Journal of KIIT. Vol. 15, No. 11, pp. 77-86, Nov. 30, 2017. pISSN 1598-8619, eISSN 2093-7571 77
http://dx.doi.org/10.14801/jkiit.2017.15.11.77

박스 알고리즘을 활용한 새로운 암호화 방법

이상일*, 민승식**

New Encryption Method Using Box Algorithm

Sangil Lee*, Seungsik Min**

요 약

온라인 활동의 활성화로 인해 개인정보의 노출 건수는 해마다 늘어나고 있으며, 이로 인해 암호화 기술의 고도화가 나날이 요구되고 있다. 민간 분야의 암호화에 대한 관심은 1970년대부터 대두되기 시작했는데 DES와 AES 등 대표적인 암호화 표준을 마련하게 되었다. 이들 알고리즘의 복잡도는 매우 높은 것으로 여겨졌지만 DES의 경우 의외로 쉽게 크래킹(cracking) 된다는 사실이 밝혀졌다. AES나 SEED의 경우도 그 안전성이 지속될 수 없다. 본 논문에서는 한국형 암호 알고리즘의 방편으로 박스 알고리즘을 제안하고, 키의 길이, 복잡도, CPU 시간을 분석하였다. 박스 알고리즘은 키의 길이에 비해 복잡도가 약간 작고, 원문에 비해 암호문의 길이가 늘어나는 단점을 지니고 있다. 하지만 타 알고리즘에 비해 키의 길이 조절이 용이하여 복잡도를 원하는 수준으로 끌어올릴 수 있고, 원문에 비해 늘어난 암호문은 유일하지 않기 때문에 원문을 추정하기 힘들다는 장점을 지니고 있다. 키의 길이, 복잡도, CPU 시간을 바탕으로 한 알고리즘의 효율성을 분석한 결과 최적값을 나타내는 의미 있는 경우를 찾을 수 있었다.

Abstract

Due to the invigorating of online activities, the number of the exposures of personal information is increasing each year, and the encryption technology is required to be improved every day. Interest in cryptography for the private field started in the 1970s, and DES, AES, and other cryptographic standards were developed. Although the complexity of these algorithms was considered to be very high, it turned out to be surprisingly easy to crack in the case of DES. Moreover, the safety of AES or SEED cannot be sustained. In this paper, we proposed a box algorithm as a Korean cryptographic algorithm and analyzed key length, complexity, and CPU time of the suggested algorithm. The box algorithm has a disadvantage in that the complexity is slightly smaller compared to the key length and the length of the ciphertext increases compared to the original text. Compared to other algorithms, however, it is easy to control the length of a key, which can increase the complexity to the desired level, and the ciphertexts are not unique. We analyzed the efficiency of the algorithm based on the key length, complexity, and CPU time and found an optimized value for the special cases.

Keywords
encryption, decryption, box algorithm, permutation, key length, complexity, CPU time, efficiency

* 해군사관학교 무기체계공학과 교수
- ORCID: http://orcid.org/0000-0003-1881-9041
** 해군사관학교 이학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-8995-7030

· Received: Oct. 19, 2017, Revised: Nov. 15, 2017, Accepted: Nov. 18, 2017
· Corresponding Author: Seungsik Min
Department of Natural Science, Korea Naval Academy, Korea
Tel.: +82-55-969-5238, Email: fieldsmin@gmail.com

Ⅰ. 서 론

기가급 유무선 인터넷망의 구축과 PC 수준의 스마트기기 보급은 온라인을 이용한 자료전송을 더욱 활성화시켰고, 클라우딩 서비스의 등장은 장소에 구애받지 않고 어디에서나 개인이 저장한 자료에 접속할 수 있게 하였다. 또한 온라인 쇼핑몰 등의 전자상거래 보편화로 온라인상의 카드 결제 및 이체 서비스가 활성화되었다. 하지만 회원가입을 위한 인증절차로 진행되는 휴대폰 번호, 신용카드 번호나 주소 등의 민감한 정보가 노출 및 탈취되는 사태가 빈번하게 나타나고 있다. 개인정보는 해킹이나 내부자 소행 등의 방법으로 외부에 자주 노출되었는데, 인터넷의 한 비영리단체의 통계에 의하면, 1991년부터 2016년 6월까지 개인정보의 유출건수는 최소 7억 5천만 건에 이른다고 한다[1]. 정보화시대에서 개인과 기업체의 정보는 하나의 중요한 자산으로 볼 수 있으며, 이러한 자료가 유출되는 것은 개인과 기업 뿐 아니라 국가의 안보에도 위협이 될 수 있다. 이에 따라 중요한 정보를 보호하기 위한 암호학과 암호기술의 발전은 필수불가결한 요건이 되었다[2][3].

민간분야에 대한 암호화가 관심을 받기 시작한 것은 1970년대 미국표준연구소(NBS)가 암호시스템의 표준으로 DES을 배포하기 시작한 때부터이다. DES는 IBM에서 Horst Feistel의 Lucifer 암호화 방법을 기반으로 만든 것으로 56비트의 키를 이용해 64비트의 평문을 암호화하는 대칭형 암호화 방법이다[4][5]. 초기 DES는 56비트 키를 사용했기 때문에 2^{56} 가량의 복잡도(Complexity)를 가지고 있다고 여겨졌으나, 1998년 EFF(Electronic Frontier Foundation)에서 만든 EFF DES cracker에 의해 단기간 해킹(56시간)이 성공하면서 더 이상 안전하지 않다고 여겨지게 되었다[6].

이후 안전한 암호를 만들기 위해 암호의 복잡도 증가를 위한 노력과 함께 크래킹(Cracking)을 위한 많은 노력이 이루어졌다. 1993년 Matshi는 DES의 평문에 대한 복잡도는 2^{47}이며 2001년 Junod는 DES의 시간 복잡도가 $2^{39\sim43}$라고 발표하였다[7][8]. DES의 복잡도가 기대했던 것보다 낮게 나타남으로 인해 최근에는 이러한 안전성 문제를 해결하기 위해 DES 알고리즘을 2~3개의 키를 이용하여 3번 실행하는 방식인 Triple DES 방식으로 대체하여 사용하고 있다. 더불어 Alvarez와 Li는 무차별 공격에 대응한 안전성 확보를 위해서 최소한 2^{100}이상의 복잡도를 갖는 키 사용을 권고하고 있다[9].

