논문 21-46-03-02 The Journal of Korean Institute of Communications and Information Sciences '21-03 Vol.46 No.03
https://doi.org/10.7840/kics.2021.46.3.420

형태보존암호 FEA 알고리즘 구현 및 성능 평가

장 원 영◆, 이 선 영°

Implementation and Performance Evaluation of the Format Preserving Encryption FEA Algorithm

Wonyoung Jang◆, Sun-Young Lee°

요 약

블록 암호는 암호화를 할 때 길이가 증가하여 블록 크기의 배수가 된다. 따라서 블록 암호는 IoT나 커넥티드 카 등 고정된 길이와 구조를 가진 데이터를 암호화하기 어려운 문제가 있다. 형태보존암호는 이와 같은 블록 암호의 문제점을 해결하여 효율적으로 암호화 할 수 있다. 형태보존암호 FEA는 국내에서 개발된 형태보존암호이다. FEA는 랭킹 함수, TBC 암호화, 역 랭킹 함수 단계로 암·복호화되지만 논문과 표준에서는 랭킹 함수의 정의가 되어있지 않고 사용자가 직접 랭킹 함수를 구현하여 사용하도록 설계되었다. 본 논문은 랭킹 함수를 직접 구현하여 FEA의 전 과정을 구현하였다. 구현한 FEA가 정상적으로 암·복호화되는지 확인하였고, 미국 NIST 표준 형태보존암호와 비교하여 암·복호화 속도가 더 빠른 것을 확인하였다.

키워드 : 형태보존암호, FEA, FF1, FF3-1, 개인정보 암호화
Key Words : Format preserving encryption, FEA, FF1, FF3-1, Private information encryption

ABSTRACT

When encrypted, block encryption increases in length by a factor of block size multiplication. Therefore, it suffers from a drawback that data encryption with fixed length and structure is challenging, such as that in the IoT and connected cars. Format preserving encryption can efficiently encrypt by solving the problem of block ciphers. Meanwhile, format preserving encryption FEA is a domestically developed format that preserves encryption; it is encrypted/decrypted in the Rank function, TBC encrypt, and Unrank function stages; but in its original paper and standard, the Rank function is not defined, this is designed to be implemented directly by the user. This study implements the entire process of FEA through direct implementation of the Rank function. We investigated whether the implemented FEA is normally encrypted/decrypted, and confirmed that the encryption/decryption speed is faster compared to the US NIST standard format for preserving encryption.

※ 이 성과는 2018년도 정부(교육과학기술부)의 재원으로 한국연구재단의 자원을 받아 수행된 연구임(No. NRF2018R1D1A1B07047656).
◆ First Author : Soonchunhyang University Department of Computer Science&Engineering, ozragwort@sch.ac.kr, 학생회원
° Corresponding Author : Soonchunhyang University Department of Information Security Engineering, sunlee@sch.ac.kr, 종신회원
논문번호 : 202007-162-A-RN, Received July 21, 2020; Revised September 23, 2020; Accepted September 25, 2020

Ⅰ. 서 론

국내외로 개인정보에 대한 위협이 증가하면서 다양한 개인정보 보호조치가 이루어지고, 개인정보에 대한 암호화가 중요해지고 있다[1]. 하지만 AES, ARIA 등 블록 암호를 이용하여 암호화할 경우 문제점이 있다. 개인정보는 정해진 형식과 길이를 가지고 있는 경우가 많아 데이터베이스에서 키로 사용되는 경우도 많기 때문에 블록 암호를 이용하여 암호화할 경우 형식이 바뀌고 패딩으로 암호문의 길이가 증가하기 때문에 데이터베이스에서 키로 사용할 수 없다. 또 암호문은 평문보다 길이가 증가하여 저장 공간이 낭비되고 형식이 바뀌면서 검색 성능이 하락하는 문제점이 있다[2,3].

형태보존암호는 이러한 블록 암호의 문제점을 해결하기 위해 처음 제안되었다. 형태보존암호를 이용하여 암호화된 데이터는 평문의 길이와 형태를 유지하는 특징이 있다. 형태보존암호는 1997년 Michael Brightwell 등에 의해 처음으로 제안되었다[4]. 그 후 2002년 John Black 등이 prefix cipher, cycle-walking cipher, generalized-Feistel cipher 세 가지 방식을 제시하면서 실질적인 형태보존암호 방식이 제안되었다[5]. 2009년에는 RTE(Rank-then-Encryption), 트윅(Tweak), 불균형 Feistel 구조를 이용하여 복잡한 도메인을 가지거나 길이가 긴 메시지에 대한 형태보존암호화가 가능해졌다[6]. 2010년 발표된 FFX mode(FF1), BPS(FF3), 2011년 발표된 VAES3(FF2)은 2013년 7월 미국 국립표준기술연구소(NIST) 표준으로 제정되었다[7-10]. 하지만 2015년 VAES3에 문제점이 발견되었고 2017년 FF3의 보안 취약점이 발견되면서 2019년 2월 기존 알고리즘을 수정한 FF1, FF3-1이 미국 국립표준기술연구소 표준으로 제정되었다[11-13]. 블록 암호는 길이가 증가하기 때문에 길이와 구조가 정해진 데이터를 암호화하기 어렵지만 형태보존암호를 이용하면 효율적으로 암호화 할 수 있다. 예를 들면 IoT(Internet of Things), 커넥티드 카 등 블록 암호를 사용하기 어려운 시스템에서 형태보존암호를 이용하여 보안성을 향상시키는 연구가 진행되고 있다[14,15].

