The Block Cipher SC2000

Takeshi Shimoyama[1], Hitoshi Yanami[1], Kazuhiro Yokoyama[1], Masahiko Takenaka[2], Kouichi Itoh[2], Jun Yajima[2], Naoya Torii[2], and Hidema Tanaka[3]

[1] Fujitsu Laboratories LTD.
4-1-1, Kamikodanaka Nakahara-ku Kawasaki 211-8588, Japan
{shimo,yanami,kayoko}@flab.fujitsu.co.jp
[2] Fujitsu Laboratories LTD.
64, Nishiwaki, Ohkubo-cho, Akashi 674-8555, Japan
{takenaka,kito,jyajima,torii}@flab.fujitsu.co.jp
[3] Science University of Tokyo
Yamazaki, Noda, Chiba, 278 Japan
tanaka@ee.noda.sut.ac.jp

Abstract. In this paper, we propose a new symmetric key block cipher SC2000 with 128-bit block length and 128-,192-,256-bit key lengths. The block cipher is constructed by piling two layers: one is a Feistel structure layer and the other is an SPN structure layer. Each operation used in two layers is S-box or logical operation, which has been well studied about security. It is a strong feature of the cipher that the fast software implementations are available by using the techniques of putting together S-boxes in various ways and of the Bitslice implementation.

1 Introduction

In this paper, we propose a new block cipher SC2000. The algorithm has 128-bit data inputs and outputs and 128-bit, 192-bit, and 256-bit keys can be used. Our design policy for the cipher is to enable high-speed in software and hardware processing on various platforms while maintaining the high-level security as cipher. The cipher is constructed by piling a Feistel and an SPN structures with S-boxes for realizing those properties.

For evaluation of security, we inspect the security in the design against differential attacks, linear attacks, higher order differential attacks, interpolation attacks, and so on. We employ a structure that facilitates security verification by using modules with established reputation concerning security. In S-box design, we use composite maps of an exponentiation over a Galois field and affine transformations[8]. For the diffusion layer in the inner function of Feistel structure, we use an MDS matrix[4,5,7,10]. We set the number of rounds as for having a sufficiently good security margin.

As high-speed implementation technology, we designed nonlinear arithmetic operations to enable high-speed implementation on a wide range of processors. Individual implementations are realized by selecting optimum parameters corresponding to the register lengths and cache memory sizes. Moreover, for the

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 312–327, 2002.

fast software encryption of SPN structure, we adopted a structure that permits the use of latest high-speed implementation called Bitslice[1,2,3,9]. In hardware implementation, we aimed at a structure that enables extremely compact implementation by using nonlinear arithmetic and logical units with up to 6-bit input/output in the encryption.

2 Preliminary

Basically, the following rules are used for notation:

- *Big-Endian* is used as *Endian*, where the highest bit in each value a is the 0-th bit of a. For example, $a = 10$ (decimal (4-bit) integer) is expressed as $(a_0, a_1, a_2, a_3) = (1, 0, 1, 0)$.
- The number of bits in a variable is expressed by each variable superscripted with a decimal number enclosed by parentheses as $a^{(4)}$. For 32-bit variables, the number of bits is omitted as a.
- A number without a prefix expresses a decimal number, and a number prefixed with `0x` expresses a hexadecimal number like as `0x01234567`.
- XOR is the exclusive-or of two variables and expressed as $a \oplus b$.
- AND is the logical product of two variables and expressed as $a \wedge b$.
- OR is the logical sum of two variables and expressed as $a \vee b$.
- NOT is an operation that inverts all bits and expressed as $\bar{b}$.
- ADD is an operation that applies modulo 2^{32} operation $(a + b \pmod{2^{32}})$ to the result of addition of two 32-bit variables and expressed as $a \boxplus b$.
- SUB is an operation that applies modulo 2^{32} operation $(a - b \pmod{2^{32}})$ to the result of subtraction of two 32-bit variables and expressed as $a \boxminus b$.
- MUL is an operation that applies modulo 2^{32} operation $(a \times b \pmod{2^{32}})$ to the result of multiplication of two 32-bit variables and expressed as $a \boxtimes b$.
- Rotate left one bit is an operation that rotates a 32-bit variable left by 1 bit $((a_0, a_1, \cdots, a_{31}) \rightarrow (a_1, a_2, \cdots, a_0))$ and expressed as $a \lll_1$.

3 Algorithm Specifications

3.1 Encryption Function

The encryption function consists of the following three functions: I function, B function, and R function, each of which has a (32 bits $\times$ 4) input/output. Among these three functions, the I function XORs the key and the B and R functions stir the data. The encryption function for a 128-bit key consists of 14 rounds of the I function, which XORs the key, and as data randomizing, 7 rounds of the B function and 12 rounds of the R function, totaling 19 rounds. The encryption function for a 192-bit or 256-bit key consists of 16 rounds of the I function, which XORs the key, and as data randomizing, 8 rounds of the B function and 14 rounds of the R function, totaling 22 rounds. Each function is executed in the order of I-B-I-$R \times R \cdots$ repeatedly. (See the table below.) For a

128-bit key, fifty-six 32-bit extended keys are used; for a 192-bit or 256-bit key, sixty-four 32-bit extended keys are used.

The following lists the entire configuration of the encryption function. Symbols used in the configuration are as follows: (See also Fig.1.)

I	I function
B	B function
$R5$	R function with $mask =$ `0x55555555`
$R3$	R function with $mask =$ `0x33333333`
$-$	Straight connection $(a, b, c, d) \rightarrow (a, b, c, d)$
$\times$	Cross connection $(a, b, c, d) \rightarrow (c, d, a, b)$

Configuration for 128-bit key:
(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-(out)
Configuration for 192-bit or 256-bit key:
(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-(out)

3.2 Decryption Function

The decryption function consists of the following three functions: I function, B^{-1} function, and R function, each of which has a (32 bits $\times$ 4) input/output. Among these three functions, the I function and R function is as same as one in the Encryption function, and the B^{-1} functin is the inverse function of the B function. Each function is executed in the order of I-B^{-1}-I-$R \times R \cdots$ repeatedly.

