_7 \| b_8$, where the a_i and b_i are 1-bit values. **f** is an affine permutation of the input, **g** is inversion over GF(2^8), and **h** is an affine transformation of the output.

For example, if the input to s_1 is {53}, then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in Table 7. This would result in s_1 having a value of {c2}.

Table 7 — The S-box s_1

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

5.3.4.5.3 S-box s_2

s_2 shown in table 8 is given as follows:

$$s_2: y = s_1(x) \lll_1.$$

Table 8 — The S-box s_2

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

5.3.4.5.4 S-box s_3

s_3 shown in Table 9 is given as follows:

$$s_3 \colon y = s_1(x) \lll 7.$$

Table 9 — The S-box s_3

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

5.3.4.5.5 S-box s_4

s_4 shown in Table 10 is given as follows:

$$s_4 \colon y = s_1(x{<}{<}{<}_1).$$

Table 10 — The S-box s_4

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

5.3.5 Camellia key schedule

The key schedule is shown in Figure 22, Tables 12 and 13. For the 128-bit key version, the key K is the 128-bit key K_L, with the 128-bit key K_R set to all zero bits. Thus,

$$K = K_L,\ K_R = 0.$$

For the 192-bit key version, the key K is the 128-bit key K_L and the leftmost 64-bits of K_R, K_{RL}, with the rightmost 64 bits of K_R, K_{RR}, set to the bitwise negation of the leftmost 64 bits of K_R, K_{RL}. Thus,

$$K = K_L \,||\, K_{RL},\ K_{RR} = \overline{K_{RL}},\ K_R = K_{RL} \,||\, K_{RR}.$$

For the 256-bit key version, the key K is the 128-bit key K_L and the 128-bit key K_R . Thus,

$$K = K_L \,||\, K_R.$$

The key schedule makes use of the F-function of the encryption module, and is the same for encryption and decryption. The key K is encrypted by means of the F-function using key schedule constants, where these constants Σ_i are defined as continuous values from the hexadecimal representation of the square root of the i-th prime. The round keys are then generated partly from rotated values of the key K (where K equals to K_L, $K_L \,||\, K_{RL}$ or $K_L \,||\, K_R$, for a 128-bit, 192-bit or 256-bit key K, respectively), and partly from rotated values of the 'encrypted' keys, K_A and K_B (where K_A and K_B are 128-bit wide).

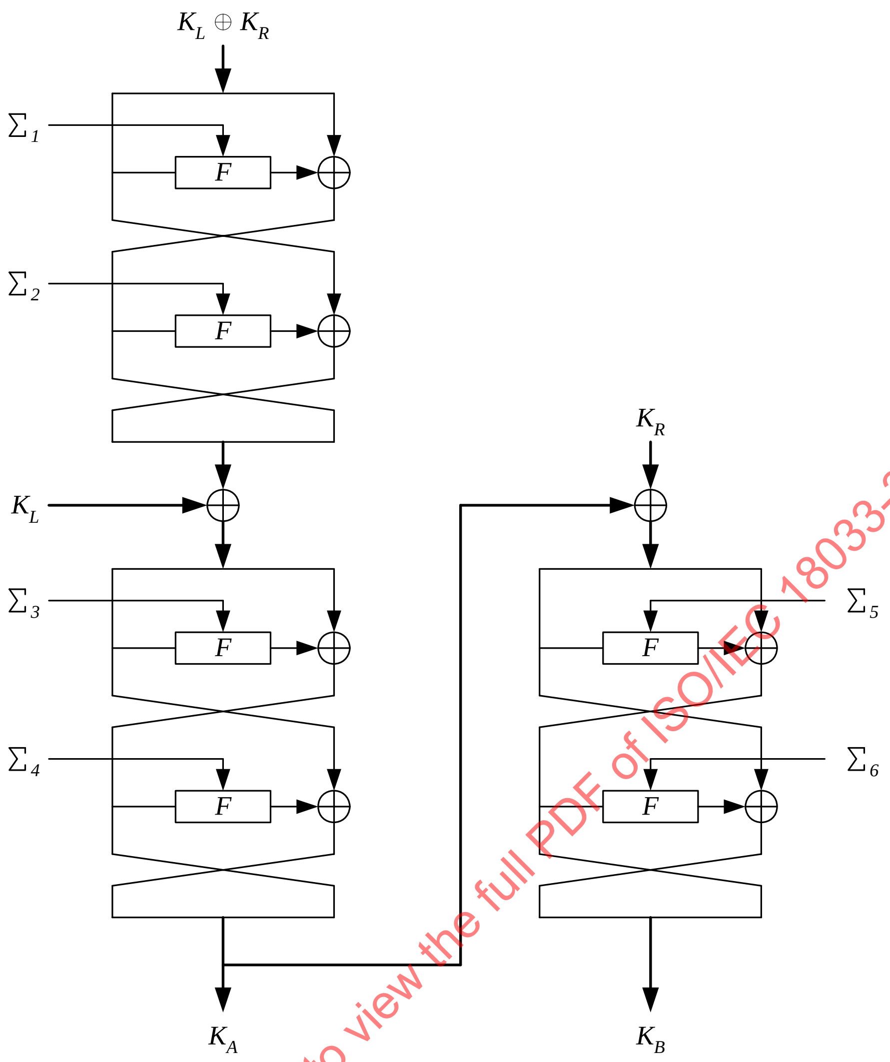

Figure 22 — Main part of key schedule

For the 128-bit key version, the output of the main part of key schedule is the 128-bit subkey K_A with the right side of Figure 22 omitted and K_B not generated or used. For the 192-bit and 256-bit key versions, the outputs of the main part of key schedule are the 128-bit subkey K_A and the 128-bit subkey K_B. The key schedule comprises two or three 2-round operations for 128-bit or 192/256-bit key versions, respectively. Each 2-round operation is 'keyed' by a pair of constants Σ_i .

