암호화 기법 조사

2021.05.21

작성자 조은희
cho108@krri.re.kr

목 차

1. 암호화 기법 조사

암호화 기법에는 크게 두 가지 분류로 나눌 수 있는데 대칭키 암호와 비대칭키 암호로 나눌 수 있다.

대칭키 암호는 암호화 알고리즘의 한 종류로, 암호화와 복호화에 같은 암호키를 쓰는 알고리즘을 의미한다. 대칭 키 암호에서는 암호화를 하는 측과 복호화를 하는 측이 같은 암호 키를 공유해야 한다. 이러한 점은 비대칭키 암호에서 공개 키와 비밀 키를 별도로 가지는 것과 구별된다. 대신, 대부분의 대칭 키 암호는 비대칭 키 암호와 비교하여 계산 속도가 빠르다는 장점을 가진다. 따라서, 많은 암호화 통신에서는 비밀 키 암호를 사용하여 대칭 키 암호의 공통 키를 공유하고, 그 키를 기반으로 실제 통신을 암호화하는 구조를 사용한다.

대칭키 암호 알고리즘으로 대표적으로는 DES, T-DES, AES, SEED, ARIA 등이 있다. 비대칭키 암호화 방식에 비해 속도가 빠르다는 장점이 있지만, 키를 교환해야 한다는 문제가 발생한다. 또한, 사람이 증가할수록 관리해야 할 키가 방대하게 많아진다는 단점이 있다.

비대칭키 암호화는 공개 키 암호방식으로도 불리운다. 사전에 비밀 키를 나눠가지지 않은 사용자들이 안전하게 통신할 수 있어야 할 때 사용된다. 공개 키와 비밀 키가 존재하며, 공개 키는 누구나 알 수 있으며 그에 대응하는 비밀 키는 키의 소유자만이 알 수 있어야 한다. 공개 키는 보안 타협 없이 공개적으로 배포가 가능하다. 일반적으로, 비대칭키 암호방식은 대칭키 암호방식보다 계산이 복잡한 단점이 있다.

대칭키와 비대칭키 암호화는 모두 키의 크기가 커질수록 해킹위험이 현저히 줄어들기 때문에 키의 크기가 핵심이다. 이는 암호화 보안강도로 판별할 수 있는데, 알고리즘의 취약성을 찾아내는 데 소요되는 작업량을 뜻한다.

현재 우리나라에서는 빠른 결제 속도를 위해 교통카드 거래 방식에 대칭키 방식으로 T-DES와 SEED암호를 사용하고 있다. 두가지 방식은 다음과 같다.

T-DES(2 key)

Triple DES 는 각 데이터 블록(64비트) 에 DES를 세 번 적용한 암호화 알고리즘이다. DES 의 키 사이즈는 64비트로 브루트포스 공격에 취약하다. 따라서 DES 알고리즘의 비도를 높이기 위해 키 사이즈를 두 배로 증가시키고(16바이트), 라운드의 단계를 3배로 증가시켜(48라운드) 그러한 공격으로부터 안전하도록 고안한 암호화 알고리즘이다.

각각의 키는 8바이트로 이를 $(k1,k2)$ 라고 가정하면, 암호화, 복호화 방법은 다음과 같다.

$$ciphertext = E_{k1}(D_{k2}(E_{k1}(plaintext)))$$
$$plaintext = D_{k1}(E_{k2}(D_{k1}(ciphertext)))$$

총 128비트의 키를 사용하는데 이 중 parity bit를 제외하면 키 사이즈는 112 비트로 이루어졌다. 가능한 키의 갯수가 2^{112}이며, 실제 암호화 보안강도는 80비트로 키를 추론하는데 2^{80}번의 시도가 필요하다. 따라서, 알려진 2^{32}개의 평문-암호문 쌍과 가능한 2^{80}개의 키, 2^{89} 번의 DES 암호화 연산, 48개의 라운드 등으로 작업량이 실행 불가능한 수준에 이르러 T-DES 에 대한 공격은 비실용적이다.