1977년에는 DES를 대체하기 위해 AES를 개발하게 되었는데, 이는 Rihndael로도 알려져 있다. AES는 128비트의 블록을 암호화하기 위해서 128, 192, 256비트의 키를 사용하며, 미국 정부뿐 아니라 전세계에 범용으로 사용되고 있다. AES는 DES보다 복잡도가 증가하였으며 키 길이와 비슷한 복잡도를 갖는 것으로 확인되었다. 즉, 비클릭 공격(Biclique Attack)에 대한 AES-128, 192, 256의 계산 복잡도는 $2^{126.1}$, $2^{189.7}$, $2^{254.4}$ 라고 알려져 있다. 하지만 연관키 공격(Related-key Attack)에 대해서는 AES-192, 256의 계산 복잡도가 2^{126}으로 비슷하게 나타나는 것으로 알려져 있다[10].

한편, SEED-128는 1999년 한국정보보호센터(현 한국정보보호진흥원)에서 개발한 블록 암호알고리즘이다. SEED-256은 기존 SEED-128이 128비트 키만 사용하여 AES 등의 국외 암호 알고리즘보다 활용도가 떨어진다고 생각되어 256비트의 키를 지원할 수 있도록 개발한 블록알고리즘이다. SEED-256은 Fiestel 구조로 이뤄져 있으며, 128비트의 평문 블록과 256비트의 키를 입력으로 사용하여 24라운드를 거쳐 128비트 암호화된 블록을 출력한다[11]. 차분공격(Differential Attack)을 이용한 SEED-128의 계산 복잡도는 2^{122}임이 알려져 있다[12]. SEED-128은 AES-128보다 약간 낮은 복잡도를 가지는데, 해킹에 대한 안정성을 높이기 위해 보다 높은 복잡도를 지니는 국내 암호알고리즘이 필요하다고 할 수 있다.

본 논문에서는 이러한 한국형 암호 알고리즘의 발전을 위한 방편으로 박스 알고리즘을 활용한 새로운 암호화 방법을 제안한다. 박스 알고리즘은 동일한 키를 사용하더라도 암호문이 다양하게 나타나며, 암호화된 내용을 동일한 평문으로 복호화를 할 수 있는 특징을 가진다. 기존 암호화 방식에서는 평문-암호문-복호문의 대응 관계가 1－1－1이었다면,

박스 알고리즘에서는 $1-n-1$의 대응 관계가 성립한다.

알고리즘을 설명하기 위해 2장에서는 박스 암호리즘의 두 가지 절차인 박스화 과정(Boxing Procedure)과 순열 과정(Permutation Procedure)에 대해 설명하고, 3장에서는 본 알고리즘을 통해 나타나는 키 길이, 복잡도, 효율성에 대해 알아본다. 4장에서는 박스 알고리즘을 바탕으로 하여, 키 길이, 복잡도, 암호화를 위한 CPU 시간, 효율성 등에 대한 결과분석 및 논의를 진행한다. 마지막으로 5장에서는 박스 암호화(Box Encryption) 알고리즘을 기존의 AES, SEED와 키 길이, 복잡도 및 CPU 시간 측면에서 비교하여 장단점에 대해 언급하고 향후 발전 방향에 대해서 제시하도록 한다.

II. 박스 암호화 과정

암호화의 목적은 송수신자 사이에서 정보를 편취하고자 하는 사람들로부터 메시지를 보호하는 데에 있다. 이를 위해 암호화 알고리즘은 암호 해독의 복잡도를 높여 제삼자에 의한 복호화 가능성을 낮추도록 발전해 왔다. 박스 암호화 방법도 복잡도를 높이는데 목적을 두고 개발하였다. k개의 박스 안에 n개의 공이 나누어 담긴 경우를 생각해 보면, 박스와 공은 일대다대응을 하게 된다. 따라서 공에 적힌 숫자만 가지고 그 공이 어느 박스 있었는지 추정하는 데에는 많은 시간이 요구된다. 여기에 착안하여 우리는 평문의 개별 데이터를 박스에 대응시키고, 그 박스 안에 담긴 공을 무작위 추출하여 나온 숫자로 암호화되도록 박스 알고리즘을 고안하였다.

본 알고리즘은 크게 두 가지 과정을 거친다. 첫째는 박스화 과정으로, n개의 숫자($n \geq k$)를 k개의 구역으로 구분하고 각 구역 번호에 대응되는 평문을 그 구역에 속한 숫자들 중에서 임의 선택된 것으로 변환시키는 과정이다. 둘째는 순열 과정으로, 박스화 과정으로 변환된 원시 암호문을 이차원 행렬화하여 표현한 후 이들의 행과 열을 섞어주는 과정이다.

모든 데이터는 이진화하여 표현할 수 있으므로 우리는 평문이 이진화된 숫자인 것으로 가정하였다. 가령 $k=16$, $n=256$인 경우, 이진화된 평문을 4비트($\lceil \log_2 16 \rceil = 4$) 단위로 구분하며, 이를 8비트($\lceil \log_2 256 \rceil = 8$) 암호문으로 변환하는 것이다. 여기서 $\lceil x \rceil$는 x를 자연수가 되도록 올림 연산한 결과를 의미한다. $\lceil \log_2 k \rceil$ 비트 단위의 평문을 $\lceil \log_2 n \rceil$ 비트 단위의 암호문으로 변환하는 것은 k개의 숫자 $\{0, 1, ..., k-1\}$를 n개의 숫자 $\{0, 1, \cdots, n-1\}$에 대응시키는 것으로 평문에 비해 암호문의 길이가 늘어나는 단점이 있는 반면, 동일한 평문에 대해 암호문이 다양하게 나타날 수 있다는 장점이 있다. k와 n의 선택은 결과적으로 암호해독의 복잡도, 암호화하는데 걸리는 시간(CPU Time), 그리고 키의 길이를 결정하는 가장 중요한 요인이 된다.

