SEED Algorithm Specification

Korea Information Security Agency

SEED Algorithm Specification

Abstract

SEED is a 128-bit symmetric key block cipher that had been developed by KISA (Korea Information Security Agency) and a group of experts since 1998. SEED is a national industrial association standard (TTAS KO-12.0004, 1999). SEED has been adopted to most of the security systems in Korea. SEED is designed to utilize the S-boxes and permutations that balance with the current computing technology. It has the Feistel structure with 16 rounds, and is strong against differential cryptanalysis and linear cryptanalysis balanced with security/efficiency trade-off.

1. Introduction

This section specifies a complete decryption of the encryption algorithm SEED, which is a secret key cipher with 128-bit data block and 128-bit secret key. The abstract features of SEED are outlined as follows:

- Feistel structure with 16 rounds
- 128-bit input-output data block size
- 128-bit key length
- Two 8×8 S-boxes
- Mixed operations of exclusive OR and modular addition

Throughout out this section, the following notations are used.

- & : bitwise AND
- + : addition in modular 2^{32}
- - : subtraction in modular 2^{32}
- $\oplus$: bitwise exclusive OR
- & : bitwise AND
- << n : left circular rotation by n bits
- >> n : right circular rotation by n bits
- || : concatenation

2. Structure of SEED

A 128-bit input is divided into two 64-bit blocks and the right 64-bit block is an input to the round function F with a 64-bit subkeys generated from the key schedule.

The structure of SEED is shown in Figure 1.

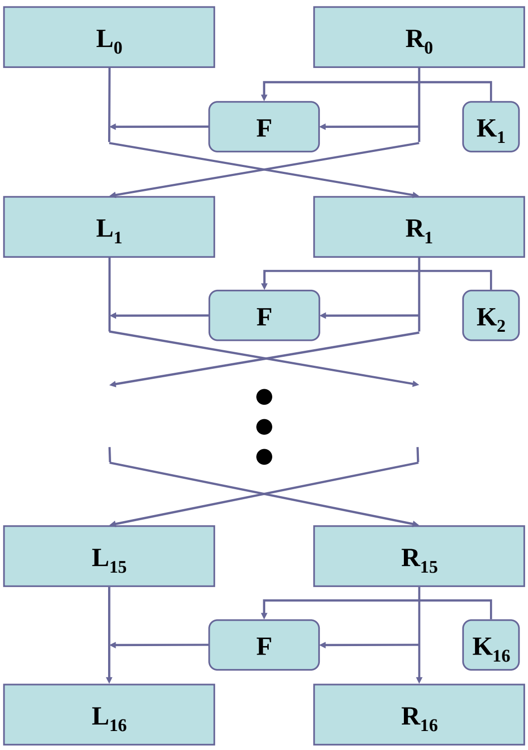

Figure 1. Structure of SEED

2.1 Round function F

A 64-bit input block of the round function is divided into two 32-bit blocks (C, D) and wrapped with 4 phases: a mixing phase of two 32-bit subkey blocks ($K_{i,0}$, $K_{i,1}$) and 3 layers of function G with additions for mixing two 32-bit blocks. The outputs C', D' of function F with two 32-bit input blocks C, D and two 32-bit subkeys $K_{i,0}$, $K_{i,1}$ are as follows:

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

The structure of round function F is shown in Figure 2.

