Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.5, Oct. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.5.751

경량 블록 암호 PIPO의 화이트박스 구현 기법*

함 은 지,[1†] 이 영 도,[2] 윤 기 순[3‡]
[1,2,3]NSHC (전임연구원, 연구원, 연구소장)

A White Box Implementation of Lightweight Block Cipher PIPO*

Eunji Ham,[1†] Youngdo Lee,[2] Kisoon Yoon[3‡]
[1,2,3]NSHC (Associate Research, Researcher, Chief Cryptographer)

요 약

최근 세계적으로 사물인터넷 부문의 지출 성장이 높아짐에 따라 이를 암호화하기 위한 경량 블록 암호의 중요성 또한 높아지고 있다. ICISC 2020에 제안된 경량 블록 암호 PIPO 암호화 알고리즘은 Unbalanced bridge 구조를 이용한 SPN 구조의 암호이다. 화이트박스 공격 모델은 공격자가 암호화 동작의 중간값까지 알 수 있는 상태를 의미한다. 이를 대응하기 위한 기법으로 2002년 Chow 등은 화이트박스 구현 기법을 제안하여 DES와 AES에 적용하였다. 본 논문에서는 경량 블록 암호 PIPO 알고리즘에 화이트박스 구현 기법을 적용한 화이트박스 PIPO를 제안한다. 화이트박스 PIPO는 Chow 등이 제안한 화이트박스 AES 대비 테이블의 크기는 약 5.8배, 연산 시간은 약 17배 감소하였다. 또한, 모바일 보안제품에 화이트박스 PIPO를 활용하였으며 적용 범위에 따른 테스트 케이스 별 실험 결과를 제시한다.

ABSTRACT

With the recent increase in spending growth in the IoT sector worldwide, the importance of lightweight block ciphers to encrypt them is also increasing. The lightweight block cipher PIPO algorithm proposed in ICISC 2020 is an SPN-structured cipher using an unbalanced bridge structure. The white box attack model refers to a state in which an attacker may know the intermediate value of the encryption operation. As a technique to cope with this, Chow et al. proposed a white box implementation technique and applied it to DES and AES in 2002. In this paper, we propose a white box PIPO applying a white box implementation to a lightweight block cipher PIPO algorithm. In the white box PIPO, the size of the table decreased by about 5.8 times and the calculation time decreased by about 17 times compared to the white box AES proposed by Chow and others. In addition, white box PIPO was used for mobile security products, and experimental results for each test case according to the scope of application are presented.

Keywords: White-box Cryptography, PIPO, Lightweight Block Cipher

I. 서 론

최근 사물인터넷(IoT, Internet of Things) 시장이 꾸준히 확장됨에 따라 경량 환경에서 사용할 수 있는 암호 기술에 대한 수요가 증가하고 있다. ICISC 2020에 제안된 블록 암호 알고리즘 PIPO는 Unbalanced bridge 구조를 이용한 SPN(Substitution Permutation Network) 구조의 경량암호이며, 효율적인 고차 마스킹 소프트웨어 구현이 가능하여 부채널 공격에 높은 보안성을 가

Received(06. 21. 2022), Modified(09. 14. 2022), Accepted(09. 15. 2022)
* 본 연구는 정보통신기획평가원의 지원으로 수행했습니다(No. 2020-0-00126).
† 주저자, ehfdl5252@naver.com
‡ 교신저자, ksyoon@nshc.net(Corresponding author)

지고 있다[1].

암호 공격 모델은 공격자에게 주어진 능력에 따라 블랙박스, 그레이박스, 화이트박스로 분류한다. 그중에서 화이트박스 공격은 암호화 알고리즘에 대한 완전한 액세스 권한을 가지고 있으며 이를 이용하여 알고리즘의 동작 과정을 관찰하고 비밀키를 추출할 수 있다.

Chow 등은 AES에 키를 주입하고 인코딩을 적용한 최초의 화이트박스 AES(이하 WB-AES)를 제안했다[2]. 그러나 이는 2005년 제안된 대수적인 공격인 BGE 공격으로 약 2^{30}의 복잡도로 키를 찾아낼 수 있다[3]. 이후 이를 보완한 화이트박스 구현 기법들이 제안되었지만[4, 5] 그에 대응되는 새로운 공격 기법들 또한 제안되었다[6, 7]. 2016년 제안된 DCA 공격은 기존 블록 암호에 적용했던 부채널 공격을 이용하여 화이트박스 구현 기법이 적용된 암호의 키를 찾아낼 수 있다[8]. 이와 같이 현재는 알려진 모든 화이트박스 구현 기법이 적용된 암호에 대한 공격이 가능하다.

그러나 실제로는 알려지지 않은 화이트박스 암호에 대한 공격이 성공한 사례가 없고 키 관리의 어려움 등을 이유로 화이트박스 암호에 대한 수요는 꾸준히 증가하고 있다. 일반적으로 사용하고 있는 블록 암호의 경우 공격자는 입력과 출력만 관찰할 수 있다는 블랙박스 모델을 가정한다. 하지만 실제로 서비스를 제공하는 환경에서 공격자는 암호 알고리즘이 동작하고 있는 기기에 대한 전력정보를 취할 수 있을 뿐만 아니라 공격자가 기기를 점령하여 연산 중간값과 비밀키 정보를 탈취할 수 있다.

본 논문에서는 기존에 제안된 경량 블록 암호 PIPO 알고리즘에 화이트박스 구현 기법을 적용한 화이트박스 PIPO 알고리즘을 제안한다. 본 논문에서 제안하는 화이트박스 구현 기법이 적용된 PIPO 알고리즘을 이하 WB-PIPO라 한다.

본 논문의 구성은 다음과 같다. 2장에서는 경량 블록 암호 PIPO와 화이트박스 공격 모델 및 암호에 관해 설명한다. 3장에서는 본 논문에서 제안하는 화이트박스 구현 기법이 적용된 WB-PIPO를 제시한다. 4장에서는 제안된 WB-PIPO에 대한 안전성 고려사항과 가능한 공격에 대한 대응 기법을 서술한다.

5장에서는 WB-PIPO의 성능을 측정한 결과를 제시하고 Chow 등이 제안한 화이트박스 AES와 비교한다. 또한 모바일 보안제품에 WB-PIPO를 적용한 활용 사례를 기술한다. 마지막으로, 6장에서는 본 논문의 결론과 추후 연구를 기술한다.

II. 관련 연구

2.1 경량 블록 암호 PIPO

블록 암호 PIPO는 ICISC 2020에 제안된 SPN 구조의 경량암호이다. PIPO 암호 알고리즘은 Unbalanced bridge 구조를 이용하여 차분, 선형 공격에 안전한 S-box를 설계했다는 특징을 갖고 있다. PIPO 암호 알고리즘에서 사용한 S-box 구조는 테이블 참조 구현 외에 bitslice 방식의 구현이 가능하며 병렬 처리가 가능하다.

PIPO의 블록 크기는 64비트이고 비밀키의 크기에 따라 PIPO-128과 PIPO-256으로 나눌 수 있으며 각각 13 라운드, 17 라운드를 사용한다. PIPO의 라운드 연산 과정에서 각 연산의 중간값은 64비트이다. 경량 블록 암호 PIPO의 규격은 Table 1.과 같으며 PIPO의 전체 구조는 Fig. 1.과 같다.