The 128-bit input to the first 2-round operation on the left side of Figure 22 is $K_L \oplus K_R$, and this 2-round operation is 'keyed' by two 64-bit constants Σ_1 and Σ_2. The 128-bit output from the first 2-round operation is then bitwise XORed with K_L before input to the second 2-round operation on the left side of Figure 22. This second 2-round operation is 'keyed' by two 64-bit constants Σ_3 and Σ_4. The 128-bit output from the second 2-round operation is K_A. For 192-bit or 256-bit key versions, K_A is then bitwise XORed with the 128-bit subkey K_R before inputting result to the third 2-round operation, which is on the right side of Figure 22. This third 2-round operation is 'keyed' by two 64-bit constants Σ_5 and Σ_6. The 128-bit output from the third 2-round operation is K_B.

The complete key schedule operation can be described as follows (K_a, K_A and K_B are 128-bit wide):

(1) K_a = 2RoundFeistel($K_L \oplus K_R$, Σ_1, Σ_2)

(2) K_A = 2RoundFeistel($K_a \oplus K_L$, Σ_3, Σ_4)

(3) K_B = 2RoundFeistel($K_A \oplus K_R$, Σ_5, Σ_6) (192/256-bit key only)

where the 128-bit input to 2RoundFeistel is split into two 64-bit parts $L_0 \parallel R_0$, the 128-bit output from 2RoundFeistel is also split into two 64-bit parts $L_2 \parallel R_2$, and the two 64-bit 'round key' inputs to 2RoundFeistel are Σ_i and Σ_{i+1}.

2RoundFeistel is then described as

(1) for j = 0, 1:

$$L_{j+1} = F(L_j, \Sigma_{i+j}) \oplus R_j$$

$$R_{j+1} = L_j$$

These 64-bit key schedule constants are defined in Table11.

Table 11 — Constants in the key schedule

	Constants
Σ_1	a09e667f3bcc908b
Σ_2	b67ae8584caa73b2
Σ_3	c6ef372fe94f82be
Σ_4	54ff53a5f1d36f1c
Σ_5	10e527fade682d1d
Σ_6	b05688c2b3e6c1fd

Finally the 64-bit round keys, k, kw and kl, are derived from the 128-bit subkeys, K_L, K_R, K_A and K_B. Table 12 is for the 128-bit key version and Table 13 is for the 192-bit or 256-bit key version.

Table 12 — Round keys for 128-bit secret key

Function	Round key	Value
	kw_1	$(K_L<<<_0)_L$
	kw_2	$(K_L<<<_0)_R$
F (Round 1)	k_1	$(K_A<<<_0)_L$
F (Round 2)	k_2	$(K_A<<<_0)_R$
F (Round 3)	k_3	$(K_L<<<_{15})_L$
F (Round 4)	k_4	$(K_L<<<_{15})_R$
F (Round 5)	k_5	$(K_A<<<_{15})_L$
F (Round 6)	k_6	$(K_A<<<_{15})_R$
FL	kl_1	$(K_A<<<_{30})_L$
FL^{-1}	kl_2	$(K_A<<<_{30})_R$
F (Round 7)	k_7	$(K_L<<<_{45})_L$
F (Round 8)	k_8	$(K_L<<<_{45})_R$
F (Round 9)	k_9	$(K_A<<<_{45})_L$
F (Round 10)	k_{10}	$(K_L<<<_{60})_R$
F (Round 11)	k_{11}	$(K_A<<<_{60})_L$
F (Round 12)	k_{12}	$(K_A<<<_{60})_R$
FL	kl_3	$(K_L<<<_{77})_L$
FL^{-1}	kl_4	$(K_L<<<_{77})_R$
F (Round 13)	k_{13}	$(K_L<<<_{94})_L$
F (Round 14)	k_{14}	$(K_L<<<_{94})_R$
F (Round 15)	k_{15}	$(K_A<<<_{94})_L$
F (Round 16)	k_{16}	$(K_A<<<_{94})_R$
F (Round 17)	k_{17}	$(K_L<<<_{111})_L$
F (Round 18)	k_{18}	$(K_L<<<_{111})_R$
	kw_3	$(K_A<<<_{111})_L$
	kw_4	$(K_A<<<_{111})_R$

Table 13 — Round keys for 192/256-bit secret key

Function	Round key	Value
	kw_1	$(K_L<<<_0)_L$
	kw_2	$(K_L<<<_0)_R$
F (Round 1)	k_1	$(K_B<<<_0)_L$
F (Round 2)	k_2	$(K_B<<<_0)_R$
F (Round 3)	k_3	$(K_R<<<_{15})_L$
F (Round 4)	k_4	$(K_R<<<_{15})_R$
F (Round 5)	k_5	$(K_A<<<_{15})_L$
F (Round 6)	k_6	$(K_A<<<_{15})_R$
FL	kl_1	$(K_R<<<_{30})_L$
FL^{-1}	kl_2	$(K_R<<<_{30})_R$
F (Round 7)	k_7	$(K_B<<<_{30})_L$
F (Round 8)	k_8	$(K_B<<<_{30})_R$
F (Round 9)	k_9	$(K_L<<<_{45})_L$
F (Round 10)	k_{10}	$(K_L<<<_{45})_R$
F (Round 11)	k_{11}	$(K_A<<<_{45})_L$
F (Round 12)	k_{12}	$(K_A<<<_{45})_R$
FL	kl_3	$(K_L<<<_{60})_L$
FL^{-1}	kl_4	$(K_L<<<_{60})_R$
F (Round 13)	k_{13}	$(K_R<<<_{60})_L$
F (Round 14)	k_{14}	$(K_R<<<_{60})_R$
F (Round 15)	k_{15}	$(K_B<<<_{60})_L$
F (Round 16)	k_{16}	$(K_B<<<_{60})_R$
F (Round 17)	k_{17}	$(K_L<<<_{77})_L$
F (Round 18)	k_{18}	$(K_L<<<_{77})_R$
FL	kl_5	$(K_A<<<_{77})_L$
FL^{-1}	kl_6	$(K_A<<<_{77})_R$
F (Round 19)	k_{19}	$(K_R<<<_{94})_L$
F (Round 20)	k_{20}	$(K_R<<<_{94})_R$
F (Round 21)	k_{21}	$(K_A<<<_{94})_L$
F (Round 22)	k_{22}	$(K_A<<<_{94})_R$
F (Round 23)	k_{23}	$(K_L<<<_{111})_L$
F (Round 24)	k_{24}	$(K_L<<<_{111})_R$
	kw_3	$(K_B<<<_{111})_L$
	kw_4	$(K_B<<<_{111})_R$

5.4 SEED

5.4.1 The SEED algorithm

The SEED algorithm is a symmetric block cipher that can process data blocks of 128 bits, using a cipher key with length of 128 bits.

