Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.2, Apr. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.2.183

사물 인터넷 프로세서 8-bit AVR 상에서의 경량암호 TinyJAMBU 고속 최적 구현*

권 혁 동,[1†] 엄 시 우,[1] 심 민 주,[1] 양 유 진,[1] 서 화 정[2‡]
[1,2]한성대학교 (대학원생, 교수)

A High Speed Optimized Implementation of Lightweight Cryptography TinyJAMBU on Internet of Things Processor 8-Bit AVR*

Hyeok-Dong Kwon,[1†] Si-Woo Eum,[1] Min-Joo Sim,[1]
Yu-Jin Yang,[1] Hwa-Jeong Seo[2‡]
[1,2]Hansung University (Graduate student, Professor)

요 약

암호 알고리즘은 많은 연산 자원을 요구하며 복잡한 수학적 원리를 통해 보안성을 가진다. 하지만 대부분의 사물 인터넷 기기는 가용 자원이 한정적이며 그에 따라 연산 성능이 부족하다. 따라서 연산량을 적게 사용하는 경량암호가 등장하였다. 미국 국립표준기술연구소는 경량암호 표준화 공모전을 개최하여 경량암호의 원활한 보급을 꾀했다. 공모전의 알고리즘 중 하나인 TinyJAMBU는 순열 기반의 알고리즘이다. TinyJAMBU는 키 스케줄을 거치지 않는 대신 많은 순열 연산을 반복하며, 이때 시프트 연산이 주로 사용된다. 본 논문에서는 8-bit AVR 프로세서상에서 경량암호 TinyJAMBU를 고속 최적 구현하였다. 제안 기법은 시프트 연산을 반대 방향으로 하여 시프트 횟수를 최소화한 리버스 시프트 기법과 키와 논스가 고정인 환경에서 일부 연산을 사전 연산한 기법이다. 제안 기법은 순열 연산에서 최대 7.03배, TinyJAMBU 알고리즘에 적용 시 최대 5.87배 성능 향상을 보였다. 키와 논스가 고정인 환경에서는 TinyJAMBU의 알고리즘이 최대 9.19배만큼 성능이 향상되었다.

ABSTRACT

Cryptographic algorithms require extensive computational resources and rely on complex mathematical principles for security. However, IoT devices have limited resources, leading to insufficient computing power. As a result, lightweight cryptography has emerged, which uses fewer computational resources. NIST organized a competition to standardize lightweight cryptography and TinyJAMBU, one of the algorithms in the competition, is a permutation-based algorithm that repeats many permutation operations. In this paper, we implement TinyJAMBU on an 8-bit AVR processor with a proposed technique that includes a reverse shift method and precomputing some operations in a fixed key and nonce environment. Our techniques showed a maximum performance improvement of 7.03 times in permutation operations and 5.87 times in the TinyJAMBU algorithm, improving up to 9.19 times in a fixed key and nonce environment.

Keywords: Lightweight cryptography, NIST, TinyJAMBU, 8-bit AVR Microcontrollers

Received(01. 17. 2023), Modified(03. 06. 2023), Accepted(03. 06. 2023)

* 본 논문은 2022년도 한국정보보호학회 충청지부 학술대회에 발표한 우수논문을 개선 및 확장한 것임

* This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government (MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 50%) and this work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 50%).

† 주저자, korlethean@gmail.com

‡ 교신저자, hwajeong84@gmail.com(Corresponding author)

I. 서 론

사물 인터넷(IoT, Internet of Things)은 다양한 사물끼리 무선 통신을 사용하여 인터넷에 연결하는 기술이다[1]. 사물을 인터넷에 연결하기 위해서는 각종 센서와 통신 장비가 내장되는데, 이를 위해서 사물 인터넷을 지원하는 프로세서가 등장하였다. 사물 인터넷 프로세서는 사물 인터넷 기기의 크기를 고려하여 크기가 작으며 그에 따라 연산 능력도 부족하다. 대부분의 암호 알고리즘은 복잡한 연산 구조와 많은 연산 자원을 요구한다. 때문에 사물 인터넷 환경에서는 원활한 암호 알고리즘의 가동이 어렵다.