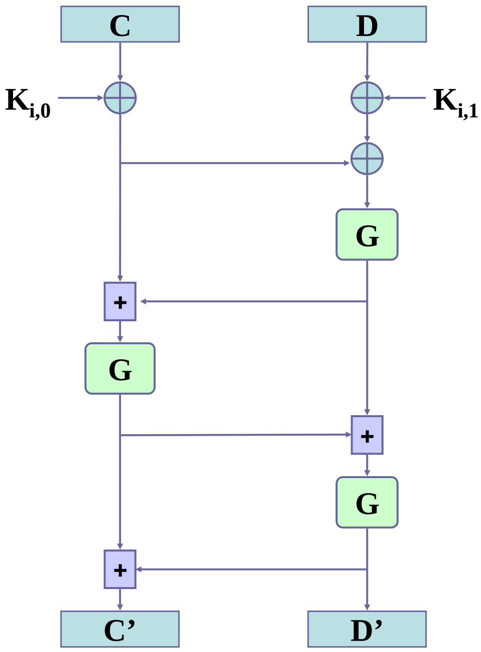

Figure 2. Round function F

2.2 Function G

The function G has two layers: a layer of two 8 × 8 S-boxes and a layer of block permutation of sixteen 8-bit sub-blocks. The first layer of two S-boxes is generated from the Boolean functions x^{247} and x^{251}. The second layer is a set of permutations in each S-box.

The outputs Z_0, Z_1, Z_2, Z_3 of the function G with four 8-bit inputs X_0, X_1, X_2, X_3 are as follows:

$$Z_0 = (S_1(X_0) \& m_0) \oplus (S_2(X_1) \& m_1) \oplus (S_1(X_2) \& m_2) \oplus (S_2(X_3) \& m_3)$$
$$Z_1 = (S_1(X_0) \& m_1) \oplus (S_2(X_1) \& m_2) \oplus (S_1(X_2) \& m_3) \oplus (S_2(X_3) \& m_0)$$
$$Z_2 = (S_1(X_0) \& m_2) \oplus (S_2(X_1) \& m_3) \oplus (S_1(X_2) \& m_0) \oplus (S_2(X_3) \& m_1)$$
$$Z_3 = (S_1(X_0) \& m_3) \oplus (S_2(X_1) \& m_0) \oplus (S_1(X_2) \& m_1) \oplus (S_2(X_3) \& m_2)$$

where, m_0 = 0xfc, m_1 = 0xf3, m_2 = 0xcf and m_3 = 0x3f.

The structure of function G is shown in Figure 3.

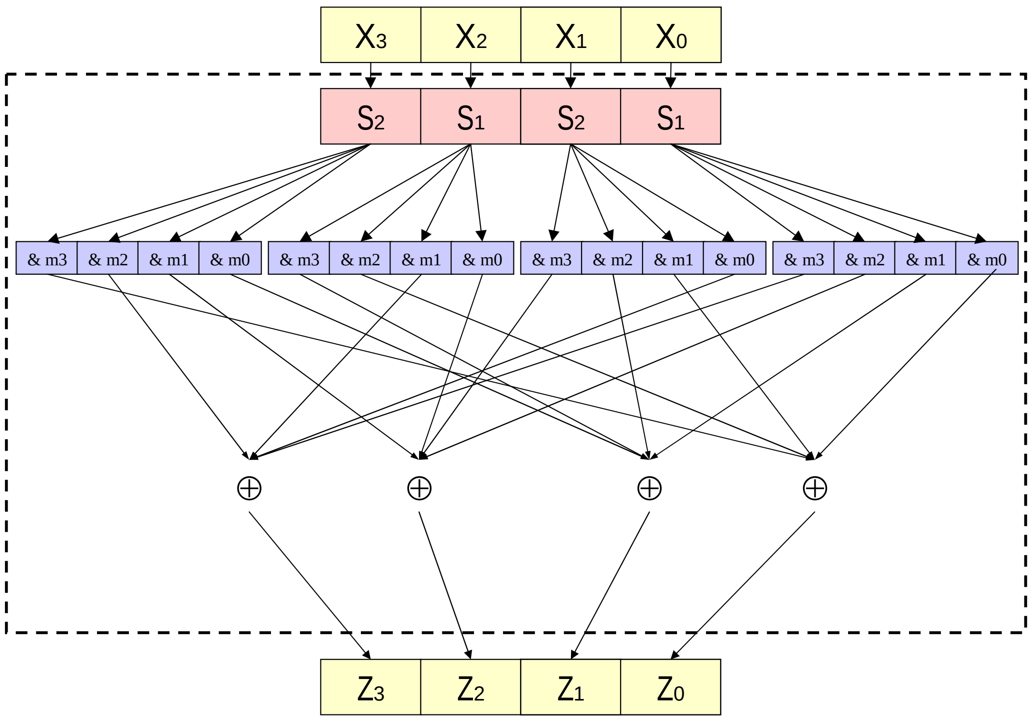

Figure 3. Function G

2.3 Design of S-box

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

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

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

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

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}.