PIPO의 라운드 키 RK_i는 128비트 마스터키일 경우 $K(=K_0\|K_1)$를 두 개의 64비트 서브키 K_0와 K_1로 나누어 라운드 키 $RK_i = K_{i \bmod 2}$ 를 생성한다. 256비트 마스터키일 경우 $K(=K_0\|K_1\|K_2\|K_3)$를 네 개의 64비트 서브키 K_0, K_1, K_2, K_3로 나누어 라운드 키 $RK_i = K_{i \bmod 4}$ 을 생성한다.

PIPO의 라운드 함수는 Fig. 1.과 같이 비선형 함수 S-Layer와 선형 함수 R-Layer, 키와 라운드 상수를 Exclusive Or (이하 XOR) 연산하는 Key Addition 단계로 이루어진다.

S-Layer는 테이블 참조 방식과 bitslice 방식으로 구현이 가능하다. 테이블 참조 방식의 경우 $2^8 \times 8 = 2048$ 비트 크기의 테이블을 저장해야 한다. S-Layer에서 사용되는 연산은 비트 연산자 AND, OR, XOR, NOT을 사용하기 때문에 비트 단위의 연산을 바이트 단위의 연산으로 확장할 수 있어 효율적인 연산이 가능하다. Bitslice 방식을 사용할 경우 Unbalanced bridge 구조를 이용하며 해당 구

Table 1. Block Cipher PIPO

Algorithm	Block size (bit)	Key size (bit)	Round
PIPO-128	64	128	13
PIPO-256	64	256	17

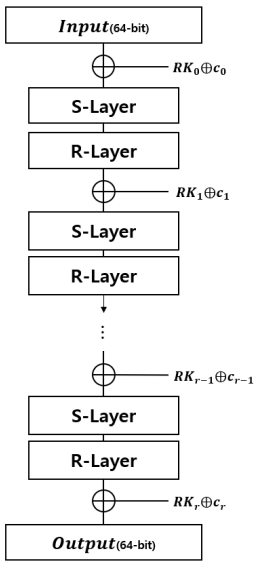


Fig. 1. PIPO Structure

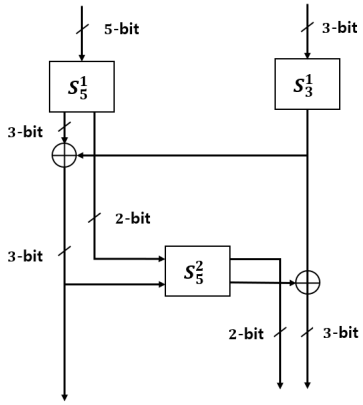


Fig. 2. PIPO S-Layer

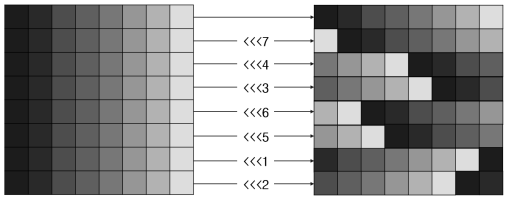


Fig. 3. PIPO R-Layer

조는 Fig. 2.와 같다. i비트 입력의 j-번째 S-box를 S_i^j로 표기할 때 S-Layer의 내부 함수는 S_5^1, S_5^2, S_3^1을 사용한다. R-Layer는 64비트 블록 8×8 배열의 각 행을 로테이션하는 선형 연산이며 해당 연산은 Fig. 3.와 같다.

2.2 화이트박스 암호

2.2.1 화이트박스 공격 모델

암호 공격 모델은 공격자에게 제공된 정보의 양과 성질에 따라 블랙박스, 그레이박스, 화이트박스 모델로 분류된다. 각 모델에 대한 설명은 다음과 같다.

- 블랙박스 모델: 입력값, 출력값, 그리고 알고리즘의 구조 외의 정보를 알 수 없는 상태
- 그레이박스 모델: 블랙박스 모델에서 전력정보, 전자파 등의 키에 대한 간접적인 정보를 추가적으로 알고 있는 상태
- 화이트박스 모델: 암호화 동작 디바이스의 메모리 접근과 수정이 모두 가능하며 중간 계산 값까지 알고 있는 상태

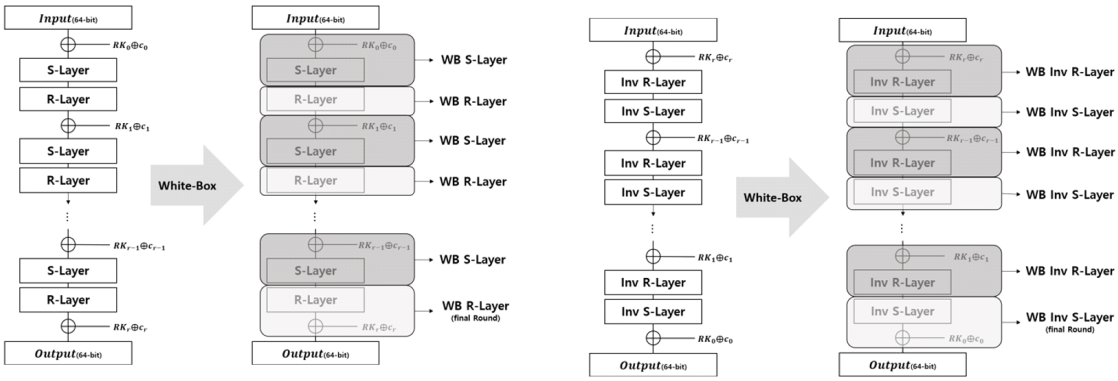
2.2.2 화이트박스 암호 관련 연구

화이트박스 구현 기법은 2002년 Chow 등에 의해서 처음 제안되었다. 화이트박스 구현 기법이 적용된 암호는 화이트박스 공격 모델의 공격자로부터 비밀키를 보호하기 위해 비밀키를 암호 알고리즘에 섞어 암호키가 드러나지 않게 설계된 암호이다. 즉, 알고리즘을 큰 참조 테이블로 생성하여 그 내부에 비밀키를 암호 알고리즘과 뒤섞인 상태로 숨김으로써 내부의 동작을 분석하더라도 키를 쉽게 유추할 수 없게 설계된 암호를 의미한다.

Chow 등이 WB-AES에 대한 설계를 제안한 이후 다양한 화이트박스 암호가 제안되었다. 그러나 블록암호 알고리즘의 대수적 특성을 이용한 BGE 공격, 통계적 분석을 이용한 DCA 공격 등 많은 화이트박스 암호 공격기법이 제안되었다. 이러한 공격기법들에 대응하기 위해 마스킹 기법을 적용한 화이트박스 암호 기법 등이 제안되었다.

2.2.3 인코딩

경량 블록 암호 PIPO는 AND, OR, XOR, NOT 그리고 좌측 회전 연산(이하 ROL)과 같은 8비트 단위의 비트 연산을 사용한다. 그러나 8비트 단위로 테이블을 생성할 경우 테이블의 크기가 매우



(1) $WB-PIPO\ Enc.$ (2) $WB-PIPO\ Dec.$

Fig. 4. WB-PIPO Structure

커지기 때문에 4비트 단위의 연산 두 개를 연접하여 연산을 진행한다. 4비트 단위의 연산 테이블에 인코딩을 적용하기 위해선 해당 비트 단위의 인코딩을 생성해야 한다. Fig. 5.[9]는 n비트 비선형 인코딩을 생성하는 알고리즘이며 Fig. 6.은 Fig. 5.에서 생성한 비선형 인코딩의 역함수를 계산하는 알고리즘이다.

