通信情報保護學會誌
第3卷 第1號, 1993. 3

Differential Cryptanalysis에 관한 고찰

조주연* · 이필중**

1. 서 론

일반적으로 iterated cryptosystem은 비도(secrecy)가 약한 암호함수를 적절히 반복 사용하여 비도를 높인 암호시스템을 지칭한다. 반복되는 매회는 라운드라고 불리며 각 라운드마다 사용되는 암호함수는 단순히 비트 연산과 이들의 재배열(permutation), 치환(substitution)을 중복하여 사용함으로써 공개키 암호시스템에 비하여 훨씬 빠른 속도를 낼 수 있는 장점이 있는 반면에 이런 종류의 암호시스템은 수학적으로 풀기 어려운 문제에 기초하지 않으므로 그 안전도에 있어서는 논란의 가능성을 내재한다고 하겠다. 그러므로 이런 종류의 암호시스템은 기존의 암호공격방식이 안전성을 측정하는 기준이 될 수 있다.

지금까지 이런 종류의 암호시스템을 공격하는 방법들이 계속 연구되어 왔으나 컴퓨터의 파워와 메모리 용량의 면에서 실제적인 공격법이 제시되지 못했었다. 그러나 최근에 발표되고 있는 Differential Cryptanalysis(이하 DC로 축약) 방법은 특정한 형식의 평문쌍의 모듈러 2 연산값이 암호문쌍의 모듈러 2 연산값에 미치는 영향을 분석하여 사용된 암호키를 찾아내는 방식으로서 그동안 안전하다고 믿어지던 iterated cryptosystem들의 안전성을 위협하는 강력한 암호분석방법이다[1]. 대표적인 예로서 1977년 미국 NBS에서 표준으로 발표된 이후로 안전성을 인정받고 있던 DES(Data Encryption Standard)[7]는 6 라운드에는 0.3초내에, 8 라운드에는 2분내에 암호키를 찾아낼 수 있으며[2] 표준 라운드인 16 라운드에서도 exhaustive search 보다 훨씬 빠르게 공격할 수 있다는 연구결과가 최근 발표되었다[5]. 그 이외에도 일본에서 상용되는 FEAL(Fast data Encipherment ALgorithm)[8]과 그 변형방식들은 31 라운드까지는 안전하기 못하며[4] DES의 전신이라고 할 수 있는 IBM의 Lucifer[9], 호주에서 연구된 LOKI[10] 등도 그 안전도가 크게 위협받고 있는 실정이다[5].

[3]에서는 이외에도 호주에서 사용되는 LOKI를 11라운드까지 exhaustive search 보다 빠르게 공격할 수 있음을 보였고 nefru[19], Khafre[20], REDOC-II[21]등을 공격하는 방법을 제시하고 있다.

국내에도 이 문제에 관한 논의가 서서히 진행되고 있으며 최근 DC의 확률을 줄일 수 있는 S-box 설계에 관한 연구가 발표된 바 있다[12],[13].

본 논문에서는 DC의 주요 개념과 공격방법을 분석하고 이를 IBM의 Lucifer[9]와 이의 변형방식[11], 그리고 미국 NBS의 DES[7]에 대하여 적용, 실제

* 포항공과대학 전자전기공학과 석사
** 포항공과대학 전자전기공학과, 부교수

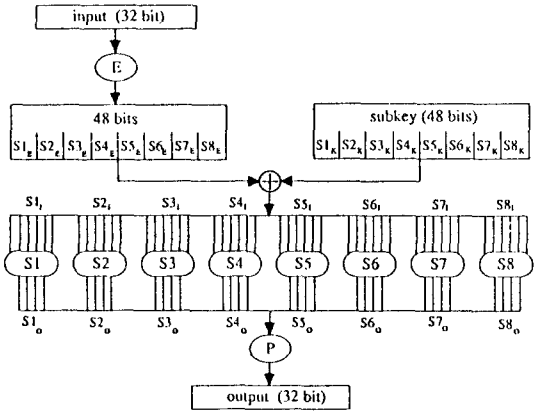

그림 1. F function of DES

프로그램으로 구현하여 암호키를 찾아내는 속도를 측정해 보며 앞으로의 연구방향을 제시해 보고자 한다.

