Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.3, Jun. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.3.391

T-table을 사용한 경량 블록 암호 PIPO의 최적화 구현*

최 민 식,[1†] 김 선 엽,[1] 김 인 성,[1] 신 한 범,[1] 김 성 겸,[2] 홍 석 희[3‡]
[1,3]고려대학교 (대학원생, 교수), [2]삼성전자 (연구원)

Optimized Implementation of Lightweight Block Cipher PIPO Using T-Table*

Minsig Choi,[1†] Sunyeop Kim,[1] Insung Kim,[1] Hanbeom Shin,[1]
Seonggyeom Kim,[2] Seokhie Hong[3‡]
[1,3]Korea University (Graduate student, Professor), [2]Samsung Electronics (Researcher)

요 약

본 논문에서는 경량 블록 암호인 PIPO-64/128, 256에 대해 T-table을 사용한 구현을 최초로 제시한다. 제안 방법은 최초 16개의 T-table을 요구하지만, 필요한 두 종류의 T-table이 순환 구조임을 보이고 T-table 개수를 줄여 구현하는 변형 방법을 추가로 제시한다. 제안 방법들의 T-table 수(코드 크기)-속도간 상충관계 분석을 위해 각각 변형 구현물을 Intel Core i7-9700K 프로세서 환경에서 평가한다. 평가를 통해 획득한 속도 최적화 구현은 TLU(Table-Look-Up) 레퍼런스 구현에 비해 PIPO-64/128, 256에서 각각 11.33, 9.31배, 비트 슬라이스(Bit Slice) 레퍼런스 구현에 비해 각각 3.31, 2.76배 향상된 속도를 갖는다.

ABSTRACT

In this paper, we presents for the first time an implementation using T-table for PIPO-64/128, 256 which are lightweight block ciphers. While our proposed implementation requires 16 T-tables, we show that the two types of T-tables are circulant and obtain variants implementations that require a smaller number of T-tables. We then discuss trade-off between the number of required T-tables (code size) and throughput by evaluating the throughput of the variant implementations on an Intel Core i7-9700K processor. The throughput-optimized versions for PIPO-64/128, 256 provide better throughput than TLU (Table-Look-Up) reference implementation by factors of 3.11 and 2.76, respectively, and bit-slice reference implementation by factors of 3.11 and 2.76, respectively.

Keywords: Block Cipher, PIPO, T-table, Implementation, Circulant Structure

I. 서 론

ICISC 2020에서 제안된 PIPO(Plug-In Plug-Out)는 64-bit 블록 크기와 128-, 256-bit의 키를 지원하는 SPN(Substitution Permutation Network) 구조의 경량 블록 암호이다[1].

T-table TLU(Table Look Up) 구현 방법은 SPN 구조를 갖는 블록 암호 구현에서 S-box와 그에 해당하는 선형 계층의 모든 가능한 입력에 대한 출력을 사전 계산하여 테이블로 만들어 실제 연산 시 사전 계산 테이블을 참조하여 출력을 얻는 방법이다.

본 논문에서는 경량 블록 암호 PIPO를 T-table을 사용하여 최적화 구현한다. 이 과정에서 생성된 PIPO의 T-table이 순환(circulant) 구조임을 보인다. 뿐만 아니라 PIPO와 유사하게 S-R Layer 기반 구조를 사용하는 블록 암호에 대해 T-table을

Received(04. 17. 2023), Modified(05. 30. 2023), Accepted(05. 31. 2023)
* 본 연구는 고려대 암호기술 특화연구센터(UD210027XD)를 통한 방위사업청과 국방과학연구소의 연구비 지원으로 수행되었습니다.
† 주저자. minsigchoi0527@korea.ac.kr
‡ 교신저자. shhong@korea.ac.kr(Corresponding author)

사용하여 최적화 구현에 사용된 T-table도 순환 구조임을 보인다.

T-table의 순환 구조는 구현에 필요한 테이블 수를 줄일 수 있기 때문에 코드 크기 측면에서 향상시킬 수 있으며, 캐시 미스(cache miss)를 고려하는 경우 적은 수의 테이블 사용이 암호화를 가속화 할 수 있다.

따라서 본 논문에서는 PIPO에 대해 T-table 수를 다르게 사용하여 T-table 수에 따른 속도와 코드 크기 사이의 상충관계를 분석해 최적화 구현을 제안한다. 그리고 기존 레퍼런스 구현[1], T-table을 이용한 AES(Advanced Encryption Standard)[2]과 속도를 비교한다.