Algorithm. n-bit nonlinear encoding (Fisher-Yates Shuffle)	
input	$X=(X[0],X[1],...,X[N-1]).\ N=(1\ll n)$
output	$X=(X[0],X[1],...,X[N-1]).\ N=(1\ll n)$
1. for $i=0$ to $N-1$ 2. $X[i]\leftarrow i$ 3. for $i=0$ to $N-1$ 4. $j\xleftarrow{\$}[0,1,...,N-1]$ 5. $T\leftarrow X[i]$ 6. $X[i]\leftarrow X[j]$ 7. $X[j]\leftarrow T$	

Fig. 5. n-bit nonlinear encoding (Fisher-Yates Shuffle)

Algorithm. n-bit nonlinear encoding inverse	
input	$X=(X[0],X[1],...,X[N-1]).\ N=(1\ll n)$
output	$X=(X[0],X[1],...,X[N-1]).\ N=(1\ll n)$
1. for $i=0$ to $N-1$ 2. $Y[X[i]]\leftarrow i$	

Fig. 6. n-bit nonlinear encoding inverse

III. WB-PIPO

본 장에서는 화이트박스 암호 기법을 적용한 경량 블록 암호 PIPO 설계 과정과 설계 과정에서 생성된 테이블의 크기에 관하여 기술한다.

3.1 WB-PIPO 알고리즘 설계

블록 암호의 경우 평문(또는 암호문)과 키를 입력하고 암호문(또는 복호문)을 출력한다. 화이트박스 구현 기법이 적용된 암호의 경우 사용되는 비밀키를 안전하게 보호하기 위해 고정된 키를 암호 알고리즘에 포함하여 비밀키에 대한 접근성을 낮춘다. 즉, 고정된 키에 의존하는 모든 연산을 테이블로 저장하고 이를 이용해 연산을 수행한다.

기존의 라운드 연산 중 Key Addition과 S-Layer 연산을 하나의 연산으로 간주하여 WB S-Layer라 한다. 암호화 알고리즘에서의 마지막 라운드 함수인 R-Layer와 Key Addition 함수를 하나의 연산으로 간주하여 WB R-Layer라 한다.

유사하게, 기존의 복호화 연산 중 Key Addition와 Inv R-Layer를 하나의 연산으로 간주하여 이하 WB Inv R-Layer라 한다. 복호화 알고리즘에서의 마지막 라운드 함수인 Inv S-Layer와 Key Additon을 하나의 연산으로 간주하여 WB Inv S-Layer라 한다. 기존의 PIPO의 구조와 본 논문에서 제안하는 WB-PIPO의 구조는 Fig. 4.와 같다.

3.2 외부 인코딩

코드 리프팅(code-lifting)이란 모듈의 실행 코드를 복제하거나 우회하는 기법이다. 코드 리프팅을 방지하기 위해서는 화이트박스 모듈의 입력과 출력 부분에 인코딩을 적용하여 정당한 사용자가 아닌 공격자는 원하는 입력에 대한 출력을 생성할 수 없도록 해야 한다. 이를 외부 인코딩이라 한다. 예를 들어, 입력과 출력이 n비트인 고정된 키 K에 대한 화이트박스 모듈 F를 $Y \leftarrow F(X)$라 하자. 공격자는 키 K를 알지 못하더라도 F의 실행 코드를 복제한다면 키 K가 적용된 화이트박스 모듈 F를 실행할 수 있다. 이를 방지하기 위해 F의 입력과 출력에 n비트 외부 인코딩 함수 IN, OUT을 다음과 같이 적용한다.

$$Y' \leftarrow F'(X') := (OUT \circ F \circ IN^{-1})(X').$$

상기 식에서 X'에 $IN(X)$을 입력하고 그에 대한 출력에 OUT^{-1}을 적용한 값은 F에 X을 입력했을 때의 출력값과 같다.

$$\begin{aligned} Y &= (OUT^{-1} \circ F' \circ IN)(X) \\ &= (OUT^{-1} \circ (OUT \circ F \circ IN^{-1}) \circ IN)(X) = F(X). \end{aligned}$$

즉, 외부 인코딩 IN, OUT을 알고 있는 정당한 사용자만이 원하는 입력에 대한 출력값을 계산할 수 있다. Fig. 7.은 4개의 비선형 인코딩 16개로 구성된 64비트 외부 인코딩을 적용하는 과정을 나타낸다.

WB-PIPO 알고리즘에서 사용하는 64비트 외부 인코딩 WI, WO은 4비트 인코딩 16개를 연접하여 생성한다.

$WI := (wi_0, wi_1, \ldots wi_{15})$, wi_i: 4비트 비선형 인코딩

$WO := (wo_0, wo_1, \ldots, wo_{15})$, wo_i: 4비트 비선형 인코딩 $(0 \le i < 16)$.

즉, 64비트 인코딩을 생성하는 방법은 Fig. 5. 알고리즘에 $n = 4$를 대입하고 이를 16번 실행하여 4비트 인코딩 16개를 생성한다.

Algorithm 7. Table Lookup Encoding	
Input	$X = x_0 \| x_1 \| \cdots \| x_{15}$, x_i: 4-bit $(0 \le i < 16)$ $G = (g_0, g_1, \ldots, g_{15})$, g_i: 4-bit nonlinear encoding $(0 \le i < 16)$
Output	$Y = y_0 \| y_1 \| \cdots \| y_{15}$, y_i: 4-bit $(0 \le i < 16)$
1. for $i = 0$ to 15 2. $y_i \leftarrow g_i(x_i)$	

Fig. 7. Table Lookup Encoding

임의의 고정된 키 K에 대한 PIPO 알고리즘을 $PIPO_K$라 할 때, WB-PIPO 알고리즘 $WBPIPO_K$을 다음과 같이 정의한다.

$$WBPIPO_K := WO \circ PIPO_K \circ WI^{-1}.$$

Y을 $PIPO_K$에 입력값 X을 입력했을 때의 출력이라 하자. 즉, $Y = PIPO_K(X)$이다. $WBPIPO_K$를 이용해 입력 X에 대한 출력 Y을 다음과 같이 3단계로 나누어 계산할 수 있다.

1. $X' := WI(X)$.
2. $Y' := WBPIPO_K(X')$
   $= (WO \circ PIPO_K \circ WI^{-1})(WI(X))$
   $= WO(PIPO_K(X)) = WO(Y)$.
3. $WO^{-1}(Y') = WO^{-1}(WO(Y)) = Y$.

상기 식으로부터 WB-PIPO 모듈만을 획득한 공격자는 원하는 입력에 대한 출력을 계산할 수 없으며, 오직 외부 인코딩을 알고 있는 정당한 사용자만이 WB-PIPO 모듈로부터 PIPO와 동일한 입력에 대한 출력을 계산할 수 있다.

결과적으로, WB-PIPO 모듈은 외부 인코딩을 적용하기 위해서 첫 번째 라운드의 첫 연산 테이블 입력 인코딩으로 WI^{-1}를 사용하고 마지막 라운드의 마지막 연산 테이블 출력 인코딩으로 WO을 사용하여 외부 인코딩을 포함한 화이트박스 모듈을 설계할 수 있다.