The following lists the entire configuration of the decryption function. Symbols used in the configuration are as follows: (See Fig.1.)

I	I function
B^{-1}	B^{-1} function
$R5$	R function with $mask =$ `0x55555555`
$R3$	R function with $mask =$ `0x33333333`
$-$	Straight connection $(a, b, c, d) \rightarrow (a, b, c, d)$
$\times$	Cross connection $(a, b, c, d) \rightarrow (c, d, a, b)$

Configuration for 128-bit key:
(in)-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)
Configuration for 192-bit or 256-bit key:
(in)-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)

3.3 *I* Function

The I function is given four 32-bit variables and four 32-bit extended keys as inputs, and it outputs four 32-bit variables. Each input data is XORed with the extended key. The extended keys are inputs only to the I functions.

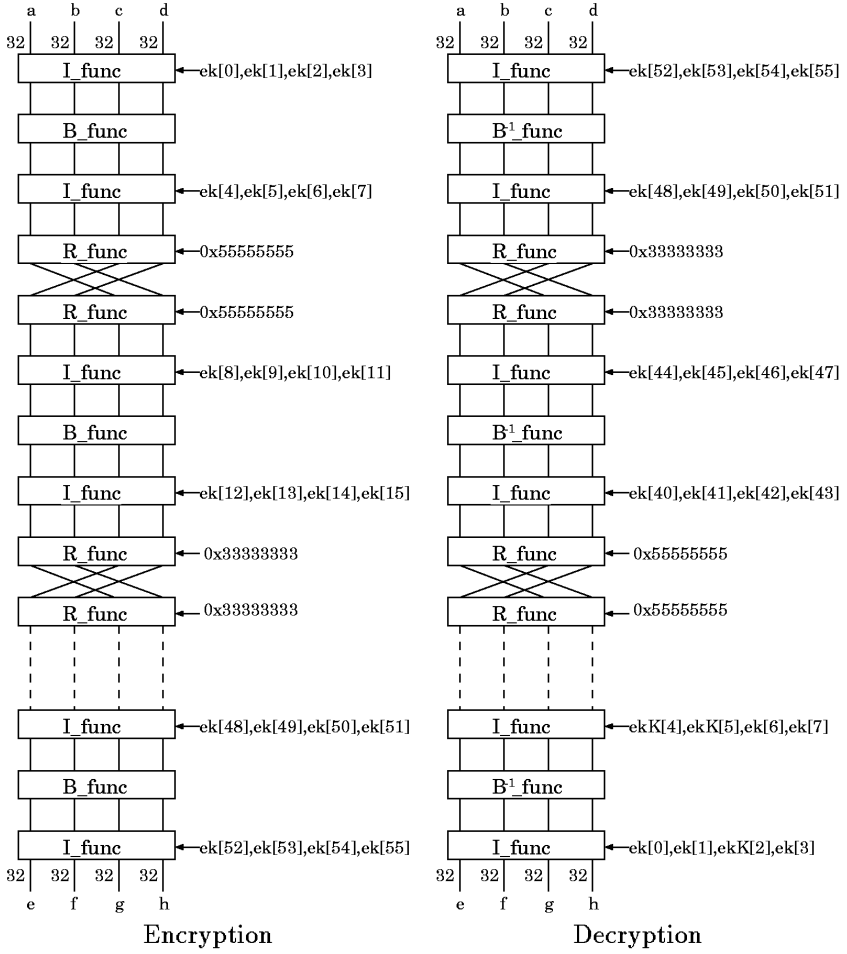


Fig. 1. Encryption and Decryption function (128-bit key)

3.4 *R* Function

The R function is a Feistel-type data randomizing function having four 32-bit variables as the input and the output. The R function inputs the third entry and the fourth entry (c, d) of the input data and the constant data mask to the F function, and it XORs the two outputs from the F function with the first entry and the second entry (a, b) of the input data. (See Fig.2.)

***F* function.** The F function is given two 32-bit variables and constant as inputs, and outputs two 32-bit variables. The F function processes two variables with the S function and the M function respectively and then processes the two outputs with the L function.

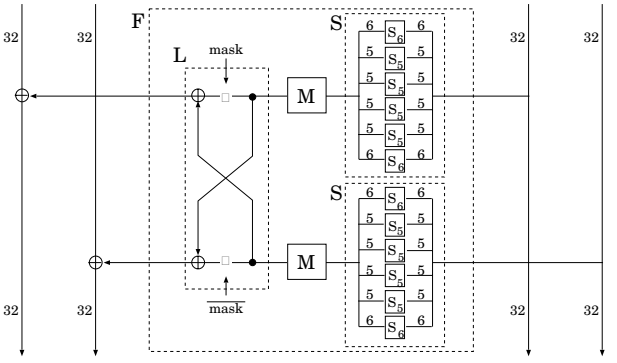


Fig. 2. R function

***S* function.** The S function splits the 32-bit input into 6 bits, 5 bits, 5 bits, 5 bits, 5 bits and 6 bits. The S function then looks up the 6-bit S-box table S_6 with the original 6 bits; it looks up the 5-bit S-box table S_5 with the 5 bits. The function then aligns individual outputs in the same order to generate 32 bits.

Each S-box is a composition of an exponentiation over a Galois field and an affine map, whose maximum differential/linear probability is 2^{-4}, and the total degree is 3 for S_5, and is 5 for S_6. See Section 4 for the elements of the tables S_5 and S_6.