논문의 구조는 다음과 같다. Ⅱ장에서는 최적화 구현을 제안하기 위한 배경지식을 소개한다. Ⅲ장에서는 T-table을 이용하여 PIPO의 최적화 구현을 제시하고, 그때 사용한 T-table이 순환 구조임을 보인다. Ⅳ장에서는 PIPO의 기존 구현물과 T-table을 이용한 AES 구현물과 각각 속도를 비교하고 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. 배경지식

2.1 표기법

Table 1. Notation

$S(\bullet)$	PIPO 8-bit S-box
$S'(\bullet)$	$S' = S \| S \| S \| S \| S \| S \| S \| S$, Application of 8-Sboxes in parallel
$R(\bullet)$	PIPO R-Layer
$C(\bullet)$	Convert operation
$R'(\bullet)$	$C \circ R \circ C$ ($\circ$: composition)
K	AddRoundKey including round constant
K′	C(K), Converted AddRoundKey
RK_i	i-th Round key
$rcon_i$	i-th Round constant
$ROL(X, b)$	$ROL(X, b) = X \ll b \mid \mid (X \gg (64-b))$, Rotation operation within 64-bit
$T_i[\bullet]$	i-th T-table required for $R' \circ S'$ $(i = 0, 1, ..., 7)$
$C_i[\bullet]$	i-th T-table required for C $(i = 0, 1, ..., 7)$
T_t	A set of T-tables T_i $(i = 0, 1, ..., 7)$
C_t	A set of T-tables C_i $(i = 0, 1, ..., 7)$
nT, nC	The Number n of used T-tables T_i and C_i respectively $(0 \le n \le 8)$

1) https://github.com/PIPO-Blockcipher/PIPO-Blockcipher

2.2 PIPO

PIPO는 경량 블록 암호로 Fig. 1과 같은 구조를 가지며 중간값의 형태는 Fig. 2와 같다. 64-bit 크기의 평문 블록을 128-, 256-bit 크기의 키를 사용해서 각 13, 17라운드를 수행하여 64-bit 크기의 암호문 블록을 생성한다.

PIPO의 한 라운드는 비선형 연산인 S-Layer, 선형 연산인 R-Layer(R)와 라운드 키, 라운드 상수를 XOR하는 AddRoundKey(K)로 구성된다.

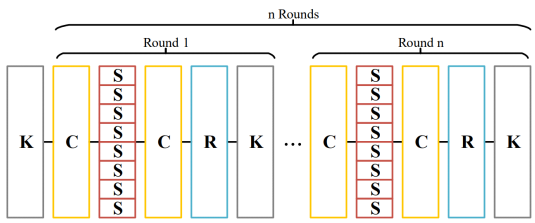


Fig. 1. Overall structure of PIPO

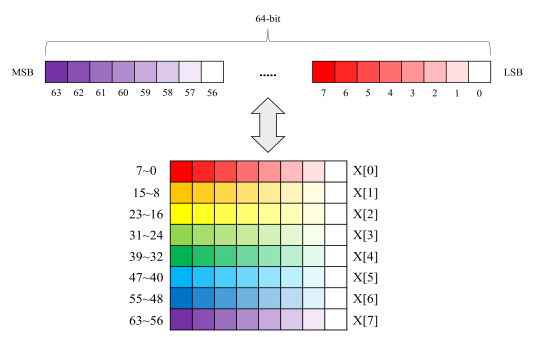


Fig. 2. Representation of 64-bit intermediate state

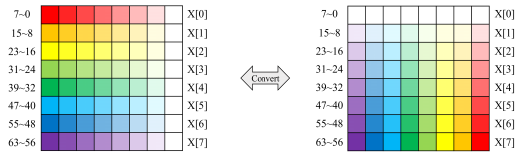


Fig. 3. Convert operation (C)

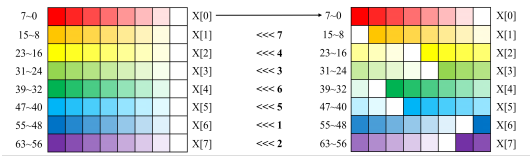


Fig. 4. R-Layer operation (R)

PIPO의 S-Layer는 비트 슬라이스 구현에서 packing, unpacking 역할을 하는 Convert(C)를 연속된 8개의 8-bit S-box(S′) 앞뒤에 배치해 S-box Look-Up 기반 구현보다 비트 슬라이스[3] 구현에 더 효율적인 구조이다.