2.1 박스화 과정

박스화 과정은 다음과 같이 4개의 과정을 거친다.

- step 1. Select $k-1$ bound positions for k regions
- step 2. Sort the bound positions to decide the range of k regions
- step 3. Select a number to decide the number of the last region, and subtract 1 to the number of regions less than or equal to the last number.
- step 4. Choose a number randomly in a region, then the number is the cipher message and the regional number is the original message.

각 과정에 대한 세부적인 설명은 다음과 같다.

step 1은 k개의 구역을 정하기 위해 $k-1$개의 경계값을 설정하는 과정이다. n개의 숫자를 무작위로 k개의 박스에 나누어 담는다면 그 정보를 키에 일일이 지정해 주어야 하므로 키의 길이가 매우 길어질 것이다. 이를 피하기 위해 $\{0, 1, 2, ..., n-1\}$의 집합 A를 k개의 구역으로 구분되도록 $k-1$개의 경계값(Bound Position)만을 설정한다. 그림 1의 (a)는 $k=16$이고, $n=256$일 경우 $k-1$에 해당하는 15개의 경계값 정보가 선택된 예시를 나타낸다. 각 숫자에 해당하는 경계값 정보는 키에 저장되며 $k-1$개의 경계값은 각각 8비트($\lceil \log_2 n \rceil = 8$)의

크기를 가지게 된다.

step 2는 선택된 $k-1$개의 경계값을 오름차순 정렬 후 구간을 결정한다. 그림 1의 (b)의 예시에서 평문 13에 대응되는 암호문의 구간이 0~7이고, 12에 대응되는 암호문의 구간이 8~11인 것을 알 수 있다. 하지만 아직 암호문의 마지막 구간인 249~255에 대응되는 평문의 숫자는 결정되지 않은 상태이다.

step 3은 마지막 구간의 숫자를 결정하는 과정이다. 이 때 선택된 숫자가 0이 아니라면 기존의 숫자(1, 2, ..., $k-1$)와 겹치게 된다. 이를 피하기 위해 선택된 숫자보다 작거나 같은 기존의 숫자들은 1씩 뺀 값을 최종적으로 취하게 된다. 그림 1의 (c)에서처럼 선택된 숫자가 8일 경우 기존에 선택된 1, 2, ..., 8 의 숫자들은 0, 1, ..., 7 로 변환된다. 그러면 평문의 숫자에 대응되는 암호문의 구간이 최종적으로 결정된다.

step 4는 이들 구간에 속한 값들을 임의 선택하는 과정으로 원시 암호문이 결정된다.

(a) Correspond the bound positions in the key to each digit

digit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bound position	95	229	223	245	160	51	248	224	181	46	165	11	7	206	85

(b) Sort the bound positions to decide the range of numbers for each digit

digit	13	12	10	6	15	1	5	11	9	14	3	8	2	4	7	-
bound position	7	11	46	51	85	95	160	165	181	206	223	224	229	245	248	-
range of number	0~7	8~11	12~46	47~51	52~85	86~95	96~160	161~165	166~181	182~206	207~223	224~224	225~229	230~245	245~248	249~255

(c) Select a number of the last region among 0 to k-1 (this case 8 is selected), then the digits lower than or equal to the selected number are substracted by 1. (1,2, ..., 8 are changeed to 0,1,...,7 for this example)

digit	13	12	10	5	15	0	4	11	9	14	2	7	1	3	6	8
bound position	7	11	46	51	85	95	160	165	181	206	223	224	229	245	248	-
range of number	0~7	8~11	12~46	47~51	52~85	86~95	96~160	161~165	166~181	182~206	207~223	224~224	225~229	230~245	245~248	249~255

그림 1. Boxing 절차 설명을 위한 예시

Fig. 1. Examples to explain the boxing procedure

(a) A sending message converted to k digits (in this case, k=16 and 0~15 digits are used)

Original Message (decimal)	15	1	12	14	14	2	15	0	3	9	9	11
Original Message (binary)	1111	0001	1100	1110	1110	0010	1111	0000	0011	1001	1001	1011
Encoded Message (decimal)	54	228	10	182	199	220	83	88	234	170	178	164
Encoded Message (binary)	00110110	11100100	00001010	10110110	11000111	11011100	01010011	01011000	11101010	10101010	10110010	10100100

(b) Binary data is formed to a matrix then cutted by l×8 (l=4 for this case) and then row and column are permuted. (For the row: 1,2,3,4→3,1,4,2; For the column: 1,2,3,4,5,6,7,8→7,1,8,2,4,5,3,6)

Matrix Formed Binary Data

	1	2	3	4	5	6	7	8
1	0	0	1	1	0	1	1	0
2	1	1	1	0	0	1	0	0
3	0	0	0	0	1	0	1	0
4	1	0	1	1	0	1	1	0
5	1	1	0	0	0	1	1	1
6	1	1	0	1	1	1	0	0
7	0	1	0	1	0	0	1	1
8	0	1	0	1	1	0	0	0
9	1	1	1	0	1	0	1	0
10	1	0	1	0	1	0	1	0
11	1	0	1	1	0	0	1	0
12	1	0	1	0	0	1	0	0

Row Permutation

	1	2	3	4	5	6	7	8
3	0	0	0	0	1	0	1	0
1	0	0	1	1	0	1	1	0
4	1	0	1	1	0	1	1	0
2	1	1	1	0	0	1	0	0
7	0	1	0	1	0	0	1	1
5	1	1	0	0	0	1	1	1
8	0	1	0	1	1	0	0	0
6	1	1	0	1	1	1	0	0
11	1	0	1	1	0	0	1	0
9	1	1	1	0	1	0	1	0
12	1	0	1	0	0	1	0	0
10	1	0	1	0	1	0	1	0