2. DES의 Differential Cryptanalysis

2.1. Main idea

DES 알고리즘은 IP(initial permutation), FP (final permutation), 그리고 매 라운드 반복되는 일정한 형식의 암호함수 F로 나눌 수 있다. 그리고 이 암호함수 F는 그림 1과 같이 E(expansion)-S (subsitution)-P(permutation)를 차례로 거치며 하위키(subkey)는 E-box를 거친 48 비트와 XOR (eXclusive OR)된다. 그러나 한 쌍의 평문을 모듈러 2 연산한 값(입력 XOR)에 대해서는 단지 함수 F 내의 8개의 S-box들만이 비선형성(nonlinearity)을 가지게 된다. 즉 한 쌍의 평문을 각각 X, X* 그리고 이를 암호화한 한 쌍의 암호문을 Y, Y*라고 하면 F 함수 내에서 E-box, 하위키 K, P-box에서는 다음과 같은 식을 만족한다.

$$E(X)\oplus E(X^*)=E(X\oplus X^*),$$
$$(X\oplus K)\oplus(X^*\oplus K)=X\oplus X^*,$$
$$P(X)\oplus P(X^*)=P(X\oplus X^*),$$
$$(X\oplus Y)\oplus(X^*\oplus Y^*)=(X\oplus X^*)\oplus(Y\oplus Y^*)$$

그러므로 입력 XOR는 함수 F 내에서 S-box를 제외한 나머지 수준에서는 선형성을 갖는다. 그러므로 사용되는 암호키와 무관하게 입력 XOR에 대한 출력 XOR와의 관계를 구할 수 있으며 이의 일부를 표 1에 나타내었다. 이 표는 8개의 S-box(S1~S8) 중의 하나인 S1 box에 있어서 한 쌍의 6비트 입력에 대한 한 쌍의 4비트 출력을 구하고 입출력 모두 각각 XOR하여 그 갯수를 카운트한 것으로 여기에서 보듯이 입력 XOR에 대한 출력 XOR의 관계는 비균일함(nonuniform)을 알 수 있을 뿐 아니라 그 정도가 크게는 25%(예 : $34_x \rightarrow 2_x$)를 차지함을 볼 수 있다(34_x에서 x는 16진수임을 나타낸다). 이와같은 성질을 이용하여 한 쌍의 S-box의 입력값을 알고 그의 출력 XOR를 알 때 암호키를 찾아내는 방법을 예를 들어 설명해 보자.

만약 입력쌍이 각각 $S1_E=1_x$, $S1_E^*=35_x$라고 하고 출력 XOR를 $S1_O'=D_x$라고 한다면 입력 XOR는 $S1_E'=S1_I'=34_x$이다(이 때 $S1_E$는 E-box의 출력, $S1_I$는 S-box의 입력, 그리고 $S1_O$는 S-box의 출력을 뜻함. 그림 1 참조). 표 1에 따르면 $34_x \rightarrow D_x$는 모두 8개의 입력값이 가능성을 가지며 이를 토대로 가능성있는 암호키들을 구하면 표 2와 같다.

표 2. Possible keys for $34_x \rightarrow D_x$ by S1 with input 1_x, 35_x(in hex)

S box input	Possible keys
06, 32	07, 33
10, 24	11, 25
16, 22	17, 23
1C, 28	1D, 29

그리고 또다른 한 쌍의 평문에 대하여 같은 과정을 거친다면 가능한 암호키에 대한 정보를 또한번 얻을 수 있다. 만약 입력쌍이 각각 $S1_E=21_x$, $S1_E^*=15_x$라고 하고 출력 XOR를 $S1_O'=3_x$라고 한다면 입력 XOR는 역시 $S1_E'=S1_I'=34_x$이며 이에 대한 가능한 암호키의 부호들을 표 3과 같이 구할 수 있다. 이 두 표에서 공통적으로 나타나는 17_x과 23_x으로 암호키의 가능성은 더욱 압축된다. 그리고 다른 형태의 여러개의 입출력 XOR($S1_E'=34_x$)에 대해서도 같은 과정을 거친 후 가장 빈번히 중복되어 나나타는 암

표 1. Partial pairs XOR distribution table of S1

Inpul	Output XOR															
XOR	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
O_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2_x	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3_x	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4_x	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5_x	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6_x	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7_x	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8_x	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9_x	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
A_x	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
B_x	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
C_x	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
D_x	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
E_x	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
32_x	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
33_x	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
34_x	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
35_x	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
36_x	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
37_x	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
38_x	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
39_x	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
$3A_x$	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
$3B_x$	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
$3C_x$	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
$3D_x$	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
$3E_x$	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
$3F_x$	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

표 3. Possible keys for $34_x \rightarrow 3_x$ by S1 with input 21_x, 15_x(in hex)

S box input	Possible keys
01, 35	03, 37
02, 36	00, 34
15, 21	17, 23

호키가 실제 암호키일 가능성이 가장 크다.

