論文 89-26-3-5

Lucifer 형태의 암호화 알고리듬에 관한 연구

(A Study on Lucifer-Type Encryption Algorithm)

姜 海 東*, 李 昌 淳**, 文 相 在***

(Hae Dong Gahng, Chang Soon Ri and Sang Jae Moon)

要 約

본 논문에서는 Lucifer의 키이바이트 사용스케쥴과 펼쳐진 콘볼루션 레지스터를 변화시키면서 32, 64, 그리고 256비트 블록크기 Lucifer 형태의 암호화 알고리듬들을 제시하고 이들의 심볼간 상호의존도와 exhaustive 암호 분석의 비도를 비교 조사한다. 또 키이 interruption 위치 변경과 autoclave 방식의 대체상자 입력 등의 심볼간 상호의존도를 개선하는 방법들도 제시한다.

Abstract

This paper presents Lucifer-type encryption algorythms with variable length of block size of 32,64, and 256 bits by varying the key byte access schedule and the unfolded convolution register of Lucifer. The intersymbol dpendence and exhaustive cryptanalysis of the algorithms are examined and compared. We also present the methods which improve the intersymbol dependence such as moving the location of key interruption and autoclave selection of S-box.

I. 서 론

주로 외교 혹은 군사적 목적으로 오래전부터 사용되어 온 암호화는 최근에 컴퓨터의 보급과 정보통신 기술의 발전으로 전자우편, 데이타베이스, 종합정보통신망 등의 활용이 보편화 됨에 따라, 상업 혹은 민간용으로도 정보를 보호하기 위하여 널리 사용되어 진다.

이를 위한 암호화에는 대체(substitution)와 치환(permutation)을 병용하여 암호화하는 것과 정수나 다항식 개념을 데이타에 도입하고 이들에게 지수승을 취하여 적절히 암호화하는 것으로 대별할 수 있다. 대표적인 것으로, Lucifer[1]와 DES[2] 등은 전자에 그리고 RSA방식[3]과 SEEK방식[4] 등은 후자에 속한다.

본 논문에서는 블록크기가 32, 64, 그리고 256비트인 Lucifer 형태의 암호화 알고리듬들을 구성하였다. 다음으로, 이들에 대한 암호의 불법적 분석을 방지하는데 중요한 측도가 되는 심볼간 상호의존도[5,6]와 exhaustive 암호분석 비도를 구하고 128비트 Lucifer의 경우와 비교 고찰한다. 라운드마다 출력되는 비트가 정보어 혹은 키이의 모든 입력비트들에 영향을 받아 변환되면 상호의존한다고 하며 본 논문에서, 심볼간 상호의존도는 Mayer와 Matyas의 계산방법[5]을 그대로 적용하여 암호어의 비트중 상호의존하는 비트의 비율을 백분율로 표시하였다. 또 exhaustive

*準會員, ***正會員, 慶北大學校 電子工學科
(Dept. of Elec. Eng., Kyungpook Nat'l Univ.)
**正會員, 大邱 工業專門大學 電算學科
(Dept. of Computer Science, Taegu Tech. Jr. College)

接受日字 : 1988年 11月 2日

암호분석 비도는 상당량의 평어와 해당 암호어를 가지고 가능한 모든 키이를 적용시켜 암호해독을 하는 exhaustive 암호분석시 평균적으로 소요되는 시간이 된다. 그리고 의존도를 향상시킬 수 있는 방법도 제시하였다.

Ⅱ. Lucifer 암호화 알고리듬

Lucifer는 128비트 정보어를 128비트의 키이를 사용하여 암호화하는 블록암호화 알고리듬이며 대체와 치환을 혼용하는 product암호법이다.[1] Lucifer는 그림 1과 같은 과정을 열여섯번 반복 수행한다. 그림 1은 한 라운드의 암호화 과정을 도시한 것이다.

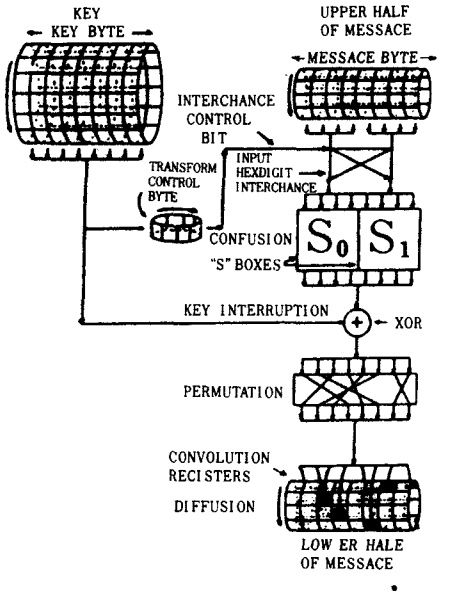


그림 1. CID (confusion key-interruption diffusion) 블록선도

Fig. 1. Block diagram of CID (confusion key-interruption diffusion) logic.