Column Permutation

	7	1	8	2	4	5	3	6
3	1	0	0	0	0	1	0	0
1	1	0	0	0	1	0	1	1
4	1	1	0	0	1	0	1	1
2	0	1	0	1	0	0	1	1
7	1	0	1	1	1	0	0	0
5	1	1	1	1	0	0	0	1
8	0	0	0	1	1	1	0	0
6	0	1	0	1	1	1	0	1
11	1	1	0	0	1	0	1	0
9	1	1	0	1	0	1	1	0
12	0	1	0	0	0	0	1	1
10	1	1	0	0	0	1	1	0

그림 2. 메시지의 2진수 변환과 행과 열의 순열 방법

Fig. 2. Examples to show the message conversion to a binary form and row/column permutation

2.2 순열 과정

본 방법론에서는 복호화의 복잡도를 더욱 높이기 위하여 행과 열을 각각 뒤섞는 순열 과정을 추가로 실시한다. $k=16$이고, $n=256$인 경우 박스화 과정을 통해 이진화하여 표현된 원시 암호문은 8비트 단위의 길이를 가지게 된다. 그림 2의 (a)는 $\{15, 1, \cdots, 11\}$로 구성된 평문을 $\{54, 228, \cdots, 164\}$의 원시 암호문으로 변환한 예시이다. 이를 이진수로 표현화면 4비트 단위의 평문이 8비트 단위의 암호문으로 변경된 것을 확인할 수 있다. 그림 2의 (b)는 8비트 단위의 이진수 암호문을 세로로 나열하여 {문자의 수}×8 크기의 행렬 형태로 나타낸 것이다.

우선 행 순열(Row Permutation)은 임의의 행 개수에 대해 순번을 교환하는 작업을 수행한다. 그림 2의 (b)는 $l=4$인 경우를 예로 든 것이다. 4개의 암호문 단위로 (1, 2, 3, 4)→(3, 1, 4, 2)의 순열 과정을 거치는 것을 확인할 수 있다. 만일 l의 개수를 메시지 송수신자가 알고 있다면 마지막 순열 (4)→(2)는 저절로 결정된다. 따라서 키에 포함되는 행 순열의 정보는 (3, 1, 4)로 충분하다. 이 경우 각각의 행 순열 정보를 이진화하여 표현할 경우 $\lceil \log_2 l \rceil = 2$비트가 필요하므로 l은 2의 거듭제곱인 경우 행 순열 과정에 필요한 키의 길이를 절약할 수 있다.

열 순열(Column Permutation)의 경우도 비슷하게 진행된다. 8비트($n=256$) 암호문을 선택하였으므로 열 순열을 위해 7개의 키가 필요하다. 그림 2의 (b)는 (1, 2, 3, 4, 5, 6, 7, 8) → (7, 1, 8, 2, 4, 5, 3, 6)인 순열을 나타내고 있다. 따라서 키에 포함되는 열 순열의 정보는 (7, 1, 8, 2, 4, 5, 3)의 7개가 된다. 행 순열과 마찬가지로 각각의 숫자 정보를 이진화하여 표현할 경우 $\lceil \log_2 \lceil \log_2 n \rceil \rceil$ 비트가 필요하므로 $\lceil \log_2 n \rceil$이 2의 거듭제곱인 경우 키의 길이를 절약할 수 있다. 즉 n이 2의 거듭제곱의 거듭제곱인 경우 키의 길이를 절약할 수 있는 것이다. 만일 $2^8 = 256 < n \le 512 = 2^9$인 경우($\lceil \log_2 n \rceil$이 2의 거듭제곱이 아닌 경우)를 예로 들면, 박스화 과정에서 경계값을 9비트로 표현해야 한다. 그러면 9비트로 이진화된 원시 암호문의 열 순열은 $\lceil \log_2 9 \rceil = 4\,(= \lceil 3.169... \rceil)$비트 크기의 숫자 8개가 필요할 것이다. 또 다른 예로 $512 < n \le 1024$이면 박스화 과정의 경계값을 10비트로 표현해야 한다. 위의 경우와 마찬가지로 이진화된 원시 암호문의 열 순열은 $\lceil \log_2 10 \rceil = 4$비트 크기의 숫자 9개가 필요할 것이다. 앞의 예처럼 열 순열을 4비트로 표현할 수 있는 n의 값은 257부터 65,636까지 가능하므로($\lceil \log_2 \lceil \log_2 n \rceil \rceil = 4$), 열 순열을 4비트로 표현하면서도 n의 값을 65,536 미만으로 설정하는 것은 키의 길이 낭비가 발생하는 것이다. 따라서 이상적인 경우는 n의 값이 $2^{2^1}=4$, $2^{2^2}=16$, $2^{2^3}=256$, $2^{2^4}=65{,}536$ 등인 경우이다. 하지만 4나 16은 너무 작은 숫자이므로 복잡도가 낮아질 것이고, 65,536 이상인 숫자들은 너무 큰 값이라 암호화하는 데에 매우 많은 시간이 소모될 것이므로 $n=256$은 가장 이상적인 선택이라 할 수 있다. 따라서 본 논문에는 $n=256$으로 고정하여 기술할 것이다.

III. 박스 암호화의 주요 특성

3.1 키 길이

박스화 과정과 순열 과정을 통한 박스 알고리즘의 정보는 키에 포함되어 전달되며, 수신자는 공유된 키 정보를 사용하여 암호화된 메시지를 복호화한다. 암호화는 $\lceil \log_2 k \rceil$ 비트 단위의 평문을 $\lceil \log_2 n \rceil$ 비트 단위의 암호문으로 변환한 후, 행 순열 및 열 순열 과정을 거친다. 이 때 키의 길이는 1) $k-1$개의 경계값 정보, 2) 마지막 구간의 숫자 정보, 3) 행 순열 정보, 마지막으로 4) 열 순열 정보에 의해 결정된다. 위 4가지 항목에 대한 세부적인 키의 길이는 다음과 같이 계산된다.

