SEED 알고리즘

상세 명세서

한국정보보호진흥원

Korea Information Security Agency

1. 개요

1999년 2월 한국정보보호센터(現 한국정보보호진흥원)에서 개발한 SEED는 민간 부분인 인터넷, 전자상거래, 무선 통신 등에서 공개 시에 민감한 영향을 미칠 수 있는 정보의 보호와 개인 프라이버시 등을 보호하기 위하여 개발된 블록암호알고리즘입니다.

대칭키 블록 암호알고리즘은 비밀성을 제공하는 암호시스템의 중요 요소이다. n비트 블록 암호알고리즘이란 고정된 n비트 평문을 같은 길이의 n비트 암호문으로 바꾸는 함수를 말한다(n비트 : 블록 크기). 이러한 변형 과정에 암·복호키가 작용하여 암호화와 복호화를 수행한다.

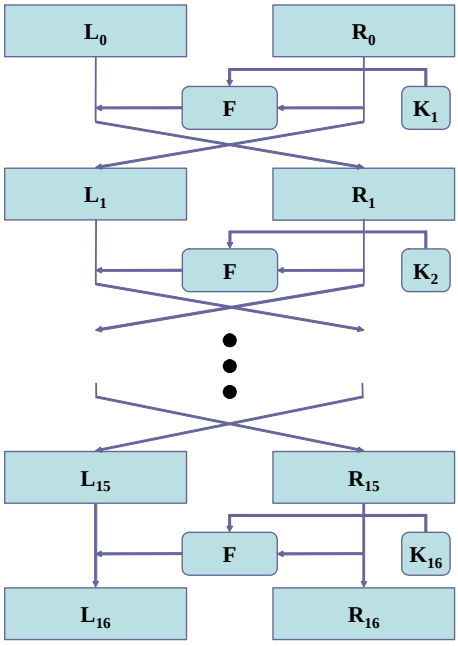
Feistel 구조란 각각 $n/2$트인 L_0, R_0 블록으로 이루어진 n비트 평문 블록 (L_0, R_0)이 r라운드($r \geq 1$)를 거쳐 암호문 (L_r, R_r)으로 변환되는 반복 구조이다.

2. 기호와 표기

- $a \oplus b$: $'a'$ bit-wise Exclusive-Or $'b'$
- $a \boxplus b$: $(a + b) \mod 2^{32}$
- a & b : $'a'$ bit-wise AND $'b'$
- $X^{\ll s}$: X를 s비트 만큼 왼쪽으로 순환 이동하는 연산
- $X^{\gg s}$: X를 s비트 만큼 오른쪽으로 순환 이동하는 연산
- L_i : i 라운드에서 출력된 왼쪽 메시지 블록(64비트)
- R_i : i 라운드에서 출력된 오른쪽 메시지 블록(64비트)
- $K_i = (K_{i,0}, K_{i,1})$: i 라운드의 라운드키(64비트)
- $K_{i,0}$: i 라운드 F 함수의 오른쪽 입력키(32비트)
- $K_{i,1}$: i 라운드 F 함수의 왼쪽 입력키(32비트)
- $X = (X_3 \| X_2 \| X_1 \| X_0)$: G함수의 입력값(32비트)
- $Y = (Y_3 \| Y_2 \| Y_1 \| Y_0)$: G함수에서 S-box(S_1, S_2)의 출력값(32비트)
- $Z = (Z_3 \| Z_2 \| Z_1 \| Z_0)$: G함수의 출력값(32비트)
- m_i : 상수
- KC_i : 라운드 키 생성과정에서 사용되는 i+1 라운드 상수

3. SEED 알고리즘의 구조

본 알고리즘의 전체 구조는 Feistel 구조로 이루어져 있으며, 128비트의 평문 블록과 128비트 키를 입력으로 사용하여 총 16라운드를 거쳐 128비트 암호문 블록을 출력한다. 다음 (그림 1)은 SEED 알고리즘의 전체구조를 도식화한 것이다.