R은 Fig. 4와 같이 행 단위 비트 회전 연산으로 S-Layer에서 나온 결과 비트를 확산시킬 수 있다.

2.3 T-table

T-table은 SPN 구조를 갖는 블록 암호 구현에서 S-box와 그에 해당하는 선형 계층의 모든 가능한 입력에 대한 출력을 사전 계산해 테이블에 저장하는 방법으로 사전에 계산된 테이블의 값을 참조해서 연산의 결과를 얻을 수 있다.

모든 가능한 입력에 대한 출력을 미리 계산한 덕분에 일반적으로 S-box, 선형 연산을 각기 독립적으로 수행하는 방법보다 빠르게 연산을 수행할 수 있다.

일반적인 경우에 S-box의 개수만큼 T-table의 개수가 요구되나 어떤 T-table을 다른 T-table을 통해 계산할 수 있다면 추가 연산을 통해 T-table의 수를 줄일 수 있다. 예를 들어 AES의 경우 순환 구조의 MixColumns 행렬을 사용하므로 하나의 T-table에 추가 회전 연산을 사용하여 나머지 T-table들을 계산하는 방식으로 구현할 수 있다.

III. T-table을 사용한 PIPO 구현

본 장에서는 T-table 사용을 위해 새롭게 정의한 비선형 연산 S′과 선형 연산 R′을 통해 PIPO를 표현한다. S′, R′의 합성인 R′∘S′과 C에 대한 각각의 T-table T_t와 C_t를 이용하여 T-table PIPO에 대한 최적화 구현을 제시한다. 또한 관찰을 통해 T_t, C_t이 순환 구조임을 확인하였고 C의 성질을 이용하여 T_t가 순환 구조임을 증명한다.

3.1 T-table 구현을 위한 PIPO의 새로운 라운드 표현

2.2절에서 언급한 바와 같이 PIPO의 라운드 함수는 〔K∘R∘C∘S′∘C〕의 형태로 표현할 수 있다. Fig. 3을 보면 C는 Involutary($C=C^{-1}$)인 연산인 것을 알 수 있다. C가 Involutary임을 이용해 결과에 영향를 주지 않는 2개의 C를 마지막 라운드 뒤에 추가할 수 있다. 다음 라운드 시작점을 C에서 S′으로 이동해서 Fig. 5와 같이 〔C∘K∘R∘C∘S′〕로 라운드 함수를 표현할 수 있다.

라운드 키, 상수에 C를 적용한 $C(RK_i)$, $C(rcon_i)$를 XOR하는 연산을 K′로 정의하면 〔C∘K〕를 〔K′∘C〕로 바꿀 수 있다. 이때 라운드 함수에서 연속된 비트 순열인 〔C∘R∘C〕를 합쳐 하나의 비트 순열 R′으로 정의할 수 있다.

최종적으로 T-table 구현을 위해 새롭게 표현한 라운드 함수를 갖는 PIPO는 Fig. 6과 같다.

다음 절에서는 R′∘S′와 C에 대한 T-table 생성 과정을 소개한다.

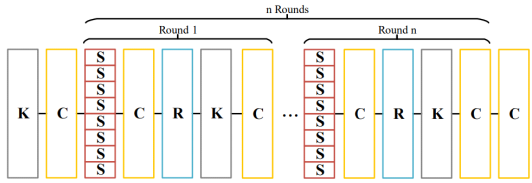


Fig. 5. A new representation of PIPO with additional C operations in last round

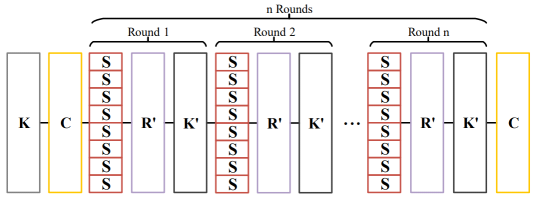


Fig. 6. A new representation of PIPO structure for T-table implementation

3.2 새롭게 표현한 PIPO의 T-table 구현

새롭게 표현한 PIPO의 S′와 R′은 다음과 같은 입출력 형태를 가진다. S′의 입출력 형태는 8-bit 이고 R′은 8-bit 입력에 대해 64-bit 결과를 출력하고 합성 연산인 R′∘S′은 8-bit 입력에 64-bit 출력을 가진다. 마찬가지로 C도 8-bit 입력에 64-bit결과를 출력한다.