3.3 테이블 생성

경량 블록 암호 PIPO의 입력과 출력의 크기는 64비트이며 이를 코드북과 같은 형태로 저장할 때 크기가 $2^{64} \times 64$비트인 테이블을 저장해야 하는데 이는 현실적으로 사용하기에는 어려움이 있다. 따라서 WB-PIPO를 구현하기 위해서는 각 라운드의 연산을 분리해야 한다. 기존의 블록 암호 PIPO는 8비트 단위로 연산을 수행하며 AND, OR, NOT 그리고 ROL와 같은 비트 연산자를 사용한다. AND, OR, XOR과 같은 8비트 단위 연산 테이블 하나의 크기는 $2^{16} \times 8$비트이며 이를 테이블화 할 때, PIPO-128 기준 총 $84 \times 13 \times 2^{16} \times 8 \approx 2^{29}$비트인 테이블을 저장해야 한다.

본 논문에서 제안하는 WB-PIPO는 두 개의 4비

트 단위의 AND, OR XOR, NOT 그리고 ROL 연산테이블을 생성하여 8비트 테이블 참조 연산을 수행한다. WB-PIPO에서 사용하는 테이블은 S-Layer에서 사용하는 $2n$비트 입력, n비트 출력의 $2n\times n$ 테이블, n비트 입출력의 $n\times n$ 테이블과 R-Layer에서 사용하는 ROL_k 테이블로 구분할 수 있다. 본 장에서는 위와 같이 구분된 테이블을 생성하는 방법과 이를 참조하는 함수를 설명한다. n비트 단위의 비트 연산자 AND, OR, XOR, NOT, ROL 함수를 각각 AND_n, XOR_n, OR_n, NOT_n, ROL_n로 표기한다.

WB S-Layer와 Inv WB S-Layer 테이블 참조 연산은 함수 $m\times n_OP_k$를 사용한다. 이때, m, n은 각각 입력 비트와 출력 비트의 크기를 의미하며 m은 $2n$ 또는 n이다.

3.3.1 $2n\times n\_OP_k$ 테이블

$2n\times n$ 테이블 참조 함수에서는 n비트 라운드키 r_0, r_1, r_2 XOR과 n비트 OP_n 연산을 수행하는 테이블 $OP_n^{r_0,r_1,r_2}$에 n비트 인코딩을 추가한 테이블 $\overline{OP_n^{r_0,r_1,r_2}}$을 다음과 같이 정의한다.

$$\overline{OP_n^{r_0,r_1,r_2}}(x,y) := g\left(OP_n^{r_0,r_1,r_2}\left(f_0^{-1}(x), f_1^{-1}(y)\right)\right).$$

이때, $OP_n^{r_0,r_1,r_2}$는 다음과 같이 정의한다.

$$OP_n^{r_0,r_1,r_2}(x,y) := OP_n(x\oplus r_0, y\oplus r_1)\oplus r_2.$$

상기 식에서 x,y는 n비트이고 f_0^{-1}, f_1^{-1}, g는 n비트 비선형 인코딩이다. f_0^{-1}, f_1^{-1}는 x과 y에 적용된 인코딩을 제거하기 위한 인코딩이며, g는 라운드 키 XOR과 OP_n 연산한 결과를 보호하기 위해 적용하는 인코딩이다.

이를 이용해 8비트 라운드키 k_0, k_1, k_2 XOR과 OP_4 연산에 인코딩을 적용한 연산 $8\times4_OP_4(X,Y,k_0,k_1,k_2)$을 다음과 같이 정의한다.

$$\begin{aligned}
&8\times4_OP_4(X,Y,k_0,k_1,k_2)\\
&:= \overline{OP_4^{k_0[0],k_1[0],k_2[0]}}(X[0],Y[0]) \| \overline{OP_4^{k_0[1],k_1[1],k_2[1]}}(X[1],Y[1])\\
&= g_0\left(OP_4^{k_0[0],k_1[0],k_2[0]}\left(f_0^{-1}(X[0]), f_1^{-1}(Y[0])\right)\right)\\
&\quad \| g_1\left(OP_4^{k_0[1],k_1[1],k_2[1]}\left(f_2^{-1}(X[1]), f_3^{-1}(Y[1])\right)\right)
\end{aligned}$$

상기 식에서 $X[i], Y[i]$ 그리고 $k_j[i]$는 4비트이며

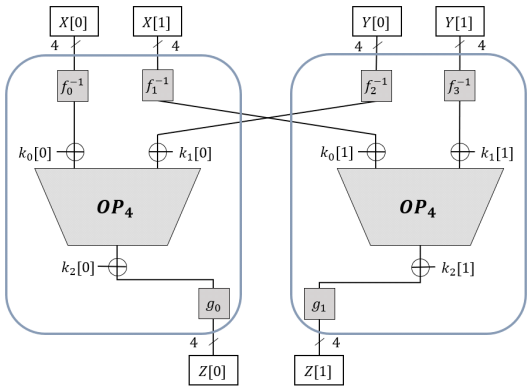


Fig. 8. $8\times4\_OP_4$

$X = X[0]\|X[1]$, $Y = Y[0]\|Y[1]$, $k_j = k_j[0]\|k_j[1]$ $j=0,1,2$ 이다.

8×4 테이블 참조 함수는 Fig. 8.과 같으며, 하나의 테이블 참조 함수에는 두 개의 테이블이 참조된다. 해당 연산에서 OP_4는 AND_4, XOR_4, OR_4 함수를 사용한다.

3.3.2 $n\times n\_OP_k$ 테이블

$n\times n$ 테이블 참조 함수에서는 n비트 라운드키 $r_0 r_1$ XOR과 n비트 OP_n 연산을 수행하는 테이블 $OP_n^{r_0,r_1}$에 n비트 인코딩을 추가한 테이블 $\overline{OP_n^{r_0,r_1}}$을 다음과 같이 정의한다.

$$\overline{OP_n^{r_0,r_1}}(x) := g\left(OP_n^{r_0,r_1}\left(f_0^{-1}(x)\right)\right).$$

이때, $OP_n^{r_0,r_1}$는 다음과 같이 정의한다.

$$OP_n^{r_0,r_1}(x) := OP_n(x\oplus r_0)\oplus r_1.$$

상기 식에서 x는 n비트이고 f_0^{-1}, g는 n비트 비선형 인코딩이다.