SEED

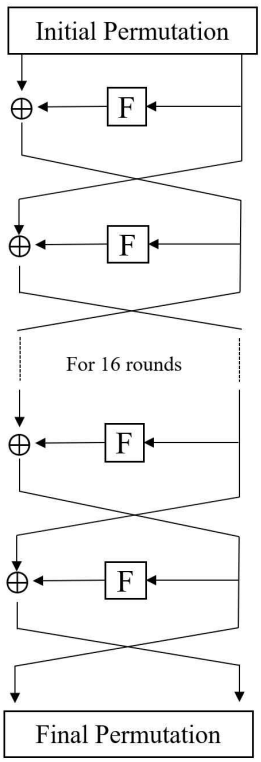
SEED는 국내에서 개발한 128비트 블록 암호 알고리즘이다. 현재 배포하고 있는 SEED 소스코드는 32비트 프로세서에 맞도록 C와 Java로 구현되어있다. 키와 데이터 블록이 모두 128비트(16바이트)로 이루어져 있어 안전성이 높다. 연산은 XOR 연산과 mod 연산만 사용되었다. 128비트 데이터를 64비트 블록(C,D)로 나누어 64비트 라운드 키 $K_i = (K_{i,0};K_{i,1})$를 F 함수에 대한 입력으로 처리하여 암호화(C',D')한다. 암호화 복호화 방법은 다음과 같다.

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

T-DES의 키 길이(112비트) 보다 키 사이즈가 더 크기 때문에(128비트) 암호화 보안강도가 128비트로 향상되었다. 이는 다른 AES-128비트 알고리즘, ARIA-128비트 알고리즘과 같은 보안강도이다.

상세한 암호화 알고리즘 내용은 1-1. DES ,1-2. SEED 와 같다.

1-1. DES



[그림 1 - Feistel 구조]

Triple DES 는 각 데이터 블록에 DES를 세 번 적용한 블록암호에 대한 일반적인 이름이다.
DES 는 64비트 블록 암호의 일종으로 대칭키 암호이며, 56 비트의 키를 사용한다. 암호화 알고리즘의 가장 대표적인 Feistel 구조를 이루고 있다. Feistel 구조란 데이터를 두 부분으로 나누어 좌, 우 데이터에 비선형 변환을 적용시키는 구조를 말한다.
DES는 크게 3가지 과정으로 나눌 수 있다.
1. Initial permutation & Final Permutation
2. 16 Round Function
3. Round-key Generator

1. Initial Permutation & Final Permutation

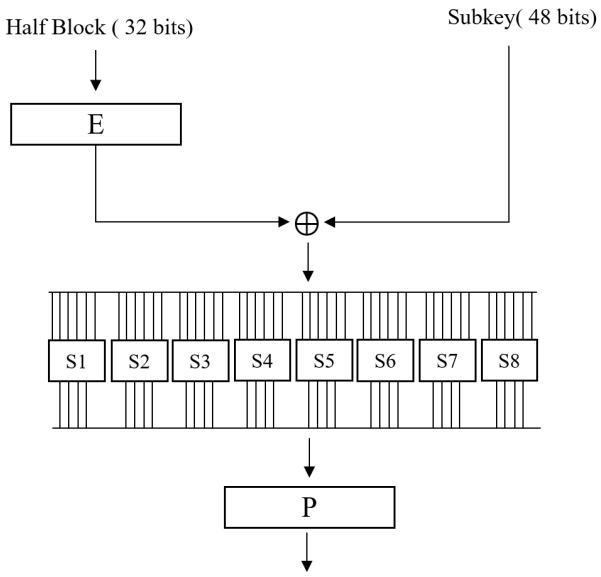
데이터의 전처리, 후처리과정이다. 이 단계에서는 암호화가 일어나지 않는다. Permutation table 에 따라서 각 bit 의 교환이 이루어진다. 테이블의 크기는 8x8 사이즈이다.
n번째 칸에 m이라는 숫자가 있으면, m번째 비트를 n번째 비트자리에 넣어 비트를 교환하는 방식이다. 즉, 1번째 칸의 숫자가 58이라면, 58번째 비트를 1번째 비트 자리에 넣는 방식으로 교환하는 방식이다. 이러한 방식으로 총 64비트 모두 교환된다.