To increase the efficiency of G function, we define extended S-boxes(SS-boxes, 4Kbyte)

$$SS_3(X) = S_2(X) \,\&\, m_2 \,||\, S_2(X) \,\&\, m_1 \,||\, S_2(X) \,\&\, m_0 \,||\, S_2(X) \,\&\, m_3$$
$$SS_2(X) = S_1(X) \,\&\, m_1 \,||\, S_1(X) \,\&\, m_0 \,||\, S_1(X) \,\&\, m_3 \,||\, S_1(X) \,\&\, m_2$$
$$SS_1(X) = S_2(X) \,\&\, m_0 \,||\, S_2(X) \,\&\, m_3 \,||\, S_2(X) \,\&\, m_2 \,||\, S_2(X) \,\&\, m_1$$
$$SS_0(X) = S_1(X) \,\&\, m_3 \,||\, S_1(X) \,\&\, m_2 \,||\, S_1(X) \,\&\, m_1 \,||\, S_1(X) \,\&\, m_0$$

and implement the function G like below equation.(Z is 32bit output of the function G such that $Z=Z_3||Z_2||Z_1||Z_0$)

$$Z = SS_3(X_3) \oplus SS_2(X_2) \oplus SS_1(X_1) \oplus SS_0(X_0)$$

The followings are the tables of two S-boxes S_1, S_2.

i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)	i	S1(i)
0	169	1	133	2	214	3	211	4	84	5	29	6	172	7	37
8	93	9	67	10	24	11	30	12	81	13	252	14	202	15	99
16	40	17	68	18	32	19	157	20	224	21	226	22	200	23	23
24	165	25	143	26	3	27	123	28	187	29	19	30	210	31	238
32	112	33	140	34	63	35	168	36	50	37	221	38	246	39	116
40	236	41	149	42	11	43	87	44	92	45	91	46	189	47	1
48	36	49	28	50	115	51	152	52	16	53	204	54	242	55	217
56	44	57	231	58	114	59	131	60	155	61	209	62	134	63	201
64	96	65	80	66	163	67	235	68	13	69	182	70	158	71	79
72	183	73	90	74	198	75	120	76	166	77	18	78	175	79	213
80	97	81	195	82	180	83	65	84	82	85	125	86	141	87	8
88	31	89	153	90	0	91	25	92	4	93	83	94	247	95	225
96	253	97	118	98	47	99	39	100	176	101	139	102	14	103	171
104	162	105	110	106	147	107	77	108	105	109	124	110	9	111	10
112	191	113	239	114	243	115	197	116	135	117	20	118	254	119	100
120	222	121	46	122	75	123	26	124	6	125	33	126	107	127	102
128	2	129	245	130	146	131	138	132	12	133	179	134	126	135	208
136	122	137	71	138	150	139	229	140	38	141	128	142	173	143	223
144	161	145	48	146	55	147	174	148	54	149	21	150	34	151	56
152	244	153	167	154	69	155	76	156	129	157	233	158	132	159	151
160	53	161	203	162	206	163	60	164	113	165	17	166	199	167	137
168	117	169	251	170	218	171	248	172	148	173	89	174	130	175	196
176	255	177	73	178	57	179	103	180	192	181	207	182	215	183	184
184	15	185	142	186	66	187	35	188	145	189	108	190	219	191	164
192	52	193	241	194	72	195	194	196	111	197	61	198	45	199	64
200	190	201	62	202	188	203	193	204	170	205	186	206	78	207	85
208	59	209	220	210	104	211	127	212	156	213	216	214	74	215	86
216	119	217	160	218	237	219	70	220	181	221	43	222	101	223	250
224	227	225	185	226	177	227	159	228	94	229	249	230	230	231	178
232	49	233	234	234	109	235	95	236	228	237	240	238	205	239	136
240	22	241	58	242	88	243	212	244	98	245	41	246	7	247	51
248	232	249	27	250	5	251	121	252	144	253	106	254	42	255	154