경량암호는 사물 인터넷과 같이 제한된 환경에서도 원활하게 가동할 수 있는 암호 알고리즘의 한 종류이다. 경량암호는 표준 암호 알고리즘이 있었기에 크게 주목받지는 못했다. 하지만 사물 인터넷 기기는 제한된 자원 때문에 표준 암호 알고리즘을 가동하기가 어려웠다. 사물 인터넷 환경이 발전함에 따라 보안성을 제공할 수 있는 경량암호의 중요성이 대두되었다. 이에 따라 2018년에 미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)는 경량암호의 보급과 안전한 사물 인터넷 환경 구축을 위해 표준화를 개최하였다.

본 논문에서는 경량암호 공모전의 출품작 중 하나인 TinyJAMBU를 8-bit AVR 프로세서 상에서 최적 구현하였다. 제안하는 기법은 크게 두 가지이다. 첫째는 TinyJAMBU의 주된 연산을 담당하는 keyed permutation에 리버스 시프트를 사용하여 시프트 횟수를 줄인 기법이다. 둘째는 사물 인터넷 기기 상에서 키와 논스 갱신이 적을 때 사용하는 기법으로, 초기화 과정을 사전 연산하여 연산 속도를 끌어 올린 기법이다.

본 논문의 구성은 다음과 같다. 2장에서 경량암호 공모전과 TinyJAMBU에 대해서 확인하며 대상 프로세서인 AVR 프로세서와 다른 경량암호 구현 동향에 대해 알아본다. 3장에서는 제안하는 기법인 리버스 시프트와 초기화 사전 연산 기법을 소개한다. 4장에서는 제안하는 기법과 기존 구현물 간의 성능 평가를 진행한다. 5장에서는 본 논문의 결론을 맺는다.

II. 배 경

2.1 NIST 경량암호 공모전

NIST의 경량암호 공모전은 2018년 개최되어 2019년 1라운드 결과를 발표하였다[2]. 1라운드에서는 57개의 알고리즘이 선정되었고, 2022년 최종 라운드에서는 10개의 알고리즘만이 남았다[3]. 표 1은 최종 라운드에 선정된 알고리즘의 목록과 특성을 간략하게 정리하였다.

각 후보는 동작 구조상 분류가 존재하며 블록 암호(block cipher), 트위커블 블록 암호(twekable block cipher), 스트림 암호(stream cipher), 그리고 순열(permutation)이 있다. NIST의 요구사항은 모든 알고리즘은 반드시 연관 데이터 인증 방식(AEAD, Authenticated Encryption with Associated Data) 모드를 제공해야 하며 해시 기능은 선택 제공이라고 명시하였다[4]. 최종 알고리즘 10종 중에서 5종은 해시를 지원하고 나머지 5종은 해시를 지원하지 않는다. 최종 알고리즘 중에서는 순열 기반 알고리즘이 7종으로 순열 기반이 강세를 보인다.

Table 1. Finalist of NIST lightweight cryptography standardization competition.

Name	Type	Hash	Core function
Grain-128 AEAD	Stream	-	Grain-128a
GIFT-COFB	Block	-	GIFT-128
Romulus	Tweakable block	O	SKINNY-128-256 SKINNY-128-384
ASCON	Permutation	O	ASCON-320
ISAP		-	Keccak-400 ASCON-320
PHOTON-Beetle		O	PHOTON-256
Elephant		-	Spongent-160 Spongent-176 Keccak-200
SPARKLE		O	Sparkle-256 Sparkle-384 Sparkle-512
Tiny JAMBU		-	JAMBU-128
Xoodyak		-	Xoodoo-384

2.2 TinyJAMBU

경량암호 TinyJAMBU는 CAESAR 경진대회에 출품되었던 블록 암호인 JAMBU를 변형한 알고리즘이다[5]. JAMBU는 CAESAR에서 가장 작은 블록 크기를 지닌 알고리즘으로 블록 단위의 암호화를 제공하지만, TinyJAMBU는 순열 기반 알고리즘으로 동작한다. 이런 특성으로 NIST 경량암호 공모전의 초기에는 TinyJAMBU를 블록 암호 기반으로 분류했었지만, 최종 라운드에서 순열 기반 알고리즘으로 수정되었다.

