음함수 기반 화이트박스 암호에 사용되는 행렬 연산 구현에 관한 연구

김예진[1], 고정빈[2], 강주성[1,2], 염용진[1,2]*
국민대학교 정보보안암호수학과[1] / 금융정보보안학과[2]
{alice1225kim, wjdqls1025, , jskang, *salt}@kookmin.ac.kr

A Study on the Implementation of Matrix Operation used for Implicit Whitebox Cryptography

Yejin Kim[1], Jeongbeen Ko[2], Ju-sung Kang[1,2], Yongjin Yeom[1,2]*
Dept. of Information Security, Cryptology, and Mathematics[1]
Financial information security[2], Kookmin Univ.

요 약

2002년 Chow 등이 최초로 제안한 테이블 참조 기반 화이트박스 암호 설계 방식을 CEJO framework라 하며, CEJO framework 기반 화이트박스 암호는 모두 안전하지 않음이 증명되었다. 2022년 Renea 등은 기존의 방식과는 다르게, 음함수 방정식을 풀어 해를 구하는 새로운 화이트박스 암호 설계 방식을 제안하였다. 본 논문은 음함수 기반 화이트박스 암호에 필요한 방정식의 저장 방식과 가우스 소거 연산으로 방정식의 해를 구하기 위한 확장 이진 행렬을 생성하는 방법에 대해 제안하고, 입·출력이 64비트인 음함수 방정식 연산의 구현 예시와 속도 측정 결과를 제시한다.

Ⅰ. 서 론

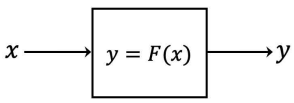
화이트박스 암호는 공격자로부터 암호 알고리즘의 암호 키를 유추할 수 없도록 키를 보호하는 기술이다. 2002년 Chow 등이 제안한 화이트박스 공격자 모델에서 공격자는 디바이스를 완전히 제어하고, 알고리즘의 중간 연산 값을 임의로 조정할 수 있다. Chow 등은 이러한 공격자에 대응하기 위해 화이트박스 암호를 최초로 설계하였고, 이러한 설계 방식을 CEJO framework라 한다. CEJO framework는 모든 입출력에 대해 랜덤 인코딩을 적용한 테이블을 저장하여 연산 시 테이블 참조 방식을 사용한다[1]. CEJO framework를 기반으로 하는 화이트박스 암호는 모두 안전하지 않음이 증명되었다[2].

2022년 Renea 등은 새로운 화이트박스 암호 설계 방식인 음함수 기반 화이트박스 암호를 제안하였다[3]. 본 논문에서는 음함수 기반 화이트박스 암호의 다항식 연산을 효율적으로 구현하기 위한 방정식 저장 방식과 행렬 연산을 위한 확장 이진 행렬의 구성 방법을 제안하고, 행렬 연산의 구현 결과를 제시한다.

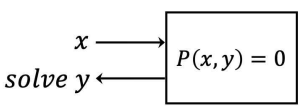
Ⅱ. 음함수 기반 화이트박스 암호[3]

정의 1. $\mathbb{F}_2^n$이 n비트 값에 대한 벡터 공간일 때, $\mathbb{F}_2^n$에서 $\mathbb{F}_2^m$로의 함수는 (n,m)비트 함수이고, $m=n$이면 n비트 함수이다.

정의 2. F가 n비트 함수일 때, $P(x,y)=0 \Leftrightarrow y=F(x)$를 만족하면, $(2n,m)$비트 함수 P는 F에 대한 음함수이다.



〔그림 1〕 양함수의 구조



〔그림 2〕 음함수의 구조

양함수(explicit)는 [그림 1]과 같이 n비트 입력 x를 함수 $F(x)$에 대입하여 n비트 출력 y를 구한다. 음함수(implicit)는 [그림 2]와 같이 n비트 입력 x를 음함수 $P(x,y)=0$에 대입하여 방정식을 풀어 n비트 출력 y를 구한다. CEJO framework 기반 화이트박스 암호는 모든 입출력에 대해 사전 계산 테이블을 저장하고, 이를 참조하여 주어진 입력에 대한 출력을 구한다. 반면, 음함수 기반 화이트박스 암호는 음함수 방정식을 저장하고, 입력값을 방정식에 대입하여 y에 대한 가우스 소거 연산으로 방정식을 풀어 해를 구해 출력한다.