국내에서는 2015년 국가보안기술연구소(NSR)가 제안한 FEA(Format-preserving Encryption Algorithm)가 한국정보통신기술협회(TTA) 표준으로 제정되었다[16,17]. FEA는 국가보안기술연구소가 독자기술로 개발한 형태보존암호로 전체 암호화 절차는 랭킹 함수, TBC 암·복호화, 역 랭킹 함수로 정의되어 있다. TBC 암·복호화는 실제 데이터가 암호화되는 과정이다. 랭킹 함수는 다양한 형식의 데이터를 공통의 형식으로 변환하고, 역 랭킹 함수는 TBC 암호문을 원래의 형식으로 변환한다. 랭킹 함수와 역 랭킹 함수는 암호문의 형식을 평문과 같게 해주는 함수이다. 하지만 FEA 표준은 TBC 암·복호화 과정에 대해서만 정의되어있고 랭킹 함수, 역 랭킹 함수에 대한 정의는 없다. 또한 FEA의 테스트 벡터는 랭킹 함수, 역 랭킹 함수를 제외하고 임의의 63-bit 또는 100-bit 비트열에 대한 TBC 암·복호화 과정만을 제공한다[17]. 10진수, 16진수 등 일반 데이터를 암호화하기 위해서 사용 환경과 구현 방식에 따라 사용자가 직접 정의하고 구현하여 사용해야 하는 문제점이 있다.

본 논문은 FEA의 랭킹 함수와 역 랭킹 함수를 직접 정의하여 FEA의 전체 과정을 구현하였다. 구현한 알고리즘은 10진수, 16진수 등 다양한 형식의 데이터를 암호화할 수 있다. 구현한 FEA를 이용하여 정상적으로 암·복호화 할 수 있는지 확인하고, 다양한 프로세서 환경과 길이가 다른 메시지를 이용하여 암호화 속도를 측정하였다. 미국 국립표준기술연구소 표준 형태보존암호 FF1, FF3-1과 비교하여 암호화 성능을 평가하였다[12].

2장은 FEA 알고리즘의 파라미터와 암·복호화 과정에 대해서 서술한다. 3장은 FEA 알고리즘의 랭킹 함수, 키 확장과 트윅 확장, TBC 암호화 과정을 어떻게 구현하였는지 서술하고 구현 결과를 출력하여 제대로 구현이 되었음을 확인하였다. 4장은 구현한 FEA 알고리즘을 5개의 프로세서 상에서 미국 국립표준기술연구소 표준 형태보존암호와 비교하여 암·복호화 속도가 더 빠른 것을 보였고, 5장에서 결론을 낸다.

Ⅱ. FEA 알고리즘[16,17]

FEA는 2015년 국가보안기술연구소에서 개발하고 한국정보통신기술협회 표준으로 제정된 국산 형태보존암호이다. 표 1은 FEA의 파라미터이다.

FEA는 두 가지 운용 모드 FEA-1과 FEA-2가 있고 128-bit, 192-bit, 256-bit 키 길이로 암호화 할 수 있다. 두 운용 모드는 같은 구조를 가지지만 트윅 적용 방식이 다르고 키 길이에 따라 라운드 수가 다르다. $radix^{length}$의 범위가 2^8 이상, 2^{128} 이하이고, 메시지의 최대 bit 길이는 $\lceil \log_2(radix^{length}) \rceil$ 이다.

표 1. FEA의 파라미터
Table 1. Parameters of FEA.

Algorithm types		FEA-1, FEA-2		
Input message range(N)		$radix^{length}$ = [28 ⋯ 2128]		
Maximum message length(M)		⌈ log2(N)⌉ bit		
Key length		128 bit	192 bit	256 bit
Total rounds	FEA-1	12	14	16
	FEA-2	18	21	24
Tweak length (bit)	FEA-1	128 - M		
	FEA-2	128		

2.1 FEA 암·복호화 과정

FEA는 cycle-walking 방식을 사용한 형태보존암호이다. 랭킹 함수 / 역 랭킹 함수, NumToBits / BitsToNum, TBC 암·복호화의 총 세 단계로 암·복호화한다[17]. 그림 1은 FEA의 전체 암·복호화 구조를 나타낸 것이다.

랭킹 함수는 입력받은 메시지를 연산이 가능한 숫자 형태로 변환하는 함수이고, 역 랭킹 함수는 랭킹 함수의 역연산이다. NumToBits, BitsToNum은 랭킹 함수를 통해 연산 가능한 숫자로 변환된 데이터를 비트로 변환하는 과정과 그 역연산을 하는 함수이다. TBC 암호화는 데이터가 암호화되는 과정이다. 표 2는 FEA에서 사용하는 변수에 대한 설명이다.

TBC 암호화 과정은 Feistel 구조로 설계되어있다. 입력값은 평문 X, 키 K, 트윅 T이다. K와 T는 키 확장과 트윅 확장으로 라운드 키 RK_a와 RK_b, 라운드 트윅 T_a와 T_b이 된다. TBC 암호화 과정은 Feistel 구조이므로 평문 X는 n_1-bit 길이의 X_a, n_2-bit 길이의 X_b

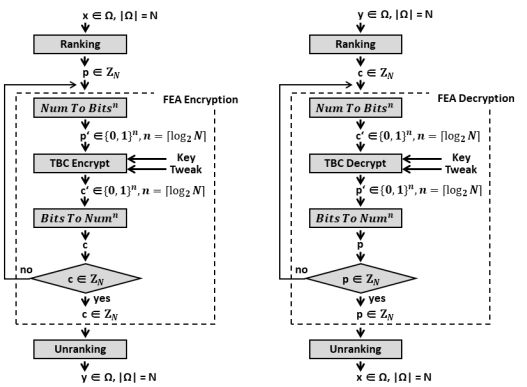

그림 1. FEA의 암·복호화 과정
Fig. 1. Encryption and Decryption process of FEA.

표 2. FEA 알고리즘 변수 정의
Table 2. FEA symbols description.