5.4.2 SEED encryption

The encryption operation is as shown in Figure 23. The transformation of a 128-bit block *P* into a 128-bit block *C* is defined as follows (*K* is a key):

(1) $P = L_0 \| R_0$

(2) for i = 1 to 15:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus \mathrm{F}(R_{i-1}, K_i)$

(3) $L_{16} = L_{15} \oplus \mathrm{F}(R_{15}, K_{16})$, $R_{16} = R_{15}$

(4) $C = L_{16} \| R_{16}$

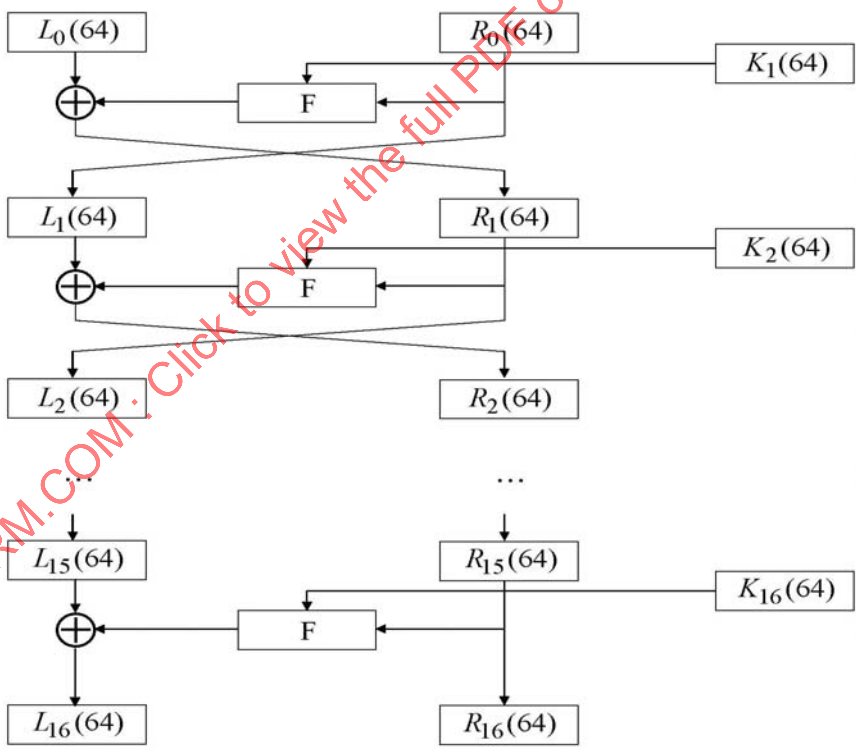

Figure 23 — Structure of SEED

5.4.3 SEED decryption

The decryption operation is identical to the encryption operation given above, except that the rounds (and therefore the subkeys) are used in reverse order to compute (L_0, R_0) from (R_{16}, L_{16}).

5.4.4 SEED functions

5.4.4.1 Round Function F

The Round Function F shown in Figure 24 is defined as follows (*C* and *D* are data, *K* is a key):

$C' = G[G[G\{(C \oplus k_{i,0}) \oplus (D \oplus k_{i,1})\} + (C \oplus k_{i,0})] + G\{(C \oplus k_{i,0}) \oplus (D \oplus k_{i,1})\}]$

$+ G[G\{(C \oplus k_{i,0}) \oplus (D \oplus k_{i,1})\} + (C \oplus k_{i,0})]$

$D' = G[G[G\{(C \oplus k_{i,0}) \oplus (D \oplus k_{i,1})\} + (C \oplus k_{i,0})] + G\{(C \oplus k_{i,0}) \oplus (D \oplus k_{i,1})\}]$

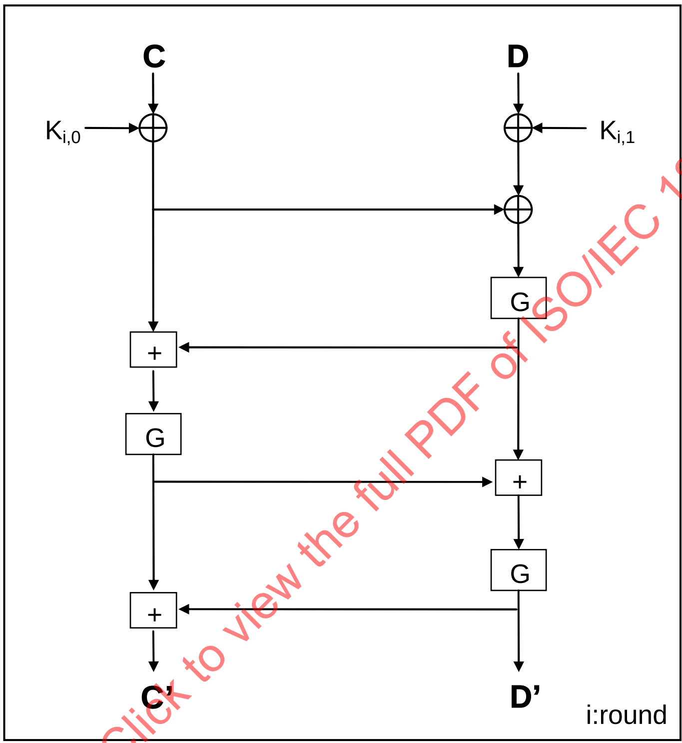

Figure 24 — Round Function F

5.4.4.2 Function G

The Function G shown in Figure 25 has two layers: a layer of two 8 × 8 S-boxes and a layer of block permutation of sixteen 8-bit sub-blocks.