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

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

[표 1 - Initial permutation table / Final permutation table]

2. Round Function (Feistel 함수)



[그림 2 - Round Function 구성도]

DES 암호화는 실제로 이 과정에서 일어난다. 암호 알고리즘마다 이 Round Function 암호화 함수는 각기 다를 수 있다.

먼저 Subkey(48bits) 와 XOR연산을 하기 위해서 암호화해야하는 64비트중 오른쪽의 32비트를 expansion permutation table을 이용하여 48 비트로 확장한다. [표 2]는 Expansion Permutation Table이다.

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

[표 2 - Expansion Permutation table]

Subkey와 XOR 연산 이후 출력된 48 비트는 S-box (Substitution box)를 이용하여 32비트로 축소된다. S-box는 8개 모두 각기 다른 테이블을 사용한다. [표 3]은 S1의 S-box 테이블을 나타낸 것이다.

S-box의 경우 6비트 중 양 끝에 있는 2비트를 이용하여 행의 값을 도출하고, 중앙에 있는 4비트를

이용하여 열의 값을 도출하여 표에서 행과 열에 해당하는 값으로 대체된다. 결과적으로 48비트가 32비트로 출력된다.

S1	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x	x1000x	x1001x	x1010x	x1011x	x1100x	x1101x	x1110x	x1111x
0yyyy0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0yyyy1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1yyyy0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1yyyy1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

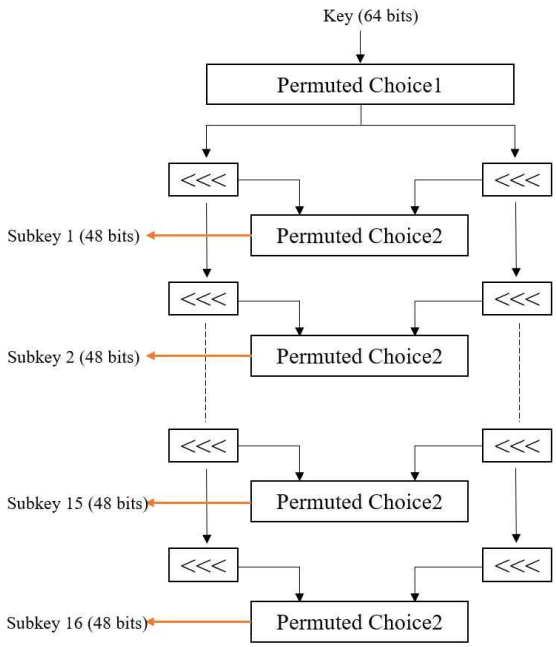
[표 3 - Substitution box table 예: S1-box]

출력된 32비트는 좌우를 교체하는 permutation 연산(표 4)을 거쳐 본 Feistel 함수 과정이 완료된다.

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

[표 4 - Permutation table]

3. Round-key Generator



[그림 3 - Round key 생성]

DES에서 사용되는 대칭키는 총 64비트로 실제 키 56비트와 parity bit 8비트로 이루어져 있다. 64비트의 키를 Permuted Choice 1단계를 거쳐 56비트로 축소한다.
이 후, 28비트로 나누어 각각 left circular shift 연산을 실행 후, 다시 결합하여 Permuted Choice 2단계를 거쳐 48비트의 subkey 하나를 만든다.
이를 반복하여 총 16개의 subkey를 만드는 것이 DES 암호화의 key generator 방식이다.

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

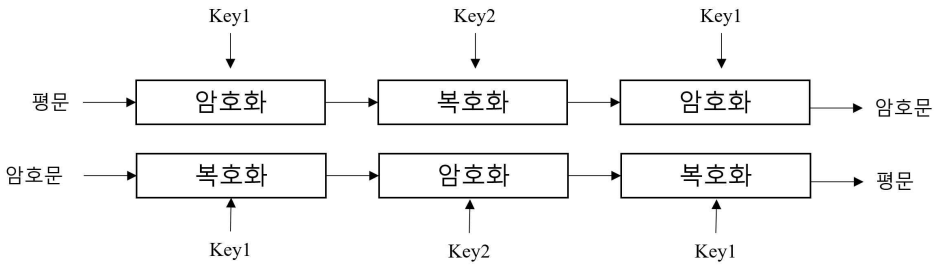
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

[표 5 - Permuted Choice 1(PC-1) / Permuted Choice 2(PC-2)]

T-DES 는 이러한 DES 암호화 과정을 세 번 반복하는 것으로 DES의 보안성 문제로 인하여 사용하게 되었다. 키의 개수가 2개이기 때문에 키가 112 bit로 기존의 DES 보다 안전하다.