이를 이용해 8비트 라운드키 k_0, k_1 XOR과 OP_n 연산에 인코딩을 적용한 연산 $4\times4_OP_4(X,k_0,k_1)$을 다음과 같이 정의한다.

$$\begin{aligned}
&4\times4_OP_4(X,k_0,k_1)\\
&:= \overline{OP_4^{k_0[0],k_1[0]}}(X[0]) \| \overline{OP_4^{k_0[1],k_1[1]}}(X[1])\\
&= g_0\left(OP_4^{k_0[0],k_1[0]}\left(f_0^{-1}(X[0])\right)\right) \| \| \; g_1\left(OP_4^{k_0[1],k_1[1]}\left(f_1^{-1}(X[1])\right)\right).
\end{aligned}$$

상기 식에서 $X[i], k_j[i]$는 4비트이며 $X = X[0]\|X[1]$, $k_j = k_j[0]\,\|\,k_j[1]$, $j=0,1$ 이다. 4×4

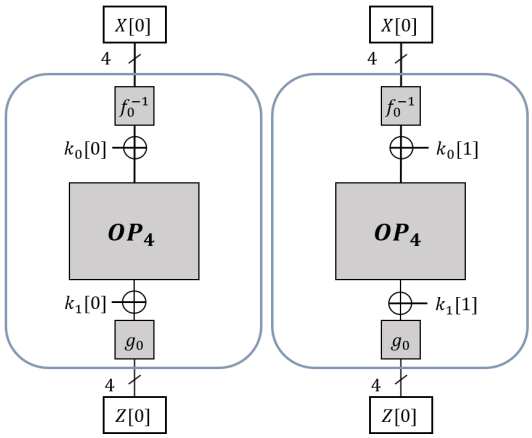


Fig. 9. $4\times4\_OP_4$

테이블 참조 함수는 Fig. 9.와 같으며, OP_4는 NOT_4 함수를 사용한다.

3.3.3 $n\times n\_ROL_k$ 테이블

WB R-Layer와 Inv WB R-Layer 테이블 참조 연산은 $n\times n\_ROL_k$ 함수를 사용한다.

기존의 PIPO에서 사용하는 8비트 ROL 연산 ROL_8은 다음과 같다.

$$ROL_8(X,L) = (X \ll L) \| (X \gg (8-L)),\ 0 \le L < 8.$$

4비트 ROL 연산 ROL_4을 다음과 같이 정의한다.

$$ROL_4(x,y,l) = (x \ll l) \| (y \gg (4-l)),\quad 0 \le l < 4.$$

8비트 ROL 연산 ROL_8을 4비트 ROL 연산 ROL_4를 이용해 다음과 같이 계산할 수 있다.

$$ROL_8(X,n) := ROL_4(X[d \bmod 2], X[d+1], l) \| ROL_4(X[d+1 \bmod 2], X[d], 4-l).$$

상기 식에서 $d = n/4$, $l = n \bmod 4$, $X = X[0] \| X[1]$이며, $X[i]$는 4비트이다.

4비트 라운드키 r_0, r_1 XOR과 4비트 ROL_4 연산을 수행하는 테이블 $ROK_4^{r_0 r_1 r_2}$에 4비트 인코딩을 추가한 테이블 $\overline{ROL_4^{r_0 r_1 r_2}}$을 다음과 같이 정의한다.

$$\overline{ROL_4^{r_0 r_1 r_2}}(x,y,l) := g\left(ROL_4^{r_0 r_1 r_2}\left(f_0^{-1}(x), f_1^{-1}(y), l\right)\right),\ 0 \le l < 4.$$

이때, $ROL_4^{r_0 r_1, r_2}$는 다음과 같이 정의한다.

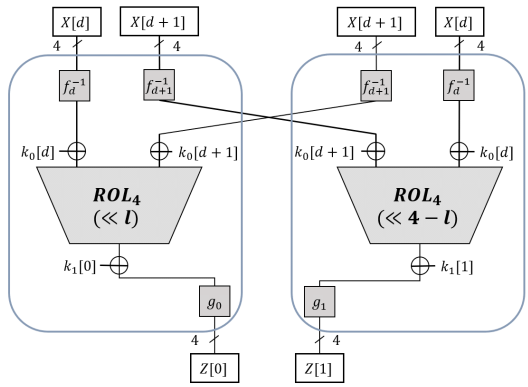


Fig. 10. $8\times4\_ROL_4$

$$ROL_4^{r_0 r_1 r_2}(x,y,l) = ROL_4(x \oplus r_0, y \oplus r_1, l) \oplus r_2,\ 0 \le l < 4.$$

상기 식에서 f_0^{-1}, f_1^{-1}는 x과 y에 적용된 인코딩을 제거하기 위한 인코딩이며, g는 라운드키 XOR과 ROL_4 연산 결과를 보호하기 위해 적용하는 인코딩이다.

이를 이용해 8비트 라운드키 k_0, k_1 XOR과 8비트 단위의 로테이션에 인코딩을 적용한 $n\times n_ROL_4(X, L, k_0, k_1)$을 다음과 같이 정의한다.

$$\begin{aligned} & n\times n\_ROL_4(X, L, k_0, k_1) \\ & := \overline{ROL_4^{k_0[d], k_0[d+1], k_1[0]}}(X[d], X[d+1], l) \\ & \| \overline{ROL_4^{k_0[d+1], k_0[d], k_1[1]}}(X[d+1], X[d], 4-l) \\ & = g_0\left(ROL_4^{k_0[d], k_0[d+1], k_1[0]}\left(f_d^{-1}(X[d]), f_{d+1}^{-1}(X[d+1]), l\right)\right) \\ & \| g_1\left(ROL_4^{k_0[d+1], k_0[d], k_1[1]}\left(f_{d+1}^{-1}(X[d+1]), f_d^{-1}(X[d]), 4-l\right)\right). \end{aligned}$$

상기 식에서 $d = n/4$, $l = n \bmod 4$, $X = X[0] \| X[1]$, $k_j = k_j[0] \| k_j[1]$이며 $X[i]$, $k_j[i]$는 각각 4비트이다. ROL_4 테이블 참조 함수는 Fig. 10.과 같다.

3.4 테이블 크기

본 장에서는 앞서 설명했던 WB-PIPO의 테이블 크기에 관해 설명한다.

WB-PIPO에서 사용하는 4비트 연산 테이블의 크기는 Table. 2.와 같다. Table. 3.은 WB-PIPO의 암호화 및 복호화의 테이블 크기와 이를 Chow의 WB-AES의 테이블 크기를 정리한 것이다. 128비트 키 크기 기준으로 Chow의 WB-AES

Table 2. Size by operation table

Operation Table	Input (bit)	Output (bit)	Size (bit)
$\overline{AND_4^{k_0,k_1,k_2}}$	8	4	1024
$\overline{OR_4^{k_0,k_1,k_2}}$	8	4	1024
$\overline{XOR_4^{k_0,k_1,k_2}}$	8	4	1024
$\overline{NOT_4^{k_0,k_1}}$	4	4	64
$\overline{ROL_4^{k_0,k_1,k_2}}$	8	4	1024

Table 3. Table size of WB-PIPO and Chow WB-AES

White Box	Key Size (bit)	Round	Table Size (byte)
WB-PIPO Enc.	128	13	133,344
	256	17	174,368
WB-PIPO Dec.	128	13	136,656
	256	17	178,704
Chow WB-AES	128	10	770,048

WB-AES와 비교했을 때 본 논문에서 제안한 WB-PIPO의 테이블의 크기가 약 5.8배 작은 것을 확인할 수 있다.

IV. 안전성 고려사항

본 장에서는 WB-PIPO 구조에 대해 알려진 화이트박스 공격인 코드 리프팅, 부채널 분석, 리버스 엔지니어링 공격과 이에 대한 대응 기법을 기술한다.

4.1 외부 인코딩

3.2절에서 언급한 바와 같이 코드 리프팅이란 응용 프로그램 일부 혹은 전체를 추출하는 공격이다. 화이트박스 구현상에서도 다른 장치의 구현 전체를 추출할 수 있다면 비밀키가 이 프로그램에 포함되어 있으므로 화이트박스 구현의 기능도 함께 추출된다. 이러한 공격을 코드 리프팅이라고 하며 이 문제에 대한 가능한 해결책은 외부 인코딩을 사용하는 것이다. 본 논문에서 제안하는 외부 인코딩 방식은 3.2절에 기술되어 있다.