The outputs a', b', c', d' of G with four 8-bit inputs a, b, c, d are as follows:

$a' = (S_1(a)\wedge m_0) \oplus (S_2(b)\wedge m_1) \oplus (S_1(c)\wedge m_2) \oplus (S_2(d)\wedge m_3)$

$b' = (S_1(a)\wedge m_1) \oplus (S_2(b)\wedge m_2) \oplus (S_1(c)\wedge m_3) \oplus (S_2(d)\wedge m_0)$

$c' = (S_1(a)\wedge m_2) \oplus (S_2(b)\wedge m_3) \oplus (S_1(c)\wedge m_0) \oplus (S_2(d)\wedge m_1)$

$d' = (S_1(a)\wedge m_3) \oplus (S_2(b)\wedge m_0) \oplus (S_1(c)\wedge m_1) \oplus (S_2(d)\wedge m_2)$

where, m_0 = {fc}, m_1 = {f3}, m_2 = {cf} and m_3 = {3f}.

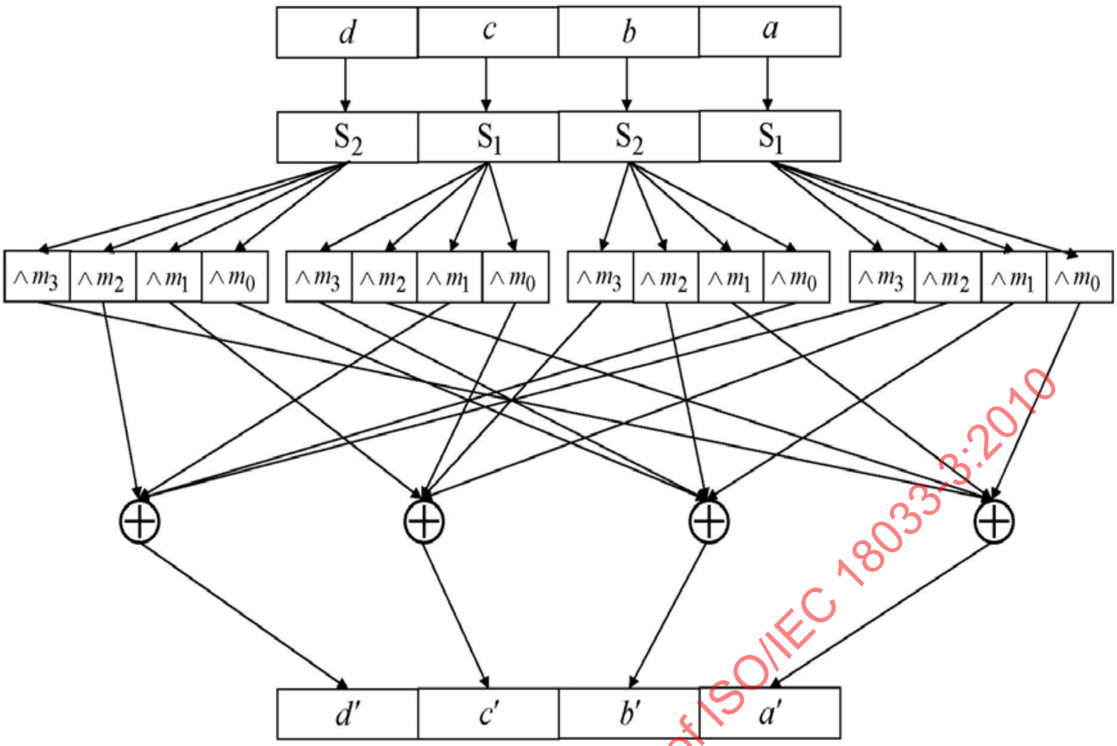

Figure 25 — The Function G

5.4.4.3 S-boxes

Two S-boxes S_1, S_2 are part of G and defined as follows:

$$S_i : Z_{2^8} \to Z_{2^8}, S_i(x) = A^{(i)} \bullet x^{n_i} \oplus b_i$$

where n_1=247, n_2=251, b_1=169, b_2=56 and

$$A^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}, A^{(2)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Notice that $A^{(i)} \bullet x^{n_i} \oplus b_i$ is an affine transformation of x^{n_i}. For any x in Z_{2^8}, x can be expressed as a binary vector form x = (x_7, … , x_0) (that is, $x = x_7 2^7 + x_6 2^6 + \ldots + x_1 2 + x_0$). We use the primitive polynomial $p(x) = x^8 + x^6 + x^5 + x + 1$ to represent x^{n_i} in Z_{2^8}.

S-boxes S_1 and S_2 are described in Tables 14 and 15, respectively.

Table 14 — S_1 – box

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

Table 15 — S_2 – box

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

5.4.5 SEED key schedule

The key schedule generates for each round subkeys. It uses the function G, additions/subtractions, and (left/right) rotations. A 128-bit input key is divided into four 32-bit blocks (a, b, c, d) and the two 32-bit subkeys of the 1st round, $k_{1,0}$ and $k_{1,1}$ are generated as following:

$$k_{1,0} = \mathrm{G}(a + c - KC_0)\ ,\ k_{1,1} = \mathrm{G}(b + KC_0 - d)$$

The two 32-bit subkeys of the 2nd round, $k_{2,0}$ and $k_{2,1}$ are generated from the input key with 8-bit right rotation of the first 64-bits($a \| b$) as follows:

$$a||b \leftarrow (a||b){>>>}_8$$

$$k_{2,0} = \mathrm{G}(a + c - KC_1)\ ,\ k_{2,1} = \mathrm{G}(b + KC_1 - d)$$

The two subkeys of the 3rd round, $k_{3,0}$ and $k_{3,1}$ are generated from the 8-bit left rotation of the last 64-bit(c || d) as follows:

$$c||d \leftarrow (c||d){<<<}_8$$

$$k_{3,0} = \mathrm{G}(a + c - KC_2)\ ,\ k_{3,1} = \mathrm{G}(b + KC_2 - d)$$

The rest of the subkeys are generated iteratively. A pseudo code for the key schedule is as follows:

(1) for i = 1 to 16:

$k_{i,0} = \mathrm{G}(a + c - KC_{i-1})$

$k_{i,1} = \mathrm{G}(b + KC_{i-1} - d)$

i: odd: $a \,||\, b = (a \,||\, b) >>>_8$

i: even: $c \,||\, d = (c \,||\, d) <<<_8$

where the constants KC_i (described in Table 16) are generated from a part of the golden ratio number $\frac{\sqrt{5}-1}{2}$.

