블록암호 LEA의 음함수 기반 화이트박스 구현 기법에 관한 연구

류지은[1], 강주성[1,2], 염용진[1,2]*
국민대학교 금융정보보안학과[1] / 정보보안암호수학과[2]
{ofryuji, jskang, *salt}@kookmin.ac.kr

Implicit White-Box Implementation of Block Cipher LEA

Jieun Ryu[1], Ju-sung Kang[1,2], Yongjin Yeom[1,2]*
Dept. of Financial information security[1] /
Information Security, Cryptology, and Mathematics[2], Kookmin Univ.

요 약

본 논문은 2022년 Ranea 등이 제안한 새로운 화이트박스 구현 기법인 ARX 구조의 블록암호에 대한 음함수 기반 화이트박스 구현 기법을 블록암호 LEA에 적용하고 구현 가능성을 분석한다. LEA는 Ranea 등이 구현에 활용한 블록암호 Speck와 비교할 때 동일한 보안 파라미터에 대하여 적은 수의 라운드로 구성되므로, 화이트박스 구현에 요구되는 메모리가 적다. 또한, 한 라운드의 구조가 복잡하여 보다 높은 안전성을 제공할 것으로 기대된다. 라운드 입출력 길이와 비밀키 길이가 128 비트일 때, 음함수 기반 LEA 화이트박스 구현에 필요한 메모리는 Speck을 사용할 때보다 25% 적은 540.11MB이며, 안전성은 $O(2^{46})$ 이상일 것으로 예상된다.

Ⅰ. 서 론

화이트박스는 암호 서비스의 소프트웨어에 접근 권한을 가진 공격자로부터 비밀키를 보호하기 위해 연구된 구현 기법이다. 2002년 Chow 등은 고정된 비밀키에 대한 AES 암호화 알고리즘을 인코딩된 테이블 형식으로 구현하는 CEJO 프레임워크(framework)를 제안했다[1]. 그러나 해당 구조는 큰 비선형 인코딩을 적용할 수 없다는 한계가 있으며, 이후 CEJO 프레임워크에 기반하여 화이트박스 구현에 관하여 많은 연구가 이루어졌으나, 공개된 구조는 모두 안전하지 않음이 밝혀졌다. Ranea 등은 2020년에 AES와 같이 substitution permutation network(SPN) 구조를 갖는 블록암호의 새로운 화이트박스 구현 기법으로 라운드 함수에 대한 자기 동치(self- equivalence) 쌍을 이용한 화이트박스 구현 가능성을 연구했으나, 안전성 개선에는 기여하지 못했다[2].

2022년 Ranea 등은 새로운 화이트박스 구현 기법으로 addition rotation XOR(ARX) 구조를 갖는 블록암호의 음함수(implicit function) 기반 화이트박스 구현 방식을 제안하고[3], 블록암호 Speck을 제안한 방식의 화이트박스로 구현하여 성능을 측정하였다. 음함수 기반의 화이트박스 구현은 기존 비선형 인코딩 기반 화이트박스나 자기 동치 쌍 기반 화이트박스에 수행된 공격에 저항성을 가지므로, 더 높은 안전성을 가지는 화이트박스 구현이 가능할 것으로 기대된다. 본 논문에서는 해당 연구에 기반하여 블록암호 LEA에 대한 음함수 기반 화이트박스 구현 가능성을 연구한다. 또한, 제시한 음함수 기반 LEA 화이트박스 구현 시 필요할 메모리와 안전성을 분석한다.

Ⅱ. ARX 블록암호의 음함수 기반 화이트박스 구현 [3]

ⅰ. 음함수와 자기 동치

정의 1. n비트 $x,y\in \mathrm{GF}(2)^n$에 대하여 양함수(explicit function) F가 $y=F(x)$일 때, 다음을 만족하는 함수 P를 F의 음함수라고 한다.

$$P(x,y)=0 \Leftrightarrow y=F(x)$$

주어진 x에 대하여 양함수는 x를 대입하여 계산한 결과로 y를 얻고, 음함수는 x를 대입한 뒤 y에 대한 연립방정식을 풀어 그 해로 y를 구한다.