1) $k-1$ bounds : $(k-1)\lceil \log_2 n \rceil$ [bit]
2) last digit : $\lceil \log_2 k \rceil$ [bit]
3) row permutation : $(l-1)\lceil \log_2 l \rceil$ [bit]
4) column permutation
: $(\lceil \log_2 n \rceil - 1)\lceil \log_2(\lceil \log_2 n \rceil) \rceil$ [bit]

예를 들어 $n=256$, $k=16$인 경우 15개의 경계값은 8비트의 단위 길이가 필요하므로 총 120비트의 경계값 정보가 필요하다. 또한 마지막 구간의 숫자는 $k=16$이므로 4비트가 필요하다. 한편, 행 순열을 위해서는 l 이내의 숫자로 구성된 $l-1$개의 $\lceil \log_2 l \rceil$ 비트 순열 정보가 필요하다. 비슷한 방법으로 열 순열을 위해서는 $\lceil \log_2 n \rceil$ 이내의 숫자로 구성된 $\lceil \log_2 n \rceil - 1$개의 $\lceil \log_2 \lceil \log_2 n \rceil \rceil$ 비트 순열 정보가 필요하다. 그러면 전체 키는 앞에서 언급한 4가지 항목의 키를 차례대로 나열한 값이 되며, 길이는 다음과 같이 표현된다.

$$\begin{aligned} &Key\ Length \\ &= (k-1)\lceil \log_2 n \rceil \\ &\quad + \lceil \log_2 k \rceil + (l-1)\lceil \log_2 l \rceil \\ &\quad + (\lceil \log_2 n \rceil - 1)\lceil \log_2(\lceil \log_2 n \rceil) \rceil \end{aligned} \tag{1}$$

일반적으로 k와 n은 2의 거듭제곱으로 나타내는 것이 자연적이므로(ASCII, UTF 코드 등의 문자열은 모두 2의 거듭제곱만큼의 개수를 가진다.) 키 길이는 다음과 같이 간략하게 나타낼 수도 있다.

$$\begin{aligned} &Key\ Length \\ &= (k-1)\log_2 n + \log_2 k + (l-1)\log_2 l \\ &\quad + (\log_2 n - 1)\lceil \log_2(\log_2 n) \rceil \end{aligned} \tag{2}$$

3.2 해킹 복잡도(Hacking Complexity)

암호화 알고리즘에서 복호화에 대한 복잡도는 보안성의 가장 중요한 요인이다. 박스 알고리즘에서의 복잡도는 1) $k-1$개의 경계값을 구하는 경우의 수, 2) 마지막 구간의 숫자를 결정하는 경우의 수, 3) 행 순열의 수, 마지막으로 4) 열 순열의 수를 구한 후 모두 곱하여 구하게 된다. 위 4가지 항목에 대한 세부적인 복잡도(경우의 수)는 다음과 같이 계산된다.

1) $k-1$ bounds : ${}_{n-1}P_{k-1}$
2) last digit : k
3) row permutation : $l \times (l-1) \times \cdots \times 1 = l!$
4) column permutation : $\lceil \log_2 n \rceil !$

박스화 과정에서는 n개의 숫자를 k개의 박스에 나누어 담는 경우의 수는 일렬로 나열된 n개의 숫자를 $k-1$개의 경계선으로 구분하는 경우의 수와 같다. n개의 숫자 사이 경계 $n-1$개 중 $k-1$개를 순서를 고려하여 뽑는 경우의 수는 ${}_{n-1}P_{k-1}$이 된다. 또한 마지막 구간의 숫자는 0에서 $k-1$ 중에서 선택될 수 있으므로 k가지의 경우의 수가 있다. 한편, 길이 l의 행 순열은 $l!$, 길이 $\lceil \log_2 n \rceil$의 열 순열은 $\lceil \log_2 n \rceil !$의 경우의 수를 가진다. 이를 바탕으로 각 과정의 경우의 수를 곱하면 다음과 같이 해킹 복잡도를 구할 수 있다.

$$\begin{aligned} &Hacking\ Complexity \\ &= {}_{n-1}P_{k-1} \times k \times l! \times \lceil \log_2 n \rceil ! \\ &= {}_{n-1}C_{k-1} \times k! \times l! \times \lceil \log_2 n \rceil ! \end{aligned} \tag{3}$$

3.3 효율성

암호 알고리즘의 효율성은 여러 측면에서 살펴볼 수 있다. 일반적으로 키의 길이가 q비트라면 가질 수 있는 정보의 최대량은 2^q이 된다. 따라서 복잡도는 2^q을 넘어설 수 없다. 이러한 관점에서 $complexity/2^q$ 값은 주어진 길이의 키가 주어질 때, 허용된 정보량에 대한 전송되는 정보량의 비율이 되므로 알고리즘의 성능을 측정하는 하나의 지표가 된다. 하지만 비용 측면에서 보면 키의 길이보다는 메시지를 암호화하는데 걸리는 시간이 보다 중요할 것이다. 한편 보안성을 높이기 위해서는 복호화에 대한 복잡도가 높은 알고리즘이 선호될 것이다. 따라서 본 절에서는 암호 알고리즘의 효율성을 다음과 같이 정의한다.

DEFINITION

$$Efficiency = \frac{\log_2(Complexity)}{\{Key\ Length\} \times \log_2(CPU\ time)} \tag{4}$$

암호화하는데 걸리는 시간은 컴퓨터의 성능과 암호화 프로그램에 따라 다소 차이가 나겠지만 동일한 환경 하에서 여러 경우의 결과들을 상대 비교하여 알고리즘의 우열을 판단할 수 있을 것이다.

Ⅳ. 결과 분석 및 토의

4.1 키 길이와 복잡도

2장에서 설명한 것과 같이, 박스 알고리즘은 이진화한 평문을 임의의 길이로 구분한 후, 다른 길이로 암호화하는 것이 가능하다. 우리는 $\log_2 k$비트 $(1, 2, 3, ..., 8\,bit)$ 단위의 평문을 $\log_2 n$비트 단위의 암호문으로 변경하고, 길이 l의 행 순열과 길이 $\log_2 n$의 열 순열을 시행한 결과를 기술할 것이다.

