JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 22, No. 7, pp. 1031-1037, Jul. 2021

Check for updates

NIST 표준 경량암호 후보 Pyjamask의 향상된 구현 방법 연구

박 종 현[1] · 전 용 진[2] · 김 종 성[3*]
[1]국민대학교 금융정보보안학과 석사과정
[2]국민대학교 금융정보보안학과 박사과정
[3*]국민대학교 정보보안암호수학과 교수

Study on the Improved Implementation Method of NIST Standard Lightweight Cipher Candidate Pyjamask

Jong-Hyun Park[1] · Yong-Jin Jeon[2] · Jong-Sung Kim[3*]

[1]Master's Course, Department of Financial Information, Kookmin University
[2]P.HD student, Department of Financial Information, Kookmin University
[3*]Professor, Department of Information Security, Cryptology, and Mathematics, Kookmin University

[요 약]

가용 가능한 자원이 적은 사물인터넷 (IoT) 환경에서, 블록암호의 하드웨어 구현 비용을 줄이는 것은 중요하다. 블록암호 구현에 필요한 연산 개수를 최소화하면 시간이나 공간적인 비용을 절감할 수 있다. 일반적으로 현대 블록암호는 선형계층과 비선형계층으로 이루어진다. 특히, 행렬을 사용하는 선형계층은 타 연산자에 비해 비교적 비용이 많이 드는 XOR (eXclusiveOR) 연산자로 구현되기 때문에, 이를 줄이는 것은 하드웨어 구현 관점에서 큰 이점이 된다. 본 논문에서는 행렬 최적화 알고리즘을 이용하여 NIST 표준 경량암호 후보 Pyjamask의 선형계층과 키 스케줄에 대한 기 제안된 XOR 연산자 개수를 각각 7.81%와 9.44% 가량 줄인다. 이는 Pyjamask의 기존 효율성을 5.16% 향상시킨 결과이다.

[Abstract]

In Internet of Things (IoT) environments with fewer available resources, reducing the cost of hardware implementing block ciphers is important. Minimizing the number of operations required to implement block ciphers can save time or space costs. In general, modern block ciphers consist of linear and nonlinear layers. In particular, since linear layers using matrices are implemented as relatively expensive XOR (eXclusivOR) operations over other operations, reducing them is a great advantage in terms of hardware implementation. In this paper, we reduce the number of proposed XOR operations for the linear layer and key schedule of NIST standard lightweight cipher candidate Pyjamask by 7.81% and 9.44%, respectively, using matrix optimization algorithms. This is the result of a 5.16% improvement in Pyjamask's existing efficiency.

색인어 : Pyjamask, 경량 구현, 행렬 최적화 알고리즘, 선형계층, XOR 연산

Key word : Pyjamask, Lightweight Implementation, Matrix Optimization Algorithm, Linear Layer, XOR operation

http://dx.doi.org/10.9728/dcs.2021.22.7.1031

Received 05 July 2021; **Revised** 20 July 2021
Accepted 20 July 2021

***Corresponding Author; Jong-Sung Kim**

Tel: +82-2-910-5750
E-mail: jskim@kookmin.ac.kr

Ⅰ. 서 론

경량암호는 연구 개발 및 표준화가 활발하게 진행되고 있는 암호기술 분야 중 하나이다. 특히, 사물인터넷 기기와 같이 하드웨어의 면적과 메모리 크기, 전력 소비량에 제약을 받는 환경에서 사용되므로 중요하게 다뤄지고 있다. 암호기술들의 표준화를 담당하는 미국 국립표준기술연구소(NIST)에서는 2015년부터 경량암호 알고리즘을 표준화하기 위한 공모사업을 진행하고 있다[1].

암호 알고리즘을 경량화하는 다양한 방법 중에는 구현에 필요한 논리 연산자의 수를 최소화하는 방법이 있다. 일반적으로 암호는 Shannon의 혼돈-확산 정리[2]를 따라 선형계층과 비선형계층으로 이루어진다. 행렬을 사용하는 선형계층은 타 연산자(NOT, NOR, AND, OR)보다 비용이 많이 드는 XOR 연산자로 구현이 되기 때문에[3], 이를 개선하는 것은 회로에서 무시할 수 없는 비용 절감 효과를 줄 수 있다.

행렬 최적화 알고리즘은 구현에 필요한 XOR 연산자의 개수를 줄여 블록암호의 구현 비용을 절감시키는 효과를 가져온다. 기존 행렬 최적화 알고리즘으로는 Paar의 알고리즘[4]과 BP (Boyar Peralta) 알고리즘[5]이 있다. 이 두 알고리즘은 임시변수를 활용한 행렬의 확장을 통해서 행렬의 XOR 연산자 개수를 계산한다. 2020년에 제안된 RNBP 알고리즘은 BP 알고리즘을 향상시킨 알고리즘이다[6]. 이 알고리즘은 임시변수의 결정에 랜덤 요소를 추가시킴으로써 더 최적화된 행렬 구현 방법을 찾는다. 반면, 동일년도에 제안된 Xiang et al.의 알고리즘은 위 세 알고리즘과 다르게 행렬 분해 방식을 사용한다[7]. 이러한 방식을 사용하면 임시변수 없이 행렬을 구현할 수 있다.