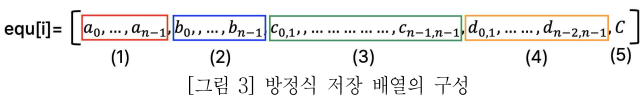
Ⅲ. 음함수 구현을 위한 방정식 저장 방식

ⅰ. 음함수 구현을 위한 방정식 저장 방식

본 논문에서 다루는 음함수 방정식은 차수를 최대 2차로 제한하고, x의 최대 차수는 2차, y의 최대 차수는 1차로 제한한다. 따라서, n비트 입력 x,y에 대한 음함수 $P(x,y)=0$는 $a_i, b_i, c_{ij}, d_{ij}, C \in \{0,1\}$에 대하여 식 (1)을 따른다. x가 주어질 때 방정식의 해를 찾기 위해 필요한 음함수 방정식은 최소 n개이다.

$$P(x,y)=\sum_{i=0}^{n-1} a_i x_i + \sum_{i=0}^{n-1} b_i y_i + \sum_{i,j=0}^{n-1} c_{ij} x_i y_j + \sum_{i<j}^{n-1} d_{ij} x_i x_j + C = 0 \quad (1)$$

방정식 하나를 저장할 때, [그림 3]과 같이 배열을 구성한다.



[그림 3] 방정식 저장 배열의 구성

n비트 입력 x에 대해 n비트 y를 출력하는 음함수 방정식 배열 equ[i]를 구성하는 계수의 종류는 [표 1]과 같다.

	미지수	계수	크기(비트)
(1)	x_i	$a_i\,(i=0,1,\ldots,n-1)$	n
(2)	y_i	$b_i\,(i=0,1,\ldots,n-1)$	n
(3)	$x_i y_j$	$c_{ij}\,(i,j=0,1,\ldots,n-1)$	n^2
(4)	$x_i x_j$	$d_{ij}(i<j)$	$\binom{n}{2}$
(5)	C	$C\,(C\in\{0,1\})$	1

[표 1] 방정식 테이블을 구성하는 미지수에 대한 계수의 종류 및 배열 내 크기

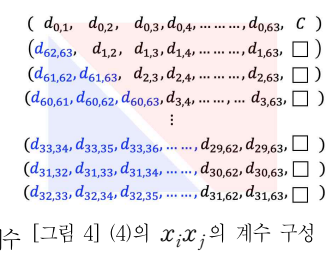
배열은 방정식의 계수 a_i, b_i, c_{ij}, d_{ij}와 상수 C를 저장하고 있고, 각 항의 계수는 0 또는 1의 값을 가지므로 한 비트로 표현 가능하다. 따라서, 하나의 방정식을 저장할 때 필요한 최소 메모리는 $n+n+n^2+\binom{n}{2}+1 = \frac{3}{2}n^2+\frac{3}{2}n+1$ 비트이다.

ii. 음함수 방정식 저장 방식의 예

[그림 3]과 같이 배열을 구성하고, 자료형의 크기를 64비트로 구현했을 때, 필요한 자료형의 개수와 이에 해당하는 인덱스는 [표 2]와 같다.

	자료형 개수	해당하는 인덱스
(1)	1	0
(2)	1	1
(3)	64	2~65
(4)	32	66~97
(5)	(4)에 포함	66

[표 2] 64비트 구현에서 필요한 자료형의 개수

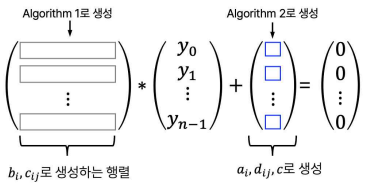


[그림 4] (4)의 $x_i x_j$의 계수 구성

$x_i x_j$의 계수를 효율적인 메모리 사용을 위해 [그림 4]와 같이 d_{ij}인 경우와 $d_{(63-i)j}$를 동일한 인덱스에 속하도록 구성하고, 구현의 용이성을 위해 하위 1비트를 사용하지 않는다. 따라서, 1개의 방정식 배열을 저장하기 위해 필요한 메모리는 6,272 bits ≈ 0.78 KB, 64개의 방정식 테이블을 저장하기 위해 필요한 메모리는 401,408 bits ≈ 49 KB이다.

IV. 이진 행렬 구성

가우스 소거 연산으로 y의 해를 구하기 위해, 방정식을 [그림 5]와 같이 행렬로 구성하는 Algorithm 1과 2의 출력은 [그림 6]과 같이 확장 이진 행렬 형태로 저장된다. Algorithm 1로 y의 계수 행렬의 행벡터를 생성하며, Algorithm 2로 상수 벡터를 생성한다.