평어 128비트는 64비트씩 상반부, 하반부로 나누어진다. 매 라운드마다 상반부와 키이의 영향을 받아서 새로운 64비트를 구성한다. 이 변형된 64비트는 다음 라운드에서 상반부가 되고 하반부는 직전 라운드의 상반부를 그대로 사용한다. 128비트 키이는 16바이트로 구분되고 64비트 평어 상반부는 8바이트로 구분되어 각기 그림 1의 원통과 같이 구성된다. 라운드마다 사용되는 키이 비트수는 평어의 상반부 비트수와 같은 64비트, 즉 8바이트가 사용된다. 매 라운드에서 마지막 키이바이트가 수행되고 나면 그 바이트는 회전하지 않고 정지하여 다음 라운드의 첫 키이바이트가 된다. 표 1은 매 라운드의 64비트 키이 발생을 위한 스케쥴이다. 매 라운드마다 사용되는 키이의 첫 바이트(표 1의 0열)는 변환 조정바이트(transform-control-byte)로 사용된다. 이 바이트의 8개 비트들은 상호교환 조정비트(interchange-control-bit)가 되어, 매 비트는 평어 상반부가 어떤 방법으로 2개의 대체상자에 입력되는 지를 결정한다. 상호교환 조정비트가 "0"이면 평어 상반부 바이트의 좌우 4비트의 두 십진값이 각각 S_0와 S_1의 입력이 되고, "1"이면 두 십진값이 서로 바뀌어 대체상자에 입력된다.

표 1. 키이바이트 사용스케쥴

Table 1. Key byte access schedule.

round (CID) \ message byte	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	7	8	9	10	11	12	13	14
3	14	15	0	1	2	3	4	5
4	5	6	7	8	9	10	11	12
5	12	13	14	15	0	1	2	3
6	3	4	5	6	7	8	9	10
7	10	11	12	13	14	15	0	1
8	1	2	3	4	5	6	7	8
9	8	9	10	11	12	13	14	15
10	15	0	1	2	3	4	5	6
11	6	7	8	9	10	11	12	13
12	13	14	15	0	1	2	3	4
13	4	5	6	7	8	9	10	11
14	11	12	13	14	15	0	1	2
15	2	3	4	5	6	7	8	9
16	9	10	11	12	13	14	15	0

라운드마다 2개의 대체상자와 한번의 치환상자를 거치는 과정을 8번 반복 수행한다. 한번 수행시마다 그 과정은 표 2에 의하여 이루어지는데 그림 1에서와 같이 키이바이트와 대체상자 출력바이트 간에 비트별 mod-2 가산이 행하여지고 그 결과가 치환상자에 입력된다.

치환과정을 거쳐 얻어진 한 바이트의 8비트들은 평어 하반부의 특정 비트들과 비트별 mod-2(XOR) 가산을 하는데 그 연산이 행해지는 방법은 그림 2에 도시되어 있다. 예로서 치환상자의 출력인 첫 라운드

표 2. 대체상자와 치환상자
Table 2. Substitution and permutation.

S-BOX INTERNAL PERMUTATION		FIXED PERMUTATION
S_0	S_1	
0…12	0… 7	0…3
1…15	1… 2	1…5
2… 7	2…14	2…0
3…10	3… 9	3…4
4…14	4… 3	4…2
5…13	5…11	5…1
6…11	6… 0	6…7
7… 0	7… 4	7…6
8… 2	8…12	
9… 6	9…13	
10… 3	10… 1	
11… 1	11…10	
12… 9	12… 6	
13… 4	13…15	
14… 5	14… 8	
15… 8	15… 5	

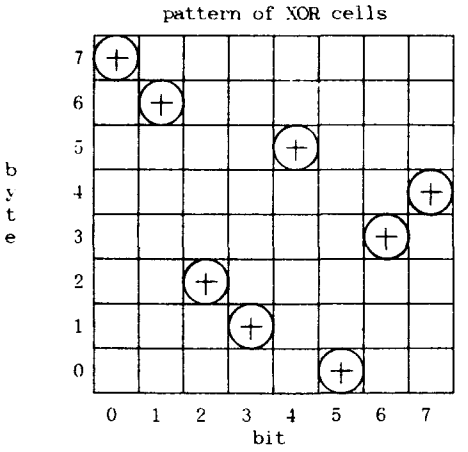


그림 2. 펼쳐진 콘볼루션 레지스터
Fig. 2. Unfolded convolution registers.