***M* function.** The M function is a 32-bit input/output bitwise linear function. The M function considers the input a as a vector of 32 entries of 1 bit and outputs $a_0 \cdot \mathtt{M}[0] \oplus \cdots \oplus a_{31} \cdot \mathtt{M}[31]$. See Section 4 for the elements of the matrix `M`.

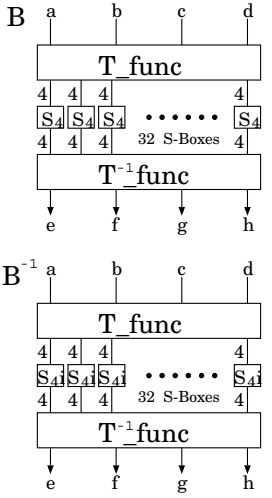
This function is designed as an MDS matrix with respect to the S function. The summation of the number of an input active S-box and the number of the output active S-box correspond to the input is greater than or equal to 6.

***L* function.** The L function is given two 32-bit variables as inputs and outputs two 32-bit variables, and a constant. The L function outputs two data $((a \wedge mask) \oplus b, (b \wedge (\overline{mask})) \oplus a)$ by processing two input values (a, b). As the $mask$ values used in the L function, there are two values: `0x55555555` and `0x33333333` for increasing the security.

Software implemenation techniques of the *F* function. In the description of the F function, the S function, the M function and the L function are executed individually in this order. However, the S and the M functions can be composed because the M function is a linear map. In the explanation of the S function, 5-bit and 6-bit S-box tables are looked up by (6,5,5,5,5,6). However, executing the table lookups by (6,10,10,6) or (11,10,11), which is obtained by jointing two consecutive S-boxes into a new one, speeds up the operations because the number

of table lookups is reduced. In this case, creating the table $M \circ S_6$(6-bit input, 32-bit output) and $M \circ (S_5 S_5)$(10-bit input, 32-bit output) by composing two tables can reduce the number of table lookups, leading to a speedup of operations. Also, creating the table $M \circ (S_6 S_5)$ and $M \circ (S_5 S_6)$ (11-bit input, 32-bit output) or composing the L function can realize a further speedup in the processor with a larger cashe memory.

Implemenataion	# of Table Lookups	Size of Tables
(6,5,5,5,5,6)	6	1 KByte
(6,10,10,6)	4	9 KByte
(11,10,11)	3	20 KByte



$B\_func()$

$t1 = \overline{b}$
$t2 = a \vee d$
$t3 = c \oplus t2$
$t4 = t1 \wedge t3$
$e = d \oplus t4$
$t6 = t2 \oplus e$
$t7 = a \oplus b$
$t8 = t6 \vee t7$
$f = c \oplus t8$
$t10 = c \wedge t3$
$t11 = t1 \oplus t10$
$t12 = c \oplus t4$
$t13 = t6 \wedge t12$
$h = t7 \oplus t13$
$t15 = c \vee h$
$g = t11 \oplus t15$

$B^{-1}\_func()$

$t1 = c \oplus d$
$t2 = t1 \vee b$
$t3 = t2 \oplus c$
$t4 = t3 \wedge a$
$t5 = \overline{c}$
$t6 = t5 \vee b$
$t7 = t6 \oplus d$
$e = t7 \oplus t4$
$t8 = a \oplus t6$
$t9 = t8 \vee d$
$g = t9 \oplus b$
$t10 = c \oplus g$
$t11 = t10 \vee t4$
$t12 = t11 \wedge t9$
$f = t12 \oplus e$
$t13 = t6 \oplus g$
$t14 = t13 \wedge t1$
$h = t14 \oplus a$

Fig. 3. B/B^{-1} function and an example of Bitslice implementation

3.5 B/B^{-1} Function

The B function converts four 32-bit inputs into thirty two 4-bit values with the T function, each of which is processed with the same 4-bit S-box table S_4. This function then recover the output values into four 32-bit values and outputs those values with the T^{-1} funciton. (See Fig.3.) The B^{-1} function uses $S_4 i$ which is the inverse of the 4-bit S-box table S_4. See Section 4 for the elements of the tables S_4 and $S_4 i$.

The B/B^{-1} function can execute Bitslicing by expressing the S_4 table in logical form[1,2]. Bitslicing enables an aggregate processing of thirty-two S-box lookups, leading to calculation speedup. For example, the B and B^{-1} function can be expressed in logical form as shown in the Fig. 3.

T/T^{-1} function. The T function converts four 32-bit inputs to the (4,32)-matrix, and it converts the matrix into the transposed (32,4)-matrix, and then converts it into thirty two 4-bit data values. The T^{-1} function is the inverse function of T.