2.2. N-라운드 특성

앞 절에서 기술한 DC 공격법을 효율적으로 적용할 수 있는 일정한 입력 XOR와 출력 XOR의 형태를 N-라운드 특성(N-round characteristic)이라고 하며 이 때 각 라운드마다 표 1에 따른 특성확률을 가진다. 이 확률이 높을수록 암호키의 정보를 가지고 있을 평문쌍의 갯수가 늘어난다. 가장 기본적이면서도 아주 유용한 예로서 그림 2와 같은 확률 1을 갖는 1-라운드 특성을 생각할 수 있다. 표 1에서 보듯이 입력 R가 0_x이면 항상 출력 XOR도 0_x이므로 확률은 1이 된다.

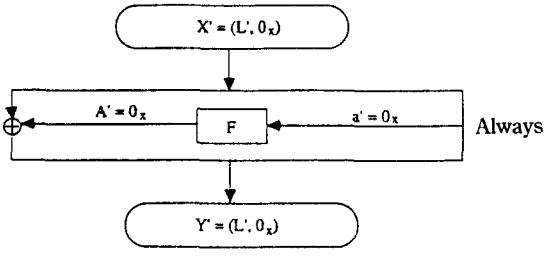

그림 2. 1-round characteristic with probability 1

4 라운드 DES에 대해서는 위의 특성을 이용하여 입력 XOR $X'=20\ 00\ 00\ 00\ 00\ 00\ 00\ 00_x$를 선택함으로써 42비트의 하위키(subkey)를 쉽게 찾을 수 있으며 나머지 6비트를 위해 $X'=02\ 22\ 22\ 22\ 00\ 00\ 00\ 00_x$를 선택하면 이를 쉽게 발견할 수 있다. 이의 특성은 확률 1이므로 모든 입력쌍들이 암호키에 대한 정보를 갖고 있다고 할 수 있다(암호키를 찾을 수 있는 평문쌍을 right pair라고 한다).

6 라운드 DES에 대해서는 가장 효율적인(확률이

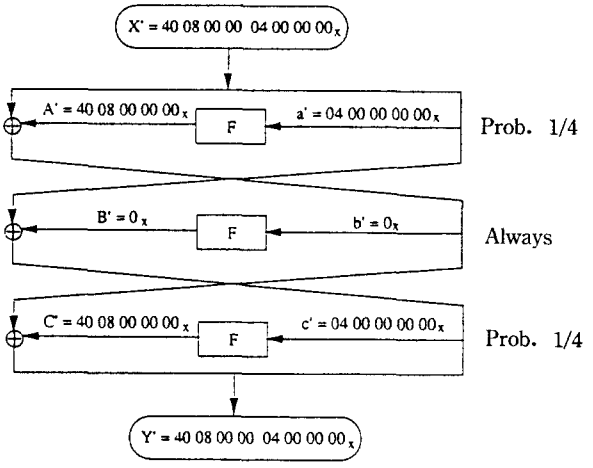

그림 3. 3-round characteristic with probability 1/16

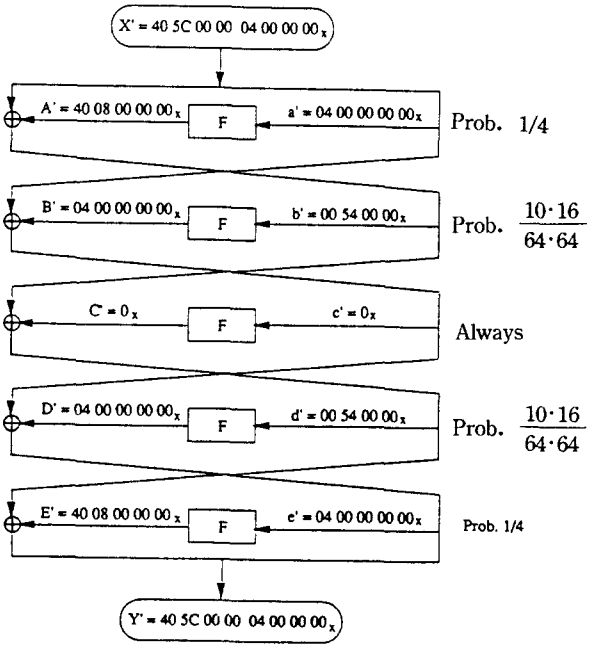

그림 4. 5 round characteristic with prob. 1/10,486

높은) 특성으로 그림 3과 같은 확률$(16/64)^2 \approx 0.06$을 갖는 3 라운드 특성이 제안되었다. 그리고 또다른 입력 XOR로서 $X'=00\ 20\ 00\ 80\ 00\ 00\ 04\ 00_x$를 선택한다면 이 두 특성을 이용하여 약 120 쌍의 데이타를 통하여 암호키를 모두 결정할 수 있다.

8 라운드 DES를 공격하기 위한 characteristic은 그림 4와 같다. 효과적인 공격을 위하여 단계별로 데이타의 filtering이 필요하며 이를 이용하면 공격 속도를 훨씬 줄일 수 있다. 특성확률은 1/10,486 으로서 약 25,000쌍의 데이타를 통하여 95% 의 성공률로서 암호키를 찾을 수 있다.