첫 바이트의 각 비트들이 순서대로 평어 하반부의 7, 6, 2, 1, 5, 0, 3, 과 4번 바이트의 해당 비트들과 (⊕로 표시됨) 각각 연산이 수행되고 나면 상반부와 하반부의 원통들은 한 바이트씩 회전을 한다. 즉 그림 2에서는 아래쪽으로 이동한다. 치환 과정에서 출력되는 그 다음 바이트의 각 비트들도 이전 바이트처럼 회전된 하반부와 똑같이 mod-2 연산을 수행하게 된다. 이렇게 형성된 64비트는 다음 라운드의 상반부에, 그리고 하반부는 직전 라운드의 상반부를 그대로 사용한다.

위 설명과 같이 매 라운드 암호화 과정이 16번 반복되어 최종출력 128비트가 얻어진다. 수신측에서의 암호해독을 위한 복호는 암호화의 역순으로 그림 1의 CID과정을 수행하되 키이바이트는 키이바이트 사용스케쥴 표 1의 아래쪽에서 위쪽으로 왼쪽에서 오른쪽의 차례로 키이바이트를 사용하게 된다.

Ⅲ. 대체상자의 성질

Lucifer의 대체상자는 Lucifer 암호화 알고리듬의 중요 구성요소이다. NSA(national security agency)가 DES 설계자들에게 DES의 대체상자들이 만족해야할 성질들을 제시하였다.[7] 본 논문에서는 NSA가 제시한 성질들중, 입력비트수가 4비트인 Lucifer의 대체상자에도 적용될 수 있는 것들만 적용하여 Lucifer의 대체상자의 성질을 조사하였다. Lucifer에서도 적용될 수 있는 성질들을 나열하면,

P0. 대체상자의 각 열은 0에서 15까지의 십진수를 치환함.

P1. 비선형이고 Affine 하지 않음.

등이다.

성질 P0는 Lucifer의 대체상자에도 만족됨을 표 2로 쉽게 알 수 있다. 대체상자의 출력 4비트들은 각각 입력 4비트의 함수로 볼 수 있다. 입력비트들을 w, x, y, 와 z로 두고, i번째 출력비트를 부울린 함수 $f_i(w, x, y, z)$, i=0, 1, 2, 3로 나타내면 S_0와 S_1의 부울린 함수들은 각각 식(1)과 (2)이다.

$$\left.\begin{aligned} f_0 &= \bar{w}\bar{y}+\bar{w}\bar{x}z+\bar{w}x\bar{z}+x\bar{y}\bar{z}+wxyz \\ f_1 &= \bar{w}\bar{y}+\bar{y}z+w\bar{x}\bar{z}+\bar{w}\bar{x}yz \\ f_2 &= \bar{w}\bar{x}z+\bar{w}x\bar{z}+w\bar{x}\bar{y}+\bar{x}y\bar{z} \\ f_3 &= y\bar{z}+w\bar{x}y+wx\bar{z}+\bar{w}\bar{y}z \end{aligned}\right\} \quad (1)$$

$$\left.\begin{aligned} f_0 &= \bar{w}\bar{x}y+w\bar{x}\bar{y}+w\bar{x}z+x\bar{y}z+wxy\bar{z} \\ f_1 &= w\bar{y}+\bar{w}\bar{x}\bar{z}+xyz \\ f_2 &= \bar{w}\bar{y}+wx\bar{y}+wxz+w\bar{x}y\bar{z} \\ f_3 &= wxz+w\bar{y}z+\bar{w}\bar{y}z+x\bar{y}z+\bar{w}\bar{x}yz+w\bar{x}yz \end{aligned}\right\} \quad (2)$$

식(1)과 (2)에서 알 수 있듯이 선형이거나 Affine한 함수는 없으므로 Lucifer의 대체상자는 성질 P1도 만족함을 알 수 있다.

그런데 각 대체상자의 출력비트들이 대체상자에 입력되는 모든 비트의 함수라는 것이 모든 입력비트에 대체가 균등히 이루어지는 것은 아니다. 예로서, S_0

상자에 1010이 입력되면 0011이 출력되므로 입력의 양쪽 2비트만 대체된 것이다. 또 S_1상자에 1110이 입력되면 1000이 출력되므로 가운데 2비트만이 대체된 것이 된다. 표 3 은 0000에서 1111까지, 16가지의 입력에 대하여 각 출력비트의 대체된 횟수를 대체된 총 출력비트수에 대한 백분율로 나타낸 것이다. 표 3 으로부터 S_0에서는 0번 비트가 그리고 S_1에서는 2번 비트가 가장 많이 대체되고, S_0가 S_1보다 균등한 분포를 가짐을 알 수 있다.