암호화시 첫 번째 키를 사용하여 암호화, 두 번째 키를 사용하여 복호화 그리고 다시 첫 번째 키를 사용하여 암호화한다.

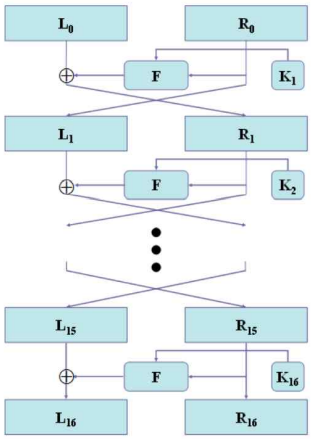
복호화시에는 첫 번째 키를 사용하여 복호화 두 번째 키를 사용하여 암호화 그리고 첫 번째 키를 사용하여 복호화한다.



[그림 4 - T-DES 암호화 / 복호화]

1-2. SEED

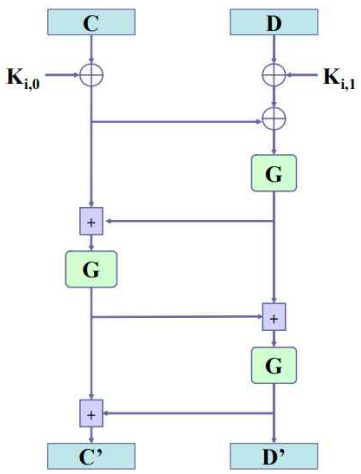
DES 와 마찬가지로 SEED 또한 Feistel 구조를 이루고 있다. DES는 64비트 블록 암호임에 반해 SEED 는 128 비트의 블록 암호로, 128비트 키를 입력으로 사용하여 총 16라운드를 거쳐 128 비트 암호문 블록을 출력한다. [그림 5]는 SEED 알고리즘의 전체구조를 도식화한 것이다.



[그림 5 - SEED 전체 구조도]

- $a \oplus b$: 'a' bit-wise Exclusive-Or 'b'
- $a \boxplus b$: $(a + b) \bmod 2^{32}$
- a & b : 'a' bit-wise AND 'b'
- $X = (X_3 \| X_2 \| X_1 \| X_0)$: 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)

1. F 함수



[그림 6 - F 함수 전체 구조도]

SEED 의 F 함수는 각각 32비트 블록 두 개, 총 64비트를 입력으로 받아, 32비트 블록 2개를 출력한다. 암호화 과정에서 128 비트의 키를 각각 64비트로 나누어 블록 두 개에 대한 입력으로 사용한다.
각각의 블록은 각각의 라운드 키와 XOR 연산을 거쳐, G 함수에 입력값으로 들어간다.

2. G 함수

G 함수의 입력값으로 32비트가 입력되어 이를 4블록으로 나누어 각각 8비트씩 계산된다. G 함수에 대한 입력값을 (X3,X2,X1,X0) 이라고 가정한다.
각각의 8비트들은 S-box(Substitute box) 에 의해, 비트값이 교환된다. S-box 의 종류는 2가지로 X3, X1은 S-box2를 사용하며, X2, X0은 S-box1을 사용한다. 이를 간단히 나타내면 다음과 같으며, [그림 9]는 SEED에서 사용하는 S-box 이다.