4.2 코드 난독화

소프트웨어상의 리버스 엔지니어링은 하나의 컴퓨터 프로그램을 역으로 분석하는 기술을 의미한다. 이러한 리버스 엔지니어링은 목적 코드 프로그램을 조사 및 분석하여 프로그램의 정보를 추출하고, 개발 원리를 파악하는 과정이다. 암호 회로 내부의 메모리 접근까지도 가능한 공격 모델을 가정한 화이트박스 공격 모델에서는 위와 같은 리버스 엔지니어링을 통한 공격이 가능하다. 리버스 엔지니어링에 대응하기 위하여 복잡하게 설계된 암호 알고리즘 회로에 비밀키를 주입하고 리버스 엔지니어링이 불가능하도록 난독화를 적용해야만 한다. 고수준의 난독화를 통하여 암호 알고리즘이 동작하는 과정 내에서 비밀키가 메모리에 로드되지 않도록 해야만 한다.

4.3 마스킹 기법 적용

부채널 분석은 암호 모듈 상에서의 암호 알고리즘이 구동될 때 발생 되는 전력 및 전자파 등의 부가적인 정보를 이용하여 분석하는 기법이다. 이러한 부채널 분석에 대응하는 기법으로는 마스킹 기법이 존재한다. 마스킹 기법은 암호화 회로의 중간값을 랜덤하게 생성하기 위한 기법으로, 본 절에서는 부채널 분석에 대응하기 위하여 마스킹 기법이 적용된 WB-PIPO를 제안한다. 앞서 제안한 WB-PIPO의 암호화 및 복호화 연산 중 비밀키가 수행되는 과정만을 고려하여 마스킹 기법을 적용한다.

WB-PIPO는 PIPO에서 사용한 8비트 연산 AND, XOR, OR, NOT 그리고 8비트 단위 ROL 연산을 4비트 단위의 테이블 두 개를 연접하였다. 본 마스킹 기법이 적용된 화이트 박스 함수는 8비트 테이블 연산을 사용하며 마스킹 기법이 적용된 WB-PIPO를 이하 Masked WB-PIPO라 한다. 본 장에서는 Masked WB-PIPO에서 사용되는 테이블 중 키를 입력받는 함수에 대한 마스킹 테이블과 마스킹을 제거하는 마스킹 제거 테이블 생성에 관한 부분만을 기술한다.

Masked WB-PIPO에서 사용되는 마스킹 테이블 참조 함수는 S-Layer와 Inv S-Layer에서 사

용되는 $2n$비트 입력 $2n$비트 출력의 $2n\times 2n$테이블, R-Layer와 Inv R-Layer에서 사용되는 n비트 입력 $2n$비트 출력의 $n\times 2n$ 테이블을 사용한다. 그리고 $2n\times 2n$ 테이블과 $n\times 2n$ 테이블에서 사용된 마스킹을 제거하는 테이블인 마스킹 제거 테이블은 $2n$ 비트 입력 n비트 출력의 $2n\times n$ 테이블을 사용한다.

4.3.1 마스킹 테이블

Masked WB S-Layer에서 사용되는 마스킹 테이블은 $2n\times 2n$ 테이블을 사용한다. $2n\times 2n$ 테이블 참조 함수에서는 n비트 라운드키 r_0, r_1, r_2 XOR과 n비트 OP_n 연산을 수행하는 테이블 $OP_n^{r_0 r_1, r_2}$에 n비트 인코딩을 추가한 테이블 $\overline{OP_n^{r_0 r_1, r_2}}$을 사용한다.

이를 이용해 8비트 라운드키 k_0, k_1, k_2 XOR과 OP_8 연산에 인코딩을 적용한 연산 $MT\_16\times 16\_OP_8(X, Y, k_0, k_1, k_2, \alpha)$ 을 다음과 같이 정의한다.

$$
\begin{aligned}
&MT\_16\times 16\_OP_8(X, Y, k_0, k_1, k_2, \alpha)\\
&:= \overline{OP_8^{k_0 k_1, k_2}}(X, Y) \,\|\, g_1(\alpha)\\
&= g_0\left(OP_8\left(f_0^{-1}(X)\oplus k_0, f_1^{-1}(Y)\oplus k_1\right)\oplus k_2 \oplus \alpha\right) \| g_1(\alpha)
\end{aligned}
$$

상기 식에서 X, Y는 각각 8비트이며, α는 8비트 랜덤 마스킹 값이다. 16×16 테이블 참조 함수는 Fig. 11.와 같으며, OP_8은 XOR_8, OR_8, AND_8 함수를 사용한다.

Masked WB R-Layer에서 사용되는 마스킹 테이블은 $n\times 2n$ 테이블을 사용한다. $n\times 2n$ 테이블 참조 함수에서는 n비트 라운드키 r_0, r_1 XOR과 n비트 ROL_8 연산을 수행하는 테이블 $ROL_8^{r_0 r_1}$에 n비트 인코딩을 추가한 테이블 $\overline{ROL_8^{r_0 r_1}}$을 사용한다.

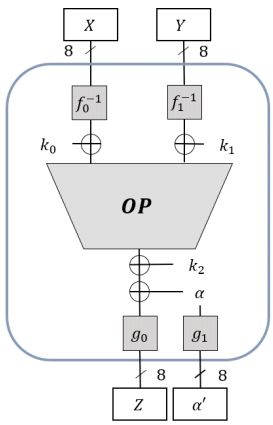


Fig. 11. $MT\_16\times 16\_OP_8$

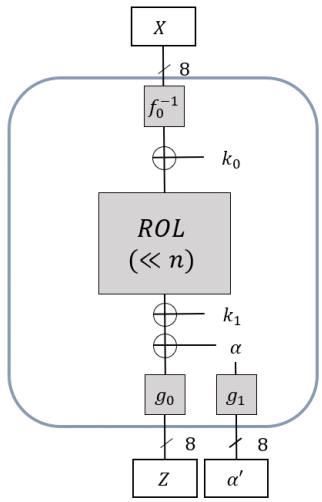


Fig. 12. $MT\_8\times 16\_ROL_8$

이를 이용해 8비트 라운드키 k_0, k_1, k_2 XOR과 ROL_8 연산에 인코딩을 적용한 연산 $MT\_8\times 16\_ROL_8(X, Y, k_0, k_1, \alpha)$ 을 다음과 같이 정의한다.

$$
\begin{aligned}
&MT\_8\times 16\_ROL_8(X, l, k_0, k_1, \alpha)\\
&:= \overline{ROL_8^{k_0 k_1, k_2}}(X, l) \,\|\, g_1(\alpha)\\
&= g_0\left(ROL_8\left(f_0^{-1}(X)\oplus k_0, l\right)\oplus k_1 \oplus \alpha\right) \| g_1(\alpha)
\end{aligned}
$$

상기 식에서 X는 8비트이며, α는 8비트 랜덤 마스킹 값이다. 8×16 테이블 참조 함수는 Fig. 12.와 같다.