R′∘S′와 C 모두 8-bit 입력에 64-bit 출력이므로 8-bit의 모든 가능한 입력(0x00~0xFF)에 대한 64-bit 결과를 미리 계산해 T-table을 만들 수 있다.

예를 들어 8-bit 입력 X[0]은 R′∘S′을 통해 Fig. 7과 같이 64-bit가 출력되므로 모든 가능한 8-bit 입력에 대한 64-bit 결과를 저장하는 T-table T_0을 만들 수 있다. 8-bit 입력 $X[i](0 \le i \le 7)$ 8개에 대해 8개의 T-table T_i을 생성할 수 있다. 마찬가지로 C에 대한 8개의 T-table C_i를 생성할 수 있다. T-table의 생성에 사용하는 입력과 함수는 Fig. 8과 같다.

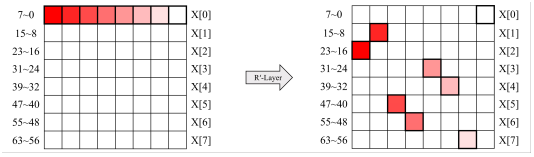


Fig. 7. 64-bit output after R′ on an 8-bit input X[0]

$T_t: R'(S(\forall X[t])), (0 \le X[t] \le 255)$		$C_t: C(\forall X[t]), (0 \le X[t] \le 255)$	
$T_0: R'(S(\forall X[0]))$	$T_1: R'(S(\forall X[1]))$	$C_0: C(\forall X[0])$	$C_1: C(\forall X[1])$
$T_2: R'(S(\forall X[2]))$	$T_3: R'(S(\forall X[3]))$	$C_2: C(\forall X[2])$	$C_3: C(\forall X[3])$
$T_4: R'(S(\forall X[4]))$	$T_5: R'(S(\forall X[5]))$	$C_4: C(\forall X[4])$	$C_5: C(\forall X[5])$
$T_6: R'(S(\forall X[6]))$	$T_7: R'(S(\forall X[7]))$	$C_6: C(\forall X[6])$	$C_7: C(\forall X[7])$

Fig. 8. Generating T-tables T_t and C_t

3.3 순환 구조 T-table

하나의 T-table로 다른 T-table들을 추가 회전 연산만으로 표현할 수 있을 때 T-table이 순환 구조를 가진다고 말한다.

PIPO의 T-table T_t, C_t가 관찰을 통해 모두 순환 구조임을 확인했고 ROL(bit rotation left) 연산에 대해 T_0에서 i-바이트만큼의 ROL, C_0에서 i-비트만큼의 ROL을 적용하면 T_i, C_i을 생성할 수 있다.

관찰 1.

$$T_i[x] = ROL(T_0[x], i \times 8)$$
$$C_i[x] = ROL(C_0[x], i)$$

T_t는 C의 성질을 이용해 순환 구조임을 보일 수 있다.

성질 1.

Fig. 9와 같은 비트 순열 A와 Fig. 10과 같은 바이트 순열 B는 어떤 입력 X에 대해

$$C(A(X)) = B(X), C(B(X)) = A(X)$$

를 만족한다.

관찰 1 증명.

Fig. 8에서 T_0, T_i 생성은 다음과 같이 쓸 수 있다.

$$T_0 : C(R(C(S'(X[0])))) \quad (1)$$

$$T_i : C(R(C(S'(X[i])))) \quad (2)$$

64-bit 블록 X의 두 8-bit 원소 $X[0], X[i]$는 i-byte 순환 관계이므로 $X[i]$를 $X[0]$로 치환하여 식

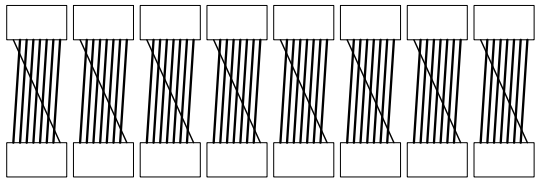


Fig. 9. Example of a bit permutation (A)

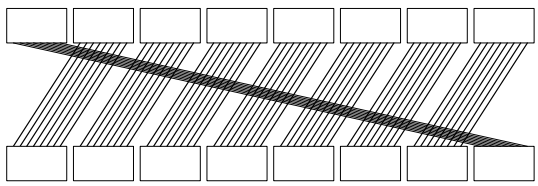


Fig. 10. Example of a byte permutation (B)

(2)를 다음과 같이 표현할 수 있다.