Symbols	Description
X	Plaintext
K	Secret key
T	Tweak
n	Plaintext's length
i	Round count
r	Total rounds
RK_a^i, RK_b^i	Round key
T_a^i, T_b^i	Round Tweak
F_o( ), F_e( )	Round function (odd round : F_o( ), even round : F_e( ))

로 나눈다. $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$ 으로 정의한다. 이때 X의 길이가 홀수 일 경우 X_a와 X_b의 길이가 달라지고 하나의 라운드 함수로 암호화를 할 수 없다. 따라서 TBC 암호화 과정에서 두 개의 라운드 함수 F_o와 F_e가 필요하다. F_o와 F_e는 현재 라운드가 홀수일 경우 F_o, 짝수일 경우 F_e 함수를 사용하여 암호화한다. F_o, F_e과정에서 X_b는 T_a와 XOR 연산을 계산한 값과 RK_a, RK_b, T_b를 입력값으로 사용한다. 출력된 값은 X_a와 XOR하여 Y_b가 된다. X_b는 Y_a가 된다. 그림 2는 TBC의 Feistel 구조이다.

라운드 함수 F_o와 F_e는 $(X_b \oplus T_a)$, T_b, RK_a, RK_b의 입력을 받는다. 라운드 키 RK_a와 RK_b는 64-bit이고 $X_b \oplus T_a$는 m_1 비트, T_b는 64 - m_1 비트의 길이를 가진다. n_1과 n_2의 길이가 다를 경우 라운드마다 라운드 함수의 입력이 되는 X_b의 길이가 달라지기 때문이다. F_o, F_e의 구조는 그림 3이다.

F 함수는 치환과 확산을 수행하는 함수이다. S는 치환 계층으로 S-box를 이용하여 8-bit 단위로 데이터

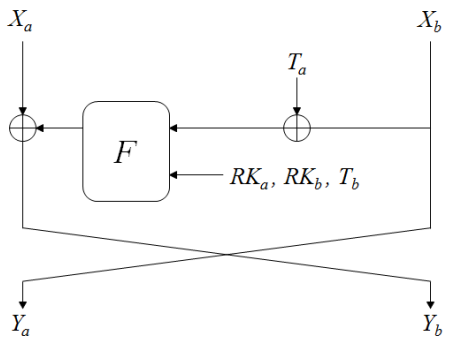

그림 2. FEA 알고리즘의 Fo, Fe 함수 구조
Fig. 2. Fo, Fe function structure of FEA algorithm.

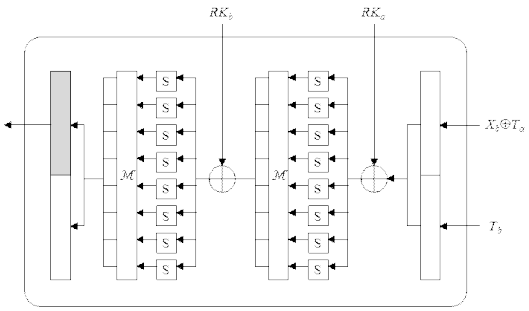

그림 3. 라운드 함수 F
Fig. 3. Round function F.

를 치환한다. M은 확산 계층으로 데이터를 이진 유한체 상의 8차 기약 다항식 $p(t) = t^8 + t^6 + t^5 + t^4 + 1$을 이용하여 정의된 유한체 $GF(2^8)$ 상의 선형 함수이다. 확산 계층에 입력을 X라고 하면 $X = \{X_0, \cdots, X_7\}$이고 $X_0, \cdots, X_7$은 각각 8-bit 값이다. 8×8 행렬 M의 i행 j열 값을 M(i, j)라고 정의하면 확산 계층의 출력값의 i번째 바이트 값은 $\oplus_7^j=0$ M(i, j) · X_j이다. 행렬 M은 표 3으로 정의한다. 바이트 곱셈 · 은 유한체 $GF(2^8)$ 상의 곱셈이다. 두 개의 8-bit 값 $a = a_7 \cdots a_0$과 $b = b_7 \cdots b_0$에 대한 바이트 곱셈 · 의 정의는 수식 1과 같다. mul_2는 8-bit 입·출력을 가지는 함수로 수식 2로 정의한다. C_p값은 p(t)에 대응하는 8-bit 값인 0x71로 정의한다.

표 3. 행렬 M
Table 3. Matrix M.

	0	1	2	3	4	5	6	7
0	28	1a	7b	78	c3	d0	42	40
1	1a	7b	78	c3	d0	42	40	28
2	7b	78	c3	d0	42	40	28	1a
3	78	c3	d0	42	40	28	1a	7b
4	c3	d0	42	40	28	1a	7b	78
5	d0	42	40	28	1a	7b	78	c3
6	42	40	28	1a	7b	78	c3	d0
7	40	28	1a	7b	78	c3	d0	42

$$a \cdot b = a_0 b \oplus a_1 mul_2(b) \oplus a_2 mul_2^2(b) \oplus \cdots \oplus a_7 mul_2^7(b) \quad (1)$$

$$mul_2(x) = (x \ll 1) \oplus x_7 C_p. \quad (2)$$

2.2 키, 트윅 확장

FEA의 키 확장은 128-bit, 192-bit, 256-bit의 비밀키 K, 평문의 비트 길이 n, 라운드 상수 RC를 사용하여 128-bit의 라운드 키를 출력한다. 4개의 64-bit 값인 K_a, K_b, K_c, K_d를 초기 상태값으로 가진다. K_a, K_b, K_c, K_d는 다음과 같이 설정한다.

- 키 길이(128-bit) : $K_a \| K_b = K$이고, $K_c = K_d = 0^{64}$
- 키 길이(192-bit) : $K_a \| K_b \| K_c = K$이고, $K_d = 0^{64}$
- 키 길이(256-bit) : $K_a \| K_b \| K_c \| K_d = K$

초기 상태 값을 이용하여 각 라운드마다 두 개의 64-bit 라운드 키를 출력한다. 그림 4는 키 확장의 과정을 나타낸 것이다.