정의 2. 다음을 만족하는 치환함수(permutation) 쌍 (A,B)를 함수 F의 자기 동치라고 한다.

$$F(x) = (B\circ F\circ A)(x)$$

ⅱ. 블록암호 Speck의 음함수 화이트박스 구현

Speck의 암호화 함수 E는 라운드 키 덧셈 연산과 순환(rotation) 연산을 수행하는 선형 레이어 $L^{(i)}$와 모듈러 덧셈(modular addition) 연산을 수행하는 비선형 레이어 S로 구성된 라운드 함수 $E^{(i)}$의 합성 함수이다.

$$E^{(i)}=L^{(i)}\circ S \quad (i=1,2,\dots,r)$$
$$E=E^{(r)}\circ E^{(r-1)}\circ\cdots\circ E^{(1)}$$

이때 임의의 $x,x'\in \mathrm{GF}(2)^{n/2}$에 대하여 S는 다음과 같다.

$$S(x,x')=(x\boxplus x',\ x')$$

라운드 함수는 자기 동치 쌍을 통해 인코딩된다. 1차 치환함수 $A^{(i)}$와 2차 치환함수 $B^{(i)}$에 대하여 $(A^{(i)},B^{(i)})$가 라운드 함수 $E^{(i)}$의 자기 동치 쌍일 때, $(A^{(i)},B^{(i)})$와 1차 치환함수 $(C^{(i)})^{-1},C^{(i+1)}$을 합성하여 인코딩한 라운드 함수 $\overline{E^{(i)}}$ 및 이에 대한 음함수 $P^{(i)}$는 다음과 같다.

$$\overline{E^{(i)}} = C^{(i+1)}\circ E^{(i)}\circ A^{(i)}\circ B^{(i-1)}\circ (C^{(i)})^{-1}$$
$$P^{(i)}=U^{(i)}\circ T\circ V^{(i)}\circ (Id,(L^{(i)})^{-1})\circ (I^{(i)},O^{(i)})$$

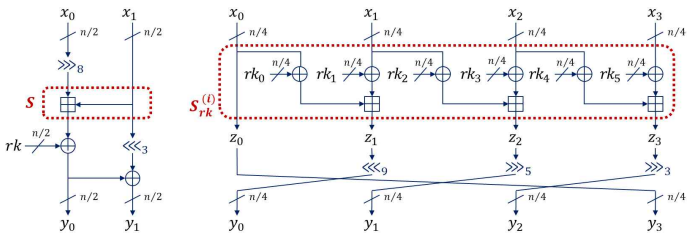
이때 T는 S의 음함수 $T(x,x',y,y')=((x\boxplus x')\oplus y,\ x'\oplus y')$이고, $(U^{(i)},V^{(i)})$는 T의 자기 동치 쌍이며, $I^{(i)}=A^{(i)}\circ B^{(i-1)}\circ (C^{(i)})^{-1}$, $O^{(i)}=(C^{(i+1)})^{-1}$이다. 또한, 임의의 $x,x',y,y'\in \mathrm{GF}(2)^{n/2}$에 대하여 T는 모듈러 덧셈 연산을 이차 함수 Q로 변환하여 표현할 수 있다.

$$Q(x,x') = (0, x_0x_0',\ \dots, x_0x_0'\oplus\cdots\oplus x_{n/2-2}x_{n/2-2}')$$
$$T(x,x',y,y')=(x\oplus x'\oplus y\oplus Q(x\oplus y,x'\oplus y),\ x'\oplus y')$$

Ⅲ. LEA의 음함수 기반 화이트박스 구현

ⅰ. ARX 기반 블록암호 LEA [4]

LEA는 Speck이 라운드 입력을 두 블록으로 나눠 연산하는 것과 달리 라운드 입력을 네 블록으로 나눠 연산을 수행한다. 또한 Speck이 한 라운드에 블록 길이의 라운드 키 하나를 사용하는 것과 달리 LEA는 블록 길이의 라운드 키 여섯 개를 사용한다. Speck과 LEA의 한 라운드 구조를 비교하면 [그림 1]과 같다.



[그림 1] (좌) Speck의 라운드 함수 (우) LEA의 라운드 함수

Speck은 라운드 시작부의 x_0에 수행되는 순환 연산을 이전 라운드의 종료부로 이동시킴으로써 한 라운드를 $E^{(i)} = L^{(i)} \circ S$로 표현 가능하다. 그러나 LEA의 경우, 위와 같은 방법을 적용하기 위하여 라운드 키 덧셈 연산을 이전 라운드로 이동시키면 라운드 입출력의 길이가 n에서 $7n/4$로 증가한다. 이는 음함수 화이트박스 구현 시 저장해야 하는 방정식의 크기를 약 9배 증가시키므로 화이트박스 구현에 적합하지 않다. 따라서 LEA의 음함수 기반 화이트박스 구현은 라운드 키 덧셈 연산을 S 레이어에 포함시킨다.

ii. LEA의 음함수 화이트박스 구현

LEA의 라운드 키 $rk_0, rk_1, rk_2, rk_3, rk_4, rk_5$에 대한 덧셈 연산 레이어를 $K^{(i)}$는 다음과 같다.