$$T_i : C(R(C(S'(ROL(X[0], i\times 8)))))) \quad (3)$$

바이트 단위 연산인 S′은 바이트 순환에 영향을 받지 않으므로 식 (3)을 식 (4)로 표현할 수 있다.

$$T_i : C(R(C(ROL(S'(X[0]), i\times 8)))) \quad (4)$$

식 (4)를 성질 1을 적용해 식 (5)로 표현한다.

$$\begin{aligned} T_i : C(R(&ROL(C(S'(X[0]))[0], i) \\ &\| ROL(C(S'(X[0]))[1], i) \\ &\| \ldots \\ &\| ROL(C(S'(X[0]))[7], i)) \end{aligned} \quad (5)$$

ROL과 R 둘 다 비트 회전 연산이므로 순서를 바꿀 수 있다.

$$\begin{aligned} T_i : C(&ROL(R(C(S'(X[0])))[0], i) \\ &\| ROL(R(C(S'(X[0])))[1], i) \\ &\| \ldots \\ &\| ROL(R(C(S'(X[0])))[7], i)) \end{aligned} \quad (6)$$

식 (6)에 대해 성질 1을 적용하면 식 (7)을 얻을 수 있고 식 (7)에 대해 식 (1)을 이용하여 식 (8)을 얻을 수 있다.

$$T_i : ROL(C(R(C(S'(X[0])))), i\times 8) \quad (7)$$

$$T_i : ROL(T_0, i\times 8) \quad (8)$$

따라서 식 (8)에서 ROL 연산을 사용해 T_0에서 T_i를 얻을 수 있고 이를 통해 T_t가 순환 구조를 가짐을 확인할 수 있었다. □

뿐만 아니라 R이 비트 회전이라면 식 (5), (6)이 성립하는 사실을 알 수 있었다. 즉, R이 어떤 형태의 비트 회전이든 T_t가 순환 구조임을 알 수 있었다.

3.4 S-R Layer 기반 블록 암호의 T-table

S-R Layer 기반 구조를 갖는 블록 암호들은 비트슬라이스 구현에 효율적이다. 따라서 마스킹 대응 기법 적용에도 효율적이므로 최근 많은 연구가 진행되고 있다.

PIPO와 유사한 SPN 구조를 가지는 블록 암호들에 대해 T-table을 이용하여 최적화 구현하였을 때, T-table이 모두 순환 구조를 갖는다는 것을 3.3절과 유사하게 증명할 수 있다. 따라서 T-table PIPO와 유사한 방법으로 T-table 수를 줄일 수 있는 방법을 고안할 수 있다.

본 논문에서 제시하는 방법이 적용될 수 있는 선형 계층을 갖는 암호로는 PIPO와 같은 S-R Layer 형태를 가지는 Fig. 11과 같은 FLY[6]와 Fig. 12과 같은 RECTANGLE[5] 등이 있다.

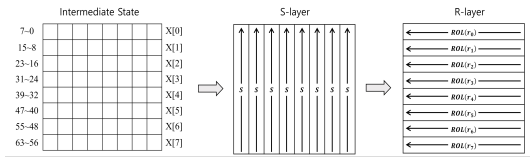


Fig. 11. PIPO/FLY S-R layer

IV. 실험 결과

실험은 프로세서 Intel Core i7-9700K 64-bit 프로세서 환경에서 GCC 컴파일러 -O3 옵션을 사용해서 진행했다. 실험에는 모두 동일한 키를 사용해 5,000,000번 암호화에 대한 속도(Gbps)를 측정했다.

4.1 코드 크기-속도 상충관계분석

T-table이 순환 구조임을 이용하면 하나의 T-table에 추가 회전 연산만을 이용하여 다른 T-table을 계산하여 사용하는 T-table의 수를 줄일 수 있다. 하지만 추가한 회전 연산의 수만큼 속도가 감소한다. 이러한 사실을 구현 예시인 Algorithm 1, 2를 통해 확인할 수 있다.