표 3. 대체상자 입력에 각 출력비트가 받는 영향

Table 3. Effect on each output bit by the input of the S-boxes.

출력비트자리	S_0	S_1
0 (MSB)	31.58%	18.75%
1	21.05	25.00
2	21.05	37.50
3	26.32	18.75

Ⅳ. Lucifer형태의 암호화 알고리듬

1. 구성

Lucifer가 완성된 1970년의 LSI기술 수준은 Lucifer 암호화 알고리듬 구조에 많은 제약을 주었다.[1] 그 결과 Lucifer의 하드웨어는 시프트레지스터(shift register)를 이용하여 키이와 평어를 한 바이트씩 처리한다. 본 논문에서 제시한 Lucifer 형태의 암호화 알고리듬은 표 1 의 키이바이트 사용스케쥴과 그림 2 의 펼쳐진 콘볼루션 레지스터를 32, 64, 그리고 256 비트 등의 블록크기에 맞게 적절히 수정하여 구성하였다.

키이바이트의 사용은 표 1 에서 보듯이, 한 블록의 암호화 과정이 끝나고 나면 그림 1 의 키이를 저장한 원통형 레지스터는 그 블록을 시작할 때의 위치로 되돌아와야 한다. 즉 모든 블록의 각 라운드마다 쓰이는 키이바이트는 동일해야 한다. 그리고 표 1 의 각 열에는 동일한 키이바이트가 없음을 알 수 있다. 이것은 각 열에 해당되는 평어의 각 바이트는 매 라운드마다 다른 키이바이트에 의해 암호화됨을 뜻한다. 즉 모든 키이바이트는 균등이 사용되게 되는 것이다. 이러한 성질을 이용하여 구성한 각 블록크기의 구체적인 키이바이트 사용스케쥴은 표 4 와 같다.

표 4. Lucifer 형태의 암호화 알고리듬의 키이바이트 사용스케쥴

Table 4. Key byte access schedules for Lucifer-type encryption algorithms.

message byte

CID round	0	1
1	0	1
2	1	2
3	2	3
4	3	0

(a) 32비트

message byte

CID round	0	1	2	3
1	0	1	2	3
2	3	4	5	6
3	6	7	0	1
4	1	2	3	4
5	4	5	6	7
6	7	0	1	2
7	2	3	4	5
8	5	6	7	0

(b) 64비트

message byte

CID round	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
3	30	31	0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
5	28	29	30	31	0	1	2	3	4	5	6	7	8	9	10	11
6	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
7	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
27	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
28	21	22	23	24	25	26	27	28	29	30	31	0	1	2	3	4
29	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
30	19	20	21	22	23	24	25	26	27	28	29	30	31	0	1	2
31	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
32	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	0

(c) 256비트

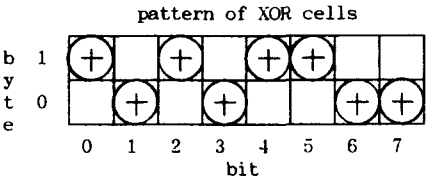
블록당 수행되는 전 라운드수는 전체 키이바이트 수와 동일하다. 대체상자와 치환상자는 바이트 단위로 연산이 수행되므로 128비트 Lucifer의 그것들을 그대로 이용한다.

그림 1 의 암호화 과정에서 마지막 과정인 확산(diffusion)은 빠른 시간에 넓게 이루어지는 것이 바람직하다. 이러한 확산기능을 얻기 위해서 Lucifer에서는 XOR 셀(cell)을 하반부의 각 바이트에 고르게 분포시켰음을 그림 2 에서 알 수 있다. XOR 셀을 각 바이트에 고르게 분포시켜야 바람직한 확산 기능을 얻을 수 있다는 것은 반대로 XOR 셀을 한 바이트에 편중되게 분포시킬 경우의 확산 형태를 조사해 보면 알 수 있다. 그림 2 의 8개 XOR 셀들을 모두 0번 바