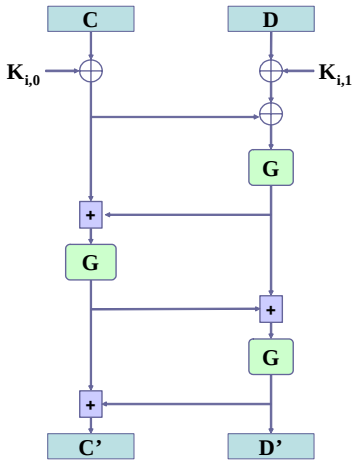


(그림 1) SEED 전체 구조도

3.1 F 함수

Feistel 구조를 갖는 블록 암호알고리즘은 F 함수의 특성에 따라 구분될 수 있다. SEED의 F 함수는 수정된 64비트 Feistel 형태로 구성된다. F 함수는 각 32비트 블록 2개(C, D)를 입력으로 받아, 32비트 블록 2개(C′, D′)를 출력한다. 즉, 암호화 과정에서 64비트 블록(C, D)와 64비트 라운드 키 $K_i=(K_{i,0};K_{i,1})$를 F 함수의 입력으로 처리하여 64비트 블록(C′, D′)을 출력한다.(i : 라운드 수)

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



(그림 2) F-함수 구조도

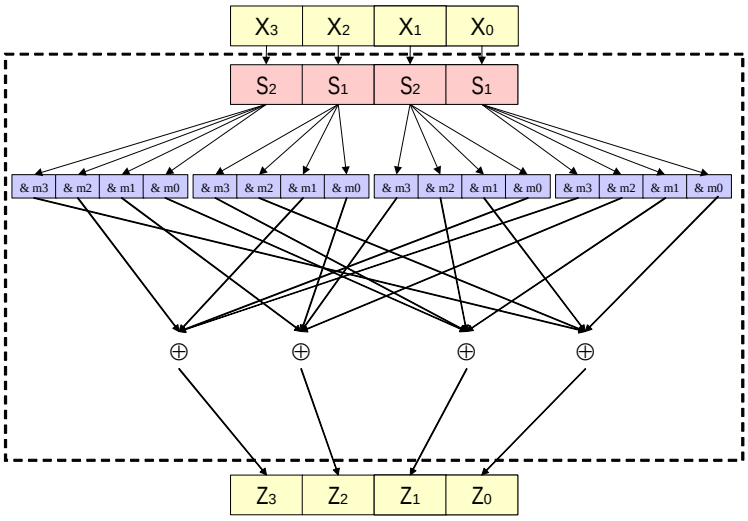
3.2. G 함수

전체 G 함수는 다음과 같이 기술된다:

$$Y_3 = S_2(X_3),\quad Y_2 = S_1(X_2),\quad Y_1 = S_2(X_1),\quad Y_0 = S_1(X_0),$$

$$\begin{aligned} Z_3 &= (Y_0 \& m_3) \oplus (Y_1 \& m_0) \oplus (Y_2 \& m_1) \oplus (Y_3 \& m_2) \\ Z_2 &= (Y_0 \& m_2) \oplus (Y_1 \& m_3) \oplus (Y_2 \& m_0) \oplus (Y_3 \& m_1) \\ Z_1 &= (Y_0 \& m_1) \oplus (Y_1 \& m_2) \oplus (Y_2 \& m_3) \oplus (Y_3 \& m_0) \\ Z_0 &= (Y_0 \& m_0) \oplus (Y_1 \& m_1) \oplus (Y_2 \& m_2) \oplus (Y_3 \& m_3) \end{aligned}$$

$$(m_0 = 0\text{xfc},\ m_1 = 0\text{xf3},\ m_2 = 0\text{xcf},\ m_3 = 0\text{x3f})$$



(그림 3) G 함수

참고> 위의 G 함수는 구현의 효율성을 위해 4개의 확장된 4바이트 SS-box들(4K bytes)의 exclusive-or로 구현할 수 있다. 이를 위해 다음과 같은 4개의 SS-box들을 저장해야 한다.

$$
\begin{aligned}
SS_3 &= S_2(X_3)\&m_2 \,\|\, S_2(X_3)\&m_1 \,\|\, S_2(X_3)\&m_0 \,\|\, S_2(X_3)\&m_3,\\
SS_2 &= S_1(X_2)\&m_1 \,\|\, S_1(X_2)\&m_0 \,\|\, S_1(X_2)\&m_3 \,\|\, S_1(X_2)\&m_2,\\
SS_1 &= S_2(X_1)\&m_0 \,\|\, S_2(X_1)\&m_3 \,\|\, S_2(X_1)\&m_2 \,\|\, S_2(X_1)\&m_1,\\
SS_0 &= S_1(X_0)\&m_3 \,\|\, S_1(X_0)\&m_2 \,\|\, S_1(X_0)\&m_1 \,\|\, S_1(X_0)\&m_0,
\end{aligned}
$$

(여기서, || 는 concatenation).