2.2절에서 키의 길이를 최적화 시키는 n의 값은 2의 거듭제곱의 거듭제곱인 경우($2^{2^1}=4$, $2^{2^2}=16$, $2^{2^3}=256$, $2^{2^4}=65{,}536$)이므로 암호화 조건을 만족하는 n의 값은 256이 이상적임을 설명하였다. 따라서 우리는 n을 256으로 고정시키고, k와 l의 값만을 2의 거듭제곱으로 변화시켜 가며 키 길이와 복잡도 변화를 분석하였다.

표 1은 기준 수 k 값과 순열 길이 l 값의 변화에 따른 키 길이의 변화를 나타낸 표이고, 이를 k와 l에 대해 별도의 경향성을 살펴본 것이다. 표 1에서 $k=2^8$인 경우는 n의 값과 같으므로 박스 알고리즘은 사실상 일대일대응을 이루는 변환을 거친다. 표 2는 기준 수 k 값과 순열 길이 l 값의 변화에 따른 복잡도 변화를 나타낸 표이다. 이 때 복잡도는 2를 밑으로 하는 로그값을 취하였다. 표에서 추정할 수 있듯이 키의 길이와 복잡도는 k와 l에 대해 비교적 정확한 선형성을 나타낸다. 즉, k와 l의 값이 커질수록 그에 비례하는 키의 길이와 복잡도를 지니는 것이다.

표 1과 2는 비슷한 경향성을 지니는데, 이는 3.3절에서 언급한 바와 같이 주어진 길이의 키에 저장할 수 있는 최대 정보량 대비 실제 저장된 정보량의 비가 0.8~0.95 범위에서 유지되는 것을 의미하며 박스 알고리즘이 안정된 성능을 지니고 있음을 보여주는 것이다.

4.2 암호화 및 복호화 시간

박스 알고리즘의 성능을 추정하기 위해 메시지 길이 변화에 따른 암/복호화 시간을 측정하였다. 본 실험에는 i5-4570 3.2GHz Quad Core CPU, 4GB 메모리 성능의 사무용 컴퓨터를 이용하였으며, 암호화를 위한 프로그램은 매트랩(Matlab) 2012를 사용하였다.

표 3은 메시지의 용량을 1MB로 고정하였을 때 기준 수 k 값과 순열 길이 l 값의 변화에 따른 암호화 및 복호화 시간을 나타낸 표이다. 메시지 용량이 고정되어 있으므로 기준 수 k 값이 커지면 암호화하는 문자의 수는 반비례하여 줄어든다는 점에 유의해야 한다.

표 1. 기준 수 k, 순열길이 l에 따른 키 길이 변화
Table 1. Key length of changing the base k and permutation length l with $n=256$

k \ l	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
2^1	31	36	51	90	185	408	919	2070
2^2	48	53	68	107	202	425	936	2087
2^3	81	86	101	140	235	458	969	2120
2^4	146	151	166	205	300	523	1034	2185
2^5	275	280	295	334	429	652	1163	2314
2^6	532	537	552	591	686	909	1420	2571
2^7	1045	1050	1065	1104	1199	1422	1933	3084
2^8	2070	2075	2090	2129	2224	2447	2958	4109

표 2. 기준 수 k, 순열길이 l에 따른 복잡도 변화
Table 2. Hacking complexity of changing the base k and permutation length l with $n=256$ (log-scaled value by log2[complexity])

k \ l	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
2^1	25.3	28.9	39.6	68.5	142.0	320.3	740.5	1708
2^2	42.3	45.9	56.6	85.5	158.9	337.3	757.4	1725
2^3	75.1	78.7	89.4	118.4	191.8	370.1	790.3	1758
2^4	139.6	143.2	153.9	182.9	256.3	434.6	854.8	1823
2^5	266.4	270.0	280.7	309.6	383.0	561.4	981.5	1949
2^6	513.9	517.4	528.2	557.1	630.5	808.9	1229	2197
2^7	983.1	986.7	997.4	1026	1100	1278	1698	2666
2^8	1700	1704	1715	1744	1817	1995	2415	3383

표 3. 기준 수 k, 순열길이 l에 따른 암/복호화 시간
Table 3. CPU time in second of changing the base k and permutation length l with $n=256$ for 1MB message

k \ l	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
2^1	59.8	53.1	49.2	46.7	45.4	45.4	46.1	45.6
2^2	30.2	26.4	24.1	23.5	22.3	22.5	22.6	23.3
2^3	20.7	17.4	16.5	15.6	15.1	15.2	15.1	15.3
2^4	15.5	13.4	12.6	12.4	11.9	11.7	12.0	11.7
2^5	12.9	11.4	10.8	10.2	10.2	10.0	10.1	10.3
2^6	12.1	10.6	10.0	9.4	9.5	9.9	9.4	9.4
2^7	12.3	11.1	10.5	10.4	10.1	10.3	10.3	9.8
2^8	15.1	13.7	14.4	13.4	12.5	12.5	12.4	12.6

따라서 k가 늘어나면 오히려 CPU 시간이 줄어들 수 있는 것이다. 각각의 k와 l 값에 따라 암호화 및 복호화에 걸리는 시간은 9.4 ~ 59.8초 범위를 가지고 있는 것을 확인할 수 있다. 흥미로운 점은 기준 수가 늘어감에 따라 소모 시간의 극소값들이 존재한다는 것이다. 주어진 메시지를 암호화시키는데 걸리는 시간은 곧 비용과 관련되는데, $n=256$인 경우 $k=64(=2^6)$일 때 가장 적은 시간으로 암호화가 가능하다는 사실을 알 수 있다. l에 대해서는 $16(=2^4)$ 이상인 경우 뚜렷한 감소를 나타내지는 않고 있다. 따라서 고려 대상이 오직 소모 시간일 경우에는, $k=64$, $l \geq 16$인 값을 택해 암호화할 경우 가장 적은 시간에 암호화 및 복호화를 시행할 수 있는 것이다.