본 논문에서는 Pyjamask[8]에서 사용하는 행렬에 RNBP 알고리즘과 Xiang et al.의 알고리즘을 적용하여 효율적인 구현을 제시한다. Pyjamask는 NIST 표준 경량암호 공모사업 후보이며, 부채널 공격에 높은 수준의 보안을 목표로 한다. 특히, 효율적인 고차 마스킹 구현을 위해 회로에서 사용되는 비선형 연산자의 개수를 최소화하는 설계 사상을 가진다.

본 논문의 구성은 다음과 같다. 2장에서는 Paar의 알고리즘과 RNBP 알고리즘, Xiang et al.의 알고리즘을 소개한다. 3장에서는 Pyjamask에서 사용하는 행렬들을 소개한다. 4장에서는 앞서 소개한 알고리즘들을 적용하여 Pyjamask의 제시된 행렬 최적 구현 방법[9]보다 향상된 결과를 제시하고 분석한다. 마지막 5장에서는 결론과 향후 연구 계획으로 본 논문을 마무리한다.

Ⅱ. 행렬 최적화 알고리즘

2-1 행렬 최적화 알고리즘 사용 방법

본 장에서는 3가지 행렬 최적화 알고리즘을 소개한다. 이 알고리즘들은 이진 행렬을 대상으로 동작한다. 만약 이진 행렬이 아닌 유한체로 생성된 행렬을 최적화한다면 이진 행렬로 변환해준 후 알고리즘에 적용해야 한다.

2-2 Paar의 알고리즘

Paar의 알고리즘은 Christof Paar이 1997년에 제안한 행렬 최적화 알고리즘이다. 이 알고리즘은 앞으로 설명하게 될 RNBP 알고리즘과 Xiang et al.의 알고리즘과는 다르게 결정론적인 알고리즘이다. 또한, 크기가 큰 행렬들에 대해서 빠르게 결과를 얻을 수 있는 장점이 있다.

알고리즘 설명을 위해 행렬의 행렬방정식과 선형 연립방정식을 고려한다. Paar의 알고리즘에 대한 설명은 다음과 같다.

1단계: AND 연산했을 때 1의 개수가 가장 많은 두 열을 찾는다. 만약 2개 이상의 조합이 나온다면 먼저 찾은 두 열을 선택한다. 그때의 두 열을 i, j라고 하자. 다음은 4×4 행렬 A의 1열과 2열을 AND 연산했을 때 1의 개수를 구하는 과정의 예시이다.

$$A\begin{bmatrix}1&1&0&0\\1&1&1&0\\1&1&1&1\\0&1&1&1\end{bmatrix}\Rightarrow\begin{bmatrix}1\\1\\1\\0\end{bmatrix}\wedge\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\Rightarrow\begin{bmatrix}1\wedge1\\1\wedge1\\1\wedge1\\0\wedge1\end{bmatrix}=\begin{bmatrix}1\\1\\1\\0\end{bmatrix}\Rightarrow(1+1+1+0)=3. \quad (1)$$

2단계: 두 열 i, j에 대해, 행렬과 연관된 선형 연립방정식에 $x_i\oplus x_j$를 새로운 변수 v_1으로 치환한 후, v_1을 포함하여 행렬방정식으로 표현한다. 이때 변수가 추가되었으므로 행렬이 확장된다.

3단계: 다시 1단계로 돌아가 반복 시행한다. 만약 모든 AND 연산 결과의 1의 개수가 모두 0개이면 알고리즘을 종료한다. 이때, 반복 시행한 횟수가 XOR 연산자의 개수이다.

그림 3은 4×4 행렬 A에 대해 위 알고리즘의 행렬 확장 과정 예시를 표현한 것이다. 행렬 A의 1열과 2열은 AND 연산했을 때 1의 개수가 가장 많은 조합이다. 따라서 $v_1=x_1\oplus x_2$으로 치환한다.