이 확장 SS-box들을 이용하면 G 함수는 다음과 같이 구현될 수 있다:

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

3.3. S-Box

G 함수의 내부에 사용되는 비선형 S-box S_1, S_2는 다음의 식을 이용하여 생성된다. (n_1=247, n_2=251, b_1=159, b_2=56)

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

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

i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$	i	$S_1(i)$
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

(표 1) S_1-Box

i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$	i	$S_2(i)$
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

(표 2) S_2-Box

3.4. 라운드 키 생성과정

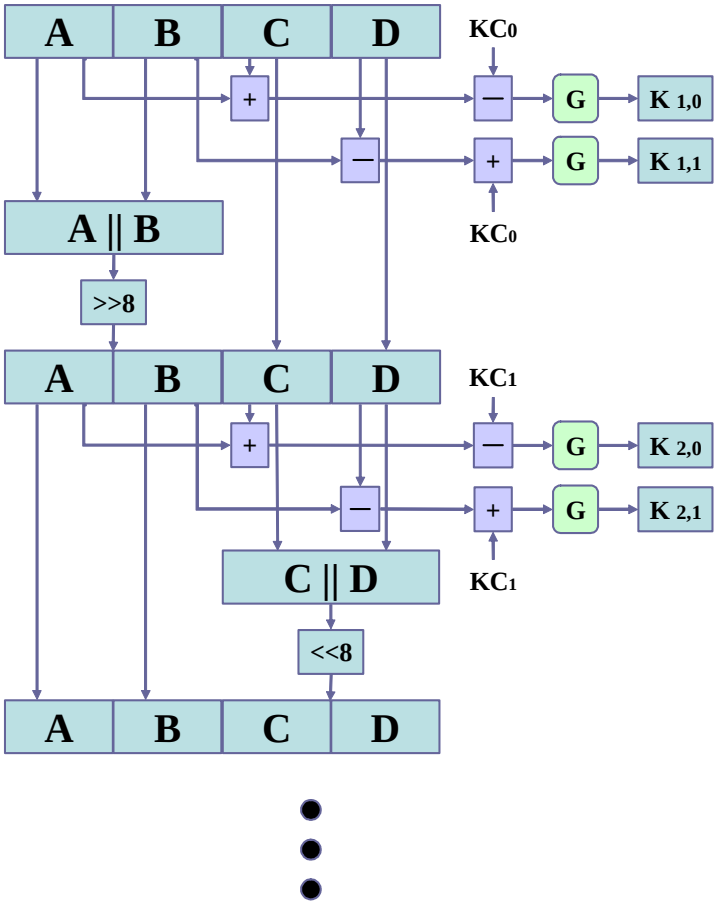
SEED의 라운드 키 생성과정은 128비트 암호키를 64비트씩 좌우로 나누어 이들을 교대로 8비트씩 좌/우로 회전이동한 후, 결과의 4워드들에 대한 간단한 산술연산과 G 함수를 적용하여 라운드 키를 생성한다. 라운드 키 생성과정은 기본적으로 하드웨어니(모든 라운드 키를 저장할 수 없는) 제한된 자원을 갖는 스마트카드와 같은 응용에서의 효율성을 위하여, 암호화나 복호화시 암호키로부터 필요한 라운드 키를 간단히 계산할 수 있도록 설계하였다.

주어진 128비트 암호키 K=A||B||C||D를 32비트 레지스터 A, B, C, D로 나눈다. 각 라운드 i에 사용되는 라운드 키 $K_i=(K_{i,0};K_{i,1})$는 다음과 같은 방식으로 생성한다:

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

라 운 드 상 수	
KC_0 = 0x9e3779b9	KC_8 = 0x3779b99e
KC_1 = 0x3c6ef373	KC_9 = 0x6ef3733c
KC_2 = 0x78dde6e6	KC_{10} = 0xdde6e678
KC_3 = 0xf1bbcdcc	KC_{11} = 0xbbcdccf1
KC_4 = 0xe3779b99	KC_{12} = 0x779b99e3
KC_5 = 0xc6ef3733	KC_{13} = 0xef3733c6
KC_6 = 0x8dde6e67	KC_{14} = 0xde6e678d
KC_7 = 0x1bbcdccf	KC_{15} = 0xbcdccf1b

(표 3) 라운드키 생성과정에 사용된 상수



(그림 4) 라운드키 생성과정 구조도