Table 16 — Constants KC_i (in hexadecimal form)

i	KC_i	i	KC_i
0	9e3779b9	8	3779b99e
1	3c6ef373	9	6ef3733c
2	78dde6e6	10	dde6e678
3	f1bbcdcc	11	bbcdccf1
4	e3779b99	12	779b99e3
5	c6ef3733	13	ef3733c6
6	8dde6e67	14	de6e678d
7	1bbcdccf	15	bcdccf1b

Annex A
(normative)

Description of DES

A.1 Introduction

The DES algorithm is a symmetric block cipher that can process data blocks of 64 bits, using a cipher key with length of 64 bits. Every eighth bit of the cipher key is usually used for parity checking and is ignored.

A.2 DES encryption

The encryption operation is as shown in Figure A.1.

The 64-bit plaintext is first subjected to the initial permutation ***IP***. After the permutation, the block is split into two halves, $\boldsymbol{L_0}$ and $\boldsymbol{R_0}$, each of 32-bits. Then there are 16 rounds of identical operations called function ***f***, in which the data are combined with the key. During each round the right half is input to a keyed function ***f*** which accepts a 32-bit input and a 48-bit subkey $\boldsymbol{K_i}$ and produces a 32-bit output. This output is then XORed with the left half to produce a modified left half. At the end of each round except last round, the left and right halves are swapped to give $\boldsymbol{L_i}$ and $\boldsymbol{R_i}$, respectively. After last round, the left half and the right half are concatenated and the 64-bit block is then subjected to the final permutation $\boldsymbol{IP}^{-1}$ which is the inverse of the initial permutation. The output is the 64-bit ciphertext.

The encryption operation is thus defined as follows (*P* and *C* are data, K_i is a key).

(1) $IP(P) = L_0 \| R_0,$

(2) for *i* = 1, 2, …, 16:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

(3) $C = IP^{-1}(R_{16} \| L_{16})$

A.3 DES decryption

The decryption operation is the same as the encryption one. The only difference is that the subkeys $\boldsymbol{K_i}$ shall be used in the reverse order.

A.4 DES functions

A.4.1 Initial permutation *IP*

The initial permutation ***IP*** is shown in Table A.1. It accepts a 64-bit input and yields a 64-bit output.
That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit.

Table A.1 — Initial permutation *IP*

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

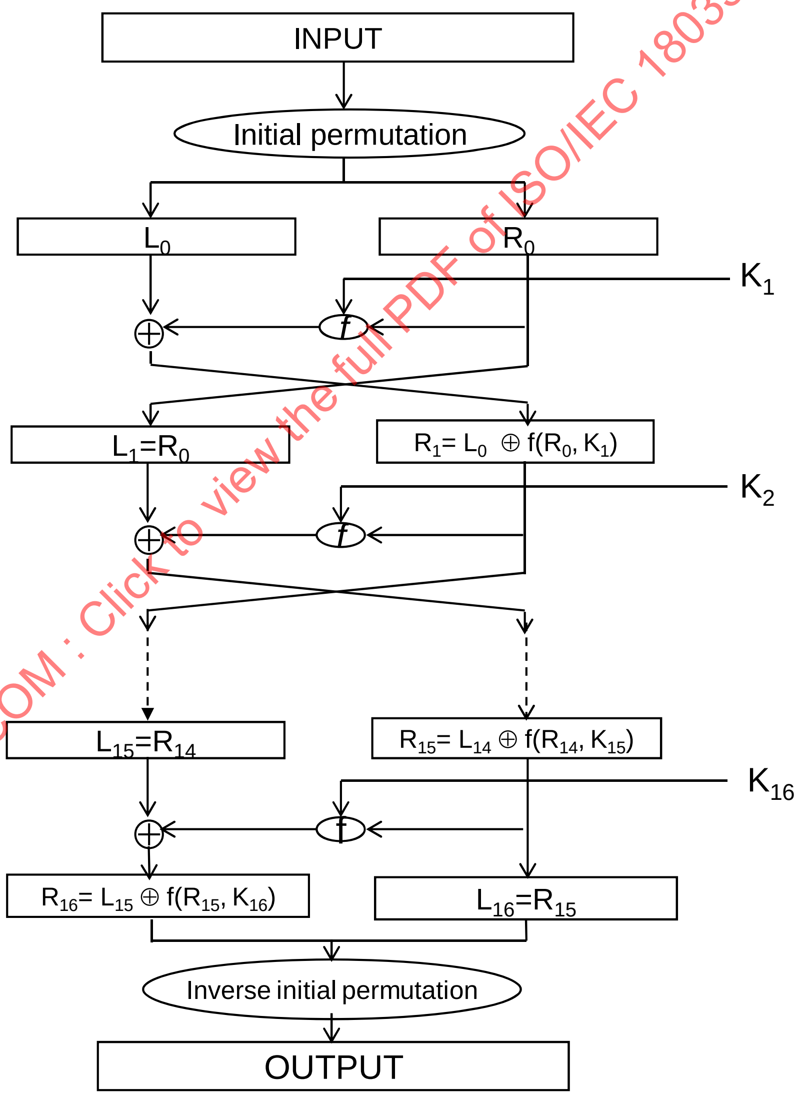

Figure A.1 — Encryption procedure

A.4.2 Inverse initial permutation IP^{-1}

The inverse initial permutation $\boldsymbol{IP}^{-1}$ is shown in Table A.2. It also accepts a 64-bit input and yields a 64-bit output. The output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

Table A.2 — Inverse initial permutation IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

A.4.3 Function *f*

The function ***f*** is shown in Figure A.2.