TinyJAMBU의 암▪복호화 과정은 크게 다섯 단계로 구성되며 각 단계는 초기화(initialization), 연관 데이터 생성(processing Associated Data, AD), 암호화/복호화(encryption/decryption), 마무리(finalization), 그리고 검증(verification)이 있다. 이때 TinyJAMBU의 모든 과정에서 공통으로 keyed permutation이 필요하다. 즉, 해당 연산이 TinyJAMBU에서 중심 역할을 하는 것을 알 수 있다.

Keyed permutation의 가장 큰 특징은 키 스케줄을 거치지 않고 비밀키 값을 그대로 state 변수에 반영한다. Keyed permutation의 내부에서는 그림 1과 같은 NLFSR(NonLinear Feedback Shift Register) 구조를 사용한다[6]. NLFSR은 일종의 시프트 레지스터로 난수 생성과 같은 역할을 할 수 있다.

TinyJAMBU에서는 그림 1과 같이 128 비트의 keyed permutation을 사용한다. 이때 각 permutation을 P라하고, 최대 라운드는 n, 현재 라운드는 i라 한다. 이를 의사 코드 형태로 표현한다면 표 2와 같은 코드로 작성된다.

Keyed permutation은 state 변수의 순서를 섞고 중간에 키 값을 삽입하여 state 변수를 갱신시키는 역할을 한다. 단계별로 keyed permutation을 실행하는 횟수는 정해져 있으며, 이는 표 3에서 확인할 수 있다. 특이사항으로 finalization의 횟수

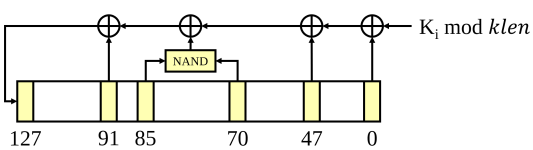

Fig. 1. Structure of Nonlinear Feedback Shift Register.

Table 2. Pseudo code of the keyed permutation.

Input: State $s[4]$(128-bit), Key k, Round n
Output: State $s[4]$(128-bit)
StateUpdate(s, k, n)
for i = 0 to n
t1 = ($s[1]$>>15)|($s[2]$<<17)
t2 = ($s[2]$>>6)|($s[3]$<<26)
t3 = ($s[2]$>>21)|($s[3]$<<11)
t4 = ($s[2]$>>27)|($s[3]$<<5)
feedback = $s[0]$^t1^(~(t2&t3))^t4^k
$s[0]$ = $s[1]$
$s[1]$ = $s[2]$
$s[2]$ = $s[3]$
$s[3]$ = feedback
end for

Table 3. Number of keyed permutation list.

Key length	128	192	256
Initialization: key setup	1024	1152	1280
Initialization: nonce setup	640	640	640
Processing AD	640	640	640
Encryption Decryption	1024	1152	1280
Finalization	1024, 640	1152, 640	1280, 640

는 두 가지가 있는데, 이는 태그 값의 finalization 과정에서 전반부는 1024회, 후반부는 640회를 하기 때문이다. 전체적으로 Keyed permutation은 반복 횟수가 매우 많으므로 TinyJAMBU의 연산 중에서 가장 큰 부하를 일으킨다. 따라서 본 논문에서는 keyed permutation을 최적화하고 그 횟수를 줄이는 것에 집중하여 TinyJAMBU를 고속 최적 구현한다.

2.3 관련 연구 동향

본 논문에서 대상으로 하는 8-bit AVR 상에서의 다른 경량암호 최적화 구현을 확인한다. 국산 경량암호인 CHAM은 ARX 구조로 되어있는 암호로, 2017년 첫 제안, 2019년에는 개정판이 제안되었다[7]. [8]는 경량암호 CHAM의 카운터 모드를 구현

한 것으로, 카운터 모드 적용 시에는 논스와 카운터 값이 항상 고정된다는 특징을 활용하였다. 연산의 첫 8라운드의 일부 구간이 값이 항상 같으므로 이를 look-up table에 저장하여 호출하는 방식을 사용했다. 구현 결과, 기존 대비 최대 12.8%의 성능 향상이 있었다. [9]는 라운드 키를 미리 레지스터에 상주시키는 방법을 추가로 적용하였다. 이는 [8]에 비해서 최대 6.8%의 성능 향상을 가진다.