$$\begin{aligned}x_1\oplus x_2&=y_1\\x_1\oplus x_2\oplus x_3&=y_2\\x_1\oplus x_2\oplus x_3\oplus x_4&=y_3\\x_2\oplus x_3\oplus x_4&=y_4\end{aligned}\Leftarrow\overset{A}{\begin{bmatrix}1&1&0&0\\1&1&1&0\\1&1&1&1\\0&1&1&1\end{bmatrix}}\cdot\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}$$

$$\Downarrow$$

$$\begin{aligned}v_1&=y_1\\x_3\oplus v_1&=y_2\\x_3\oplus x_4\oplus v_1&=y_3\\x_2\oplus x_3\oplus x_4&=y_4\end{aligned}\Rightarrow\begin{bmatrix}0&0&0&0&1\\0&0&1&0&1\\0&0&1&1&1\\0&1&1&1&0\end{bmatrix}\cdot\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\v_1\end{bmatrix}=\begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}$$

그림 1. 1열과 2열에 대한 행렬 확장 과정
Fig. 1. Matrix Expansion Process for Columns 1 and 2

2-3 RNBP 알고리즘

RNBP 알고리즘은 Quan Quan Tan과 Thomas Peyrin이 2020년에 제안한 행렬 최적화 알고리즘이다. 이 알고리즘은 BP 알고리즘의 향상된 알고리즘이므로 BP 알고리즘을 먼저 소개한다.

알고리즘 설명에서 다음과 같은 용어들을 사용한다. $Dist$는 행렬의 각 행의 1의 개수에서 1을 뺀 수들을 벡터로 표현한 것이다. 예시로 4×4 행렬의 1행부터 4행까지 1의 개수가 각각 2, 3, 3, 2라고 하면 $Dist=[1,2,2,1]$이 된다. En은 $Dist$의 유클리드 노름을 뜻하며, $\sum Dist$는 $Dist$ 원소들의 합을 뜻한다. 즉, $Dist=[1,2,2,1]$이면, $En=\sqrt{10}$이고 $\sum Dist=6$이다.

알고리즘 설명을 위해 행렬의 행렬방정식과 선형 연립방정식을 고려한다. BP의 알고리즘에 대한 설명은 다음과 같다.

1단계: 두 열 i,j에 대해, 다음 선별 과정을 거친다.

확장된 행렬이 가장 작은 $\sum Dist$가 되는 후보를 선택한다. 가장 작은 $\sum Dist$를 가지는 i,j 후보가 2개 이상이라면, 가장 큰 En이 되는 후보를 선택한다. 만약, 가장 큰 En이 되는 후보가 2개 이상이라면 먼저 후보가 된 두 열 i,j를 선택한다.

2단계: 선별 과정을 거친 두 열 i,j에 대해, Paar의 알고리즘의 2단계와 같은 방법으로 새로운 변수 v_1을 추가하여 행렬을 확장한다.

3단계: 다시 1단계로 돌아가 반복 시행한다. 만약 확장된 행렬의 $Dist=[0,0,0,0,0,0]$을 만족한다면 알고리즘을 종료한다. 이때, 반복 시행한 횟수가 XOR 연산자 개수이다.

그림 2는 위 알고리즘의 행렬 확장 과정 예시를 표현한 것이다. 행렬 A에서 2, 3열과 1, 6열의 조합 모두 가장 작은 $\sum Dist=9$인 조합이다. 각각의 $Dist$는 $[1,1,3,1,1,2]$와 $[2,1,3,1,0,2]$이다. 그중 $En=\sqrt{19}$로 더 큰 1, 6열의 조합을 선택한다. 따라서 $v_1=x_1\oplus x_6$으로 치환한다.

$$
\begin{aligned}
x_1\oplus x_2\oplus x_3&=y_1\\
x_5\oplus x_6&=y_2\\
x_1\oplus x_3\oplus x_4\oplus x_5&=y_3\\
x_3\oplus x_6&=y_4\\
x_1\oplus x_6&=y_5\\
x_3\oplus x_5\oplus x_6&=y_6
\end{aligned}
\Leftarrow
\overset{A}{\begin{bmatrix}1&1&1&0&0&0\\0&0&0&0&1&1\\1&0&1&1&1&0\\0&0&1&0&0&1\\1&0&0&0&0&1\\0&0&1&0&1&1\end{bmatrix}}
\cdot
\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{bmatrix}
=
\begin{bmatrix}y_1\\y_2\\y_3\\y_4\\y_5\\y_6\end{bmatrix}
$$

$$
\Downarrow
$$

$$
\begin{aligned}
x_1\oplus x_2\oplus x_3&=y_1\\
x_5\oplus x_6&=y_2\\
x_1\oplus x_3\oplus x_4\oplus x_5&=y_3\\
x_3\oplus x_6&=y_4\\
v_1&=y_5\\
x_3\oplus x_5\oplus x_6&=y_6
\end{aligned}
\Rightarrow
\begin{bmatrix}1&1&1&0&0&0&0\\0&0&0&0&1&1&0\\1&0&1&1&1&0&0\\0&0&1&0&0&1&0\\0&0&0&0&0&0&1\\0&0&1&0&1&1&0\end{bmatrix}
\cdot
\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\v_1\end{bmatrix}
=
\begin{bmatrix}y_1\\y_2\\y_3\\y_4\\y_5\\y_6\end{bmatrix}
$$

그림 2. 1열과 6열에 대한 행렬 확장 과정
Fig. 2. Matrix Expansion Process for Columns 1 and 6

RNBP 알고리즘은 BP 알고리즘의 1단계에서 가장 큰 En이 되는 후보가 2개 이상일 때, 랜덤하게 하나를 선택한다. 이를 통해 AES의 행렬 구현에 필요한 XOR 연산자를 97회에서 95회로 줄임으로써 향상된 성능을 보인다.