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

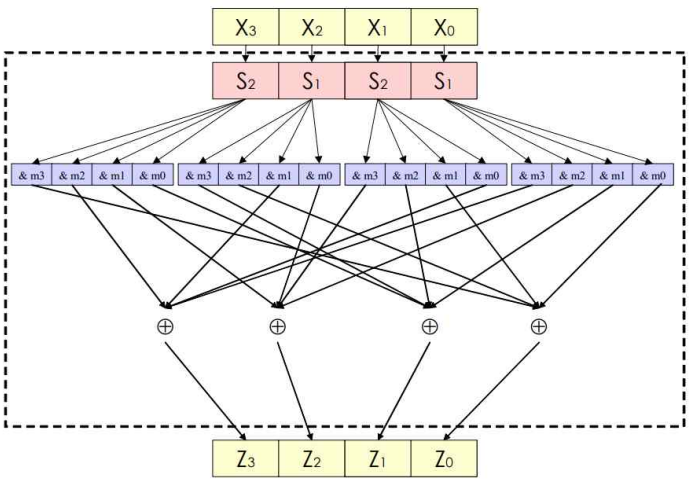
[그림 7 - S-box 사용]

이 후, 정해진 각각의 Y값들이 m값들과 AND 연산을 하고, XOR 연산을 통해 최종 Z 값을 추출한다.
m값은 다음으로 정해져 있다.

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

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

[그림 8 - G 함수]



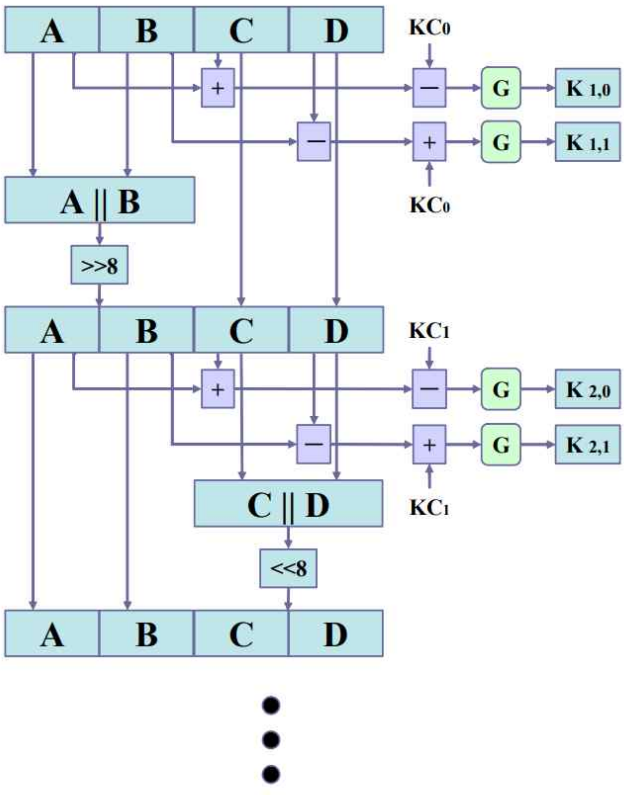
[그림 8 - G 함수 전체 구조도]

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

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

[그림 9 - S-box1 / S-box2]

3. Round-key Generator



[그림 10 - 라운드키 생성과정 구조도]

라운드 키 생성과정은 기본적으로 하드웨어나 모든 라운드 키를 저장할 수 없는 제한된 자원을 갖

는 응용프로그램에서의 효율성을 위하여, 암호화나 복호화시 암호키로부터 필요한 라운드 키를 간단히 계산할 수 있도록 설계된 과정이다.
SEED의 라운드 키는 128비트 암호키를 64비트씩 나누어 생성한다. [그림 10]에서 보이는 A, B, C, D는 각각 16비트로 이루어져 있다. 교대로 8비트씩 회전이동하여, G 함수를 적용하여 생성한다.
간단히 나타낸 수식은 다음과 같다.

for(i=1; i<=16; i++) {

$K_{i,0} \leftarrow G(A+C-KC_{i-1})$;

$K_{i,1} \leftarrow G(B-D+KC_{i-1})$;

if(i%2==1) $A||B \leftarrow (A||B)^{>>8}$;

else $C||D \leftarrow (C||D)^{<<8}$;

}

라운드키 생성과정에 사용되는 상수는 [그림 11]과 같다.

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

[그림 11 – 라운드 생성과정에 사용된 상수]

출처

- SEED 128 알고리즘 상세 명세서 (한국인터넷진흥원 Korea Internet & Security Agency)
- Chris J.Mitchell, “On the Security of 2-key triple DES” (2016)
- 한국정보보호센타, 128비트 블록 암호 알고리즘(SEED) 개발 및 분석 보고서(1998)