경량블록 암호 SIMON은 2013년 미국의 국가안보국에서 제안한 알고리즘으로 ARX 구조를 지닌 Feistel 구조의 암호이다[10]. SIMON은 다양한 블록 크기를 지원하여 긴 메시지도 효과적으로 암호화할 수 있게 설계되었다. [11]은 SIMON의 카운터 모드를 AVR 상에서 최적 구현하였다. 통상적으로 SIMON의 입력 값은 카운터에 영향을 받으나, AVR 상에서는 8-bit 단위로 연산이 진행된다는 점에 착안하여 각 블록 크기 별로 생략이 가능한 범위가 다르다. 제안하는 기법은 기존 SIMON에 비해 최대 5.3%의 성능 향상을 보였다.

III. 제안 기법

제안 기법은 크게 두 종류로 keyed permutation의 내부를 최적화한 리버스 시프트 기법, 그리고 키와 논스를 재사용하는 환경에서 사용 가능한 초기화 생략 방법을 제안한다.

3.1 구현 환경

대상 프로세서는 8-bit AVR 마이크로컨트롤러이다. AVR 프로세서는 8-bit 프로세서로 하나의 레지스터가 8-bit까지 저장할 수 있다. 범용 레지스터의 수는 32개로 효과적인 구현을 위해서 그림 2와 같이 레지스터 할당 계획을 수립한다.

제안하는 기법은 주로 AVR의 어셈블리 명령어를 사용하여 구현한다. 실제 사용한 명령어는 표 4의

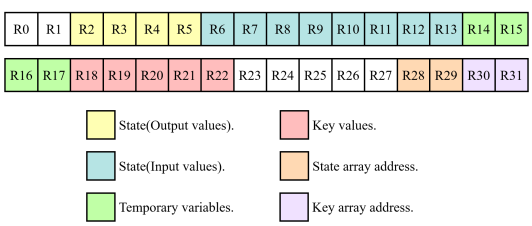

Fig. 2. Registers allocation plan.

Table 4. Instruction set list.

Instruction	Description
AND	Logical AND.
CLR	Clear register.
COM	One's complement.
EOR	Rotate right.
LDD	Load indirect.
LSL	Logical shift left.
MOVW	Copy register word.
OR	Logical OR.
POP	Pop register from stack.
PUSH	Push register on stack.
ROL	Rotate left.
STD	Store indirect.

명령어들을 사용하였다[12].

3.2 Reverse shift

제안하는 첫 번째 기법은 리버스 시프트이다. Keyed permutation 과정에서 하나의 상태 변수는 한 쪽 방향으로만 시프트된다. 가령 표 2에서 t1을 계산할 때를 제외하고 s[2]는 우측으로 6회, 우측으로 21회, 우측으로 27회 시프트가 진행됨을 알 수 있다. 이때 하나의 상태 변수는 32-bit이므로, 4개의 레지스터를 사용한다. 레지스터별로 단계별 상태 변수 s[2]의 변화는 그림 3과 같이 표현할 수 있다. 그림 3에서 한 칸은 레지스터의 1-bit를 뜻한다. 비트마다 값의 이동을 확인할 수 있도록 서로 다른 색으로 되어있으며, 같은 색은 같은 값을 의미한다. 흰색은 비어있는 값을 뜻한다. 가장 위의 값은 작업하지 않았으므로 0단계라 칭한다. 1단계 값은 0단계에서 우측으로 6회 시프트 하여 생성한다. 2단계는 0단계에서 21회 시프트된 값이지만, 1단계에서 6회 시프트를 이미 했으므로 15회만 추가로 진행한다. 마찬가지로 마지막 값을 생성하는 단계에서도 5

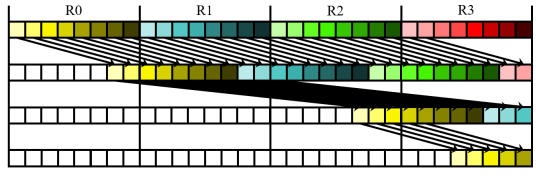

Fig. 3. Computation steps of original s[2] values.