2-4 Xiang et al.의 알고리즘

Xiang et al.의 알고리즘은 AES의 행렬 구현에 필요한 XOR 연산자의 개수를 92회로, 현재까지 최적의 구현 방법을 제시했다. 알고리즘 설명에서는 다음과 같은 용어들이 사용된다. $E(i+j)$와 $E(i\leftrightarrow j)$는 각각 j행을 i행에 XOR 연산을 해주는 기본 행렬과 i행과 j행의 위치를 서로 바꿔주는 연산을 해주는 기본행렬을 의미한다. 또한, 이 기호는 열의 연산을 표기할 때도 똑같이 적용한다. 알고리즘에는 다음과 같은 정리가 사용된다.

정리 1. 임의의 가역 이진 행렬은 $E(i+j)$와 $E(i\leftrightarrow j)$의 곱으로 표현될 수 있다.

그림 3은 Xiang et al.의 알고리즘 순서도이다. Xiang et al.의 알고리즘은 크게 2개의 알고리즘으로 행렬 분해 과정과 $E(i+j)$를 줄이는 과정이 있다. 행렬 M을 알고리즘에 적용했을 때, 1~4단계는 행렬 분해 과정이고 5단계는 $E(i+j)$ 줄이기 과정이다.

1단계: 현재의 행렬에 $E(i+j)$ 행렬을 곱했을 때, 1의 개수가 가장 적게 만드는 두 행(또는 열) i,j를 모두 찾는다. 찾아진 조합 중 랜덤하게 하나를 선택한다.

2단계: 선택된 i,j에 대해, 현재의 행렬을 $E(i+j)$를 곱한 행렬로 갱신하고 다시 1) 단계로 돌아가 반복 시행한다. 만약 1의 개수가 더 줄지 않는다면 행(또는 열) 기준으로 가우스 조르단 소거법을 사용하여 행렬을 기본행렬의 곱 형태로 만들어준다.

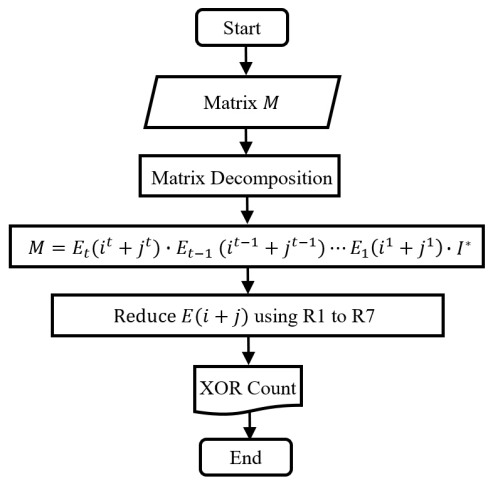

그림 3. Xiang et al.의 알고리즘 순서도
Fig. 3. Flowchart for algorithm of Xiang et al.

3단계: 결과적으로 정리 1에 의해 행렬 M은 $E(i+j)$와 $E(i\leftrightarrow j)$의 곱으로 표현된다.

4단계: 3단계에서 표현된 M을 식 (2)을 사용하여 식 (3)과 같은 형태로 만든다(s,t는 자연수). 이때, $E(i\leftrightarrow j)$들을 재결합한 행렬을 I^*라고 하자.

$$\begin{aligned}&E(i+j)E(k\leftrightarrow l)=E(k\leftrightarrow l)E(f_{k,l}(i)+f_{k,l}(j)),\\&E(k\leftrightarrow l)E(i+j)=E(f_{k,l}(i)+f_{k,l}(j))E(k\leftrightarrow l),\\&where\, f_{k,l}(x)\equiv\begin{cases}k, & \text{if } x=l,\\ l, & \text{if } x=k,\\ x, & else.\end{cases}\end{aligned}\tag{2}$$

$$\begin{aligned}M=&E(i_t+j_t)\cdots E(i_2+j_2)E(i_1+j_1)\\&E(i_s^{'}\leftrightarrow j_s^{'})\cdots E(i_2^{'}\leftrightarrow j_2^{'})E(i_1^{'}\leftrightarrow j_1^{'}).\end{aligned}\tag{3}$$