이 두 그림에서 알 수 있듯이 중앙의 라운드를 중심으로 대칭인 특성을 가질때 효율적인 공격이 가능한데 이런 특성을 iterated characteristic이라고 한다.

2.3. S-box설계와 DC

앞 절에서 설명한 바와 같이 DC는 iterated cryptosystem이 가지는 S-box의 비선형성을 효과적으로 공격할 수 있는 암호분석법이다. S-box의 설계에 대해서는 그간 기본적인 설계기준(design criteria)들이 존재해 왔으며[16] 최근에는 DC 공격을 어느정도 방어할 수 있는가 하는 점 역시 S-box 설계기준이 되고 있다. 최근 국내에서도 [12]에서 DC에 강한 S-box설계가 연구되었으나 S-box의 메모리가 4배로 늘어나는 단점이 있고 [13]에서 논의된 S-box 설계도 안전하지 못하다는 것이 밝혀졌다[17]. [14]에서 J.Adams는 DC가 S-box의 입출력 XOR의 관계가 비균일함에 착안하는 점을 중시하여 이를 균일화하기 위하여 bent 함수를 이용하는 방법을 제시하고 있다. [15]에서는 정보누출(information leakage)을 보다 억제할 수 있는 SP 네트워크의 설계방법을 제시하고 있다. 이와같이 당분간 DC를 효과적으로 방어할 수 있는 S-box의 설계문제는 암호학의 주요 쟁점이 되리라고 생각된다.

3. 실험결과

이 장에서는 지금까지 소개한 DC방법을 실제 프로그래밍하여 암호키를 찾아내기까지 속도를 측정한 결과이다. 실험은 편의상 Lucifer 알고리즘과 그 변형방식, 그리고 DES에 대하여 시도되었다.

3.1. Lucifer와 그 변형

Lucifer 알고리즘은 IBM에서 제정되어 상용되던 것으로서 DES의 전신이라고 할 수 있다[6]. 128비트인 한 블럭 입력이 두 개의 S-box에서 치환(substitution)을 거쳐 128비트의 하위키와 XOR 연산을 한 후 재배열(permutation), 확산(diffusion)을 차례로 거치게 된다. 각 S-box는 4비트 입력, 4비트 출력으로 하나의 키 비트에 의해서 두 box가 교환될 수 있도록 설계되었다. 또한 [11]에서는 Lucifer 형태의 암호시스템의 심볼간 상호의존도를 개선한 수정된 암호알고리즘을 발표하였다.

이에 대한 공격으로서 [3]에서 Biham과 Shamir는 Lucifer의 S-box의 출력비트들의 패턴을 관찰하고 8라운드의 Lucifer를 24~30쌍의 데이타를 사용하여 PC로 수 초내에 암호키를 찾을 수 있음을 밝혔다. 본 논문에서는 Lucifer 알고리즘은 물론 [11]에서 발표된 Lucifer의 심볼간 상호의존도를 개선한 암호알고리즘도 역시 DC에 대해 안전하지 못하다는 것을 실험으로 증명하였다. 수정된 128비트 암호키를 사용한 8라운드의 Lucifer를 IBM PC로 4~6초만에 거의 100% 암호키를 찾아낼 수 있었다. 프로그램은 C언어를 사용하였고 그 코드를 부록으로 수록하였다.

3.2. DES

Crypto'90에서 Biham과 Shamir가 DC를 이용하여 축소된 DES(reduced DES)를 공격하는 방법을 제시한 이래 마침내 Crypto'92에서 2^{55}의 복잡도(complexity)를 가진 표준 라운드인 16 라운드의 DES를 exhaustive search 보다 빠른 2^{37}단계만에 공격할 수 있었다고 발표하였다. 이는 2^{47}개의 선별된 데이타 중에서 2^{36}개의 데이타를 분석함으로써 이루어졌다.

본 실험에서는 축소된 4라운드, 6라운드, 그리고 8라운드의 DES에 대하여 C언어를 이용하여 DC를 프로그래밍하였으며 SUN sparc station 1+에서 수행하였을 때 6 라운드에서 240개의 데이타로 0.8초 정도에, 그리고 8 라운드에서 150,000개의 데이타로 2분 정도에 30 비트의 암호키를 찾아낼 수 있었다. [1]에서는 6 라운드 DES를 240개의 데이타로 0.3

초내에, 그리고 8 라운드 DES를 50,000개의 선별된 데이타 중에서 15,000개의 데이타를 분석하거나 랜덤한 150,000개의 데이타를 분석함으로써 2분내에 공격할 수 있었음을 발표하였으나 본 연구에서 실험된 공격속도는 상대적으로 느린 속도를 보였다.