$$K^{(i)}(x_0, x_1, x_2, x_3)$$
$$= (x_0, x_0 \oplus rk_0, x_1 \oplus rk_1, x_1 \oplus rk_2, x_2 \oplus rk_3, x_2 \oplus rk_4, x_3 \oplus rk_5)$$

그리고 라운드 키를 포함하는 비선형 레이어 $S_{rk}^{(i)}$와 이에 대한 음함수 $T_{rk}^{(i)}$를 다음과 같이 정의한다. $z_i\,(i = 0,1,2,3)$는 $S_{rk}^{(i)}$ 레이어 연산 결과를 표현하는 기호이다.

$$S_{rk}^{(i)}(x_0, x_1, x_2, x_3)$$
$$= (S \circ K^{(i)})(x_0, x_1, x_2, x_3)$$
$$= (x_0, (x_0 \oplus rk_0) \boxplus (x_1 \oplus rk_1),$$
$$(x_1 \oplus rk_2) \boxplus (x_2 \oplus rk_3), (x_2 \oplus rk_4) \boxplus (x_3 \oplus rk_5))$$
$$= (z_0, z_1, z_2, z_3)$$

$$T_{rk}^{(i)}(x_0, x_1, x_2, x_3, z_0, z_1, z_2, z_3)$$
$$= (x_0 \oplus z_0,$$
$$x_0 \oplus x_1 \oplus rk_0 \oplus rk_1 \oplus z_1 \oplus Q(x_0 \oplus rk_0 \oplus z_1,\ x_1 \oplus rk_1 \oplus z_1),$$
$$x_1 \oplus x_2 \oplus rk_2 \oplus rk_3 \oplus z_2 \oplus Q(x_1 \oplus rk_2 \oplus z_2,\ x_2 \oplus rk_3 \oplus z_2),$$
$$x_2 \oplus x_3 \oplus rk_4 \oplus rk_5 \oplus z_3 \oplus Q(x_2 \oplus rk_4 \oplus z_3,\ x_3 \oplus rk_5 \oplus z_3))$$
$$= (0,0,0,0)$$

상기 식들을 통해 표현한 인코딩한 LEA의 i번째 라운드 함수의 음함수 $P_{LEA}^{(i)}$는 다음과 같다.

$$P_{LEA}^{(i)} = U^{(i)} \circ T_{rk}^{(i)} \circ V^{(i)} \circ (Id, (L_{LEA}^{(i)})^{-1}) \circ (I^{(i)}, O^{(i)})$$

이때 $L_{LEA}^{(i)}$는 라운드 키 덧셈 연산 이후 수행되는 LEA 라운드 함수의 선형 레이어이다. 또한, 상기 식에서 $(U^{(i)}, V^{(i)})$는 $T_{rk}^{(i)}$의 자기 동치 쌍이며, LEA 라운드 함수의 자기 동치 쌍 $(A_{LEA}^{(i)}, B_{LEA}^{(i)})$에 대하여 $I^{(i)} = A_{LEA}^{(i)} \circ B_{LEA}^{(i-1)} \circ (C^{(i)})^{-1}$, $O^{(i)} = (C^{(i+1)})^{-1}$이다.

iii. 음함수 기반 LEA 화이트박스의 효율성

음함수 기반 화이트박스 구현은 암호화 및 복호화 함수의 각 라운드를 인코딩된 개별 연립 방정식의 형태로 저장하기 때문에, 구현에 많은 메모리를 소모한다. 그러나 이 음함수 방정식의 크기는 라운드의 입출력 길이가 아닌 다항식의 차수에 따라 결정되므로, 인코딩 크기에 제약이 있는 CEJO 프레임워크에 기반한 화이트박스 구현 방식과 달리 인코딩을 큰 인코딩을 사용하여 안전성을 높일 수 있다는 장점이 있다.

또한, Speck의 라운드 함수가 입력의 절반을 그대로 출력하는 것과 달리 LEA의 라운드 함수는 입력의 1/4만을 라운드 키 덧셈 연산이나 모듈러 덧셈 연산 없이 출력한다. 따라서 LEA 라운드 함수의 음함수는 Speck 라운드 함수의 음함수보다 더 많은 항으로 표현되며, 이는 공격 복잡도를 증가시켜 더 높은 안전성 제공하는 데 기여할 것으로 기대된다.