5단계: 행렬 분해 과정이 종료되면, 이어서 $E(i+j)$ 줄이기 과정이 시작된다. 이때, 다음 성질 $R1\sim R7$을 사용한다.

$$\begin{aligned}&R1: E(k+i)E(k+j)E(i+j)=E(i+j)E(k+i),\\&R2: E(i+k)E(k+j)E(i+j)=E(k+j)E(i+k),\\&R3: E(i+k)E(j+k)E(i+j)=E(i+j)E(j+k),\\&R4: E(j+k)E(i+k)E(i+j)=E(i+j)E(j+k),\\&R5: E(k+j)E(k+i)E(i+j)=E(i+j)E(k+i),\\&R6: E(k+j)E(i+k)E(i+j)=E(i+k)E(k+j),\\&R7: E(j+i)E(i+j)=E(i\leftrightarrow j)E(j+i).\end{aligned}\tag{4}$$

식 (3)의 꼴로 분해된 행렬에서는 $E(i+j)$만이 XOR 연산이 필요로 한다. 따라서 성질 $R1\sim R6$은 사용되는 XOR 연산을 3개에서 2개로 줄이고, $R7$은 XOR 연산을 2개에서 1개로 줄인다.

행렬 분해 과정에 랜덤 요소가 있으므로, 행렬 분해 과정과 $R1\sim R7$을 이용한 $E(i+j)$을 줄이기를 반복 실행하면서 최적 행렬 구현 방법의 XOR 연산자 개수를 계산한다. 행렬이 확장되지 않고 $E(i+j)$를 사용하여 XOR 연산을 구현하기 때문에, 앞의 두 알고리즘과 다르게 구현에 임시변수를 사용하지 않는다.

Ⅲ. NIST 표준 경량암호 후보 Pyjamask

Pyjamask는 NIST 표준 경량암호 공모사업 후보로 블록암호 기반 AEAD 알고리즘이다. Pyjamask의 핵심 함수는 각각 96-bit와 128-bit의 블록 크기를 가지는 블록암호 Pyjamask-96과 Pyjamask-128이 있다. 라운드 함수에는 AddRounkey, SubBytes, MixRows가 있다. 총 라운드는 14라운드이다. 그림 4은 Pyjamask-128의 라운드 함수를 도식화한 그림이다.

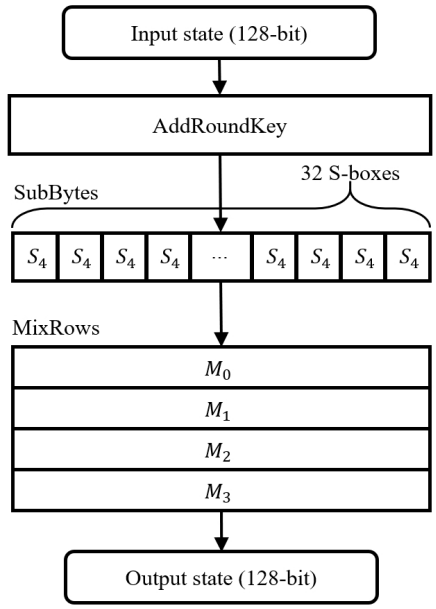

그림 4. Pyjamask-128의 라운드 함수
Fig. 4. Round function of Pyjamask-128

AddRoundkey는 128-bit 라운드 키를 XOR 연산하는 함수이고 SubBytes는 S-box인 S_4를 32개 사용하는 함수이다. 그리고 MixRows는 4개의 행렬이 사용되는 함수이다. Pyjamask에는 순환 이진 행렬 $Cir(\cdot)$가 사용된다. 예시로 16×16 순환 이진 행렬 $Cir(0xd084)$은 다음 식 (5)와 같다.

$$Cir(0xd084)=\begin{bmatrix}1101000010000100\\0110100001000010\\0011010000100001\\1001101000010000\\0100110100001000\\0010011010000100\\0001001101000010\\0000100110100001\\1000010011010000\\0100001001101000\\0010000100110100\\0001000010011010\\0000100001001101\\1000010000100110\\0100001000010011\\1010000100001001\end{bmatrix}.\tag{5}$$

Pyjamask-128의 경우에는 32×32 순환 이진 행렬 5개와 4×4 순환 이진 행렬 1개를 사용한다. 확산계층인 MixRows에서는 식 (6)의 32×32 순환 이진 행렬 M_0,M_1,M_2,M_3을 사용한다. Pyjamask-96의 경우, M_3를 사용하지 않는다.

$$\begin{aligned}&M_0=Cir(0xd08430e2),\\&M_1=Cir(0x42074163),\\&M_2=Cir(0x00a79a4b),\\&M_3=Cir(0x64095289).\end{aligned}\tag{6}$$

Key Schedule 함수에서는 32×32 순환 이진 행렬 M_k와 4×4 순환 이진 행렬 M_4가 사용된다. 행렬은 다음 식 (7)과 같다.

$$M_k = Cir(0xa9cec08e),$$
$$M_4 = Cir(0x7). \quad (7)$$

기존에 제시된 Pyjamask의 라운드 기반 하드웨어 구현 관점에서 UMC 180 공정을 사용하는 ASIC에 필요한 면적은 7500 GE(Gate Equivalents) 이다. 그중 이진 행렬 구현에 필요한 면적은 4632 GE로 약 62%를 차지한다.