Table 1. ***S_1* *- box***

i	S2(i)	i	S2(i)	i	S2(i)	i	S2(i)	i	S2(i)	i	S2(i)	i	S2(i)	i	S2(i)
0	56	1	232	2	45	3	166	4	207	5	222	6	179	7	184
8	175	9	96	10	85	11	199	12	68	13	111	14	107	15	91
16	195	17	98	18	51	19	181	20	41	21	160	22	226	23	167
24	211	25	145	26	17	27	6	28	28	29	188	30	54	31	75
32	239	33	136	34	108	35	168	36	23	37	196	38	22	39	244
40	194	41	69	42	225	43	214	44	63	45	61	46	142	47	152
48	40	49	78	50	246	51	62	52	165	53	249	54	13	55	223
56	216	57	43	58	102	59	122	60	39	61	47	62	241	63	114
64	66	65	212	66	65	67	192	68	115	69	103	70	172	71	139
72	247	73	173	74	128	75	31	76	202	77	44	78	170	79	52
80	210	81	11	82	238	83	233	84	93	85	148	86	24	87	248
88	87	89	174	90	8	91	197	92	19	93	205	94	134	95	185
96	255	97	125	98	193	99	49	100	245	101	138	102	106	103	177
104	209	105	32	106	215	107	2	108	34	109	4	110	104	111	113
112	7	113	219	114	157	115	153	116	97	117	190	118	230	119	89
120	221	121	81	122	144	123	220	124	154	125	163	126	171	127	208
128	129	129	15	130	71	131	26	132	227	133	236	134	141	135	191
136	150	137	123	138	92	139	162	140	161	141	99	142	35	143	77
144	200	145	158	146	156	147	58	148	12	149	46	150	186	151	110
152	159	153	90	154	242	155	146	156	243	157	73	158	120	159	204
160	21	161	251	162	112	163	117	164	127	165	53	166	16	167	3
168	100	169	109	170	198	171	116	172	213	173	180	174	234	175	9
176	118	177	25	178	254	179	64	180	18	181	224	182	189	183	5
184	250	185	1	186	240	187	42	188	94	189	169	190	86	191	67
192	133	193	20	194	137	195	155	196	176	197	229	198	72	199	121
200	151	201	252	202	30	203	130	204	33	205	140	206	27	207	95
208	119	209	84	210	178	211	29	212	37	213	79	214	0	215	70
216	237	217	88	218	82	219	235	220	126	221	218	222	201	223	253
224	48	225	149	226	101	227	60	228	182	229	228	230	187	231	124
232	14	233	80	234	57	235	38	236	50	237	132	238	105	239	147
240	55	241	231	242	36	243	164	244	203	245	83	246	10	247	135
248	217	249	76	250	131	251	143	252	206	253	59	254	74	255	183

Table 2. ***S_2 - box***

2.4 Key schedule

The key schedule generates each round subkey. It uses the function G, addition, subtraction, and (left/right) circular rotation. 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:

$K_{1,0} = G(A + C - KC_0)$, $K_{1,1} = G(B - D + KC_0)$.

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:

$A||B \leftarrow (A||B) >> 8$.

$K_{2,0} = G(A + C - KC_1)$, $K_{2,1} = G(B + KC_1 - D)$.

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:

$C||D \leftarrow (C||D) << 8$.

$K_{3,0} = G(A + C - KC_2)$, $K_{3,1} = G(B - D + KC_2)$.