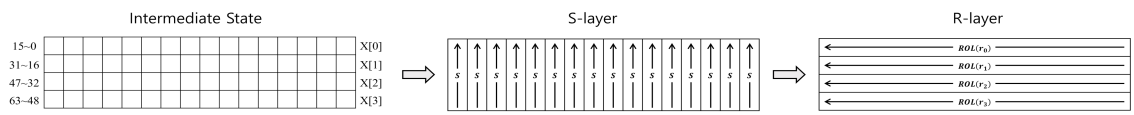


Fig. 12. RECTANGLE S-R layer

Table 2. Throughput, code size, and throughput-to-code size ratio of PIPO according to the number of T-tables T_t and C_t
(0C : Omit the Convert operation before and after the PIPO Round function)

		Throughput (Gbps)					Code Size (KB)					Throughput/Code Size (Kbps/KB)				
		0C	1C	2C	4C	8C	0C	1C	2C	4C	8C	0C	1C	2C	4C	8C
128	1T	1.22	1.11	1.10	1.11	1.09	27.14	30.21	32.26	35.84	44.03	44.80	36.64	34.22	30.85	24.66
	2T	1.24	1.13	1.13	1.13	1.14	29.18	31.74	33.79	37.89	46.08	42.40	35.59	33.58	29.70	24.64
	4T	1.45	1.31	1.32	1.32	1.33	32.26	35.33	37.38	41.47	49.15	45.04	37.12	35.29	31.77	27.01
	8T	2.80	2.33	2.33	2.38	2.37	41.47	44.54	46.59	50.69	58.37	67.46	52.39	49.95	46.94	40.57
256	1T	0.93	0.87	0.87	0.87	0.87	28.67	31.74	33.79	37.89	46.08	32.56	27.51	25.88	22.97	18.98
	2T	0.95	0.89	0.89	0.89	0.89	30.72	33.79	35.84	39.94	47.62	30.96	26.25	24.79	22.24	18.73
	4T	1.13	1.03	1.04	1.04	1.04	33.79	36.86	38.91	43.01	50.69	33.33	27.94	26.77	24.10	20.60
	8T	1.67	1.48	1.49	1.48	1.49	39.42	43.01	45.06	48.64	56.83	42.47	34.31	33.17	30.51	26.19

Best Case

본 절에서는 T-table 수 변화에 따른 PIPO-64/128, 256에 대한 최적화 구현의 속도와 코드 크기에 대한 상충 관계를 분석한다. T_t, C_t의 수를 각각 nT, nC라고 표현할 때 1T, 2T, 4T, 8T와 1C, 2C, 4C, 8C 그리고 C를 하지 않는 경우(0C)들을 조합하여 속도와 코드 크기를 구하고 코드 크기 당 속도를 분석한다.

Table 2를 보면 0C인 경우를 제외하고 속도는 키가 128-bit인 경우 〔8T, 4C〕에서 2.38Gbps, 256-bit는 〔8T, 8C〕에서 1.49Gbps로 가장 높은 속도를 얻었고, 코드 크기는 키가 128-, 256-bit일 때 모두 〔1T, 1C〕일 때 각각 30.21KB, 31.74KB로 가장 작은 코드 크기를 가지는 것을 알 수 있었다. 속도/코드 크기 부분에서는 키가 128-, 256-bit일 때 모두 〔8T, 1C〕일 때 각각 52.39Kbps/KB, 34.31Kbps/KB로 가장 큰 값을 가진다는 것을 알 수 있었다.

Table 3에서 nT, nC의 n이 커질수록 *ROL* 연산 수는 적어지고, n이 작아질수록 *ROL* 연산 수는 많아진다는 것을 확인할 수 있었다. 또한 nC보다 nT가 *ROL* 연산 수의 영향을 많이 주는 것을 확인할 수 있었다.

Table 3. Comparison of the number of *ROL* operations according to the number of T-tables T_t and C_t

		1C	2C	4C	8C
128	1T	105	103	99	91
	2T	92	90	86	78
	4T	66	64	60	52
	8T	14	12	8	0
256	1T	133	131	127	119
	2T	116	114	110	102
	4T	82	80	76	68
	8T	14	12	8	0

4.2 속도 비교

4.1절에서 PIPO-64/128에서는 〔8T, 4C〕, PIPO-64/256에서는 〔8T, 8C〕일 때 가장 높은 속도를 갖는다는 것을 알 수 있었다. 가장 높은 속도의

Table 4. Throughput evaluation

	Implementation	Gbps	
PIPO-64/128	T-table (8T, 4C)	2.38	Ours
	Reference TLU	0.21	Ref
	Reference Bit Slice	0.72	
	MultiBlock Bit Slice	3.82	
AES-128 (T-table)		2.76	
PIPO-64/256	T-table (8T, 8C)	1.49	Ours
	Reference TLU	0.16	Ref
	Reference Bit Slice	0.54	
	MultiBlock Bit Slice	3.02	
AES-256 (T-table)		2.07	

제안 구현물과 기존 PIPO 레퍼런스 TLU, 비트 슬라이스, 8개의 멀티 블록 비트 슬라이스 구현물, T-table을 이용한 AES와 속도를 비교한다. (단, 멀티 블록 비트 슬라이스는 준비된 평문과 병렬 처리가 가능한 CTR, ECB 운영모드라 가정한다.)