It takes a 32-bit input *R* and expands this to a 48-bit *R*' by using the expansion permutation ***E***. The 48-bit *R*' is XORed with a 48-bit subkey *K* and the computed 48-bit data, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the table. Each of the unique selection functions, called S-Boxes, $S_1, S_2, \ldots, S_8$, takes a 6-bit block r_i as input and yields a 4-bit block $S_i(r_i)$ as output. The permutation function ***P*** yields a 32-bit output *R*'''' from a 32-bit input *R*''' by permuting the bits of the input block. *R*'''' is the output of the function ***f***.

The function ***f*** is thus defined as follows (*P* and *C* are data, K_i is a key).

(1) $R' = E(R)$

(2) $R'' = R' \oplus K$

(3) $R'' = r_1 \| r_2 \| r_3 \| r_4 \| r_5 \| r_6 \| r_7 \| r_8$

$$R''' = S_1(r_1) \| S_2(r_2) \| S_3(r_3) \| S_4(r_4) \| S_5(r_5) \| S_6(r_6) \| S_7(r_7) \| S_8(r_8)$$

(4) $R'''' = P(R''')$

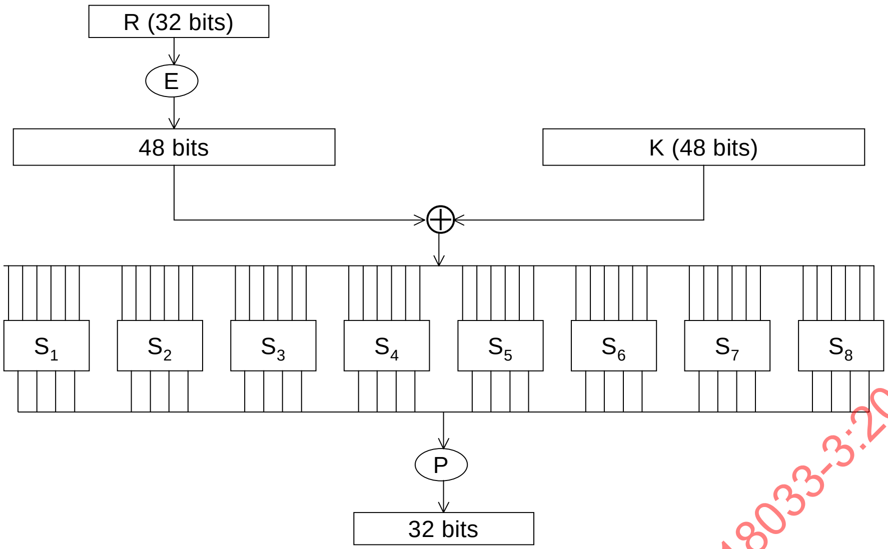

Figure A.2 — Calculation of f(R, K)

A.4.4 Expansion permutation *E*

The expansion permutation ***E*** is shown in Table A.3. It accepts a 32-bit input and yields a 48-bit output. The first three bits of ***E*** are the bits in positions 32, 1 and 2 while the last 2 bits are the bits in positions 32 and 1.

Table A.3 — Expansion permutation *E*

32	1	2	3	4	5	64	32
4	5	6	7	8	9	63	31
8	9	10	11	12	13	62	30
12	13	14	15	16	17	61	29
16	17	18	19	20	21	60	28
20	21	22	23	24	25	59	27
24	25	26	27	28	29	58	26
28	29	30	31	32	1	57	25

A.4.5 Permutation *P*

The permutation function ***P*** is shown in table A.4. It accepts a 32-bit input and yields a 32-bit output. The output ***P(L)*** for the function ***P*** defined by this table is obtained from the input ***L*** by taking the 16th bit of ***L*** as the first bit of ***P*(*L*)**, the 7th bit as the second bit of ***P*(*L*)**, and so on until the 25th bit of ***L*** is taken as the 32nd bit of ***P*(*L*)**.

Table A.4 — Permutation *P*

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

A.4.6 S-Boxes

The S-Boxes are shown in Table A.5. Each of them accepts a 6-bit input and yields a 4-bit output.

If S_1 is the function defined in the table and ***B*** is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of ***B*** represent in base 2 a number in the range 0 to 3. Let that number be *i*. The middle 4 bits of ***B*** represent in base 2 a number in the range 0 to 15. Let that number be *j*. Look up in the table the number in the *i*'th row and *j*'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input ***B***. For example, for input `011011` the row is `01`, that is row 1, and the column is determined by `1101`, that is column 13. In row 1 column 13 appears 5 so that the output is `0101`.

Table A.5 — S-Boxes

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

A.5 DES key schedule

The key scheduling part is shown in Figure A.3. It accepts a 64-bit key ***KEY*** and yields sixteen 48-bit subkeys $K_1, K_2, \ldots, K_{16}$.

Recall that $\boldsymbol{K_n}$, for $1 \le n \le 16$, is the block of 48 bits in (2) of the algorithm. Hence, to describe ***KS***, it is sufficient to describe the calculation of $\boldsymbol{K_n}$ from ***KEY*** for n = 1, 2,..., 16. That calculation is illustrated in Figure A.3. To complete the definition of ***KS*** it is therefore sufficient to describe the two permuted choices, as well as the schedule of left shifts. One bit in each 8-bit byte of the ***KEY*** may be utilized for error detection in key generation, distribution and storage. Bits 8, 16,..., 64 are for use in assuring that each byte is of odd parity. Permuted choice 1 is determined by Table A.6:

Table A.6 — Key permutation PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

The table has been divided into two parts, with the first part determining how the bits of $\boldsymbol{C_0}$ are chosen, and the second part determining how the bits of $\boldsymbol{D_0}$ are chosen. The bits of ***KEY*** are numbered 1 through 64. The bits of $\boldsymbol{C_0}$ are respectively bits 57, 49, 41,..., 44 and 36 of ***KEY***, with the bits of $\boldsymbol{D_0}$ being bits 63, 55, 47,..., 12 and 4 of ***KEY***. With $\boldsymbol{C_0}$ and $\boldsymbol{D_0}$ defined, we now define how the blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$ are obtained from the blocks $\boldsymbol{C_{n-1}}$ and $\boldsymbol{D_{n-1}}$, respectively, for n = 1, 2,..., 16. That is accomplished by adhering to the following schedule of left shifts of the individual blocks:

Table A.7 — Number of key bits shifted per round

Iteration Number	Number of Left shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
Iteration Number	**Number of Left shifts**
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

For example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1. Permuted choice 2 is determined by Table A.8:

Table A.8 — Compression permutation *PC-2*

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

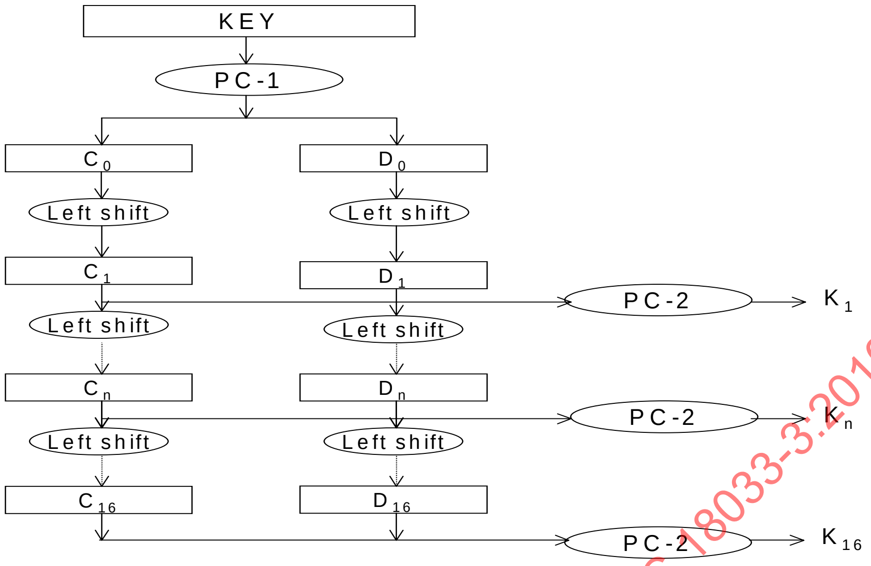

Figure A.3 — Key schedule calculation

Annex B
(normative)

Object identifiers

This annex lists the object identifiers assigned to algorithms specified in this part of ISO/IEC 18033.

NOTE In applications where a combination of algorithms is used to provide security services or when an algorithm is parameterised by the choice of a combination of other algorithms such a combination may be specified as a sequence of object identifiers assigned to these algorithms or by including the object identifiers of lower layer algorithms (for example by specifying the object identifier of a key encapsulation mechanism as a parameter in the algorithm identifier structure specifying a hybrid encryption algorithm). The algorithm identifier structure is defined in ISO/IEC 9594-8.

```
--
-- ISO/IEC 18033-3 ASN.1 Module
--

EncryptionAlgorithms-3 {
   iso(1) standard(0) encryption-algorithms(18033) part(3)
      asn1-module(0) algorithm-object-identifiers(0) version(1)}
   DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

-- IMPORTS None; --

OID ::= OBJECT IDENTIFIER -- Alias

-- Synonyms --
is18033-3 OID ::= {iso(1) standard(0) encryption-algorithms(18033) part(3)}

id-bc64   OID ::= { is18033-3 cipher-64-bit(1)  }
id-bc128  OID ::= { is18033-3 cipher-128-bit(2) }

-- Assignments --

id-bc64-tdea        OID ::= {id-bc64 tdea(1)      }
id-bc64-misty1      OID ::= {id-bc64 misty1(2)    }
id-bc64-cast128     OID ::= {id-bc64 cast128(3)   }
id-bc64-hight       OID ::= {id-bc64 hight(4)      }
id-bc128-aes        OID ::= {id-bc128 aes(1)      }
id-bc128-camellia   OID ::= {id-bc128 camellia(2) }
id-bc128-seed       OID ::= {id-bc128 seed(3)     }

EncryptionAlgorithmIdentifier{ALGORITHM:BlockAlgorithms} ::= SEQUENCE {
   algorithm    ALGORITHM.&id({BlockAlgorithms}),
   parametersALGORITHM.&Type({BlockAlgorithms}{@algorithm}) OPTIONAL
}

BlockAlgorithms ALGORITHM ::= {

    { OID id-bc64-tdea        PARMS KeyLengthID } |
    { OID id-bc64-misty1      PARMS KeyLength   } |
    { OID id-bc64-cast128     PARMS KeyLength   } |
```



```
     { OID id-bc64-hight       PARMS KeyLength   } |
     { OID id-bc128-aes        PARMS KeyLengthID } |
     { OID id-bc128-camellia   PARMS KeyLengthID } |
     { OID id-bc128-seed       PARMS KeyLength   },

    ...   –- Except 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  -- EncryptionAlgorithms-3 --
```

Annex C
(informative)

Algebraic forms of MISTY1 and Camellia S-boxes

C.1 Introduction

In MISTY1 and Camellia algorithms, S-boxes can be described by the following algebraic forms respectively.

C.2 MISTY1 S-boxes

C.2.1 The S-boxes S_7 and S_9

This clause describes the algebraic forms over GF(2) of S_7 and S_9 in the MISTY1 algorithm.

For example, if the input to S_7 is {53}, then $X = (x_6, x_5, x_4, x_3, x_2, x_1, x_0) = (1, 0, 1, 0, 0, 1, 1)$. The substitution value would be derived by the algebraic form in Figure C.1 and $Y = (y_6, y_5, y_4, y_3, y_2, y_1, y_0) = (1, 0, 1, 0, 1, 1, 1)$. This would result in S_7 having a value of {57}.