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

```
for (i=1; i<=16; i++){
    Ki,0 ← G(A + C – KCi-1);
    Ki,1 ← G(B – D + KCi-1);
    if (i%2 == 1) A||B ←(A||B) >>8
    else C||D ←(C||D) <<8
}
```

where, each constant KC_i is generated from a part of the golden ratio number $\frac{\sqrt{5}-1}{2}$.

Constants in hexadecimal form (KCi)			
i	Value	i	Value
0	0x9e3779b9	8	0x3779b99e
1	0x3c6ef373	9	0x6ef3733c
2	0x78dde6e6	10	0xdde6e678
3	0xf1bbcdcc	11	0xbbcdccf1
4	0xe3779b99	12	0x779b99e3
5	0xc6ef3733	13	0xef3733c6
6	0x8dde6e67	14	0xde6e678d
7	0x1bbcdccf	15	0xbcdccf1b

Table 3. Constant KC_i

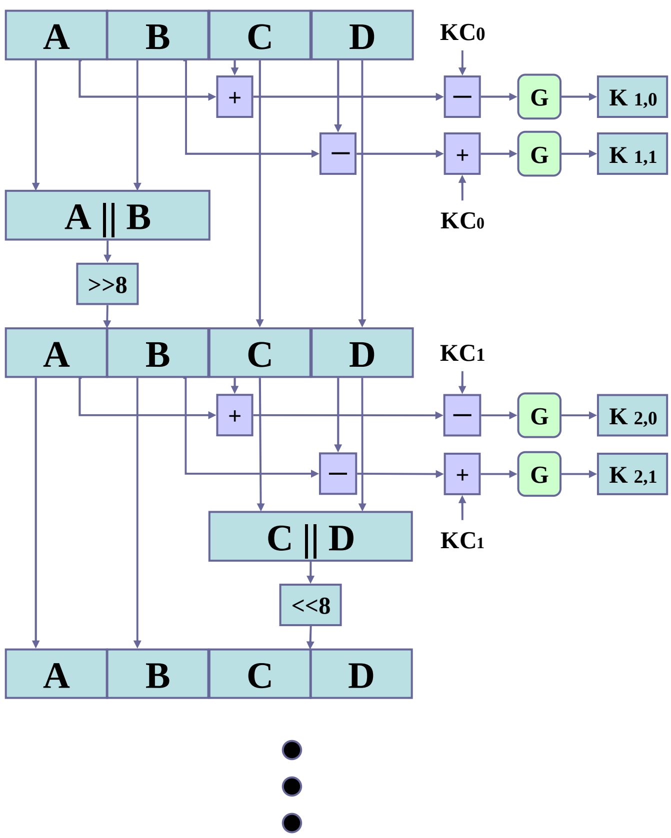

Figure 4. Key Schedule

3. Modes of Operation

SEED doesn't have any restriction for modes of operation that are used in block ciphers.

4. Assessments

SEED is robust against known attacks including DC, LC and related key attacks, etc. The results of the assessments are described in detail in Clause 1 (cryptanalysis) and Clause 2 (statistical test) of THE SELF-EVALUATION ON SEED.

5. The Adoption of The Algorithm

SEED has been widely used in Korea for confidential services such as electronic commerce and financial service, etc. SEED is an industrial association standard of Korea (TTAS.KO-12.0004, 1999).

As of Sep 2003, its source code in C has been distributed to 600 businesses including academics and research institutes by KISA through e-mail. Moreover, there are several international corporations in the number of businesses, such as BULL-Korea, RSA Security, IBM-Korea, Entrust-Korea, etc. KISA has up-loaded the related documents on its homepage and http://www.kisa.or.kr/seed/.

The usage of SEED has been covered the security service applications such as, e-Commerce, e-mail, dedicated receiver with Broadcasting, financial service, data storage, electronic toll collection, VPN, Digital Right Management, etc.

In particular, under the auspices of the Bank of Korea, eleven banks and one credit card company has launched a pilot service of K-cash for about 600 franchisees in Seoul since July of the year 2000. SEED has been used to protect the privacy of the users and the transaction data in this service.