트윅 확장은 FEA-1과 FEA-2 알고리즘에 따라 다르다. 입력 받은 트윅은 라운드 트윅 T_a^i, T_b^i로 출력된다. i는 라운드 트윅이 사용되는 라운드를 나타낸다.

FEA-1은 입력받은 트윅 T를 T_L과 T_R로 나눈다. T_L은 T[0 : 64-n_1-1]이고, TR은 T[64-n_2 : 128-n-1]이다. 라운드 트윅 T_a^i와 T_b^i에서 T_a^i는 0이고, T_b^i는 i가 홀수인 경우 T_L이 되고, i가 짝수인 경우 T_R이 된다.

FEA-2는 입력받은 트윅을 T_L, T_R로 나눈다. T_L은 T[0 : 63]이고, T_R은 T[64 : 127]이다. i를 3으로 나누어 나머지가 1인 경우 $T_a^i \| T_b^i = 0$, 나머지가 2인 경우 $T_a^i \| T_b^i = T_L$, 나머지가 0인 경우 $T_a^i \| T_b^i = T_R$로 정의한다.

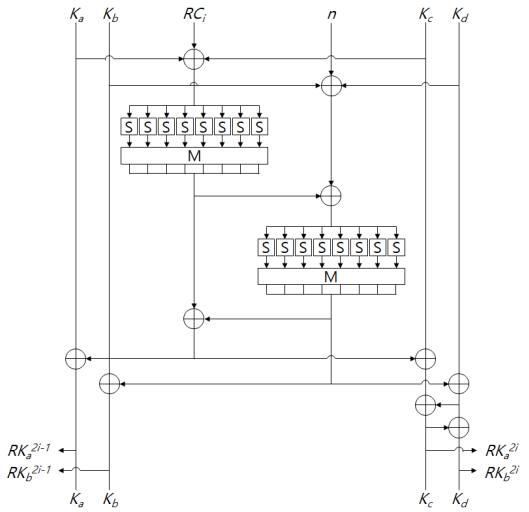

그림 4. 키 확장 라운드
Fig. 4. Key schedule round.

Ⅲ. FEA 알고리즘 구현

FEA 알고리즘에서 사용하는 용어에 대해서 알과

벳(Alphabet)은 데이터가 가질 수 있는 부호의 집합을 나타내고 밑(Base)은 알파벳 부호의 개수를 의미한다. 예를 들어 10진수 숫자라면 알파벳은 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}로 표현하고 밑은 10이다. 10진수 데이터 1234를 표현할 때 1234 = $[1, 2, 3, 4]_{10}$ 으로 표현한다. 데이터의 형식은 알파벳과 밑에 따라 형식이 달라진다. 형태보존암호는 임의의 형식을 가진 데이터를 암호화할 때 암호문도 같은 형식을 유지해야한다. 랭킹 함수는 다양한 형식의 데이터를 TBC 암호화를 하기 위하여 공통의 형식으로 변환하고, 역 랭킹 함수는 TBC 암호문을 원래의 형식으로 변환한다. 본 논문에서 구현한 랭킹 함수는 FEA 표준에서 공식적으로 정의되어있지 않아 새로 구현하였다.

3.1 랭킹 함수 구현

본 논문에서 구현한 랭킹 함수는 FEA 암호화 과정 앞에서 다양한 밑의 데이터를 공통의 밑으로 변환하는 함수이고, 역 랭킹 함수는 FEA 암호화 과정 뒤에서 암호화된 데이터를 원래의 밑으로 변환하는 함수이다. 구현한 랭킹 함수는 다음과 같은 방식이다.

평문의 길이를 m, 평문의 밑을 b라고 한다. 그때 데이터 a는 a = $[a_{m-1}, \cdots, a_1, a_0]b$ 로 표현한다. 표현된 데이터 a를 TBC 함수로 암호화할 수 있도록 10진수로 변환한다. 따라서 어떠한 값이 입력되어도 랭킹 함수의 출력값은 10진수 숫자가 된다. 이를 이용한 랭킹 함수 $Rank(x)$의 식은 수식 3과 같다.

$$Rank(a) = \sum_{i=0}^{m-1} (a_i \times b^i) \tag{3}$$

예를 들어 16진수 데이터 $89AB_{16}$ = $[8, 9, 10, 11]_{16}$ 을 랭킹 함수로 계산하는 과정은 다음과 같다.

$$Rank(89AB_{16}) = 8 \cdot 16^3 + 9 \cdot 16^2 + 10 \cdot 16^1 + 11 \cdot 16^0$$

$89AB_{16}$를 랭킹 함수로 계산한 값은 10진수 값 35243이다. 역 랭킹 함수는 FEA 암호화한 데이터를 다시 원래의 밑으로 변환하는 과정이다. 표 4는 역 랭킹 함수 $Unrank(x)$ 과정을 나타낸 것이다.

10진수 데이터 35243을 역 랭킹 함수로 계산한 값은 다음과 같다.

$$Unrank(35243_{10}) = 8 \cdot 16^3 + 9 \cdot 16^2 + 10 \cdot 16^1 + 11 \cdot 16^0$$

표 4. 역 랭킹 함수
Table 4. Unranking function.

$$Unrank(x) = y$$

$q_0 \leftarrow \lfloor x/b \rfloor$
$r_0 \leftarrow x \bmod b$
for $i = 1\ to\ (m-1)\ do$
  $q_i \leftarrow \lfloor q_{i-1}/b \rfloor$
  $r_i \leftarrow q_{i-1} \bmod b$
end for
$y \leftarrow [r_{m-1}, \cdots r_1, r_0]_b$

따라서 35243을 역 랭킹 함수로 계산한 값은 [8, 9, 10, 11]$_{16}$ = $89AB_{16}$가 된다.