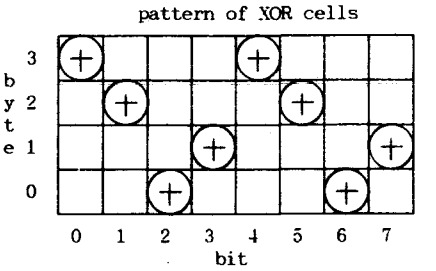
이트에 편중시키거나 또 다른 한 바이트에만 편중시켰을 때는 전혀 확산이 되지 않거나 확산이 되더라도 비트 단위가 아닌 바이트 단위로 확산이 되어 확산 형태가 주기를 갖게 된다. 예를들어, 그림 2 처럼 XOR 셀들을 두면, 고정치환의 첫번째 출력 8비트들은 하반부의 7, 6, 2, 1, 5, 0, 3, 그리고 4번 바이트의 각 해당 비트로 확산이 된다. 그러나 XOR 셀을 모두 0번 바이트에 두면 고정치환의 첫번째 출력바이트는 하반부 0번 바이트와 mod-2 연산이 되므로 확산은 전혀 이루어지지 않는다. 또 XOR 셀들을 모두 1번 바이트에 두면 고정치환의 첫번째 출력 8비트는 하반부 1번 바이트에만 확산이 된다. 특히, Lucifer의 고정치환은 4라운드의 주기를 갖고 있다. 고정치환 상자에 임의의 8비트를 입력하여 출력되는 8비트를 다시 고정치환상자에 입력하는 방법으로 4번만 반복하면 4번째로 출력되는 8비트는 치환이 전혀 되지않은 상태, 즉 첫번째 입력비트들과 동일하게 되는 것이다. 따라서 바이트를 연산의 기본 Lucifer 암호화 과정의 특성을 고려한다면 XOR 셀을 하반부의 각 바이트에 균등하게 분포시키는 것은 꼭 필요하다 하겠다.

그림 2 와는 달리 XOR 패턴을 7, 6, 5, 4, 3, 2, 1, 0 번 바이트의 순서로 혹은 0, 1, 2, 3, 4, 5, 6, 7번 바이트의 순서로도 균등하면서 이해하기 쉽게 구성할 수도 있겠다. 이처럼 XOR 패턴을 구성하여도 심볼간 상호의존도에 큰 차이는 없었으나 그림2 처럼 바이트를 랜덤성있게 선택한 구성이 평균적으로 좋은 (0.1% 정도) 심볼간 상호의존도를 얻을 수 있었다.

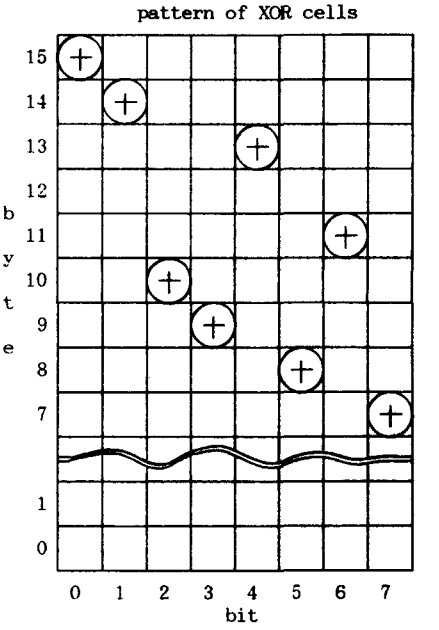
전술한 특성을 고려하여 구성한 각 블록크기의 펼쳐진 콘볼루션 레지스터는 그림 3 과 같다. 32비트와 64비트의 경우, 각각 30개와 5개의 키이에 대하여 가능한 모든 XOR 패턴에 해당되는 심볼간 상호의존도를 구하여 의존도가 가장 좋은 패턴중에 XOR 셀의 하반부 각 바이트에 균등한 분포를 고려한 패턴의 한 예이며, 256비트의 경우는 XOR 셀의 균등한 분포를 고려하여 1000개 패턴을 샘플링하고 이들에 해당되는 심볼간 상호의존도를 비교하여 구성한 예이다. 특히 256비트의 경우는 고정치환의 출력비트수 8보다 더 많은 16바이트의 하반부가 있으므로 어느 바이트에 XOR 셀을 두느냐가 문제인데, 심볼간 상호의존도를 조사한 결과, 대체로 하반부 16바이트 중 상위바이트(8~15번 바이트)에 XOR 셀을 두는 경우가 빠른 확산을 보였다.



(a) 32비트



(b) 64비트



(c) 256비트

그림 3. Lucifer 형태의 암호화 알고리듬의 펼쳐진 콘볼루션 레지스터

Fig. 3. Unfolded convolution registers for lucifer-type encryption algorithms.

2. 심볼간 상호의존도

라운드마다 출력되는 비트는 정보(평어)비트들과 키이비트들의 함수로 볼 수 있다. 즉, 각 출력비트는 서로 다른 입력비트들에 영향을 받게(혹은 의존하게)된다. 이러한 입출력 각 비트간의 의존도(이를 심볼간 상호의존도로 함)를 조사함으로써 암호 방법의 복잡성을 비교할 수 있다.