실험 결과 본 논문 구현물의 속도는 TLU 레퍼런스 구현에 비해 PIPO-64/128, 256에서 각각 11.33, 9.31배, 비트 슬라이스 레퍼런스 구현에 비해 각각 3.31, 2.76배 향상된 속도를 갖는다. 하지만 멀티 블록 비트 슬라이스 구현에 비해 각각 0.62, 0.49배, T-table을 이용한 AES-128, 256 구현에 비해 각각 0.86, 0.72배 속도를 갖는다는 것을 Table 4을 통해 확인할 수 있었다.

PIPO 구현 시 병렬 처리가 가능한 CTR, ECB 운영모드에서는 멀티 블록 비트 슬라이스 구현이 효율적이라 볼 수 있지만, 병렬 처리가 불가능한 운영모드(예: CBC)에서는 T-table을 사용한 단일 블록 암호화 구현이 가장 속도가 높고 효율적임을 알 수 있다.

V. 결 론

본 논문에서는 T-table을 사용하여 경량 블록 암호인 PIPO를 구현하였다. 그 과정에서 PIPO의 $R' \circ S'$ T-table T_t와 C T-table C_t가 순환 구조임을 보이고 이를 통해 사용하는 T-table 개수를 줄여 코드 크기를 줄일 수 있는 기법을 제시했다. 뿐만 아니라 PIPO와 유사한 S-R Layer 기반 블록 암호에 대해 T-table을 이용한 최적화 구현 시, 사용하는 T-table이 항상 순환 구조임을 증명했다.

추가적으로 T-table 수에 따른 속도를 분석한 결과 PIPO-64/128, 256일 때 모두 〔8T, 1C〕에서 코드 크기 당 속도가 가장 뛰어남을 알 수 있었고 테이블 수에 따른 ROL 연산 수를 계산하여 구현 간 속도 차이를 이론적으로 분석하였다.

T-table PIPO 구현물은 TLU 레퍼런스 구현에 비해 PIPO-64/128, 256에서 각각 11.33, 9.31배, 비트 슬라이스 레퍼런스 구현에 비해 각각 3.31, 2.76배 향상된 속도를 갖지만 멀티 블록 비트 슬라이스 구현에 비해 각각 0.62, 0.49배, T-table을 이용한 AES-128, 256구현에 비해 각각 0.86, 0.72배 속도가 느린 것을 확인했다.

그러나 멀티 블록 비트 슬라이스를 이용하기 위해서 미리 준비된 평문이 필요하고 CTR, ECB 모드에서만 실현 가능하다는 제약 조건이 존재하므로 CBC와 같은 병렬 처리가 불가능한 운영모드에서는 T-table 기법을 적용한 본 논문의 결과는 기존 단일 블록에 대한 레퍼런스 구현보다 속도를 향상시킨다는 점에서 의미가 있다.

References

〔1〕 H. Kim, Y. Jeon, G. Kim, J. Kim, B. Sim, D. Han, H. Seo, S. Kim, S. Hong, J. Sung, and D. Hong, "PIPO: A Lightweight Block Cipher with Efficient Higher-Order Masking Software Implementations," ICISC'20, LNCS12593, pp. 99 - 122, 2020.

〔2〕 National Institute of Standards and Technology (NIST), FIPS-197: Advanced Encryption Standard (November 2001).

〔3〕 Eli Biham, "A Fast New DES Implementation in Software," FSE'97,LNCS 1267, pp. 260-272, Jan. 1997.

〔4〕 H. Kim, H. Kim, S. Eum, H. Kwon, Y. Yang, and H. Seo, "Parallel Implementation of PIPO and Its Application for Format Preserving Encryption," IEEE Access, vol. 10, pp. 99963-99972, Sep. 2022.

〔5〕 W. Zhang, Z. Bao1, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "RECTANGLE: A Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms," Science China Information Sciences, vol. 58, pp. 1-15, Nov. 2015.

〔6〕 P. Karpman and B. Grégoire, "The LITTLUN S-box and the FLY block cipher," in Proc. Lightweight Cryptogr. Workshop, 2016, pp. 17 - 18.