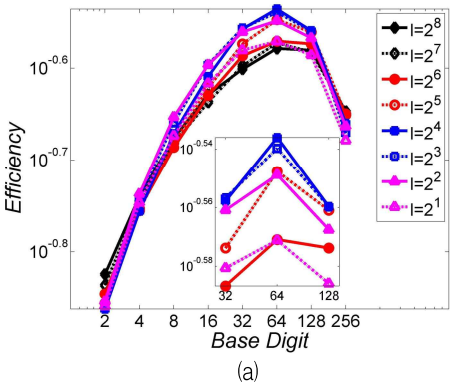
4.3 효율성

암호 알고리즘의 성능은 여러 가지 기준으로 측정할 수 있을 것이다. 우리는 3.3절 식 (4)~(6)의 정의를 바탕으로 앞서 측정한 키의 길이와, 복잡도, CPU 시간을 조합하여 효율성을 계산하였다. 키의 길이는 정보 전송의 효과성을, 복잡도는 해킹에 대비한 안정성을, 그리고 CPU 시간은 암호화 비용의 경제성과 관련되어 있다.

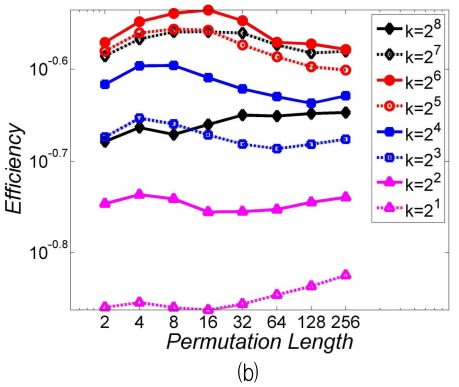
표 4 및 그림 3은 3.3절의 식 (4)에서 정의한 것과 같이 키의 길이와 CPU 시간 대비 복잡도를 바탕으로 한 효율성을 나타낸 표와 그래프이다.

표 4. 1MB 메시지를 n=256을 사용하여 암호화 할 때 기준수와 순열길이에 따른 효율성 비교
Table 4. Efficiency of changing the base k and permutation length l with $n=256$ for 1MB message

k \ l	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
2^1	0.138	0.140	0.138	0.137	0.139	0.143	0.146	0.150
2^2	0.179	0.183	0.181	0.175	0.176	0.177	0.180	0.182
2^3	0.212	0.222	0.219	0.213	0.208	0.206	0.208	0.211
2^4	0.242	0.253	0.253	0.246	0.239	0.235	0.231	0.235
2^5	0.263	0.275	0.278	0.277	0.267	0.259	0.253	0.251
2^6	0.268	0.283	0.289	0.291	0.283	0.269	0.267	0.264
2^7	0.260	0.271	0.276	0.276	0.275	0.267	0.262	0.263
2^8	0.210	0.217	0.213	0.219	0.224	0.223	0.225	0.225



(a)



(b)

그림 3. 기준 수와 순열 길이에 따른 효율성 비교
Fig. 3. Plot of the efficiency as functions of base digit and permutation length

그래프를 보면 모든 l에 대해 $k=64(=2^6)$일 때 효율성이 극대화 되는 것을 알 수 있다. 이 때 l의 값은 2^3~2^4 범위일 때 효율성이 높게 나타난다. 따라서 우리의 박스 알고리즘은 $n=256$, $k=64$, $l=2^3$~2^4인 값을 선택하는 것이 효율성을 높이는 합리적인 선택이라 할 수 있다.

V. 결론 및 향후과제

미국 등 세계 여러 나라에서 암호화 표준으로 채택하고 있는 AES와 한국의 암호화 표준인 SEED 등 여러 암호화 표준들이 점차 해킹으로부터 위협을 받고 있는 현 상황에서 우리는 박스 알고리즘을 제시하고 그 특성들을 분석하였다.

박스 알고리즘은 박스화 과정과 순열 과정을 거쳐 암호문을 만든다. 박스화 과정에서 평문은 원시 암호문으로 변환되는데, 이 때 동일한 평문과 키를

가지고서도 암호문은 무작위 변형된 형태로 나타나며, 다시 복호화할 경우 평문으로 변환된다 ($1-n-1$ 대응). 알고리즘의 복잡도를 더욱 높이기 위해 행렬 형태의 원시 암호문은 행 순열과 열 순열을 거친다.

박스 알고리즘의 장점은 1) 복잡도, 2) 계산 시간, 3) 효율성 측면에서 설명할 수 있다.

박스 알고리즘에서 복잡도는 키의 길의에 비례하여 증가하며, 키 길이 조절에 따른 복잡도 조절이 가능한 특징을 지닌다. 키 길이와 복잡도는 매우 유사한 패턴을 가지는데, 이는 본 알고리즘의 효과성이 일정하게 유지되는 것을 의미한다(표 1 및 2). 비클릭 공격에 대한 복잡도/키의 길이 관계는 AES-128에서 126.1/128, AES-192는 189.7/192, AES-256이 254.4/256으로 나타났고, 차분 공격에 대해서는 SEED-128이 122/128로 나타났다[10][12]. AES가 98.5% 이상, SEED도 95% 이상인 셈이다. 한편, 우리의 알고리즘은 $k=4$, $l=2$일 때 143.2/151이었고, $k=4$, $l=3$일 때 153.9/166, $k=5$, $l=2$일 때 270/280, $k=5$, $l=3$일 때 280/295였다. 단순히 복잡도/키의 길이 비만 보면 박스 알고리즘이 92.5~96.5% 정도로 SEED와 비슷하고, AES보다는 낮은 것처럼 보인다. 하지만 암호화 비용 측면에서 키의 길이는 크게 중요하지 않을 뿐만 아니라 박스 알고리즘은 k와 l 값의 조절을 통해 키의 길이를 자유롭게 바꿀 수 있기 때문에 원하는 복잡도를 얻기에 충분하리라 판단된다.