각 블록크기에 맞는 그림 3과 표 4를 작성하고 각 블록크기 Lucifer 형태의 암호화 알고리듬을 구성하고, Mayer와 Matyas의 계산방법[6]을 그대로 적용하여 이들의 심볼간 상호의존도를 키이의존도와 정보의존도로 나누어 구한 결과를 표 5에 나타내었다. Lucifer의 심볼간 상호의존도는 DES와는 달리 키이에 따라 각 의존도가 달라지게 되는데 이는 키이의 상호교환 조정비트가 각 대체상자에 입력될 정보비트들(대체상자당 4비트)을 조정하기 때문이다. 본 논문에서는 상호교환 조정비트가 1이 나올 확률과 0이 나올 확률이 똑같이 1/2이라 가정하는 것이 타당하므로 키이비트를 랜덤하게 1과 0이 나올 확률이 각각 1/2이 되도록 발생시켰다.

표 5. Lucifer형태 암호화 알고리듬의 심볼간 상호의존도

Table 5. Intersymbol dependencies of lucifer-type encryption algorithms.

	정 보				키 이			
라운드	32	64	128	256	32	64	128	256
1	9.38	4.69	2.34	1.17	3.03	1.54	0.78	0.39
2	36.13	19.92	10.05	5.06	14.55	8.65	4.53	2.31
3	75.39	53.21	32.98	18.64	43.66	27.20	16.87	9.54
4	97.07	87.89	68.25	48.60	78.62	54.37	46.47	30.25
5		100.0	93.36	81.83		77.64	77.89	66.66
6			100.0	97.79		89.80	93.53	93.73
7				100.0		95.61	98.75	99.70
8						99.17	99.76	100.0
9							100.0	

키이에 따라 각 의존도가 달라지므로 전체 라운드 수가 비교적 적은 32비트와 64비트 Lucifer들은 입력되는 키이에 따라서는 최대 100%의존도를 얻지 못할 경우도 있다.

3. 심볼간 상호의존도 개선

Lucifer 키이 interruption 위치는 대체과정 이후에 이루어져 키이비트에는 대체과정이 수행되지 못한다. 따라서 본 논문에서는 키이 interruption 위치를 대체과정 이전에 오게하여 키이비트에도 대체과정이 수행되게 함으로써 심볼간 상호의존도(특히 키이의존도)를 크게 개선시킬 수 있었다. 그러나 키이 interruption의 위치를 대체과정 이전으로 옮긴다 하더라도 Lucifer의 심볼간 상호의존도는 키이에 따라 변하게 된다.

그런데 좋은 키이를 선택한다면 암호화 알고리듬을 바꾸지 않더라도 심볼간 상호의존도를 높일 수도 있겠다. 그러나 이 방법은 키이를 선택하는데 노력과 시간이 많이 든다거나 큰 개선(심볼간 상호의존도가 첫번째로 100%가 되는 라운드가 줄어들 정도의 개선)을 얻지 못한다는 점등을 제쳐두고라도 좋은 키이만을 선택하여 쓴다는 것이 오히려 비도를 줄이는 치명적인 결과를 낳을 수도 있다. 따라서, Lucifer도 DES처럼 모든 키이에 대해 심볼간 상호의존도가 변하지 않고 일정하도록 알고리듬을 구성하여[2,5] 전 라운드가 끝나기 전에 100%의 상호의존도를 얻게 한다면 이 의존도는 모든 키이에 대해 믿을 수 있을 것이다.

이러한 특성을 지니게 하기 위해서는 Lucifer에서도 DES처럼 대체상자 입력 이전에 평어를 확장(expansion)시키고 각 대체상자가 4개의 대체표를 갖게 함으로써 키이가 바뀌어도 높은 상호의존도를 일정하게 유지할 수 있다. 그림 4는 위 설명과 같이 키이 interruption의 위치조정과 autoclave방식의 대체상자 입력등을 도입하여 Lucifer를 다시 구성한 알고리듬의 블록 선도이며, 표 6은 그림 4에 있는 암호화 알고리듬의 심볼간 상호의존도를 구한 것이다.