〔7〕 S. W. Eum, H. D. Kwon, H. J. Kim, K. B. Jang, H. J. Kim, J. H. Park, G. J. Song, M. J. Sim, and H. J. Seo, "Optimized Implementation of Block Cipher PIPO in Parallel-Way on 64-bit ARM Processors," KIPS Transactions on Computer and Communication Systems, vol. 10, no. 8, pp. 223-230, Aug. 2021.

〔8〕 H. Choi and S. Seo, "Efficient Parallel

Implementations of PIPO Block Cipher on CPU and GPU," IEEE Access, vol. 10, pp. 85995-86007, Aug. 2022.

[9] Y. Kwak, Y. Kim, and S. Seo, "Parallel Implementation of PIPO Block Cipher on 32-bit RISC-V Processor," WISA'21, LNSC13009, pp. 183-193, 2021.

[10] Y. Kim and S. Seo, "Optimized Implementation of PIPO Block Cipher on 32-Bit ARM and RISC-V Processors," IEEE Access, vol. 10, pp. 97298-97309, Sep. 2022.

Appendix

Algorithm 1. T-table PIPO Enc - Using $8T, 1C$

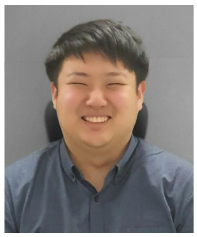
1. // T-table : $T_0, T_1, \ldots, T_7, C_0$
2. // Input : pt, Output : ct
3. // Whitening
4. $K(pt, RK_0, rcon_0)$
5. // Convert
6. $pt = C_0[pt[0]] \oplus ROL(C_0[pt[1]], 1) \oplus \ldots \oplus ROL(C_0[pt[7]], 7)$
7. for i from 1 to n do
8. // S'-R' Layer T-table Look Up
9. $pt = T_0[pt[0]] \oplus T_1[pt[1]] \oplus \ldots \oplus T_7[pt[7]]$
10. // XOR Convert(RK_i), Convert($rcon_i$)
11. $K'(pt, RK_i, rcon_i)$
12. // Convert
13. $pt = C_0[pt[0]] \oplus ROL(C_0[pt[1]], 1) \oplus \ldots \oplus ROL(C_0[pt[7]], 7)$
14. $ct = pt$
15. return ct

Algorithm 2. T-table PIPO Enc - Using $1T, 1C$

1. // T-table : T_0, C_0
2. // Input : pt, Output : ct
3. // Whitening
4. $K(pt, RK_0, rcon_0)$
5. // Convert
6. $pt = C_0[pt[0]] \oplus ROL(C_0[pt[1]], 1) \oplus \ldots \oplus ROL(C_0[pt[7]], 7)$
7. for i from 1 to n do
8. // S'-R' Layer T-table Look Up
9. $pt = T_0[pt[0]] \oplus ROL(T_0[pt[1]], 8\times 1) \oplus \ldots \oplus ROL(T_0[pt[7]], 8\times 7)$
10. // XOR Convert(RK_i), Convert($rcon_i$)
11. $K'(pt, RK_i, rcon_i)$
12. // Convert
13. $pt = C_0[pt[0]] \oplus ROL(C_0[pt[1]], 1) \oplus \ldots \oplus ROL(C_0[pt[7]], 7)$
14. $ct = pt$
15. return ct

〈저 자 소 개〉

최 민 식 (Minsig Choi) 학생회원
2023년 2월: 홍익대학교 컴퓨터공학과 졸업
2023년 3월~현재: 고려대학교 정보보호학과 석사과정
〈관심분야〉 부채널 분석, 암호 알고리즘 설계 및 분석



김 선 엽 (Sunyeop Kim) 학생회원
2019년 8월: 고려대학교 수학과 졸업
2019년 9월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

김 인 성 (Insung Kim) 학생회원
2022년 2월: 서울시립대 수학과 졸업
2022년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

신 한 범 (Hanbeom Shin) 학생회원
2022년 2월: 서울시립대 수학과 졸업
2022년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

김 성 겸 (Seonggyeom Kim) 정회원
2016년 8월: 한양대학교 수학과 졸업
2016년 9월~2018년 8월: 고려대학교 정보보호학과 석사
2019년 3월~2023년 2월: 고려대학교 정보보호학과 박사
2023년 3월~현재: 삼성전자
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호

홍 석 희 (Seokhie Hong) 종신회원
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀리지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털 포렌식