4. 결론 및 연구방향

본 논문에서는 지금까지 상용되던 많은 iterated cryptosystem들을 효과적으로 공격할 수 있는 DC의 방법을 분석하고 지금까지 발표된 실험내용을 소개하였으며 그 중 IBM의 Lucifer와 그의 변형, 그리고 축소된 DES에 대하여 실제로 프로그래밍하여 그 결과를 제시하였다. 실험결과는 아직 미흡한 바가 많으나 앞으로 계속될 연구의 자료로 활용될 수 있으리라 믿는다.

최근 이 방법을 이용하여 각종 iterated cryptosystem이나 일방함수(one-way hash function)을 공격한 사례가 속속 발표되고 있으며 이에 대응하는 개선된 암호시스템이 역시 계속 발표되고 있으며[18] 이제는 이런 종류의 암호시스템을 설계하는 기준(criteria)의 하나로서 자리잡고 있다. 또한 DC를 방어할 수 있는 S-box의 입출력 XOR를 균일(uniform)하게 만들어 특성확률(prob. of charcteristic)을 낮추는 방안이나 정보이론적 접근방법이 제시되고 있다.

국내에서도 표준화할 수 있는 고유한 암호시스템의 개발이 시급한 당면과제라고 할 수 있으며 Differnetial Cryptandalysis는 그 설계의 중요한 기준의 하나로서 충분히 고려되어야 할 사항임에 분명하다. 최근 발표된 16 라운드 DES도 아직 완벽히 공격했다고 할 수 없으나 컴퓨터 프로세서 속도가 급속히 향상되고 있는 점으로 미루어 이제 DES의 안전성은 제고되어야 할 것이다. 그런 만큼 국내에서 개발되는 iterated형 암호시스템은 Differnetial Cryptanalysis의 검증을 충분히 거쳐서 설계되어야 할 것으로 생각된다.

참 고 문 헌

1. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, Vol. 4, No. 1, 1991.

2. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," in *Proc. of Crypt'90*, 1990.

3. E. Biham and A. Shamir, "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer," in *Proc. of Crypt'91*, 1991.

4. E. Biham and A. Shamir, "Differential Cryptanalysis of Feal and N-Hash," in *Proc. of Eurocrypt'91*, April. 1991, pp.1-8.

5. E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-round DES," in *Proc. of Crypt'92*, 1992.

6. H. Feistel, "Cryptography and data security," *Scientific American*, Vol. 228, No. 5, May. 1973, pp.15-23.

7. NBS, "Data Encryption Standard," *FIPS PUB* 46, Jan. 1977.

8. Shoji Miyaguchi, "FEAL-N specifications," *NTT*, 1989.

9. A. Sorkin, "Lucifer, a cryptographic algorithm," Cryptology, Vol. 8, No. 1, Jan. 1984.

10. L. Brown, J. Pieprzyk and J. Seberry, "LOKI-A cryptographic primitive for authentication and secrecy applications," *in Proc. of Auscrypt'90*, 1990, pp. 229-236.

11. 강해동, 이창순, 문상재, "Lucifer 형태의 암호화 알고리듬에 관한 연구," 대한전자공학회논문지, 제28권, 제3호, 1989, 3월, pp.339-346.

12. 임웅택, 남길현, "Differential Cryptanalysis에 강한 S-box설계에 관한 연구," 데이타 보호 기반기술 Workshop, 1992.

13. Kwangjo Kim, "Construction of DES-like S-boxes based on boolean functions satisfying the SAC," in *Proc. of Asiacrypt'91*, 1991.

14. C.M. Adams, "On immunity against Biham and Shamir's differential cryptanalysis," Info. proc. lett. 41, No. 2, pp.77-80.

15. M. Sivabalan, S.E. Tavares and L.E.

Peppard, "On the design of SP networks from an information theoritic point of view," in *Proc. of Crypto'92*, 1992.

16. E.F. Brickell, J.H. Moore and M.R. Purtill, "Structure in the S-boxes of the DES," in *Proc. of Crypto'86*, 1986.

17. L.R. Knudsen and N. Munkegade, "Iterative characteristics of DES and s^2-DES," in. *Proc. of Crypto'92*, 1992.

18. L. Brown, M. Kwan, J. Pieprzyk and J. Seberry, "Improving resistance to differential cryptanalysis and the redesign of LOKI," in Proc. of Asiacrypt'91, 1991.