CPU 시간은 k값에 영향을 받았으며, 기존 연구[13]에서의 AES나 DES의 속도보다 빠른 것으로 나타났다. 암호화에 걸리는 시간을 측정하기 위해 $n=256$으로 두고 시작한 본 알고리즘은 $k=64$일 때 CPU 시간이 최저로 나타났으며, l의 값에는 큰 영향을 받지 않았다(표 3). 최적의 CPU 시간을 만드는 값이 아닌 $k=16$, $l=4$인 경우에서도 매트랩을 이용하여 1MB의 메시지를 암호화 및 복호화하는데 13.4초가 걸렸다(표 3). 즉 박스 알고리즘의 암호화 속도는 74.6 kB/sec로, $k=64$인 최적의 경우에서 C나 Java 프로그램을 이용하여 암호화 시간만 측면하면 훨씬 빠른 속도를 낼 수 있을 것이다. 이에 비해 2015년 발표된 논문에는 Java를 이용한 AES의 암호화 속도는 27.76~31.65 kB/sec, DES는 10.13~ 10.31 kB/sec로 나타났다[13]. AES나 DES는 곱셈 연산을 수반하는 데 반해 박스 알고리즘은 추출의 과정만 거치면 되기 때문에 연산 시간이 훨씬 짧게 나오는 것이다.

마지막으로 박스 알고리즘은 효율성을 올리기 위한 k와 l값을 조정할 수 있다. 암호화의 효과성과 관련되는 키의 길이, 보안성을 나타내는 복잡도, 비용에 관련되는 CPU 시간을 바탕으로 효율성을 정의하여 분석하였다(표 4, 그림 3). 그 결과 $k=64$, $l=8, 16$을 선택하여 암호화 하는 것이 가장 합리적이라는 결과를 얻을 수 있었다.

박스 알고리즘의 단점은 평문에 비해 암호문의 길이가 늘어나는 점이다. 하지만 그로 인해 동일한 평문에 대해 암호문이 다양한 형태로 나타난다는 장점이 있다. 따라서 일대일의 암호화 방법인 AES나 SEED에 비해 암호 해독 알고리즘을 구사하기가 힘들 것이다. 또한 본 알고리즘의 암호화 시간은 AES나 DES에 비해 훨씬 빠른 장점이 있다. 박스 알고리즘은 키의 길이 대비 복잡도가 AES에 비해 다소 떨어지는 단점이 있다. 하지만 키의 길이가 암호화의 절대적인 요인은 아니므로 박스 알고리즘의 장점인 키의 길이를 변형시킴으로써 원하는 수준의 복잡도를 구사할 수 있다.

향후 본 알고리즘을 통해 만들어진 암호문을 다양한 해킹 알고리즘을 통해 실질적인 보안성 검토를 거치는 과정이 필요할 것으로 사료된다.

References

[1] Information Right, http://www.privacy.or.kr/archives/4891 [Accessed: Sep. 05, 2017]

[2] I. K. Kim, "Fundamentals of cryptography", Kyowoosa, pp. 13-16, Jan. 2015.

[3] B. Schneier, "Applied Cryptograpy", 2nd. ed. John Wiley & Sons, Inc., pp. 11-14, Jan. 1996

[4] NBS, "Data Encryption Standard," FIPS Pub. 46, U.S, National Bureau of Standards, Washington DC, pp. 1-7, Oct. 1999.

[5] W. Barker, "Introduction to the Analysis of the

Data Encryption Standard(DES)", Laguna Hills, Calif., Aegean Park Press, Vol. C-55, 1989.

[6] Electronic Frontier Foundation, "Cracking DES - Secrets of Encryption Research, Wiretap Politics & Chip Design", Oreilly & Associates Inc., pp 1-15, May 1998

[7] M. Matsui, "Linear cryptanalysis method for DES cipher", Advances in Cryptology - EUROCRYPT pp. 386-397, 1993.

[8] Wikipedia contributors, "Data Encryption Standard", Wikipedia, The Free Encyclopedia, 24 Aug. 2017.

[9] G. Alvarez and S. Li, "Some basic cryptographic requirements for chaos-based cryptosystems", International Journal of Bifurcation and Chaos, Vol. 16, No. 8, pp. 2129-2151, Aug. 2006

[10] Wikipedia contributors. "Advanced Encryption Standard", Wikipedia, The Free Encyclopedia, 9 Sep. 2017. Web. 13, Sep. 2017

[11] Korea Internet & Security Agency, "Specification and detailed description of SEED-256 Encrypting Algorithm", Korea Internet & Security Agency, pp. 1-2, Jul. 2009

[12] J. Sung, "Differential cryptanalysis of eight-round SEED", Information Processing Letters, Vol. 111, No. 10, pp. 474-478, Apr. 2011

[13] S. Rihan, A. Khalid, and S. Osman, "A Performance Comparison of Encryption Algorithms AES and DES", International Journal of Engineering Research & Technology, Vol. 4, No. 12, pp. 151-154, Apr. 2011

저자소개

이 상 일 (Sangil Lee)

2001년 3월 : 해군사관학교 기계조선공학과(공학사)
2006년 7월 : 서울대학교 기계항공공학부(공학석사)
2013년 7월 : Texas A&M Univ. 재료공학과(박사수료)
2013년 7월 ~ 현재 : 해군사관학교 무기체계공학과 조교수
관심분야 : 암호화 알고리즘, 전산재료역학, 소성역학

민 승 식 (Seungsik Min)

2007년 2월 : KAIST 수학과, 물리학과(이학사)
2009년 2월 : KAIST 물리학과(이학석사)
2016년 2월 : 부경대학교 물리학과 (이학박사)
2009년 6월 ~ 현재 : 해군사관학교 이학과 조교수
관심분야 : 암호화 알고리즘, 통계물리학, 복잡계, 해군무기체계