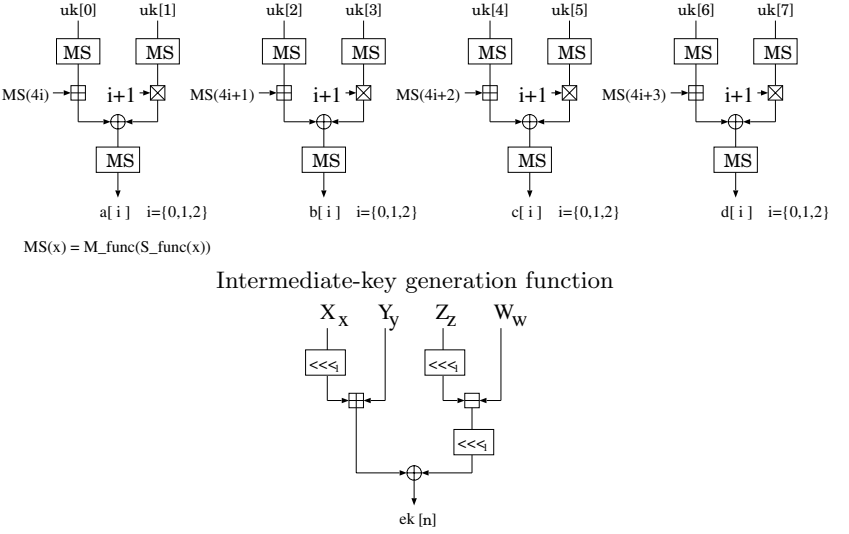


Fig. 4. Key Schedule

3.6 Key Schedule Specifications

The key schedule generates fifty-six 32-bit extended keys (for 128-bit key) or sixty-four 32-bit extended keys (for 192-bit or 256-bit key) from the secret key. The key schedule is a function consisting of the intermediate-key generation function and the extended-key generation function. For a 256-bit key, the key schedule splits the 256-bit secret key into eight 32-bit user keys $uk[0], \ldots, uk[7]$. For a 192-bit key, the key schedule splits the 192-bit secret key into six 32-bit user keys $uk[0], \ldots, uk[5]$ like as 256-bit key, then the function expands the 6 user keys to 8 user keys by $uk[6] = uk[0]$ and $uk[7] = uk[1]$. In the same way, for a 128-bit key, the key schedule splits the 128-bit secret key into four 32-bit user keys $uk[0], \ldots, uk[3]$, then expands the 4 user keys to 8 user keys by $uk[4] = uk[0]$, $uk[5] = uk[1]$, $uk[6] = uk[2]$ and $uk[7] = uk[3]$. From the user keys $uk[]$, it generates the intermediate keys $imkey[]$ with the intermediate-key generation function and then executes the extended-key generation function to generate 32-bit extended keys $ek[]$.

Intermediate-key Generation Function. This function generates twelve 32-bit intermediate keys $a[i], b[i], c[i], d[i]$ for $i \in \{0, 1, 2\}$ from eight 32-bit user keys by the following processing.

for $(i = 0; i < 3; i++)$ {
$a[i] = M(S((M(S(4i)) \boxtriangle M(S(uk[0])))\oplus (M(S(uk[1])) \boxminus (i+1))))$
$b[i] = M(S((M(S(4i+1)) \boxtriangle M(S(uk[2])))\oplus (M(S(uk[3])) \boxminus (i+1))))$
$c[i] = M(S((M(S(4i+2)) \boxtriangle M(S(uk[4])))\oplus (M(S(uk[5])) \boxminus (i+1))))$
$d[i] = M(S((M(S(4i+3)) \boxtriangle M(S(uk[6])))\oplus (M(S(uk[7])) \boxminus (i+1))))$
}

Extended-key Generation Function. This function generates fifty-six 32-bit extended keys from twelve 32-bit intermediate keys for a 128-bit key; and also generates sixty-four 32-bit extended keys for a 192-bit or 256-bit key.

if $(KeyLength == 128)$ $num\_ekey = 56$ else $num\_ekey = 64$
for $(n = 0; n < num\_ekey; n++)$ {
$s = n \pmod 9$
$t = (n + \lfloor n/36 \rfloor) \pmod{12}$
$X = Order[t][0]$ $Y = Order[t][1]$ $Z = Order[t][2]$ $W = Order[t][3]$
$x = Index[s][0]$ $y = Index[s][1]$ $z = Index[s][2]$ $w = Index[s][3]$
$ek[n] = ((X[x] \lll n_1) \boxtriangle Y[y]) \oplus (((Z[z] \lll n_1) \ggg W[w]) \lll n_1)$
}

4 Table

The following shows the tables used by functions for the cipher:

```
S6[64]  = {47,59,25,42,15,23,28,39,26,38,36,19,60,24,29,56,
           37,63,20,61,55, 2,30,44, 9,10, 6,22,53,48,51,11,
           62,52,35,18,14,46, 0,54,17,40,27, 4,31, 8, 5,12,
            3,16,41,34,33, 7,45,49,50,58, 1,21,43,57,32,13};
S5[32]  = {20,26,7,31,19,12,10,15,22,30,13,14, 4,24, 9,18,
           27,11, 1,21, 6,16, 2,28,23, 5, 8, 3, 0,17,29,25};
S4[16]  = { 2, 5,10,12, 7,15,1,11,13, 6, 0, 9, 4, 8, 3,14};
S4i[16] = {10, 6, 0,14,12, 1, 9,4,13,11, 2, 7, 3, 8,15, 5};
M[32] = {0xd0c19225,0xa5a2240a,0x1b84d250,0xb728a4a1,
         0x6a704902,0x85dddbe6,0x766ff4a4,0xecdfe128,
         0xafd13e94,0xdf837d09,0xbb27fa52,0x695059ac,
         0x52a1bb58,0xcc322f1d,0x1844565b,0xb4a8acf6,
         0x34235438,0x6847a851,0xe48c0cbb,0xcd181136,
         0x9a112a0c,0x43ec6d0e,0x87d8d27d,0x487dc995,
         0x90fb9b4b,0xa1f63697,0xfc513ed9,0x78a37d93,
         0x8d16c5df,0x9e0c8bbe,0x3c381f7c,0xe9fb0779};
```

Order

t	0	1	2	3	4	5	6	7	8	9	10	11
X	a	b	c	d	a	b	c	d	a	b	c	d
Y	b	a	d	c	c	d	a	b	d	c	b	a
Z	c	d	a	b	d	c	b	a	b	a	d	c
W	d	c	b	a	b	a	d	c	c	d	a	b

Index

s	0	1	2	3	4	5	6	7	8
x	0	1	2	0	1	2	0	1	2
y	0	1	2	1	2	0	2	0	1
z	0	1	2	0	1	2	0	1	2
w	0	1	2	1	2	0	2	0	1