Ⅳ. 행렬 최적화 알고리즘 적용 결과

4-1 알고리즘 적용 결과

본 논문에서는 식 (8)에 있는 9개의 행렬을 최적화했다. (이때, M^{-1}는 M의 역행렬이다.)

$$M_0, M_1, M_2, M_3, M_k, M_0^{-1}, M_1^{-1}, M_2^{-1}, M_3^{-1} \quad (8)$$

크기가 비교적 작은 M_4는 최적화 알고리즘을 사용하지 않아도 간단히 최적화할 수 있기에 제외했다.

[9]에서는 Paar의 알고리즘을 사용하여 (8)의 행렬들에 대한 최적 구현 방법을 제시했다. 우리는 RNBP 알고리즘과 Xiang et al.의 알고리즘을 적용했다. 표 1은 우리의 결과와 기존에 제시된 결과를 비교하여 향상된 결과를 정리한 표이다. #(XOR)은 구현에 필요한 XOR 연산자 개수를 말한다.

표 1. Pyjamask 행렬 구현에 필요한 XOR 연산자 개수 비교
Table 1. Compare the number of XOR operations required to implement the Pyjamask matrices

Matrix	Existing results[9]	Our Results	
	#(XOR)	#(XOR)	Algorithm
M_0	175	166	RNBP
M_2	199	174	RNBP
M_k	200	163	Xiang et al.
M_0^{-1}	169	162	RNBP
M_1^{-1}	201	178	RNBP
M_2^{-1}	147	144	RNBP
M_3^{-1}	203	162	Xiang et al.

본 논문의 결과 중 M_k 구현 방법을 표 2에 제시한다. 모든 구현 결과들은 [10]을 통해 얻을 수 있다.

표 2. 표 1에서 제시한 Pyjamask의 행렬 M_k에 대한 구현
Table 2. Our implementation for matrix M_k of Pyjamask in Table 1

# operation		# operation	
1	x[6] ^= x[22]	43	x[7] ^= x[16]
2	x[20] ^= x[4]	44	x[31] ^= x[9]
3	x[19] ^= x[3]	45	x[16] ^= x[11]
4	x[5] ^= x[21]	46	x[11] ^= x[28]
5	x[8] ^= x[24]	47	x[21] ^= x[14]
6	x[23] ^= x[7]	48	x[11] ^= x[6]
7	x[11] ^= x[27]	49	x[7] ^= x[1]
8	x[29] ^= x[13]	50	x[3] ^= x[16]
9	x[30] ^= x[14]	51	x[21] ^= x[3]
10	x[9] ^= x[19]	52	x[3] ^= x[22]
11	x[1] ^= x[6]	53	x[22] ^= x[14]
12	x[28] ^= x[12]	54	x[14] ^= x[26]
13	x[27] ^= x[5]	55	x[22] ^= x[18]
14	x[25] ^= x[9]	56	x[18] ^= x[5]
15	x[25] ^= x[11]	57	x[5] ^= x[29]
16	x[27] ^= x[6]	58	x[26] ^= x[6]
17	x[15] ^= x[31]	59	x[26] ^= x[13]
18	x[24] ^= x[15]	60	x[13] ^= x[25]
19	x[27] ^= x[14]	61	x[22] ^= x[1]
20	x[14] ^= x[13]	62	x[27] ^= x[12]
21	x[11] ^= x[23]	63	x[10] ^= x[8]
22	x[1] ^= x[17]	64	x[0] ^= x[28]
23	x[10] ^= x[26]	65	x[6] ^= x[20]
24	x[26] ^= x[20]	66	x[31] ^= x[12]
25	x[13] ^= x[7]	67	x[13] ^= x[4]
26	x[12] ^= x[17]	68	x[4] ^= x[0]
27	x[0] ^= x[16]	69	x[1] ^= x[19]
28	x[30] ^= x[19]	70	x[3] ^= x[19]
29	x[13] ^= x[31]	71	x[12] ^= x[13]
30	x[31] ^= x[5]	72	x[16] ^= x[24]
31	x[17] ^= x[29]	73	x[13] ^= x[22]
32	x[4] ^= x[17]	74	x[29] ^= x[10]
33	x[17] ^= x[24]	75	x[14] ^= x[16]
34	x[18] ^= x[2]	76	x[10] ^= x[18]
35	x[7] ^= x[8]	77	x[19] ^= x[10]
36	x[4] ^= x[7]	78	x[10] ^= x[11]
37	x[7] ^= x[31]	79	x[3] ^= x[0]
38	x[31] ^= x[26]	80	x[22] ^= x[26]
39	x[26] ^= x[24]	81	x[16] ^= x[9]
40	x[24] ^= x[27]	82	x[16] ^= x[6]
41	x[7] ^= x[21]	83	x[9] ^= x[4]
42	x[27] ^= x[3]	84	x[24] ^= x[18]