3.2 키 및 트윅 확장 구현

키 확장은 키 K, 라운드 상수 RC, 블록 비트 길이 n을 이용한다. 라운드 상수는 암호화 키의 길이에 따라 다른 값이 정의되어있다. 표 5는 키 확장에서 사용하는 파라미터이고 그림 5는 키 확장의 구조이다.

트윅 확장은 각 라운드마다 T_a와 T_b가 정의된다. FEA-2는 입력받은 128-bit의 트윅을 64-bit씩 나누어 첫 번째 64-bit를 T_L, 두 번째 64-bit를 T_R로 정의한다. 현재 라운드가 i라고 할 때 i를 3으로 나누어 나머지가 1인 경우 $T_a^i \| T_b^i$ = 0, 나머지가 2인 경우 $T_a^i \| T_b^i$ = T_L, 나머지가 0인 경우 $T_a^i \| T_b^i$ = T_R로 정의한다. 표 6은 트윅 확장에서 사용하는 파라미터이고 그림 6은 트윅 확장의 구조이다.

표 5. 키 확장 파라미터
Table 5. Parameters of key schedule.

Parameter	Description
byte[] key	Secret key
long[] RC	Round constant
int n	Bit length of input data

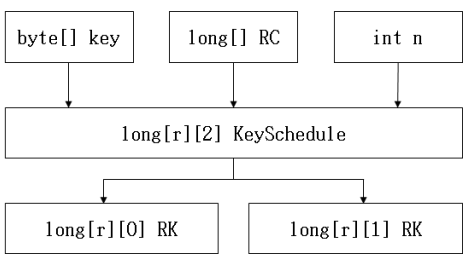

그림 5. 키 확장 구조
Fig. 5. Key schedule diagram.

표 6. 트윅 확장 파라미터
Table 6. Parameters of tweak schedule.

Parameter	Description
byte[] tweak	Tweak
int r	Total rounds

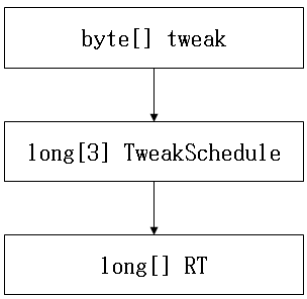

그림 6. 트윅 확장 구조
Fig. 6. Tweak schedule diagram.

3.3 TBC 암호화 함수 구현

TBC 암호화 함수는 FEA에서 메시지가 실제로 암호화되는 과정이다. TBC 암호화 함수는 암호화할 평문 블록 X, 라운드 키 RK_a, RK_b, 라운드 트윅 T_a, T_b를 이용하여 암호화한다. X는 랭킹 함수에 의해 변환된 평문 블록을 의미한다. 내부에서 현재 라운드에 따라 F_o와 F_e 함수를 이용하여 암호화한다. F_o와 F_e에 치환 계층과 확산 계층이 있다. F_o, F_e 함수는 내부에서 64-bit로 연산이 되기 때문에 다양한 길이의 X를 64-bit로 변환해야한다. 이때 X를 T_a와 XOR 연산 하고 T_b와 연접한다. $T_a^i \| T_b^i$의 길이는 트윅 확장에서 64-bit로 생성되었기 때문에 X를 64-bit로 변환할 수 있다. 표 7은 TBC 암호화 함수의 파라미터이고, 그림 7은 TBC 암호화의 구조이다.

plaintext는 입력한 평문이다. 랭킹 함수에 의해 변환된 데이터를 2개로 나눈 것이 X이다. TBC encrypt의 입력값은 X의 한 블록인 X[1], 키 확장으로 출력된 두 개의 라운드 키 RK_a, RK_b, 트윅 확장으로 생성된 라운드 트윅 T_a, T_b이다. X[1]과 라운드 트윅을 이용하여 TBC 암호화에서 사용할 roundX를 생성한다.

표 7. TBC 암호화 파라미터
Table 7. Parameters of TBC Encryption.

Parameter	Description
byte[] plaintext	Input plaintext data
long[] X	Plaintext bit string split by two
long[][] RK	Round Key
long[] RT	Round Tweak

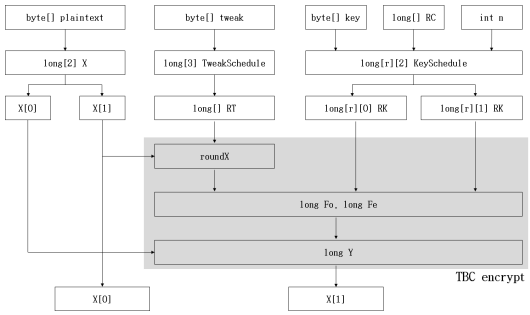

그림 7. TBC 암호화 구조
Fig. 7. TBC Encryption diagram.

roundX와 두 개의 RK_a, RK_b를 라운드에 따라 F_o, F_e 함수를 통해 Y를 출력한다. 출력된 Y는 다음 라운드에서 사용할 X[1]이 되고 기존의 X[1]은 X[0]이 된다. 라운드 수 만큼 반복 후 X[0]과 X[1]을 역 랭킹 함수로 암호문을 생성한다.

3.4 구현 결과

한국정보통신기술협회 표준에 정의된 FEA는 랭킹 함수가 정의되어있지 않아서 그대로 사용하면 실제 데이터를 암호화할 수 없다. 따라서 본 논문에서는 랭킹 함수를 포함한 FEA의 전체 과정을 구현하였다. 그림 8은 실제 구현한 FEA를 이용하여 암·복호화 실험 결과이다.