Table 1. Distribution tables for differential/linear approximations of S_4

differential

ΔIn \ ΔOut	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	0	0	2	2	2	2	2	2	2	2	0	0
0x2	0	0	0	0	2	0	4	2	2	2	0	0	0	2	0	2
0x3	0	0	0	0	2	0	2	0	0	0	2	2	2	0	4	2
0x4	0	0	0	2	0	2	0	4	0	2	2	2	0	0	2	0
0x5	0	2	4	0	0	2	0	0	0	0	2	0	0	4	2	0
0x6	0	2	0	4	2	0	0	0	2	0	0	0	0	2	4	0
0x7	0	0	0	2	2	4	0	0	2	2	0	2	0	2	0	0
0x8	0	0	2	4	0	4	0	2	0	0	2	0	0	0	0	2
0x9	0	0	0	2	2	0	0	0	4	0	0	2	2	0	0	4
0xa	0	2	2	2	2	2	0	2	0	0	2	0	2	0	0	0
0xb	0	2	0	0	0	2	0	0	0	4	0	2	4	0	0	2
0xc	0	2	4	0	0	0	2	0	0	4	2	0	0	2	0	0
0xd	0	4	2	0	2	0	0	0	2	0	2	2	0	0	0	2
0xe	0	2	0	0	2	0	2	2	0	0	0	2	2	2	2	0
0xf	0	0	2	0	0	0	4	2	2	0	0	0	2	0	2	2

$(\#\{In|\Delta In \to \Delta Out\})$

linear

ΓIn \ ΓOut	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	2	2	-2	-2	4	0	0	4	2	-2	2	-2
0x2	0	0	0	0	-4	4	0	0	2	2	-2	-2	-2	-2	-2	-2
0x3	0	0	0	0	-2	-2	-2	-2	-2	2	-2	2	0	4	0	-4
0x4	0	2	2	-4	0	2	-2	0	0	2	-2	0	0	2	2	4
0x5	0	2	-2	0	2	4	0	2	-4	2	2	0	2	0	0	-2
0x6	0	-2	-2	-4	0	-2	-2	4	2	0	0	-2	2	0	0	-2
0x7	0	-2	2	0	2	0	0	-2	-2	0	-4	-2	4	-2	-2	0
0x8	0	-2	-2	0	0	2	-2	-4	0	-2	2	-4	0	2	2	0
0x9	0	2	-2	-4	2	0	4	-2	0	-2	-2	0	-2	0	0	-2
0xa	0	-2	-2	0	0	2	2	0	2	0	0	2	2	4	-4	2
0xb	0	2	-2	4	2	0	0	2	2	0	-4	-2	0	2	2	0
0xc	0	4	0	0	0	0	-4	0	0	-4	0	0	0	0	-4	0
0xd	0	0	-4	0	2	-2	-2	-2	0	4	0	0	-2	-2	-2	2
0xe	0	0	4	0	4	0	0	0	2	2	2	-2	-2	2	-2	-2
0xf	0	4	0	0	-2	-2	2	-2	2	2	2	-2	4	0	0	0

$(\#\{In|In{\cdot}\Gamma In \oplus Out{\cdot}\Gamma Out = 0\} - 8)$

5 Security Evaluation Policy

Since differential attacks and linear attacks were presented, designers of block cipher need to consider the security specifically against these attacks among standard attacks, and trials to show the security against these attacks by several methods have been made.

While a method of finding the theoretical upper bounds of security can show the security of cipher theoretically, such a method has certain constraints in the structure of the cipher to apply the theory. The block cipher SC2000 cannot be directly applied those techniques because the cipher uses a *combined* structure of the Feistel structure and the SPN structure. It is also difficult to construct a new technique to evaluate the security of the structure of the cipher. Therefore, we employ the following method to show the security of the cipher:

> Search for characteristics that give differential probability or linear probability value greater than or equal to a certain threshold value, and then show that such characteristics do not exist.

6 The Security against Differential Attack

In this session, we describe an evaluation of the security against differential attack. Here we study the cases where the input differential of the F function in the R function is repeated by two rounds by the effect of the B function. Since the structure of SC2000 is constructed by the iteration of 3-round non-linear functions B-R-R, it is expected that the *effective* differential characteristics are obtained from the concatenation of 3-round characteristics.

In the B function, each S-box S_4 may sometimes give effects other than exchanging upper and lower 2-bit differentials. For example, in the differential

Table 2. Differential characteristic probabilities for each round

round	1	2	3	4	5	6	7	8	9	10	11	12	13
	B	R_5	R_5	B	R_3	R_3	B	R_5	R_5	B	R_3	R_3	B
active S-box	5	0	4	5	0	4	5	0	4	5	0	4	5
probability($\log_2$)	-15	-15	-33	-48	-48	-66	-81	-81	-99	-114	-114	-132	-147

distribution table for S_4, (0x6,0x8)- and (0x2,0x8)-entry have nonzero probabilities. (See Table 1.) Therefore, the B function in which thirty-two S-boxes S_4 are aligned may possibly have the following input/output differential:

$$(\Delta a, 0, 0, 0) \xleftarrow{B} (\Delta c, \Delta c, \Delta a, 0),$$

$\Delta d \prec \Delta a, \Delta c = mask \wedge \Delta d$, where $mask$ is the constant value used in the R function, and $\Delta d \prec \Delta a$ means $\Delta d \vee \Delta a = \Delta a$. By using this, the following differential may pass through: (see Figure 5.)