회의 시프트를 진행한다. 따라서 총 시프트 횟수는 27회가 된다.

이때 21회 시프트된 값인 2단계의 값을 주목한다. 2단계에서는 0단계 R0의 전체 값과 0단계 R1의 상위 3비트에서 유래한 값들이 존재한다. 이는 1단계에서 상당히 쉽게 생성할 수 있는데, 1단계의 R0, R1, R2 값을 우측으로 1회 시프트 하면 의도한 값을 생성할 수 있다. 또한, 3단계에서는 0단계 R0의 상위 5비트 값만 필요하다. 이는 2단계에서 5회 시프트를 하는 대신, 우측으로 2회 시프트를 해주면 완성할 수 있다. 이를 리버스 시프트 방법이라 칭한다. 이 과정을 그림으로 표현한다면 그림 4와 같다. 원래 시프트 방향과 반대 방향으로 시프트 할 때는 다른 색상의 화살표로 표시하였다. 2단계와 3단계에서는 기존과는 반대 방향으로 시프트되며 그 횟수도 1회와 2회로 전체 시프트 횟수는 9회가 된다.

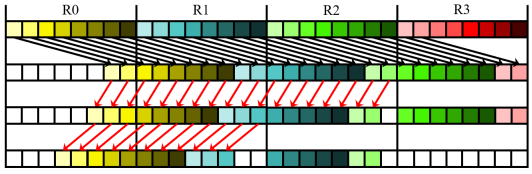

Fig. 4. Reverse shift technique for s[2] values.

s[2]가 우측으로 시프트를 진행했다면, s[3]는 좌측으로 시프트를 진행한다. 그러나 s[3]는 s[2]와는 다르게 시프트가 많은 횟수부터 먼저 진행하게 된다. 이럴 경우, 원본값이 소실되는 문제가 있어서 시프트를 누적시키며 진행할 수 없다. 따라서 26회, 11회, 5회 시프트 값 생성을 역순으로 진행하여 5회, 11회 26회 시프트 값 생성을 한다. 이를 그림으로 표현하면 그림 5와 같다.

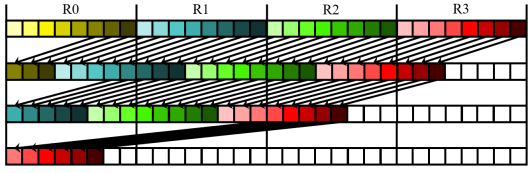

Fig. 5. Computation steps of original s[3] values.

s[2]와 마찬가지로 2단계, 3단계부터는 리버스 시프트를 적용하여 좌측이 아닌 우측으로 시프트를 진행한다. 2단계 값은 1단계 값에서 오른쪽으로 2회 시프트를 진행하는 것으로 완성할 수 있다. 3단계 값은 2단계 값에서 1회 우측으로 시프트 하면 완성할 수 있다. 이 과정을 시각화하면 그림 6과 같다. 기존 전체 시프트 횟수는 26회지만, 리버스 시프트를 사용하게 되면 8회로 모든 값을 생성할 수 있다.

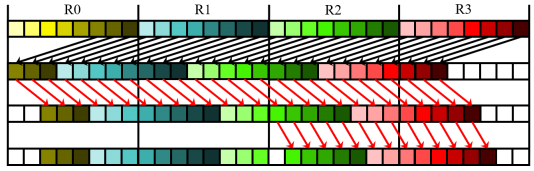

Fig. 6. Reverse shift technique for s[3] values.