실험은 전화번호, 주민등록번호, 카드번호의 길이와 같은 11, 13, 16 자리의 10진수 숫자로 암·복호화 하였다. 구현 결과 여러 길이의 10진수 데이터가 정상

```
Length : 11
(phone number)
plaintext  : 12345678901
ciphertext : 40858636675
decryption : 12345678901

Length : 13
(resident registration number)
plaintext  : 1234567890123
ciphertext : 6219678049385
decryption : 1234567890123

Length : 16
(card number)
plaintext  : 1234567890123456
ciphertext : 1430027967785616
decryption : 1234567890123456
```

그림 8. 구현된 FEA 암·복호화 결과
Fig. 8. Implemented FEA encryption / decryption results.

적으로 암·복호화되는 것을 확인하였다.

Ⅳ. FEA 암호화 속도 비교

실험은 FEA, FF1, FF3-1의 암호화 속도를 측정하여 FEA의 속도를 평가한다. FEA는 FEA-2 알고리즘을 이용하여 실험하였다. FEA의 암호화 속도를 측정하기 위해 5가지 프로세서 환경에서 테스트하였다. 표 8은 실험에서 사용한 5가지 프로세서 환경이다.

실험은 다양한 메시지 길이에서 암호화 속도를 측정하기 위해 최소 6 자리, 최대 16 자리의 10진수 숫자를 이용하였다.

표 8. 실험 환경
Table 8. Experiment environment.

	Processor	Clock	Memory
1	(intel) i7-8700	3.2 GHz	8 GB
2	(intel) i5-8250U	1.6 GHz	8 GB
3	(AMD) Ryzen5 1600	3.2 GHz	8 GB
4	(ARM) Cortex-A53	1.4 GHz	1 GB
5	(ARM) Cortex-A7	0.9 GHz	1 GB

4.1 intel i7-8700 암호화 속도

intel의 i7-8700 프로세서의 성능을 기본 클럭인 3.2 GHz로 설정하여 실험하였다. 표 9는 i7-8700 프로세서 환경에서 FEA, FF1, FF3-1의 암·복호화 속도를 측정한 결과이고 시간은 ms로 나타내었다.

표 9. i7-8700 환경에서 FEA 암·복호화 시간
Table 9. Measurement of the FEA encryption / decryption time in i7-8700.

	Encryption time			Decryption time		
	FEA	FF1	FF3-1	FEA	FF1	FF3-1
6	0.0579	0.0575	0.0629	0.0583	0.0556	0.0622
7	0.0586	0.0668	0.0626	0.0590	0.0610	0.0620
8	0.0595	0.0715	0.0683	0.0596	0.0657	0.0659
9	0.0606	0.0817	0.0685	0.0629	0.0754	0.0657
10	0.0645	0.0814	0.0712	0.0632	0.0745	0.0648
11	0.0643	0.0760	0.0671	0.0636	0.0763	0.0611
12	0.0650	0.0783	0.0697	0.0645	0.0761	0.0631
13	0.0658	0.0806	0.0716	0.0660	0.0791	0.0691
14	0.0668	0.0781	0.0696	0.0687	0.0804	0.0669
15	0.0681	0.0825	0.0654	0.0691	0.0825	0.0667
16	0.0690	0.0852	0.0640	0.0732	0.0791	0.0652
avg.	0.0636	0.0763	0.0674	0.0644	0.0732	0.0648

4.2 intel i5-8250U 암호화 속도

intel의 i5-8250U 프로세서의 성능을 기본 클럭인 1.6 GHz로 설정하여 실험하였다. 표 10은 i5-8250U 프로세서 환경에서 FEA, FF1, FF3-1의 암·복호화 속도를 측정한 결과이고 시간은 ms로 나타내었다.

표 10. i5-8250U 환경에서 FEA 암·복호화 시간
Table 10. Measurement of the FEA encryption / decryption time in i5-8250U.

	Encryption time			Decryption time		
	FEA	FF1	FF3-1	FEA	FF1	FF3-1
6	0.1072	0.1313	0.1497	0.1093	0.1251	0.1566
7	0.1175	0.1533	0.1345	0.1146	0.1442	0.1439
8	0.1217	0.1501	0.1479	0.1189	0.1495	0.1544
9	0.1142	0.1520	0.1466	0.1302	0.1644	0.1596
10	0.1242	0.1582	0.1458	0.1216	0.1545	0.1569
11	0.1328	0.1629	0.1523	0.1308	0.1554	0.1599
12	0.1442	0.1513	0.1423	0.1343	0.1582	0.1534
13	0.1298	0.1788	0.1454	0.1308	0.1645	0.1545
14	0.1282	0.1652	0.1514	0.1428	0.1754	0.1520
15	0.1403	0.1717	0.1384	0.1429	0.1841	0.1569
16	0.1445	0.1911	0.1328	0.1424	0.2146	0.1463
avg.	0.1277	0.1605	0.1443	0.1290	0.1627	0.1540

4.3 AMD Ryzen 5 1600 암호화 속도

AMD의 Ryzen 5 1600 프로세서의 성능을 기본 클럭인 3.2 GHz로 설정하여 실험하였다. 표 11은 Ryzen 5 1600 프로세서 환경에서 FEA, FF1, FF3-1의 암·복호화 속도를 측정한 결과이고 시간은 ms로 나타내었다.

표 11. Ryzen 5 1600 환경에서 FEA 암·복호화 시간
Table 11. Measurement of the FEA encryption / decryption time in Ryzen 5 1600.