$$\begin{aligned}(\Delta a, 0, 0, 0) &\xleftarrow{B} (\Delta c, \Delta d, \Delta a, 0)\\ (0, 0) &\xleftarrow{F} (0, 0)\\ (\Delta c, \Delta d) &\xleftarrow{F} (\Delta a, 0)\\ (\Delta a, 0, 0, 0) &\xleftarrow{B} (\Delta c, \Delta d, \Delta a, 0).\end{aligned}$$

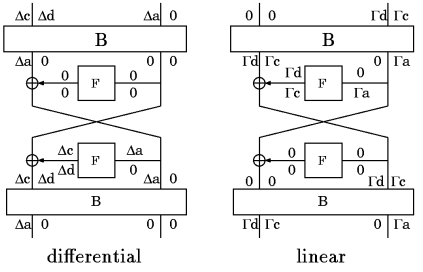


Fig. 5. 3-round cycle differential/linear characteristics

Since the (0xe,0x8)-entry of the differential distribution table is zero, Δc must be 0. Also in the example above, we set lowermost 32 bits in the input differential of the F function zero, but it is also necessary to search for cases where both upper and lower bits of the input differential of the F function are nonzero. This types of characteristics may be formed depending on the entry of the differential distribution table of S_4.

As a result of calculations of differentials in this form by computer, we obtained the following pattern which is one of the best differential characteristic whose probability is 2^{-33} for 3-round B-R-R described above. (See Table 2.)

Table 3. Linear characteristic probabilities for each round

round	1	2	3	4	5	6	7	8	9	10	11	12	13
	B	R_5	R_5	B	R_3	R_3	B	R_5	R_5	B	R_3	R_3	B
active S-box	6	4	0	6	4	0	6	4	0	6	4	0	6
probability($\log_2$)	-16	-34	-34	-50	-68	-68	-84	-102	-102	-118	-136	-136	-152

$$\begin{aligned}(0x08090088, 0, 0, 0) &\xleftarrow{B} (0, 0x00080008, 0x08090088, 0) && 2^{-15}\\ (0, 0) &\xleftarrow{F} (0, 0) && 1\\ (0, 0x00080008) &\xleftarrow{F} (0x08090088, 0) && 2^{-18}.\end{aligned}$$

The differential attack can be applicable to the 13-round SC2000 with 2^{117} data and 2^{162} computational complexity for deriving a part of the extended keys by using the 11-round (2nd to 12th round) differential characteristic with probability 2^{-117} described above and by using two-round elimination technique. Although it is still an open problem whether there exist the effective differential patterns for more rounds or not, we believe that the cipher SC2000 with each of 19-round for 128-bit key and 22-round for other keys is secure against differential attacks.

7 The Security against Linear Attacks

This section describes an evaluation of the security against linear attacks. The symbols used in the evaluation of the security against differential attacks are also used in this session. In the same reason as in the differential searches, we will focus on the iteration of the forms of 3-round linear characteristics.

From the linear distribution table for S_4, (0x2,0x8)-, (0x2,0x9)-, (0x3,0xd)-entry have nonzero probabilities. (See Table 1.) Therefore, the B function in which thirty-two S-boxes S_4 are aligned may possibly have the following input/output linear:

$$(\Gamma d, \Gamma c, 0, \Gamma a) \xleftarrow{B} (0, 0, \Gamma d, \Gamma c),$$

$\Gamma c = mask \wedge \Gamma d$, $\Gamma a \prec \Gamma d$, where $mask$ is the constant value used in the R function. By using this, the following linear characteristic patterns may pass through: (See Figure 5.)

$$\begin{aligned}(\Gamma d, \Gamma c, 0, \Gamma a) &\xleftarrow{B} (0, 0, \Gamma d, \Gamma c)\\ (\Gamma d, \Gamma c) &\xleftarrow{F} (0, \Gamma a)\\ (0, 0) &\xleftarrow{F} (0, 0)\\ (\Gamma d, \Gamma c, 0, \Gamma a) &\xleftarrow{B} (0, 0, \Gamma d, \Gamma c).\end{aligned}$$

As a result of search of linear characteristic patterns of this form by using a computer, we found that the following 3-round linear characteristic is one of the best pattern, whose linear probability is 2^{-34} for 3-round. (See Table 3.)

Table 4. Lower bound of total degrees of algebraic relationals and of numbers of terms in polynomial representations about input bits in n-round input/output bits

# of rounds	1	2	3	4	5	6	7
Total degrees	2	4	8	16	32	64	128

total degree of algebraic relational

# of rounds	1	2	3	4	5	6
# of terms	2^3	2^9	2^{27}	2^{81}	2^{243}	2^{729}

number of term in polynomial

$$(0x08188810, 0x00100010, 0, 0x08100810) \xleftarrow{B} (0, 0, 0x08188810, 0x00100010) \quad 2^{-16}$$
$$(0x08188810, 0x00100010) \xleftarrow{F} (0, 0x08100810) \quad 2^{-18}$$
$$(0, 0) \xleftarrow{F} (0, 0) \quad 1.$$

The linear attack can be applicable for the 13-round SC2000 with 2^{120} data and 2^{180} computational complexity for deriving a part of the extended keys by using the 11-round (2nd to 12th round) linear characteristic with probability 2^{-120} described above by using two-round elimination technique. Although it is still an open problem whether there exist the effective linear patterns for more rounds or not, we believe that the cipher SC2000 with each of 19-round for 128-bit key and 22-round for other keys is secure against linear attacks.

8 The Security against Higher Order Differential Attack and Interpolation Attack

The cipher uses three types of S-boxes: S_4, S_5 and S_6. For all of these S-boxes, nonlinear functions having three or higher degrees are used. In addition, it has been confirmed that algebraic equations for the input/output have two or higher degrees. Therefore, the increase in degrees by two or more per round is unavoidable for the improved higher order differential attacks. Accordingly, it is shown that the minimum degree of algebraic equations for the input/output of n rounds is 2^n or higher. As a result, it is found that the number of required plaintexts exceeds 2^{128} after at most seven rounds.