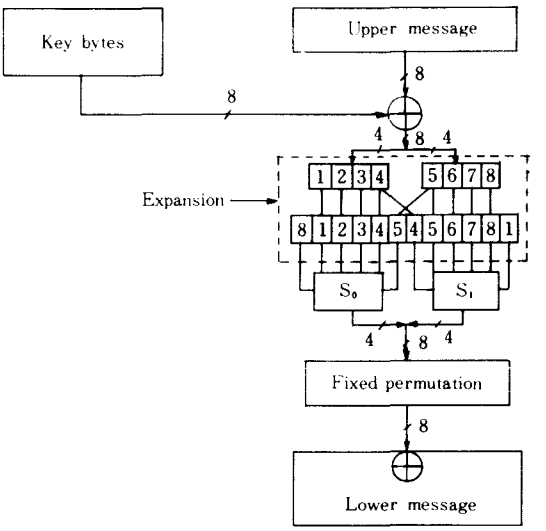


그림 4. 심볼간 상호의존도를 개선하는 수정된 암호화 알고리듬

Fig. 4. An advanced encryption algorithm that promote the intersymbol dependencies.

표 6. 개선된 심볼간 상호의존도
Table 6. Improved intersymbol dependencies.

	정 보				키 이			
라운드	32	64	128	256	32	64	128	256
1	12.50	6.26	3.13	1.56	9.38	4.69	2.35	1.17
2	43.75	25.88	19.39	9.72	38.28	28.52	18.51	9.38
3	81.25	70.51	55.57	37.79	71.10	65.24	53.42	37.27
4	98.43	95.41	88.53	76.71	92.19	90.82	85.55	75.44
5		100.0	99.61	97.27		99.42	98.24	96.21
6			100.0	100.0		100.0	99.95	99.83
7							100.0	100.0

위 표에서 볼 수 있듯이 이 절에서 설명된데로 알고리듬을 수정한다면, 수정 전 100%의 의존도를 얻지 못하던 64비트의 키이 의존도 경우, 수정 후에는 6라운드에서 100%의 의존도를 얻을 수 있다. 라운드당 평균적으로 정보의존도 4.04%, 키이 의존도 16.84% 개선되며 전체적으로 10.44%의 심볼간 상호의 존도의 개선을 얻을 수 있었다.

4. Exhaustive 암호분석

불법적인 암호분석을 막는 또 하나의 중요한 측도가 되는 것은 비도이다. 비도는 여러 측면에서 생각할 수 있으나 본 논문에서는 상당량의 평어와 해당 암호어를 가지고 가능한 모든 키이를 적용시켜 암호해독을 하는 exhaustive 암호분석시 평균적으로 소요되는 시간의 측면에서 비도를 조사하였다. 이는 한 블록을 암호화하는데 소요되는 시간에 평균적으로 암호해독에 적용되는 키이의 수를 곱한것이 된다. 이러한 비도는 암호해독에 사용되는 장비에 따라 달라질 수 있으나 어떤 장비에서 구한 비도인가를 안다면 현재 장비에 맞는 비도로 쉽게 환산할 수 있을 것이다.

Exhaustive 암호분석시, n비트 키이의 경우 반드시 2^n번의 시도를 할 필요는 없다. 물론 2^n번을 시도해야할 경우도 있으나 단 한번의 시도로 끝날수도 있다. 따라서 이 경우에는 비도를 랜덤 변수로 두고 이의 기대값($(1+2+\cdots+2^n)/2^n \times$한 블록 암호화시간)을 구하는 것이 타당하다 하겠다. 따라서 exhaustive 암호분석시 평균적으로 소요되는 시간 T_k는 식(3)으로 둘 수 있다. 여기서 k는 블록크기를 나타내고, 2^8과 2^{16}개의 키이를 계산에서 빼는 것은 이들이 각각 Lucifer의 weak key와 semi weak key[5]이기 때문이다.

$$T_k = \text{한 블록을 암호화하는 시간} \times 0.5 \times (2^k - 2^8 - 2^{16}) + 0.5 \quad (3)$$

표 7은 IBM-PC/XT호환기종(CPU : V20, CPU clock rate : 8.46MHz, OS : MS-DOS, Compiler : MSC5.0)과 IBM-PC/AT 호환기종 (CPU : i80286, CPU clock rate : 13.12MHz, OS : ZENIX-5, Compiler : CC)에서 식 3을 이용하여 계산한 시간 T_k이다.

표 7. Exhaustive 암호분석시 평균적으로 소요되는 시간
Table 7. The average time required for exhaustive attack.