	Encryption time			Decryption time		
	FEA	FF1	FF3-1	FEA	FF1	FF3-1
6	0.0530	0.0542	0.0721	0.0527	0.0602	0.0727
7	0.0543	0.0588	0.0717	0.0562	0.0649	0.0725
8	0.0548	0.0616	0.0735	0.0599	0.0619	0.0746
9	0.0613	0.0703	0.0721	0.0594	0.0638	0.0744
10	0.0627	0.0653	0.0761	0.0573	0.0678	0.0748
11	0.0603	0.0727	0.0746	0.0586	0.0664	0.0711
12	0.0604	0.0697	0.0726	0.0641	0.0696	0.0726
13	0.0655	0.0675	0.0756	0.0635	0.0692	0.0774
14	0.0617	0.0687	0.0754	0.0617	0.0703	0.0771
15	0.0632	0.0701	0.0808	0.0648	0.0728	0.0741
16	0.0653	0.0787	0.0749	0.0647	0.0732	0.0740
avg.	0.0602	0.0671	0.0745	0.0603	0.0673	0.0741

4.4 ARM Cortex-A53 암호화 속도

ARM의 Cortex-A53 프로세서의 성능을 기본 클럭인 1.4 GHz로 설정하여 실험하였다. 표 12는 Cortex-A53 프로세서 환경에서 FEA, FF1, FF3-1의 암·복호화 속도를 측정한 결과이고 시간은 ms로 나타내었다.

표 12. Cortex-A53 환경에서 FEA 암·복호화 시간
Table 12. Measurement of the FEA encryption/ decryption time in Cortex-A53.

	Encryption time			Decryption time		
	FEA	FF1	FF3-1	FEA	FF1	FF3-1
6	6.4310	10.615	17.886	6.2357	10.872	19.804
7	6.5795	10.736	17.735	6.5700	11.138	19.854
8	7.1202	11.615	17.705	7.0378	11.532	20.191
9	7.6337	11.914	17.933	7.5852	11.902	20.441
10	7.9585	12.614	18.422	7.9007	13.040	20.155
11	8.6388	13.027	18.459	8.4273	13.543	19.679
12	8.9421	13.724	17.888	8.8743	14.383	19.537
13	9.2850	14.780	18.268	9.6752	14.811	20.482
14	9.9105	15.239	17.442	9.7796	15.281	21.431
15	10.500	16.403	17.233	10.060	16.151	20.442
16	10.337	17.038	18.012	10.522	16.548	20.760
avg.	8.4851	13.427	17.907	8.4243	13.563	20.252

4.5 ARM Cortex-A7 암호화 속도

ARM의 Cortex-A7 프로세서의 성능을 기본 클럭인 0.9 GHz로 설정하여 실험하였다. 표 13은 Cortex-A7 프로세서 환경에서 FEA, FF1, FF3-1의 암·복호화 속도를 측정한 결과이고 시간은 ms로 나타내었다.

표 13. Cortex-A7 환경에서 FEA 암·복호화 시간
Table 13. Measurement of the FEA encryption/ decryption time in Cortex-A7.

	Encryption time			Decryption time		
	FEA	FF1	FF3-1	FEA	FF1	FF3-1
6	4.8762	8.5215	18.223	4.8221	7.9050	17.225
7	5.1499	8.6727	18.224	5.2945	8.6977	16.881
8	5.7247	8.7664	18.501	5.5278	9.1574	17.170
9	5.7447	9.1816	17.668	6.1892	9.7773	17.540
10	6.3428	9.6608	18.719	6.4773	9.5758	18.193
11	6.9407	9.8680	17.421	6.5427	9.9533	18.782
12	6.8210	11.158	17.753	7.2688	10.702	18.957
13	7.5345	11.479	18.787	7.6058	11.422	18.740
14	7.9300	12.037	18.162	7.8341	12.230	17.984
15	8.9204	12.594	17.992	8.8207	12.743	18.043
16	8.8157	13.108	18.562	8.3884	13.593	18.086
avg.	6.8001	10.458	18.182	6.7974	10.523	17.963

4.6 FEA 암·복호화 속도 평가

FEA와 FF1, FF3-1의 암호화 속도를 비교하여 FEA의 암호화 속도를 평가한다. 표 14와 그림 9는 다양한 프로세서 환경에서 FEA, FF1, FF3-1의 평균 암호화 시간을 나타낸 표와 그림이다. 그림 9의 좌측 세로 축은 i7-8700, i5-8250U, Ryzen 5 1600 프로세서의 암호화 시간(ms)이고, 우측 보조 세로 축은 Cortex-A53, Cortex-A7 프로세서의 암호화 시간을 나타낸 것이다.

실험 결과 모든 프로세서 환경에서 랭킹 함수를 포함한 FEA는 FF1, FF3-1보다 암·복호화 속도가 더 빠른 것으로 확인되었다. 또 FEA의 암·복호화 속도는 FF1과 비교할 때 Cortex-A53 프로세서에서 58% 더 빠르고, FF3-1과 비교할 때 Cortex-A7 프로세서에서 167% 더 빠른 것으로 확인되었다. intel 프로세서를 이용한 실험에서도 클럭이 낮을수록 암·복호화 속도가 더 차이가 났다. 따라서 FEA는 프로세서의 성능이 낮을수록 FEA가 FF1, FF3-1보다 더 효율적인 것으로 판단된다.

표 14. 실험 환경에서의 FEA, FF1, FF3-1 평균 암호화 시간 표
Table 14. Average encryption time of FEA, FF1 and FF3-1 in experiment environment.

Processor	FEA	FF1	FF3-1
i7-8700	0.063645	0.076327	0.067355
i5-8250U	0.127691	0.160536	0.144282
Ryzen 5 1600	0.060227	0.067055	0.074491
Cortex-A53	8.485118	13.427727	17.907545
Cortex-A7	6.800055	10.458818	18.182909

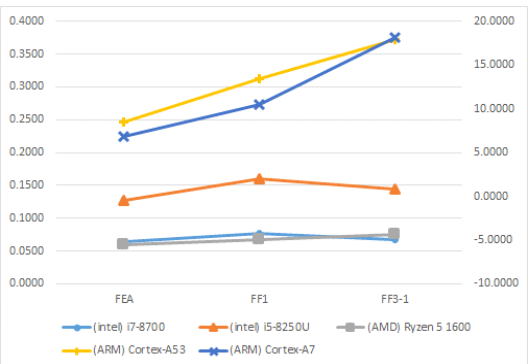

그림 9. 실험 환경에서의 FEA, FF1, FF3-1 평균 암호화 시간
Fig. 9. Average encryption time of FEA, FF1 and FF3-1 in experiment environment.