Here we study a method of finding extended keys by using linear equations, considering Boolean polynomials as vectors of a linear space over $GF(2)$. Assume that the number of terms in a polynomial is estimated by 2^s, where s is the degree of the polynomial (considering that at least two key bits affect). Since the degree of a polynomial increases three per round, the number of terms at the output of the n-th round is estimated as $s = (2^3)^n$ or greater. Therefore, the complexity exceeds 2^{243} for five rounds, and it exceeds 2^{729} for 6 rounds. From the descriptions above, it is assured that improved higher order differential attacks are not successful if the number of rounds is $5 + 5 + 7 = 17$ or greater for the 128-bit key, or $6 + 6 + 7 = 19$ or greater for the 192-bit or 256-bit keys.

The interpolation attack is effective against ciphers that use simple algebraic functions. However, interpolation attacks are often only workable against ciphers whose round functions have very low algebraic degree. The cipher SC2000 is a

combination of Feistel structure and SPN structure, and uses three kind of S-boxes. These combination of the many types of algebraic structures which are used in SC2000 should help increase the degree after a few rounds. Therefore, we believe that SC2000 is secure against interpolation attacks.

9 Study on Collisions of Intermediate-Keys

In the intermediate key generation function, the following properties concerning existence of collisions are satisfied: Let a_i $(i = 0, 1, 2)$ be intermediate keys from an 8-tuple of user keys $k = (k_0, ..., k_7)$ and a'_i $(i = 0, 1, 2)$ from $k' = (k'_0, ..., k'_7)$. Then the following holds: $k_0 \neq k'_0$ or $k_1 \neq k'_1 \Rightarrow a_0 \neq a'_0$ or $a_1 \neq a'_1$.

For intermediate keys b_i, c_i, d_i $(i = 0, 1, 2)$, the same property is satisfied. This property indicates that no collisions occur in the process of generating intermediate keys from the user keys.

Here we prove the property. Let $C_0 = M(S(0))$, $C_1 = M(S(4))$, $\tilde{k}_0 = M(S(k_0))$, $\tilde{k}'_0 = M(S(k'_0))$, $\tilde{k}_1 = M(S(k_1))$,and $\tilde{k}'_1 = M(S(k'_1))$. The i-th bit of variable x is expressed as x_i. From the conditions, we have $(C_0 + \tilde{k}_0) \oplus \tilde{k}_1 = (C_0 + \tilde{k}'_0) \oplus \tilde{k}'_1$, $(C_1 + \tilde{k}_0) \oplus (2\tilde{k}_1) = (C_1 + \tilde{k}'_0) \oplus (2\tilde{k}'_1)$.

Here we use induction on i concerning i-th bit. For the 31st bit (the least significant bit), the following is proven by the latter equation: $C_{1,31} \oplus \tilde{k}_{0,31} = C_{1,31} \oplus \tilde{k}'_{0,31}$, that is, $\tilde{k}_{0,31} = \tilde{k}'_{0,31}$, $C_{0,31} \oplus \tilde{k}_{0,31} \oplus \tilde{k}_{1,31} = C_{0,31} \oplus \tilde{k}'_{0,31} \oplus \tilde{k}'_{1,31}$, that is, $\tilde{k}_{1,31} = \tilde{k}'_{1,31}$. Assume that for any j greater than or equal to $i + 1$, the euqations $\tilde{k}_{0,j} = \tilde{k}'_{0,j}$, $\tilde{k}_{1,j} = \tilde{k}'_{1,j}$ hold. Then for i-th bit, the following equation holds:

$$\begin{aligned} &C_{1,i} \oplus \tilde{k}_{0,i} \oplus \mu(C_{1,31}, ..., C_{1,i+1}, \tilde{k}_{0,31}, ..., \tilde{k}_{0,i+1}) \\ &\quad = C_{1,i} \oplus \tilde{k}'_{0,i} \oplus \mu(C_{1,31}, ..., C_{1,i+1}, \tilde{k}_{0,31}, ..., \tilde{k}_{0,i+1}), \end{aligned}$$

where μ is a Boolean function. From the equation, $\tilde{k}_{0,i} = \tilde{k}'_{0,i}$ is proven, and in the same way $\tilde{k}_{1,i} = \tilde{k}'_{1,i}$ is proven.

10 Study on Some of Weak Keys against Slide Attack

When all values of the 12 intermediate keys a_i, b_i, c_i, d_i, that are generated from user keys are the same, all values of the extended key K_i that are generated in the extended-key generation function become the same. This means that they become weak keys against slide attacks. The following is a study on whether such weak keys exist: We consider the conditions that satisfy the following equation: $a_0 = a_1 = a_2$.

By considering the equations above as 32×2 bitwise Boolean equations, we induced algebraic equations of constraints about $C_0 = M(S(0))$, $C_1 = M(S(4))$, $C_2 = M(S(8))$, by using a computer algebra system. The relational Boolean equations below are parts of such equations. Existence of k_0 and k_1 that satisfies $a_0 = a_1 = a_2$ requires that all of the following Boolean equations must be satisfied:

Table 5. Processing speed in software

(Unit clock cycles / 1 block)

Name	Encode	Decode	ExKey
key size	128 192 256	128 192 256	128 192 256
SPARC	274 323 323	277 317 317	340 393 407
Pentium	383 438 438	403 460 460	427 487 507
Athlon	319 361 361	332 377 377	456 478 476
Alpha	238 298 298	238 298 298	288 327 327