Ranea 등이 구현한 Speck 화이트박스는 x에 대하여 2차, y에 대하여 1차인 3차 방정식을 사용할 때, 64비트 라운드 입출력을 기준으로 한 라운드 음함수를 저장하는 데 1.42MB, 전체 라운드를 저장하는 데 38.22MB의 메모리를 요구한다. 그러나 본 논문에서는 LEA와 Speck의 효율성 비교를 위해, LEA의 최소 입출력 길이인 128 비트를 기준으로 두 알고리즘의 전체 라운드 함수 구현에 필요한 메모리 크기를 비교한다. 제시한 방식으로 LEA 화이트박스를 구현할 경우, 한 라운드 구현에 필요한 메모리 크기는 22.5MB이다. Speck은 동일한 라운드 입출력 길이를 기준으로 LEA보다 많은 라운드 수행을 요구하며, 128 비트 비밀키 기준으로 LEA 화이트박스는 Speck 화이트박스 구현에 필요한 메모리의 약 3/4만으로 구현 가능하다. 이때 Speck 화이트박스의 안전성은 $O(n^9) = O(2^{46})$으로[5], LEA 화이트박스 안전성은 이보다 높을 것으로 예상된다. 두 블록암호에 대하여 전체 라운드에 대한 음함수 기반 화이트박스 구현에 필요할 것으로 예상되는 메모리 크기는 [표 1]과 같다.

입출력 길이	비밀키 길이	구분	라운드 수	메모리 크기
64	128	Speck	27	38.22
128	128	Speck	32	720.14
		LEA	24	540.11

[표 1] Speck과 LEA의 비밀키 길이에 따른
음함수 기반 화이트박스 구현의 메모리 사용량(MB)

한편, 음함수 기반 화이트박스의 동작 속도는 인코딩된 라운드 함수의 연립방정식을 풀기 위해 수행하는 가우스 소거법(Gaussian Elimination)의 연산량에 가장 큰 영향을 받는다. 가우스 소거법의 연산량은 가우스 소거 행렬 크기 t에 의해 결정되며, LEA와 Speck의 음함수 기반 화이트박스는 동일한 라운드 입출력 길이를 기준으로 가우스 소거 행렬의 크기를 결정하는 y 항의 개수가 같기 때문에 한 라운드를 수행하는 데 걸리는 시간이 비슷할 것으로 예상된다. 라운드 입출력 길이가 128 비트일 때 가우스 소거법 연산량은 $O(2^{21})$으로 다항시간 내에 실행 가능하다.

Ⅳ. 결론

음함수 기반 화이트박스 구현은 아직 안전성에 대한 증명이 충분히 이뤄지지 않은 연구 분야이다. 그러나 음함수 기반 화이트박스는 CEJO 프레임워크에 기반한 화이트박스나 자기 동치 쌍 기반 화이트박스에 수행된 공격들에 저항성을 가진다.

LEA는 Speck보다 라운드 구조가 복잡하고 동일한 보안 파라미터에 대하여 적은 라운드로 구성되므로, 특정 안전성을 목표로 음함수 기반 화이트박스를 구현할 때 비교적 적은 메모리를 사용하면서 높은 안전성을 제공할 수 있을 것으로 기대된다. 향후 연구에서는 LEA의 음함수 기반 화이트박스에 대한 구체적인 안전성 분석을 수행하고, 실제 구현을 통한 동작 성능을 평가하여 모바일 거래 시스템 등의 적용 가능성을 분석할 계획이다.

ACKNOWLEDGMENT

본 연구는 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2021-0-00046, 국가공공정보시스템 안전성 및 활용성 제고를 위한 차세대 암호체계 개발)

참 고 문 헌

[1] S. Chow, P. Eisen, H. Johnson, & P. C. Van Oorschot, “White-box cryptography and an AES implementation,” SAC 2002, Aug. 2002, Revised Papers 9, pp. 250-270, Springer, 2003.

[2] A. Ranea, & B. Preneel, “On self-equivalence encodings in white-box implementations,” SAC 2020, Oct. 2020, Revised Selected Papers 27, pp. 639-669, Springer, 2021.

[3] A. Ranea, J. Vandersmissen, & B. Preneel, “Implicit white-box implementations: White-boxing ARX ciphers,” CRYPTO 2022, pp. 33-63, Springer, 2022.

[4] D. Hong, J. K. Lee, D. C. Kim, D. Kwon, K. H. Ryu, & D. G. Lee, “LEA: A 128-bit block cipher for fast encryption on common processors,” WISA 2013, Aug, 2013,, Revised Selected Papers 14, pp. 3-27, Springer, 2014.

[5] A. Biryukov, B. Lambin & A. Udovenko, “Cryptanalysis of ARX-Based White-Box Implementations”. 2023. TCHES 2023 (3): 97-135.