85	x[6] ^= x[25]	127	x[31] ^= x[1]
86	x[25] ^= x[30]	128	x[5] ^= x[0]
87	x[18] ^= x[15]	129	x[15] ^= x[29]
88	x[22] ^= x[31]	130	x[7] ^= x[10]
89	x[9] ^= x[30]	131	x[0] ^= x[25]
90	x[31] ^= x[28]	132	x[12] ^= x[2]
91	x[11] ^= x[8]	133	x[2] ^= x[14]
92	x[0] ^= x[28]	134	x[14] ^= x[21]
93	x[25] ^= x[8]	135	x[25] ^= x[8]
94	x[28] ^= x[5]	136	x[21] ^= x[25]
95	x[10] ^= x[23]	137	x[25] ^= x[10]
96	x[5] ^= x[30]	138	x[10] ^= x[27]
97	x[30] ^= x[1]	139	x[17] ^= x[3]
98	x[4] ^= x[27]	140	x[29] ^= x[2]
99	x[9] ^= x[2]	141	x[0] ^= x[16]
100	x[1] ^= x[8]	142	x[3] ^= x[24]
101	x[27] ^= x[23]	143	x[14] ^= x[4]
102	x[29] ^= x[23]	144	x[29] ^= x[6]
103	x[2] ^= x[23]	145	x[11] ^= x[22]
104	x[10] ^= x[30]	146	x[4] ^= x[22]
105	x[30] ^= x[20]	147	x[29] ^= x[1]
106	x[20] ^= x[28]	148	x[13] ^= x[19]
107	x[28] ^= x[23]	149	x[3] ^= x[30]
108	x[23] ^= x[18]	150	x[2] ^= x[25]
109	x[26] ^= x[18]	151	x[25] ^= x[9]
110	x[8] ^= x[20]	152	x[18] ^= x[31]
111	x[20] ^= x[18]	153	x[20] ^= x[12]
112	x[23] ^= x[0]	154	x[1] ^= x[17]
113	x[21] ^= x[5]	155	x[6] ^= x[4]
114	x[1] ^= x[0]	156	x[28] ^= x[3]
115	x[8] ^= x[29]	157	x[24] ^= x[2]
116	x[18] ^= x[10]	158	x[27] ^= x[19]
117	x[22] ^= x[17]	159	x[8] ^= x[14]
118	x[30] ^= x[15]	160	x[30] ^= x[13]
119	x[24] ^= x[21]	161	x[19] ^= x[24]
120	x[4] ^= x[18]	162	x[23] ^= x[7]
121	x[26] ^= x[5]	163	x[5] ^= x[21]
122	x[18] ^= x[25]		
123	x[28] ^= x[11]		
124	x[11] ^= x[25]		
125	x[15] ^= x[26]		
126	x[7] ^= x[2]		

4-2 결과에 대한 분석

표 3는 Pyjamask-128의 MixRows와 $MixRows^{-1}$의 시간 및 공간복잡도를 기존 결과와 우리의 결과를 비교한 표이다. Pyjamask-128의 MixRows에서 발생하는 XOR 연산자의 개수를 각각 476번과 1,036번의 XOR 연산을 줄였다. 라운드 기반 하드웨어 구현 관점에서는 UMC 180 공정을 사용하는 ASIC에서 필요한 면적을 각각 91 GE와 198 GE만큼 줄였다.

표 3. Pyjamask-128의 MixRows 하드웨어 구현에 필요한 #(XOR)/면적 비교

Table 3. Compare #(XOR)/GE required to hardware implement the Pyjamask-128 MixRows

	Existing Results[9]	Our Results
MixRows*14R	9,268 XORs	8,792 XORs
$MixRows^{-1}$*14R	10,080 XORs	9,044 XORs
MixRows	1,767.54 GE	1,676.76 GE
$MixRows^{-1}$	1,922.4 GE	1,724.82 GE

Ⅴ. 결 론

본 논문에서는 Paar의 알고리즘과 RNBP 알고리즘, Xiang et al.의 알고리즘 총 3개의 행렬 최적화 알고리즘을 소개했다. 또한, 최신 알고리즘인 RNBP 알고리즘과 Xiang et al.의 알고리즘을 사용해 Pyjamask에서 사용되는 행렬을 최적화했다. 이 구현 기법을 사용한 Pyjamask의 라운드 기반 하드웨어 구현 면적 비용을 7,500 GE에서 7,113 GE로 5.16%가량 줄였다.