C.2.2 MISTY1 S-box S_7

$$y_0 = x_0 \oplus x_1 x_3 \oplus x_0 x_3 x_4 \oplus x_1 x_5 \oplus x_0 x_2 x_5 \oplus x_4 x_5 \oplus x_0 x_1 x_6 \oplus x_2 x_6 \oplus x_0 x_5 x_6 \oplus x_3 x_5 x_6 \oplus 1$$
$$y_1 = x_0 x_2 \oplus x_0 x_4 \oplus x_3 x_4 \oplus x_1 x_5 \oplus x_2 x_4 x_5 \oplus x_6 \oplus x_0 x_6 \oplus x_3 x_6 \oplus x_2 x_3 x_6 \oplus x_1 x_4 x_6 \oplus x_0 x_5 x_6 \oplus 1$$
$$y_2 = x_1 x_2 \oplus x_0 x_2 x_3 \oplus x_4 \oplus x_1 x_4 \oplus x_0 x_1 x_4 \oplus x_0 x_5 \oplus x_0 x_4 x_5 \oplus x_3 x_4 x_5 \oplus x_1 x_6 \oplus x_3 x_6 \oplus x_0 x_3 x_6 \oplus x_4 x_6 \oplus x_2 x_4 x_6$$
$$y_3 = x_0 \oplus x_1 \oplus x_0 x_1 x_2 \oplus x_0 x_3 \oplus x_2 x_4 \oplus x_1 x_4 x_5 \oplus x_2 x_6 \oplus x_1 x_3 x_6 \oplus x_0 x_4 x_6 \oplus x_5 x_6 \oplus 1$$
$$y_4 = x_2 x_3 \oplus x_0 x_4 \oplus x_1 x_3 x_4 \oplus x_5 \oplus x_2 x_5 \oplus x_1 x_2 x_5 \oplus x_0 x_3 x_5 \oplus x_1 x_6 \oplus x_1 x_5 x_6 \oplus x_4 x_5 x_6 \oplus 1$$
$$y_5 = x_0 \oplus x_1 \oplus x_2 \oplus x_0 x_1 x_2 \oplus x_0 x_3 \oplus x_1 x_2 x_3 \oplus x_1 x_4 \oplus x_0 x_2 x_4 \oplus x_0 x_5 \oplus x_0 x_1 x_5 \oplus x_3 x_5 \oplus x_0 x_6 \oplus x_2 x_5 x_6$$
$$y_6 = x_0 x_1 \oplus x_3 \oplus x_0 x_3 \oplus x_2 x_3 x_4 \oplus x_0 x_5 \oplus x_2 x_5 \oplus x_3 x_5 \oplus x_1 x_3 x_5 \oplus x_1 x_6 \oplus x_1 x_2 x_6 \oplus x_0 x_3 x_6 \oplus x_4 x_6 \oplus x_2 x_5 x_6$$

Figure C.1 — Algebraic Form of S_7 ($y_6 2^6 \oplus y_5 2^5 \oplus \ldots \oplus y_1 2^1 \oplus y_0 2^0 = S_7(x_6 2^6 \oplus x_5 2^5 \oplus \ldots \oplus x_1 2^1 \oplus x_0 2^0)$)

C.2.3 MISTY1 S-box S_9

$$y_0 = x_0 x_4 \oplus x_0 x_5 \oplus x_1 x_5 \oplus x_1 x_6 \oplus x_2 x_6 \oplus x_2 x_7 \oplus x_3 x_7 \oplus x_3 x_8 \oplus x_4 x_8 \oplus 1$$
$$y_1 = x_0 x_2 \oplus x_3 \oplus x_1 x_3 \oplus x_2 x_3 \oplus x_3 x_4 \oplus x_4 x_5 \oplus x_0 x_6 \oplus x_2 x_6 \oplus x_7 \oplus x_0 x_8 \oplus x_3 x_8 \oplus x_5 x_8 \oplus 1$$
$$y_2 = x_0 x_1 \oplus x_1 x_3 \oplus x_4 \oplus x_0 x_4 \oplus x_2 x_4 \oplus x_3 x_4 \oplus x_4 x_5 \oplus x_0 x_6 \oplus x_5 x_6 \oplus x_1 x_7 \oplus x_3 x_7 \oplus x_8$$
$$y_3 = x_0 \oplus x_1 x_2 \oplus x_2 x_4 \oplus x_5 \oplus x_1 x_5 \oplus x_3 x_5 \oplus x_4 x_5 \oplus x_5 x_6 \oplus x_1 x_7 \oplus x_6 x_7 \oplus x_2 x_8 \oplus x_4 x_8$$
$$y_4 = x_1 \oplus x_0 x_3 \oplus x_2 x_3 \oplus x_0 x_5 \oplus x_3 x_5 \oplus x_6 \oplus x_2 x_6 \oplus x_4 x_6 \oplus x_5 x_6 \oplus x_6 x_7 \oplus x_2 x_8 \oplus x_7 x_8$$
$$y_5 = x_2 \oplus x_0 x_3 \oplus x_1 x_4 \oplus x_3 x_4 \oplus x_1 x_6 \oplus x_4 x_6 \oplus x_7 \oplus x_3 x_7 \oplus x_5 x_7 \oplus x_6 x_7 \oplus x_0 x_8 \oplus x_7 x_8$$
$$y_6 = x_0 x_1 \oplus x_3 \oplus x_1 x_4 \oplus x_2 x_5 \oplus x_4 x_5 \oplus x_2 x_7 \oplus x_5 x_7 \oplus x_8 \oplus x_0 x_8 \oplus x_4 x_8 \oplus x_6 x_8 \oplus x_7 x_8 \oplus 1$$
$$y_7 = x_1 \oplus x_0 x_1 \oplus x_1 x_2 \oplus x_2 x_3 \oplus x_0 x_4 \oplus x_5 \oplus x_1 x_6 \oplus x_3 x_6 \oplus x_0 x_7 \oplus x_4 x_7 \oplus x_6 x_7 \oplus x_1 x_8 \oplus 1$$
$$y_8 = x_0 \oplus x_0 x_1 \oplus x_1 x_2 \oplus x_4 \oplus x_0 x_5 \oplus x_2 x_5 \oplus x_3 x_6 \oplus x_5 x_6 \oplus x_0 x_7 \oplus x_0 x_8 \oplus x_3 x_8 \oplus x_6 x_8 \oplus 1$$

Figure C.2 — Algebraic Form of S_9 ($y_8 2^8 \oplus y_7 2^7 \oplus \ldots \oplus y_1 2^1 \oplus y_0 2^0 = S_9(x_8 2^8 \oplus x_7 2^7 \oplus \ldots \oplus x_1 2^1 \oplus x_0 2^0)$)