t1 값을 생성하는 경우에는 s[1]과 s[2]를 사용하는데, 이 값들은 t1 생성에만 관여하고 다른 임시 값 생성에는 사용하지 않는다. 따라서 다른 값들과 다르게 여러 단계가 아닌 한 단계만 거쳐서 결과 값을 만들도록 한다. s[1]의 경우에는 우측으로 15회 시프트가 진행된다. 이 결과 s[1]의 상위에서 17비트가 남게 된다. 이를 리버스 시프트로 쉽게 계산하기 위해서는 좌측으로 1회 시프트 해준다. 마찬가지로 s[2]는 좌측으로 17회 시프트가 되는데, 이는 원본 값에서 하위 15-bit만 분리한다. 이 경우에는 리버스 시프트를 적용하지는 않지만, 하위 16-bit 값만 로드한 다음 좌측으로 1회 시프트를 하는 것으로 값을 계산할 수 있다.

즉 제안하는 기법은 시프트 횟수를 줄여서 연산 효율을 높이는 데 집중하였다. 표 5에서는 기존 기법과 제안하는 기법의 시프트 횟수 차이를 한눈에 확인할 수 있도록 전체 시프트 횟수를 정리해두었다.

이처럼 리버스 시프트는 기존 TinyJAMBU의 알고리즘보다 시프트 횟수를 줄여주는 장점을 지닌다.

Table 5. Number of shifts list.

	Reference	Proposed
s[2]>>6	6	6
s[2]>>21	15	1
s[2]>>27	5	2
s[3]<<5	5	5
s[3]<<11	6	2
s[3]<<26	15	1
s[1]>>15	15	1
s[1]<<17	17	1

또한, 리버스 시프트를 사용하게 되면 시프트 명령어를 적용하는 레지스터의 수도 줄어들게 된다. 가령 그림 3에서는 시프트 명령어를 적용하는 레지스터가 4, 4, 2개인 것을 확인할 수 있다. 반면 리버스 시프트를 적용한 그림 4에서는 4, 3, 2개로 시프트가 필요한 레지스터가 1개 줄어들게 된다. 이는 결과 값 생성에 필요 없는 레지스터는 시프트를 사용할 필요가 없기 때문이다.

3.3 Initialization skip

사물 인터넷과 같은 소형기기상에서는 키나 논스와 같은 상수 값을 재사용하는 경우가 많다. TinyJAMBU에서 초기화 단계는 키와 논스만 사용되며 그 외 입력 값은 사용되지 않는다. 즉, 키와 논스가 갱신되지 않았다면 초기화 과정의 결과물은 항상 같다. 초기화 과정의 결과물은 상태 변수에 반영되며 이는 총 128-bit이다, 따라서 한 번 키와 논스를 사용하여 초기화 과정을 진행한 후, 생성한 상태 변수 128-bit를 따로 저장한다. 그 후로는 저장한 변수를 그대로 호출하여 사용하는 것으로 초기화 과정을 생략할 수 있다. 이 경우 초기화 과정에서 필요한 keyed permutation을 그대로 생략할 수 있으므로 동작 속도 면에서 큰 이득을 볼 수 있다.

IV. 성능 평가

제안하는 기법은 TinyJAMBU의 레퍼런스 코드와 성능 비교를 진행한다. TinyJAMBU의 레퍼런스 코드는 일반 버전과 최적화 버전 두 가지가 존재하며, [6]에서 제시한 코드를 사용하였다. 일반 버전은 TinyJAMBU의 가장 기본적인 형태이다. 최적화 버전은 keyed permutation 과정에서 상태 변수 이동을 제거하여 속도를 개선한 알고리즘이다. 본 논문에서 제안하는 리버스 시프트와 초기화 생략 기법은 최적화 버전을 기반으로 구현하였다.

구현은 Microchip Studio 프레임워크를 사용하여 구현하였으며, ATmega128 프로세서를 대상으로 한다. 컴파일 옵션으로 -O3(fastest)를 적용하고 평문 길이는 8바이트를 사용하였다. 성능 평가 단위는 clock cycles를 사용하였다. 우선 keyed permutation의 성능 비교를 진행한다. 비교 결과는 표 6과 같다.