Name	CPU (clock)	Language		Implementation
		En/Decode	ExtKey	(See Section 3.4.)
SPARC	UltraSAPRC II (400MHz)	C	C	(6,10,10,6)
Pentium	PentiumIII (550MHz)	asm.	C	(6,10,10,6)
Athlon	Athlon (900MHz)	asm.	C	(11,10,11)
Alpha	Alpha21264 (500MHz)	C	C	(11,10,11)

Table 6. Performance on Intel 8051 and in JAVA language

Performance in Intel 8051 (128bit key only)

Unit: msec / 1 block

Name	Encode	Decode	ExKey
8051	8.133	8.609	21.666

Size of code: 1597Byte

Size of static data : 1154Byte

Performance of Encryption in JAVA

Unit: clock cycles / 1 block

Name	Encode			Decode			ExKey			Implementation
key length	128	192	256	128	192	256	128	192	256	(See Section 3.4.)
JAVA	3998	4599	4604	4114	4748	4742	14204	15840	15835	(6,5,5,5,5,6)
JAVA	3359	3860	3878	3453	3966	3976	13538	15140	15129	(6,10,10,6)

Binary size: 1200Byte, ROM : 1408Byte (6,5,5,5,5,6) 32790Byte(6,10,10,6)

$$\begin{cases} C_{0,31} \oplus C_{2,31} = 0, \quad C_{0,30} \oplus C_{1,31} \oplus C_{2,30} \oplus C_{2,31} = 0, \\ (C_{1,31} \oplus C_{2,31} \oplus 1)C_{0,29} \oplus (C_{1,31} \oplus C_{2,31} \oplus 1)C_{1,30} \oplus (C_{2,29} \oplus C_{2,30})C_{1,31} \\ \qquad \oplus (C_{2,31} \oplus 1)C_{2,29} \oplus (C_{2,31} \oplus 1)C_{2,30} = 0, \\ \cdots \end{cases}$$

Now, it can be confirmed that the constant parameters (C_0, C_1, C_2) used in the intermediate key generation function, where

$$(C_0, C_1, C_2) \in \{(M(S(x)), M(S(x+4)), M(S(x+8)))\}_{x=0,1,2,3}$$

do not satisfy the system of equations above. Therefore, it can be said that no user key having intermediate keys with the same value exists.

Table 7. Evaluation of hardware implementation

	Key size	Gate size	Throughput
Large†	128-bit	226K	1.26Gbps
Large	128-bit	262K	1.25Gbps
	192-/256-bit	262K	1.08Gbps
Medium	128-bit	63K	964Mbps
	192-/256-bit	63K	844Mbps
Small	128-bit	33K	264Mbps
	192-/256-bit	33K	231Mbps

†: exclusive use for 128-bit key
Verilog-HDL, 0.25μm CMOS ASIC (See [6].)

11 Evaluation of Implementation

In this section, we evaluate the processing speed of the data randomization and key schedule of the cipher.

Table 5 shows the software processing speed for encryption (Encode), decryption (Decode), and key schedule processing (ExKey) on the four CPUs, UltraSPARC II, PentiumIII, Athlon and Alpha21264. Each of the processing speed is measured by clock cycles per one block encryption. For the fast software implementation, the techniques described in Section 3.4 are used. These measurements show the maximum performances which we obtained.

Table 6 shows the software performance of SC2000 on 8-bit CPU measuring by Intel 8051 simulator, and on implementing in JAVA language. For Intel 8051 CPU, Table 6 only shows real timing data (not clock cycles) of the processing functions, since we do not know the ratio between the number of clocks of CPU and the number of cycles of processing and we can not convert from timing data to clock cycles. We note that the performance of the key schedule in our JAVA implementation is measured on the processing speed for swapping all the extended keys without re-construction of the structure of the encryption function.

Table 7 shows our evaluations in hardware implementations. There are three kinds of implementations, Large, Medium and Small. These evaluations of hardware implementation here, Large, Medium and Small, are constructed by giving priorities to the speed of processing, the ratio of the speed to the gate size and the gate size, respectively. The throughputs are measured by bit per second. The total number of gates includes the number of gates for the data randomization and for the key schedule.

References

1. E.Biham, "A Fast New DES Implementation in Software," Fast Software Encryption, 4th International Workshop Proceedings, Springer-Verlag LNCS 1267, 1997, pp.260-272.

2. E.Biham, R.Anderson, and L.Knudsen, "Serpent: A New Block Cipher Proposal," Fast Software Encryption, 5th International Workshop Proceedings, Springer-Verlag LNCS 1372, 1998, pp.222-238.
3. B.Gladman, "Serpent S Boxes as Boolean Functions," http://www.btinternet.com/~brian.gladman/cryptography_technology/serpent/index.html, 2000.
4. J.Daemen, L.Knudsen. V.Rijmen, "The Block Cipher Square," FSE4, LNCS pp.149-165, 1997.
5. J.Daemen, V.Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, Jun 1998.
6. Fujitsu, "CE71 Series," DATA SHEET DS06-20108-1, http://edevice.fujitsu.com/fj/CATALOG/AD00/00-00001/10e-5b-2.html
7. M.Kanda, Y.Takashima, T.Matsumoto, K.Aoki, K.Ohta "A strategy for constructing fast found functions with practical security agaist differential and linear cryptanalysis," Selected Areas in Cryptography, SAC'98, LNCS 1556, pp.264-279, 1998.
8. K.Nyberg and L.R.Knudsen, "Provable Security Against Differential Cryptanalysis," Journal of Cryptology, v.8, n.1, 1995, pp.27-37.
9. D.A.Osvik "Speeding up Serpent," Proceedings of The Third AEC Conference, pp 317-329.
10. B.Schneier, J.Kelsey, D.Whiting, D.Wagner, C.Hall, N.Ferguson, "The Twofish Encryption Algorithm," John Wiley and Sons, Inc. New York, 1999.