(a) XT

크기[bit]		소 요 시 간[sec]			T_k
블록	파일	데이타전송	processing	합계	[year]
32	12288 ×8	19.5	16.5	36	0.798
64			34.5	54	1.028×10^{10}
128			71.5	91	6.393×10^{29}
256			144.5	164	7.841×10^{68}

(b) AT

크기[bit]		소 요 시 간[sec]			T_k
블록	파일	데이타전송	processing	합계	[year]
32	11744 ×8	7	11	18	0.417
64			24	31	6.177×10^{9}
128			47	54	3.969×10^{29}
256			95	102	5.102×10^{68}

표 7의 T_k는 암호해독자가 해독을 위하여 상당량의 평어와 해당 암호어 뿐만아니라 그 암호어가 몇 비트의 블록크기로 암호화 되었는지 알 경우의 것이다. 그러나 Lucifer 형태의 암호화 알고리듬으로 암호화된 암호어를 불법해독자가 입수를 하였더라도 몇 비트의 블록크기로 암호화 되었는지 알 수가 없으므로(비록 블록크기의 종류가 32, 64, 128, 그리고 256 비트라는 것을 알더라도) 그가 키이를 알아내기 위해서는 블록크기가 32비트부터 256비트까지의 모든 경우를 고려하여야 할 것이다. 그리고 표 7에서 정보의 중요도에 따라 암호화 알고리듬을 선택 적용할 수 있다.

V. 결 론

본 논문에서는 키이바이트 사용스케쥴과 확산치환의 패턴을 수정하여 32, 64, 그리고 256비트의 블록

크기를 가진 Lucifer 형태의 암호화 알고리듬들을 구성하고, 이들의 심볼간 상호의존도와 비도에 대해 비교 조사하였다.

라운드수가 비교적 적은 32비트와 64비트 Lucifer 형태의 암호화 알고리듬에서는 키이에 따라서는 최대 100%의 심볼간 상호의존도를 얻지 못할 경우도 있었다. 그러나 키이비트가 대체과정 수행에 영향을 미치도록 키이 interrution 위치를 대체상자 이전으로 옮겨 심볼간 상호의존도를 높이고, DES의 autoclave 방법을 적용시켜 키이의 선택에 무관하게 높은 의존도를 유지시킬 수 있었다.

정보의 중요도에 따라 선택할 수 있도록 각 블록크기의 Lucifer 형태 암호화 알고리듬의 exhaustive 암호분석 비도를 구하고 서로 비교하였다.

參 考 文 獻

[1] A. Sorkin, "Lucifer, a cryptographic algorithm," *Cryptologia*, vol.8, no.1, pp.22-35, Jan. 1984.

[2] National Bureau of Standards, Data Encryption Standard, U.S. FIFP PUB 46, pp. 1-18, 1977.

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Comm. ACM*, vol. 21, no. 2, pp. 120-6, 1986.

[4] L. Neuwith, "A comparison of four key distribution methods," *Telecommunication*, July 1986.

[5] C. Meyer and S. Matyas, *Crypography: A New Dimension in Computer Data Security*, New York, John Wiley Sons, 1982.

[6] 이훈재, 문상재, "LUCIFER와 DES에서의 심볼간 상호의존성에 관한 연구," 경북대학교 전자기술 연구지, 제 8 권, pp. 89-92, 8 월 1987.

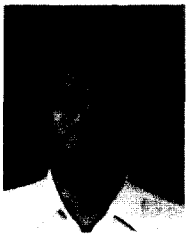
[7] E.F. Brickell and et. al., "Structure in the S-Boxes of the DES," extended abstract performed while the author was visiting Bell Communication Research. *

著 者 紹 介

姜 海 東(準會員)

1961年 10月 9日生. 1987年 2月 경북대학교 전자공학과 졸업. 1987年 3月～현재 경북대학교 대학원 전자공학과 석사과정. 주관심분야는 암호화 과정, 정보통신 등임.

●



李 昌 淳(正會員)

1958年 5月 5日生. 1981年 2月 경북대학교 전자공학과 졸업. 1983年 2月 경북대학교 대학원 전자공학과 석사학위 취득. 1986年 3月～현재 경북대학교 대학원 전자공학과 박사과정 재학중. 주관심분야는 컴퓨터 네트워크 및 암호화과정 등임.

文 相 在(正會員)

1948年 4月 20日生. 1972年 2月 서울대학교 공업교육과(전자전공) 공학사학위 취득. 1974年 2月 서울대학교 대학원 전자공학과 석사학위 취득. 1984年 6月 미국 UCLA 공학박사(통신공학전공) 학위 취득. 1974年 2月～1974年 10月 금성전기주식회사 근무. 1980年 10月～1984年 6月 미국 UCLA 연구조원 근무. 1984年 3月～1984年 7月 미국 Satellite Tech. Management Inc. 근무. 1984年 7月 ～1985年 7月 미국 UCLA Postdoctor 근무(Dept. of Elec. Eng.). 1984年 7月～1985年 7月 미국 OMNET 주식회사 Consultant 근무. 1974年 12月～현재 경북대학교 전자공학과 부교수. 주관심분야는 부호기술 및 디지틀통신 등임.