Paar의 알고리즘과 RNBP 알고리즘은 알고리즘의 특징이 비슷하지만 Xiang et al.의 알고리즘은 그 특징을 달리한다. Paar의 알고리즘과 RNBP 알고리즘은 모두 행렬의 구현 방법이 제시되면 알고리즘이 종료되지만, Xiang et al.의 알고리즘은 E(i+j) 줄이기 과정이 존재한다. 만약 E(i+j) 줄이기 과정을 일반화할 수 있다면 Paar의 알고리즘 또는 RNBP 알고리즘을 사용해 얻은 행렬 구현 방법을 더 최적화할 수 있을 것이다.

Xiang et al.의 알고리즘에서 행렬 분해 과정을 거치면 행렬은 기본행렬들의 곱으로 표현이 된다. 이때, [7]에서는 부분적인 기본행렬들을 대상으로 다시 Xiang et al.의 알고리즘을 반복적으로 적용하여 더 효과적인 구현 방법을 만들었다. 이러한 방법을 Paar의 알고리즘 또는 RNBP 알고리즘으로 응용해본다면 새로운 검색을 진행함으로써 성능 향상에 도움을 줄 수 있을 것이다.

감사의 글

본 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원으로 받아 수행된 연구임 (No.2017-0-00520).

참고문헌

[1] National Institute of Standards and Technology. COMPUTER SECURITY RESOURCE CENTER. Lightweight Cryptography [Internet]. Available: https://csrc.nist.gov/Projects/lightweight-cryptography

[2] Claude E. Shannon, “A Mathematical Theory of Cryptography”, Bell System Technical Memo MM 45-110-02, September 1, 1945.
[3] A. Poschmann, “Lightweight Cryptography from an Engineers Perspective.”, Workshop on Elliptic Curve Cryptography (ECC), 2007.
[4] C. Paar, “Optimized arithmetic for Reed-Solomon encoders.” In IEEE International Symposium on Information Theory, pp. 250, 1997.
[5] J. Boyar and R. Peralta, “A new combinational logic minimization technique with applications to cryptology.” In: Festa, P. (ed.) SEA 2010. LNCS, Vol. 6049, pp. 178–189. 2010
[6] Q. Q. Tan and T. Peyrin, “Improved Heuristics for Short Linear Programs.” IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1), pp. 203-230, 2020.
[7] Z. Xiang, X. Zeng, D. Lin, Z. Bao, and S. Zhang, “Optimizing Implementations of Linear Layers.” IACR Transactions on Symmetric Cryptology, 2020(2), pp. 120-145, 2020.
[8] National Institute of Standards and Technology. COMPUTER SECURITY RESOURCE CENTER. Lightweight Cryptography, Pyjamask [Internet]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/pyjamask-spec-round2.pdf
[9] D. Goudarzi, J. Jean, S. Kölbl, T. Peyrin, M. Rivain, Y. Sasaki, and S. M. Sim, “Pyjamask: Block Cipher and Authenticated Encryption with Highly Efficient Masked Implementation.” IACR Transactions on Symmetric Cryptology, 2020(S1), pp. 31-59, 2020.
[10] Improved-implementation-method-of-Pyjamask, Available: https://github.com/Sub-2021/Improved-implementation-method-of-Pyjamask

박종현(Jong-Hyun Park)

2021년 2월 : 국민대학교 정보보안암호수학과 졸업
2021년～현 재: 국민대학교 금융정보보안학과 석사과정
※관심분야 : 정보보호, 암호 알고리즘

전용진(Yong-Jin Jeon)

2019년 8월 : 국민대학교 정보보안암호수학과 졸업
2020년 8월 : 국민대학교 금융정보보안학과 석사
2020년～현 재: 국민대학교 금융정보보안학과 박사과정
※관심분야 : 정보보호, 암호 알고리즘

김종성(Jong-Sung Kim)

2000년 8월/2002년 8월 : 고려대학교 수학 전공 학사/이학석사
2006년 11월 : K.U.Leuven. ESAT/SCD-COSIC 정보보호 전공 공학박사
2007년 2월 : 고려대학교 정보보호대학원 공학박사
2007년 3월～2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월～2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월～2017년 2월: 국민대학교 수학과 부교수
2014년 3월～2020년 8월: 국민대학교 일반대학원 금융정보보안학과 부교수
2017년 3월～2020년 8월: 국민대학교 정보보안암호수학과 부교수
2020년 9월～현 재: 국민대학교 정보보안암호수학과/일반대학원 금융정보보안학과 교수
※관심분야 : 정보보호, 암호 알고리즘, 디지털 포렌식