4.3.2 마스킹 제거 테이블

Masked WB S-Layer와 Masked WB R-Layer에서 사용된 $MT\_16\times 16\_OP_8$ 와 $MT\_8\times 16\_ROL_8$ 테이블 참조 함수에 대한 마스킹을 제거하는 테이블인 마스킹 제거 테이블이 함께 사용된다. 마스킹 제거 테이블 $UMT_16\times 8$을 다음과 같이 정의한다.

$$UMT\_16\times 8(X, Y) := g_0(XOR_8(f_0^{-1}(X), f_1^{-1}(Y)))$$

상기 식에서 X, Y는 각각 8비트이다. $UMT\_16\times 8$

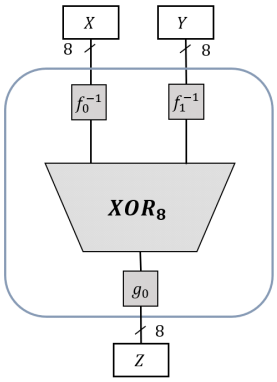


Fig. 13. *UMT*_16×8

테이블 참조 함수는 Fig. 13.와 같다.

V. 성능 측정 결과 및 활용 사례

본 장에서는 본 논문에서 제안하는 WB-PIPO의 성능을 측정한 결과를 제시하고 이를 WB-AES와 비교한다. 또한 WB-PIPO를 N사에서 출시한 모바일 보안 소프트웨어 D제품에 적용한 결과를 제시한다.

5.1 성능 측정 결과

본 절에서는 WB-PIPO의 성능을 측정하고 이를 Chow의 화이트박스와 비교한 결과를 제시한다. Table 4.는 본 논문에서 사용한 성능 측정 환경이다.

성능 측정 단위는 CPB(Cycles Per Byte)이며 CPB 측정 방법은 [10]을 참고하였다. WB-PIPO와 WB-AES의 성능을 비교하기 위해 128비트 키 길이를 사용하였다. Fig. 14.는 128비트 키를 사용한 WB-PIPO와 WB-AES의 암호화 및 복호화 성능 측정 결과를 나타낸 것이다.

Fig. 14.에서 WB-PIPO 128의 암호화와 복호화 CPB는 각각 1184.75, 1208.25이고 WB-AES 128의 암호화와 복호화 CPB는 각각 20242.62, 20808.73이다. 성능 측정 결과로부터 WB-PIPO는 WB-AES 대비 약 17배 빠르다는 것을 알 수 있다.

Table 4. Test environment

CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
OS	Ubuntu 16.04.2 LTS
RAM	16GB
Language	C
compiler	gcc version 5.4.0 20160609

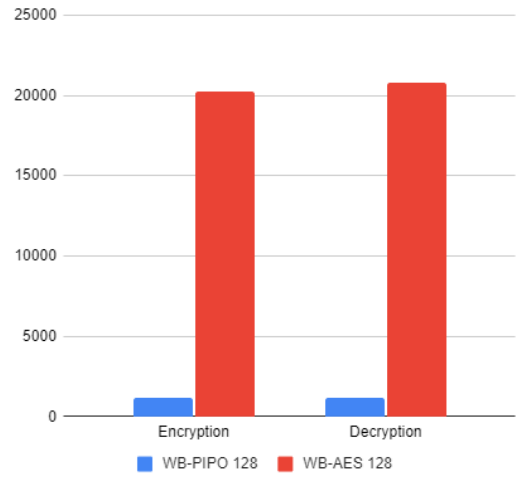


Fig. 14. Performance of WB-PIPO 128 and WB-AES 128 (unit: CPB)

WB-AES의 경우 동일한 키 크기의 WB-PIPO 대비 라운드 수가 적지만 8비트 입력, 32비트 출력 또는 128비트 출력 테이블을 사용하므로 이를 참조한 결과를 다시 취합하는 XOR 테이블을 사용하기 때문에 테이블의 크기가 크고 테이블을 참조하는 횟수도 증가한다. 그러나 WB-PIPO는 8비트 입력, 4비트 출력 테이블을 사용하므로 테이블을 참조한 값이 바로 결과값이 되기 때문에 테이블의 크기가 작고 테이블을 참조하는 횟수도 감소한다.

5.2 모바일 보안제품 활용 사례

N사의 모바일 보안 소프트웨어 D제품은 서버와 클라이언트 간의 키 합의 프로토콜을 이용해 비밀키를 공유한다. 클라이언트가 정책파일을 요청하면 서버는 블록 암호 SEED를 이용해 암호화한 정책 파일을 클라이언트에게 전송한다. 클라이언트는 공유한 비밀키를 이용해 SEED로 복호화한다. 본 절에서는 모바일 보안 소프트웨어 D제품에 WB-PIPO를 적용한 두 가지 케이스에 대한 성능을 측정한 결과를 기술한다.

첫 번째 테스트 케이스는 클라이언트에서 사용하는 비밀키를 WB-PIPO로 복호화하는 것이다. 앱

내에서는 암호화된 정책 파일을 SEED로 복호화할 때 비밀키를 사용하므로 비밀키가 메모리에 노출될 가능성이 높다. 비밀키를 안전하게 보호하기 위해 서버에서는 정책파일을 암호화하는데 사용한 비밀키를 WB-PIPO로 암호화하여 암호화된 정책 파일과 키를 클라이언트에 전송한다. 클라이언트는 암호화된 비밀키를 WB-PIPO로 복호화한 후 SEED로 정책 파일을 복호화한다.

두 번째 테스트 케이스는 키가 아닌 정책 파일을 WB-PIPO를 이용해 서버에서 암호화하고 클라이언트에서 복호화한다. 첫 번째 테스트 케이스와 다른 점은 정책 파일을 SEED가 아닌 WB-PIPO를 이용해 암호화와 복호화를 진행하므로 서버와 클라이언트는 비밀키를 공유할 필요가 없다.

WB-PIPO 성능 측정에 사용한 운영모드는 CBC 모드이며 PKCS 7 패딩을 사용했다. D제품에 사용한 블록 암호 SEED의 키 길이는 128비트이고 정책 파일의 크기는 427바이트이다. 각 테스트 케이스의 시간 측정 범위는 다음과 같다.

- 키 복호화: WB-PIPO를 통한 키 복호화 후 SEED 복호화 완료 시점까지의 시간을 측정
- 내용 복호화: 정책 파일에 대하여 WB-PIPO를 통한 복호화 완료 시점까지의 시간을 측정
- 기존: 정책 파일에 대하여 SEED를 통한 복호화 완료 시점까지의 시간을 측정

Table 5.는 테스트 케이스 별 성능 측정에 사용한 실험 환경을 나타낸 것이다.