V. 결 론

국내에서 개발되어 한국정보통신기술협회 표준으로 제정된 형태보존암호 FEA는 랭킹 함수, TBC 암호화, 역 랭킹 함수 단계로 암호화한다. 하지만 표준에서 전체 구조와 TBC 암호화에 대한 정의만 되어있고 랭킹 함수에 대한 정의가 되어있지 않다. 따라서 10진수, 16진수 등 메시지를 암호화하기 위해서는 사용자가 직접 랭킹 함수를 정의해야한다.

본 논문에서는 FEA의 랭킹 함수를 직접 구현하였고, 구현한 FEA를 이용하여 암·복호화가 정상적으로 동작하는지 실험하였다. 또 다양한 프로세서 환경에서 암호화 속도를 측정하였고 미국 국립표준기술연구소의 표준 형태보존암호 FF1, FF3-1와 비교하였다.

실험 결과 모든 프로세서 환경에서 구현한 FEA의 암·복호화 속도가 FF1, FF3-1의 암·복호화 속도보다 더 빠른 것을 보였다. 특히 Cortex-A53 프로세서에서 FF1보다 58% 더 빨랐고, Cortex-A7 프로세서에서 FF3-1보다 167% 더 빠른 것으로 확인되어 프로세서의 클럭이 낮을수록 더 효율적으로 암호화할 수 있을 것으로 판단된다.

References

[1] Personal Information Protection Commission, *Annual Report on Personal Information Protection*(2019), Retrieved Jul. 12, 2020, from www.securitya.kr.

[2] T. Spies, *Format preserving encryption*(2008), Retrieved Jul. 12, 2020, from www.voltage.com.

[3] C. Lee, G. Kim, S. Hong, and S.-Y. Lee, "Implementation of the format preservation encryption FE1 algorithm," *J. KICS*, vol. 44, no. 7, pp. 1373-1380, Jul. 2019.

[4] M. Brightwell and H. Smith, "Using datatype preserving encryption to enhance data warehouse security," *20th National Inf. Syst. Secur. Conf. Proc.*, 1997.

[5] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *Cryptographers' Track at the RSA Conf. Springer*, pp. 114-130, Berlin, Heidelberg, 2002.

[6] M. Bellare, et al., "Format-preserving encryption," *Int. Wkshps Sel. Areas in Cryptography*, Springer, Berlin, Heidelberg, 2009.

[7] M. Bellare, P. Rogaway, and T. Spies, "The FFX mode of operation for format-preserving encryption," *NIST submission*, Feb. 2010.

[8] E. Brier, T. Peyrin, and J. Stern, "BPS: A format-preserving encryption proposal," *Режим доступу до ресурсу*, 2010, http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf.

[9] J. Vance, "VAES3 scheme for FFX: An addendum to The FFX mode of operation for format preserving encryption," *Submission to NIST 6.7*, May 2011.

[10] M. J. Dworkin, "Recommendation for block cipher modes of operation: Methods for formatpreserving encryption," *NIST Special Publication 800*, 38G Draft, 2016.

[11] M. J. Dworkin and R. A. Perlner, "Analysis of VAES3 (FF2)," *Cryptology ePrint Archive*, Apr. 2015.

[12] F. B. Durak and S. Vaudenay, "Breaking the FF3-1 format-preserving encryption standard over small domains," *Annu. Int. Cryptology Conf.*, Springer, Cham, 2017.

[13] M. J. Dworkin, *Recommendation for block cipher modes of operation: Methods for format-preserving encryption*(2019), Retrieved Jul. 12, 2020, from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38Gr1-draft.pdf

[14] J.-K. Lee, B. Koo, D. Roh, W.-H. Kim, and D. Kwon, "Format-preserving encryption algorithms using families of tweakable blockciphers," *Inf. Secur. and Cryptology – ICISC 2014. LNCS*, vol. 8949, pp. 132-159, 2015.

[15] TTAK.KO-12.0275, *Format Preserving Encryption FEA*(2015), Retrieved Jul. 12, 2020, from www.committee.tta.or.kr

[16] W. Jang and S.-Y. Lee, "Partial image encryption using format-preserving encryption in image processing systems for Internet of things environment," *Int. J. Distrib. Sensor Netw.*, vol. 16, no. 3, p. 1550147720914779,

Mar. 2020.

[17] I. Oh, T. Kim, K. Yim, and S.-Y. Lee, “A novel message-preserving scheme with format-preserving encryption for connected cars in multi-access edge computing,” *Sensors*, vol. 19, no. 18, p. 3869, Sep. 2019.

장 원 영 (Wonyoung Jang)

2018년 2월 : 순천향대학교 정보보호학과 졸업

2020년 8월 : 순천향대학교 컴퓨터학과(공학석사)

<관심분야> 정보보안, 보안개발, 암호이론

[ORCID:0000-0002-2777-9830]

이 선 영 (Sun-Young Lee)

1995년 2월 : 부경대학교 전자계산학과(이학석사)

2001년 3월 : 일본 도쿄대학교 전자정보공학전공(공학박사)

2004년 3월~현재 : 순천향대학교 정보보호학과 교수

<관심분야> 암호이론, 정보이론, 콘텐츠 보안, 정보보안

[ORCID:0000-0002-4686-9436]