19. R.C. Merkel, "Fast software one-way hash function," J. of Cryptology, Vol. 3, No. 1, 1990, pp.43-58.

20. R.C. Merkel, "Fast software encryption functions," in *Proc. of Crypto'90*, 1990.

21. T.W. Cusick and M.C. Wood, "The REDOC-II cryptosystem," in *Proc. of Crypto'90*, 1990.

```
/*
 *    Luci.i  contains the fixed permutation and substitution tables
 *            for differential cryptanalysis of modified Lucifer algorithm
*/

/* S0-box substitution table*/
char S0[16]
={  12, 15, 7, 10, 14, 13, 11, 0, 2, 6, 3, 1, 9, 4, 5, 8  };
/* S1-box substitution table */
char S1[16]
={  7, 2, 14, 9, 3, 11, 0, 4, 12, 13, 1, 10, 6, 15, 8, 5  };

/* inverse P-box table*/
char iP[32]
={  2, 5, 4, 0, 3, 1, 7, 6 };

/* reverse convolution register table for S0 */
int KB0[32] [2]
={  {3, 7}, {0, 1}, {2, 3}, {1, 6}, {1, 2}, {1, 2}, {5, 7},
    {4, 5}, {2, 6}, {3, 7}, {2, 4}, {0, 5}, {1, 2},
    {0, 6}, {4, 5}, {3, 4}, {1, 5}, {4, 6}, {1, 3},
    {4, 7}, {0, 2}, {5, 7}, {3, 5}, {2, 3}, {0, 4},
    {2, 5}, {0, 2}, {3, 6}, {6, 7}, {4, 6}, {2, 4}, {1, 2}} ;
```

```
/* reverse convolution register table for S1 */
int KB0[32][2]
={  {3, 7}, {1, 4}, {6, 7}, {2, 3}, {5, 6}, {3, 5}, {1, 3},
    {0, 1}, {2, 6}, {2, 3}, {6, 7}, {1, 4}, {4, 5},
    {3, 4}, {0, 2}, {0, 7}, {1, 5}, {2, 7}, {5, 6},
    {0, 3}, {3, 4}, {1, 3}, {1, 7}, {6, 7}, {0, 4},
    {1, 6}, {3, 4}, {2, 7}, {2, 3}, {0, 2}, {0, 6}, {5, 6}} ;

/*
 *    Differential Cryptanalysis of Modified Lucifer Algorithm : ROUND 4
 *    Ref :
 *        i) J. of KITE Vol. 26, NO. 3, May, 1989. pp.32-39.
 *        ii) Proc. of Crypto'91, 1991.
 *    Programmed by Joo Yeon Cho, Dept. of E.E., POSTECH
 */

#include<stdio.h>
#include"Luci.i"
#include<time.h>
#include<io.h>
#define ROUND    4

unsigned long s0[16][16][2] = {0}, s1[16][16][2] = {0} ;
unsigned char key[16] ;

main()
{
        unsigned char fkey[8], rkey[16] ;
        int     i,j,rd,iv ;
        float   tm=0.0 ;
        clock_tclock(void) ;

        get_sbox_ioXOR(s0, s1) ;      /*get distribution table of pairs XOR */
        getkey(key) ;                 /*typing input of 128 bit key*/
        tm=clock() ;
        for(rd=ROUND ; rd>1 ; rd--) {
                iv=((rd-1)*7)% 16 ;
                roundkey(fkey, rd) ;   /*find subkey of each round*/
                for(i=0 ; i<8 ; i++) {
                        j=(iv+i)% 16 ;
                        rkey[j]=fkey[i] ;
                }
        }
        tm=clock()-tm ;
```

```
/*Monitoring main 128 bit key*/
        printf("\nResult : ") ;
        for(i=0 ; i<16 ; i++)
                pirntf("%c", rkey[i]) ;
        printf("\n") ;
        printf("The time elapsed : %f secs.\n",tm/CLK_TCK) ;
}

/*routime for finding subkey of ench round*/
roundkey(fkey, rd)
unsigned char fkey[8] ;
int     rd ;
{
        unsigned char       key1[16] ;
        static unsigned long citx0[4]={0}, citx1[4]={0} ;
        static unsigned long pitx0[4]={0}, pitx1[4]={0} ;
        static unsigned char k0[8][16]={0}, k1[8][16]={0} ;
        unsigned char pk0[8][16], pk1[8][16] ;
        int     i,j,m=2, t, max0, max1, z, w ;

        /*plaintext generation*/
        for(i=0 ; i<8 ; i++) {
                for(j=0 ; j<16 ; j++) {
                        pk0[i][j]=0 ; pk1[i][j]=0 ;
                }
        }
        while(m ! =0) {
                if(m==2) {pitx[0]=1 ; pitx0[1]=0 ; }
                if(m==1) {pitx[0]=0 ; pitx0[1]=1 ; }
                for(t=0 ; t<32 ; t++) {