[그림 5] 이진 행렬 구성

[그림 6] 확장 이진 행렬

방정식 테이블을 통해 n비트 입력 x에 대해 $(n \times n+1)$크기의 확장 이진 행렬을 구성할 때 방정식의 각 항의 계수를 하나의 비트로 구성하였으므로, 테이블을 구성한 자료형의 크기인 n비트 단위로 연산이 가능하다. 정의 1에 의해 방정식에 대한 연산은 곱셈을 AND 연산으로, 덧셈을 XOR 연산으로 처리할 수 있다.

i) y 의 계수 계산

Algorithm 1 y의 계수 계산

Input: $x = (x_0, x_1, x_2, \ldots, x_{n-1})$
Output: y_coeff $\in \{0,1\}^{n-1}$

```
for i ← 0 to n − 1 do
    y_coeff ← y_coeff ⊕ (equ[2 + i] ∧ x)
end for
y_coeff ← y_coeff ⊕ equ[1]
```

ii) 상수 계산

Algorithm 2 상수 계산

Input: $x = (x_0, x_1, x_2, \ldots, x_{n-1})$
Output: $k \in \{0,1\}$

```
c = c ⊕ equ[0]
for i ← 0 to n/2 − 1 do
    if x_i == 1 then
        c ← c ⊕ (((equ[n + 2 + i] ≫ 1) ∧ x) ≪ i)
    end if
end for
for i ← n − 2 to n/2 do
    if x_i == 1 then
        c ← c ⊕ ((equ[2n − i + 1] ≫ (i + 1)) ∧ x)
    end if
end for
for i ← 0 to n − 1 do
    k = k ⊕ (c ≫ i) ∧ 1
end for
k = k ⊕ C
```

V. 구현 결과

3장의 음함수 저장 방식의 성능 측정을 위해 랜덤한 값으로 음함수 방정식 저장 테이블을 채우고, 64비트 입력 x에 대해 4장에서 제안한 Algorithm 1, 2를 통해 확장 이진 행렬을 생성하고 가우스 소거 연산을 수행하였다. 속도 측정 결과는 1,000,000번 반복하였을 때 1회 평균 0.091ms가 소요되며, 초당 약 10,000번 연산이 가능하다. 음함수 기반 화이트박스 암호 구현은 매 라운드마다 적어도 한 번의 가우스 소거 연산이 필요하다. 예를 들어, 경량 블록 암호인 PIPO-128을 음함수 기반 화이트박스로 구현했을 때 약 39회의 가우스 소거 연산이 필요할 것으로 예상되고, 본 논문에서 제시한 구현 방법을 활용하면 음함수 기반 화이트박스 PIPO-128은 3.55ms 내에 연산이 가능할 것으로 예상된다.

VI. 결론

CEJO framework 기반 화이트박스 암호가 안전하지 않음이 밝혀지면서 새롭게 제안된 음함수 기반 화이트박스 암호는 x, y에 대한 음함수 방정식에 입력 x를 대입하고 가우스 소거 연산으로 방정식을 풀어 출력 y를 구한다. 본 논문에서는 음함수 기반 화이트박스 암호에서 사용되는 행렬 연산에 필요한 방정식 저장 방식과 저장된 방정식 테이블과 입력에 대한 이진 행렬을 구성하는 방법을 제안하였다.

음함수 기반 화이트박스 암호는 모든 라운드에서 저장된 음함수 방정식에 대한 행렬 구성과 가우스 소거 연산 과정이 필요하다. 본 논문에서 소개한 이진 행렬 구성 방식과 가우스 소거 연산을 라이브러리로 활용하면 해당 연산을 효율적으로 수행할 수 있을 것으로 기대된다. 이에 대한 후속 연구로 본 논문에서 제시한 라이브러리를 활용하여 음함수 기반 화이트박스 PIPO를 구현하는 연구를 진행할 예정이다.

ACKNOWLEDGMENT

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

참 고 문 헌

[1] Stanley Chow. et al., "White-box Cryptography and an AES Implementation", SAC 2002

[2] Patrick Derbez. et al., "On Recovering Affine Encodings in White-Box Implementations". IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018(3), 121-149 (2018)

[3] Renea A. et al., "Implicit White-Box Implementations: White-Boxing ARX Ciphers", CRYPTO 2022