Fig. 15.는 상기 테스트 케이스를 1회당 10,000번 수행했으며 이를 총 10회 수행한 평균 시간을 나타낸 것이고 단위는 나노세컨드다. 갤럭시 S21 Ultra 5G, 화웨이 HORNOR V10, 그리고 갤럭시 S7에서 테스트 별 시간을 측정한 결과 기존 대비 키 복호화는 각각 1.8배, 2.2배, 1.9배 시간이 소요

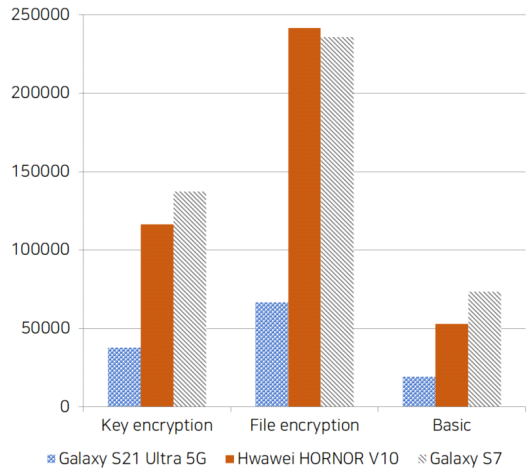


Fig. 15. Performance of WB-PIPO (Key encryption, File encryption, Basic test) (unit: nano second)

되었다. 또한 기존 대비 내용 암호화의 경우 각각 3.5배, 4.6배, 3.2배 시간이 소요됨을 알 수 있다

화이트박스 암호의 경우 테이블에서 데이터를 가져오고 이를 연산하는 과정에서 기존 블록 암호보다 연산 시간이 더 소요되므로 암호화할 데이터의 크기가 커질수록 연산 시간이 증가한다. 따라서 첫 번째 테스트와 같이 비밀키를 화이트박스 암호로 안전하게 보호하거나 크기가 작은 데이터의 암·복호화에 화이트박스 암호를 활용할 것을 권장한다.

VI. 결 론

본 논문에서는 ICISC 2020에 제안된 경량 블록 암호 PIPO를 대상으로 화이트박스 암호 기법을 적용한 WB-PIPO를 제안했다. WB-PIPO는 PIPO의 8비트 단위 비트 연산을 라운드 키가 포함된 두

Table 5. Test environment

Model	Galaxy S21 Ultra 5G	Hwawei HORNOR V10	Galaxy S7
Android Version	11	9	8
API level	30	28	26
CPU	Samsung Exynos 10 Series (2100) SoC (ARM Cortex-X1 MP1 2.91 GHz + ARM Cortex-A78 MP3 2.81 GHz + ARM Cortex-A55 MP4 2.21 GHz)	Kirin 970-HiSilicon Octa core (4x2.4GHz Cortex-A73, 4x1.8GHz Cortex-A53)	Samsung Exynos 8 Octa (8890) SoC (Samsung Exynos M1 MP4 2.29 GHz + ARM Cortex-A53 MP4 1.59 GHz)

개의 4비트 단위 비트 연산으로 변환하여 설계했다. WB-PIPO 128과 WB-AES 128을 비교하였을 때, 테이블의 크기는 약 5.8배 작았으며, CPB 단위 기준 약 17배 빠른 것을 알 수 있었다. WB-AES는 테이블을 참조한 결과값을 취합하는 과정이 포함되는 반면 WB-PIPO는 테이블 참조 값에 대한 별도의 취합 과정을 포함하지 않기 때문에 이와 같은 결과가 나타난 것으로 분석된다.

또한 본 논문에서 제안한 WB-PIPO를 모바일 보안 소프트웨어 D제품에 적용한 결과를 제시한다. 화이트박스 암호는 연산에 필요한 데이터를 테이블에서 가져오기 때문에 기존 블록암호 대비 연산 시간이 더 소요됨으로 WB-PIPO를 이용해 동일한 정책 파일을 암호화했을 때 연산 시간이 약 3.7배 증가했고, 키를 보호하기 위해 WB-PIPO를 사용했을 때 연산 시간이 약 1.9배 증가한 것을 확인했다.

화이트박스 구현은 암호 알고리즘에 주입된 비밀키에 대해 테이블화를 통해 비밀키가 메모리에 직접적으로 로드되지 않도록 하고 난독화 과정을 통하여 비밀 값을 보호하는 구현 방법이다. 또한 코드리프팅, 리버스 엔지니어링, 부채널 분석과 같은 공격 기법들에 대해서도 안전성을 고려하여 구현할 수 있는 기법이다. 이러한 화이트박스 구현 방식은 소프트웨어 기술만으로 비밀키를 안전하게 보호하는 측면에서 유의미한 기술이다. 본 연구진은 WB-PIPO에 대한 실질적으로 유효한 공격과 고차마스킹 기법과 같은 대응 방안에 대해 분석 및 구현할 예정이며, 연산 최적화를 통한 기능 향상을 목표로 연구를 진행하고 있다. 이러한 화이트박스 구현 기법을 적용한 암호 기술의 연구 및 개발과 더불어 안전성 검증과 같은 기준 등의 연구가 활발히 이뤄져 차후 실제 산업화 제품에 적용될 것을 기대한다.

References

[1] H. G. Kim, et al., "A new method for designing lightweight S-boxes with high differential and linear branch numbers, and its application," International Conference of Imformation Security and Cryptology, pp. 105-132, Dec 2020.

[2] S. Chow, P. Eisen, H. Johnson and P.C.V. Oorschot, "White-Box Cryptography and an AES Implementation," Selected Areas in Cryptography, LNCS 2595, pp. 250 - 270, 2002.

[3] O. Billet, H. Gilbert and C. Ech-Chatbi, "Cryptanalysis of a White Box AES Implementation," Selected Areas in Cryptography, LNCS 3357, pp. 227-240, 2005.

[4] Y. Xiao, X. Lai, "A Secure Implementation of White-Box AES," Computer Science and its Applications, pp. 1-6, Dec 2009.

[5] M. Karroumi, "Protecting White-Box AES with Dual Ciphers," International Conference of Imformation Security and Cryptology, pp. 278-291, 2010

[6] Y.D. Mulder, P. Roelse and B. Preneel, "Cryptanalysis of the Xiao - Lai White-Box AES Implementation," Selected Areas in Cryptography, LNCS 7707, pp. 34-49, 2012.

[7] T. Lepoint, M. Rivain, Y.D. Mulder, P. Roelse & B. Preneel, "Two Attacks on a White-Box AES Implementation," Selected Areas in Cryptography, LNCS 8282, pp. 265-285, 2013.

[8] J.W. Bos, C. Hubian, W. Michels and P. Teuwen, "Differential computation analysis: Hiding your white-box designs is not enough," Cryptographic Hardware and Embedded Systems, pp. 215-236, August 2016.

[9] R.A. Fisher and F. Yates, Statistical tables for biological, agricultural and medical research, 3rd Ed. London: Oliver & Boyd. pp. 26 - 27. 1938.

[10] Korea cryptography forum, "Reference for LSH Implementation Competition", https://kcryptoforum.or.kr/web/Board/342/detailView.do, June 2022.

〈저 자 소 개〉

함 은 지 (Eunji Ham) 정회원
2017년 2월: 국민대학교 수학과 졸업
2019년 2월: 국민대학교 금융정보보안학과 석사
2018년 2월~2022년 7월: NSHC 연구원
〈관심분야〉 화이트박스 암호, 분산 신원증명, 양자내성암호

이 영 도 (Youngdo Lee) 정회원
2021년 2월: 국민대학교 수학과 졸업
2020년 12월~현재: NSHC 연구원
〈관심분야〉 정수론, 암호학, 양자내성암호

윤 기 순 (Kisoon Yoon) 정회원
1998년 8월: 경희대학교 수학과 이학사
2007년 8월: 고려대학교 정보보호학과 공학석사
2013년 11월: Universite de Caen 수학과 이학박사
2013년 11월~현재: NSHC 암호기술연구소 연구소장
〈관심분야〉 정수론, 암호학, 정보보호