Table 6. Results of keyed permutation evaluation. (N: Normal, O: Optimized, R: Reverse shift, Diff: Clock cycles difference with the proposed technique, Unit: clock cycles)

Alg.	Type	Clock cycles	Diff.
640-128	N	21752	6.05
	O	20135	5.60
	R	**3594**	-
640-192	N	22325	6.21
	O	20968	5.83
	R	**3597**	-
640-256	N	21752	6.05
	O	20278	5.64
	R	**3596**	-
1024	N	34736	6.09
	O	32177	5.64
	R	**5706**	-
1152	N	40088	7.03
	O	37203	5.80
	R	**6415**	-
1280	N	43392	6.10
	O	40488	5.69
	R	**7118**	-

Keyed permutation의 비교 결과, 최소 5.60배에서 최대 7.03배 더 좋은 성능을 지니는 것을 확인할 수 있다. 이는 제안하는 기법이 더 적은 시프트를 통해 기존보다 빠른 속도로 keyed permutation을 종료하는 것을 알 수 있다.

다음은 TinyJAMBU 알고리즘에 적용한 결과를 확인한다. 표 7은 기존 TinyJAMBU와 본 논문에서 제안한 최적화 기법을 적용한 TinyJAMBU의 비교 결과를 보여준다. 비교 결과 리버스 시프트를 적용하였을 때는 기존 대비 최소 5.60배에서 최대 5.87배 더 좋은 성능을 가졌다. 그리고 초기화 생략 기법을 적용한다면 그 차이는 최대 9.19배까지 성능 차이가 발생한다. 초기화 생략 기법은 기존 알고리즘에도 손쉽게 적용할 수 있으며, 기존 알고리즘에만 적용할 경우 약 80% 정도의 성능 개선을 확인할 수 있다. 이는 어셈블리 명령어를 사용한 방법이 아니므로 AVR이 아닌 다른 환경에서 TinyJAMBU를 사용할 때도 적용할 수 있다. 즉, 프로세서 의존이 없는 좀 더 유연한 기법이라 할 수 있다.

Table 7. Results of TinyJAMBU evaluation. (N: Normal, O: Optimized, R: Reverse shift, I: Initialization skip, Unit: clock cycles)

Alg.	Type	Clock cycles
128-Enc	N	217043
	N+I	118679
	O	208397
	O+I	115191
	R	**37970**
	R+I	**21378**
128=Dec	N	217043
	N+I	118713
	O	209013
	O+I	115270
	R	**38009**
	R+I	**21422**
192-Enc	N	239695
	N+I	134119
	O	233214
	O+I	132882
	R	**40818**
	R+I	**23508**
192-Dec	N	239765
	N+I	134200
	O	233253
	O+I	132929
	R	**40857**
	R+I	**23555**
256-Enc	N	249906
	N+I	143017
	O	244698
	O+I	141518
	R	**43626**
	R+I	**25618**
256-Dec	N	249900
	N+I	143020
	O	244737
	O+I	141563
	R	**43665**
	R+I	**25663**

V. 결 론

본 논문에서는 리버스 시프트와 초기화 생략 기법을 적용한 8-bit AVR 프로세서상에서의 경량암호 TinyJAMBU의 최적 구현에 대해서 제안하였다. 제안하는 리버스 시프트 기법은 연산 결과 값을 생성할 때 시프트 방향을 반대로 하여 시프트 횟수를 줄이는 기법이다. 초기화 생략은 키와 논스가 같다고 가정할 시, 초기화 단계의 결과물은 항상 동일하기 때문에 연산 결과를 저장해두고 다음 연산의 초기화 과정을 생략하는 기법이다. 성능 평가 결과 keyed permutation의 성능은 최대 5.24배 성능 개선이 있었으며 제안하는 기법을 TinyJAMBU에 적용한다면 리버스 시프트를 적용할 때 최대 4.71배, 초기화 생략을 추가로 적용한다면 최대 9.19배의 성능 개선이 있었다. 제안하는 기법은 8-bit로 레지스터를 취급하는 환경이면 손쉽게 적용할 수 있다. 가령 vector register를 통해서 내부 데이터를 8-bit로 취급하는 ARM 프로세서의 경우[13], 제안하는 기법의 적용이 가능하다. 따라서 제안하는 기법은 프로세서 의존도도 다소 낮다고 평가할 수 있다.