                        if(t ! = 0) {
                                pitx0[0]<<=1 ; pitx0[1]<<=1 ;
                                pitx1[0]<<=1 ; pitx1[1]<<=1 ;
                        }

                        for(i=0 ; i<16 ; i++) key1[i]=key0[i] ;
                        Luci(pitx0, citx0, key1, rd) ;       /*modified Lucifer algorithm*/
                        for(i=0 ; i<16 ; i++) key1[i]=key0[i] ;
                        Luci(pitx1, citx1, key1, rd) ;       /*modified Lucifer algorithm*/

                        findkey(citx0, citx1, pk0, pk1, t, m) ; /*find possible key*/
                }

                /*Voting to the max value*/
```

```
            for(i=0 ; i<8 ; i++) {
                max0=pk0[i] [0] ; max1=pk1[i] [0] ; k ; k0[i] [1]=0 ; k1[i] [1]=0 ;
                for(j=1 ; j<16 ; j++) {
                    if(pk0[i] [j]>max0) {max0=pk0[i] [j] ; k0[i] [1]=j ; }
                    if(pk1[i] [j]>max1) {max1=pk1[i] [j] ; k1[i] [1]=j ; }
                }
                if((max0 ! =0))||(max1 ! =0) {
                    z=1 ; w=1 ;
                    for(j=0 ; j<16 ; j++) {
                        if((pk0[i] [j]==max0)&&(j ! =k0[i] [1])
                            k0[i] [++z]=j ;
                        if((pk1[i] [j]==max1)&&(j ! =k1[i] [1])
                            k1[i] [++w]=j ;
                    }
                    k0[i] [0]=z ; k1[i] [0]=w ;
                }
            }
            m=m-1 ;
        }
}

/*routine for finding possibel subkey*/
findeky(citx0, citx1, pk0, pk1, t, m)
unsinged long citx0[4], citxl[4] ;
unsigned char pk0[8] [16], pk1[8] [16] ;
int     t,m ;
{
        unsigned long inXOR[2], outXOR[2], tmp, ct0[2], ct1[2] ;
        unsigned char x0, x1, y0, y1, sin[2], inX, outX, max0, max1 ;
        unsigned char z, w, pi ;
        int     i, j, k=1, rem, ix, p ;

        for(p=0 ; p<2 ; p++) {
            outXOR[0]=citx0[0]^citx1[0] ;
            outXOR[1]=citx0[1]^citx1[1] ;
            inXOR[0]=citx0[2]^citx1[2] ;
            inXOR[1]=citx0[3]^citx1[3] ;
            ct0[0]=citx0[2] ; ct0[1]=citx0[3] ;
            ct1[0]=citx1[2] ; ct1[1]=citx1[3] ;
            ipbox(outXOR) ;
            if(m==2) {
                k=KB0[t] [p]/4 ; rem=(7-KB0[t] [p]) %4 ; ix=KB0[t] [p] ;
            }
            if(m==1) {
```

```
        k=KB1[t][p]/4 ; rem=(7-KB0[t][p]) %4 ; ix=KB1[t][p] ;
}
inXOR[k]>>=(rem*8) ; outXOR[k]>>=(rem*8) ;
inX=(inXOR[k] & 0xff) ; outX=(outXOR[k] & 0xff) ;
ct0[k]>>=(rem*8) ; sin[0]=(ct0[k] & 0xff) ;
ct1[k]>>=(rem*8) ; sin[1]=(ct1[k] & 0xff) ;

x1=inX & 0xf ; x0=(inX & 0xf0)>>4 ;
y1=ioutX & 0xf ; y0=(outX & 0xf0)>>4 ;

/*case 1 : unswapped s input*/
if((s0[x0][y0][0] ! =0) && (x0 ! =0 && y0 ! =0)) {
        tmp=s0[x0][y0][1] ;
        for(j=0 ; j<s0[x0][y0][0] ; j+=2) {
                pi=(tmp & 0xf) ;
                z=(pi^((sin[0] & 0xf0)>>4)) ;
                w=(pi^((sin[1] & 0xf0)>>4)) ;
                pk0[ix][z]++ ; pk0[ix][w]++ ; tmp>>4 ;
        }
}
if((s1[x1][y1][0] ! =0)) && (x1 ! =0 && y1 ! =0)) {
        tmp=s1[x1][y1][1] ;
        for(j=0 ; j<s1[x1][y1][0] ; j+=2) {
                pi=(tmp & 0xf) ;
                z=(pi^(sin[0] & 0xf)) ;
                w=(pi^(sin[1] & 0xf)) ;
                pk1[ix][z]++ ; pk1[ix][w]++ ; tmp>>=4 ;
        {
{

/*case 2 : swapped s input*/
if((S1[x0][y1][0] ! =0) && (x0 ! = && y1 ! =0)) {
        tmp=s1[x0][y1][1] ;
        for(j=0 ; j<s1[x0][y1][0] ; j+=2) {
                pi=(tmp & 0xf) ;