References

[1] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence, 1st Ed., Academic Press, April 2014.

[2] M.S. Turan, K.A. McKay, C. Çalık, D. Chang, and L. Bassham, "Status report on the first round of the NIST lightweight cryptography standardization process," NISTIR 8268, National Institute of Standards and Technology, Gaithersburg, 2019.

[3] H. Madushan, I. Salam, and J. Alawatugoda, "A Review of the NIST Lightweight Cryptography Finalists and Their Fault Analyses," Electronics, vol. 11, no. 24, pp. 4199-4220, Dec. 2022.

[4] Submission requirements and evalu-

ation criteria for the lightweight cryptography standardization process, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf, 2018.

〔5〕 H. Wu, and T. Huang. "JAMBU lightweight authenticated encryption mode and AES-JAMBU," CAESAR competition proposal, 2014.

〔6〕 H. Wu, and T. Huang. "TinyJAMBU: A family of lightweight authenticated encryption algorithms (version 2)," Submission to the NIST Lightweight Cryptography Standardization Process, May 2021.

〔7〕 D.Y. Roh, B.W. Koo, Y.H. Jung, I.W. Jeong, D.G. Lee, D.S. Kwon, and W.H. Kim, "Revised version of block cipher CHAM," International Conference on Information Security and Cryptology, pp. 1-19, Dec. 2020.

〔8〕 H.D. Kwon, S.W. An, Y.B. Kim, H.J. Kim, S.J. Choi, K.B. Jang, J.H. Park, H.J. Kim, S.C. Seo, and H.J. Seo, "Designing a CHAM block cipher on low-end microcontrollers for internet of things," Electronic, vol. 9, no. 9, pp. 1548-1564, Sep. 2020.

〔9〕 H.D. Kwon, K.B. Jang, J.H. Park, and H.J. Seo, "High-Speed Implementation to CHAM-64/128 Counter Mode with Round Key Pre-Load Technique," Journal of The Korea Institute of Information Security & Cryptology, 30(6), pp. 1217-1223, Dec. 2020.

〔10〕 R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," IACR ePrint 2023-404, June 2013.

〔11〕 H.D. Kwon, K.B. Jang, H.J. Kim, and H.J. Seo, "The fast implementation of block cipher SIMON using pre-computation with counter mode of operation," Journal of the Korea Institute of Information and Communication Engineering, 25(4), pp. 588-594, April 2021.

〔12〕 AVR Instruction Set Manual, https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf, 2012.

〔13〕 H.J. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange," IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 1-20, Aug. 2018.

〈저자소개〉

권 혁 동 (Hyeok-Dong Kwon) 학생회원
2018년 2월: 한성대학교 정보시스템공학과 학사 졸업
2020년 2월: 한성대학교 IT융합공학부 석사 졸업
2020년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보안, 암호구현

엄 시 우 (Si-Woo Eum) 학생회원
2021년 2월: 한성대학교 IT융합공학부 학사 졸업
2023년 2월: 한성대학교 IT융합공학부 석사 졸업
2023년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 정보보안, 암호구현

심 민 주 (Min-Joo Sim) 학생회원
2021년 2월: 한성대학교 IT융합공학부 학사 졸업
2023년 2월: 한성대학교 IT융합공학부 석사 졸업
2023년 3월~현재: 한성대학교 정보컴퓨터공학과 박사과정
〈관심분야〉 암호구현, 정보보안

양 유 진 (Yu-Jin Yang) 학생회원
2022년 2월: 한성대학교 IT융합공학부 학사 졸업
2022년 3월~현재: 한성대학교 IT융합공학부 석사과정
〈관심분야〉 정보보안, 양자컴퓨터

서 화 정 (Hwa-Jeong Seo) 종신회원
2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2016년 2월: 부산대학교 컴퓨터공학과 박사 졸업
2016년 1월~2017년 3월: 싱가포르 과학기술청 연구원
2017년 4월~2023년 2월: 한성대학교 조교수
2023년 3월~현재: 한성대학교 부교수
〈관심분야〉 암호구현