                z=(pi^((sin[0] & 0xf0)>>4)) ;
                w=(pi^((sin[1] & 0xf0)>>4)) ;
                pk0[ix][z]++ ; pk0[ix][w]++ ; tmp>>=4 ;
        }
}
if((S0[x1][y0][0] ! =0) && (x1 ! = && y0 ! =0)) {
        tmp=s0[x1][y0][1] ;
        for(j=0 ; j<s0[x1][y0][0] ; j+=2) {
                pi=(tmp & 0xf) ;
```

```
                        z=((pi^(sin[0] & 0xf0)) ;
                        w=(pi^((sin[1] & 0xf0)) ;
                        pk1[ix] [z]++ ; pk1[ix] [w]++ ; tmp>>=4 ;
                    }
                }
        }
}

/*routine for access 128 bit main key*/
getkey(k)
unsigned char k[16] ;
{
        int     i=0, ch ;
GK :    printf("\nEnter key : ") ;

        while((ch=getch() ! =0x0d) {
                k[i++]=(unsigned char)ch ;
        }
        if(i<8) {
                prntf("\nBAD key specification. more, please.") ;
                goto GK ;
        }
        printf("\n") ;

        for(i=0 ; i<8 ; i++)
                k[8+i]=k[i] ;

}

/*routine for reverse permutation box*/
ipbox(r)
unsigned long *r ;
{
        int     i, j, mask=0x80 ;
        unsigned char tmp, pr[8] ;

        Itoc(r, pr) ;
        for(i=0 ; i<8 ; i++) {
                tmp=0 ;
                for(j=0 ; j<8 ; j++) {
                        tmp | =((pr[i]) & mask)>>iP[j] ;
                        pr[i]<<=1 ;
                }
                pr[i]=tmp ;
```

```
        }
        ctol(pr, r) ;
}

/*routine for access pairs XOR distribution table of each S-box*/
get_sbox_ioXOR(s0, s1)
unsigned long s0[16][16][2], s1[16][16][2] ;
{
        int    x, y0, y1, i, j, k ;

        for(i=0 ; i<16 ; i++) {
                for(j=i ; j<16 ; j++) {
                        x=i^j ;
                        y0=sbox0(i)^sbox0(j) ;
                        y1=sbox1(i)^sbox1(j) ;
                        if (i==j)
                                {s0[x][y0][0]+=1 ; s1[x][y1][0] +=1 ; }
                        else
                                {s0[x][y0][0]+=2 ; 2s[x][y1][0] +=2 ; }
                        s0[x][y0][1]<<=4 ; s0[x][y0][1] | =i ;
                        s1[x][y1][1]<<=4 ; s1[x][y1][1] | =i ;
                }
        }
}
/*S0-box*/
sbox0(i)
int     i ;
{
        return(S0[i]) ;
}

/*S1-box*/
sbox1(i)
int     i ;
{
        return(S1[i]) ;
}

/*routine for conversion character to long integer*/
ctol2(tmp, work)
unsigned long *work ;
unsigned char *tmp ;
{
        int     j ;
```

```
        work[0]=0 ; work[1]=0 ;
        for(j= 0 ; j<4 ; j++) {
                work[0]<<=8 ;
                work[0] | =tmp[j] ;
        }
        for(j=4 ; j<8 ; j++) {
                work[1]<<=8 ;
                work[0] | =tmp[j] ;
        }
}

/*routine for conversion long integer to character*/
1toc2(work, tmp)
unsigned long *work ;
unsigned char *tmp ;
{
        int     i, j ;

        for(j=0 ; j<4 ; j++) {
                tmp[j]=work[0] & 0xff ;
                work[0]>>=8 ;
        }
        for(j=4 ; j<8 ; j++) {
                tmp[j]=work[1] & 0xff ;
                work[1]>>=8 ;
        }
}
```

□ 著者紹介

趙 柱 衍 (학생회원)

1968년 11월생
1991년 2월 서울대학교 제어계측공학과 학사
1993년 2월 포항공과대학 전자전기공학과 석사
1993년 2월~현재 금성정보통신 연구원

李 弼 中 (정 회 원)

1951년 12월생
1974년 2월 서울대학교 전자공학과 학사
1977년 2월 서울대학교 전자공학과 석사
1982년 6월 U.C.L.A. System Science, Engineer
1985년 6월 U.C.L.A. Electrical Engineering, Ph.D.
1980년 6월~1985년 8월 : Jet Propulsion Laboratory, Senior Engineer
1985년 8월~1990년 2월 : Bell Communications Research, M.T.S
1990년 2월~현재 : 포항공과대학 전자전기공학과, 부교수.