8-bit AVR 상에서의 Reverse shift를 사용한 경량암호 TinyJAMBU 최적 구현

권혁동* 엄시우* 심민주* 양유진* 서화정**

*한성대학교 (대학원생)
**한성대학교 (조교수)

Optimized implementation of Lightweight cryptography TinyJAMBU with Reverse shift technique on 8-bit AVR processors

Hyeok-Dong Kwon* Si-Woo Eum* Min-Joo Sim*,
Yu-Jin Yang* Hwa-Jeong Seo**

*Hansung University (Graduate student)
**Hansung University (Assistant professor)

요 약

IoT가 발전함에 따라 경량암호의 중요성이 크게 떠올랐다. NIST는 경량암호의 신규 표준 설립을 위해 표준 공모전을 개최하였다. 그 중에서 TinyJAMBU는 2022년 최종 라운드에 진출한 10종의 알고리즘 중 하나로 코드 크기가 굉장히 작으며 key schedule이 없다는 특징을 지닌다. TinyJAMBU는 keyed permutation 연산을 반복하여 암호화를 진행한다. 이때 내부적으로 shift 연산을 중점적으로 사용한다. 본 논문에서는 8-bit AVR 프로세서 상에서 경량암호 TinyJAMBU를 최적 구현한다. 제안하는 구현물은 AVR 어셈블리를 사용할 뿐만 아니라, keyed permutation의 shift를 반대 방향으로 하여 shift 횟수를 최소화한 Reverse shift 기법을 제안한다. 제안 기법은 keyed permutation에서 최대 624%, TinyJAMBU 알고리즘에 적용 시 최대 587% 더 좋은 성능을 지닌다.

I. 서론

IoT(Internet of Things) 기술의 발전에 따라 초소형 기기들이 서로 유기적인 연결이 된 초연결 사회에 진입하게 되었다. 이러한 초소형 기기들은 대체로 연산 능력과 가용 자원이 제한적이다. 때문에 연산량이 큰 암호 알고리즘을 가동하기가 어렵다. 이를 대체하기 위해 경량암호가 제안되었고, NIST(National Institute of Standards and Technology)에서는 경량암호 표준화를 위한 공모전을 개최하였다.

해당 공모전의 최종 라운드 진출 작품 중, TinyJAMBU는 굉장히 작은 코드 사이즈를 지니며 단순한 구조를 가지고 안정적인 평가를 받고 있다. 본 논문에서는 경량암호 TinyJAMBU의 AVR 프로세서 상에서의 최적 구현을 제시한다. 제시하는 기법은 TinyJAMBU에서 중심적인 연산을 담당하는 keyed permutation을 최적화하는데 집중한다.

본 논문의 구성은 다음과 같다. 2장에서 대상 알고리즘인 TinyJAMBU에 대해서 확인한다. 3장에서 제안하는 기법인 Reverse shift에 대해서 소개한다. 4장에서 성능 평가를 진행하고 5장에서 결론을 맺는다.

II. 배경

경량암호 TinyJAMBU는 CAESAR 경진대회에 출품되었던 JAMBU[1]의 변형 알고리즘이다. CAESAR에서 JAMBU는 가장 작은 블록 크기를 지닌 알고리즘으로, TinyJAMBU는 JAMBU를 기반으로 동작한다.

TinyJAMBU는 내부 상태와 키를 사용하여 난수를 생성한 다음, 해당 값을 평문과 XOR 연산하여 암호문을 생성한다. 따라서 암호 생성은 스트림암호로 볼 수 있다. 하지만 내부 동작은 블록암호 방식이므로 NIST 공모전의 분류상으로 블록암호 기반의 알고리즘으로 분류되었다. 이와 같은 방식의 최종 후보군 알고리즘으로

GIFT-COFB[2]가 존재한다. 다만 최종 라운드에서는 순열 기반 알고리즘으로 재분류 되었다.

TinyJAMBU의 암·복호화 과정은 Initialization, Processing AD(Associated Data), Encryption/Decryption 과정으로 구성된다. 이때 모든 과정에서 keyed permutation이 중점적인 연산으로 사용된다.

Keyed permutation은 키 스케줄 없이 입력받은 키 값을 그대로 상태 변수에 반영한다. 내부적인 구조는 그림 1과 같은 Nonlinear Feedback Shift Register를 사용한다[3].

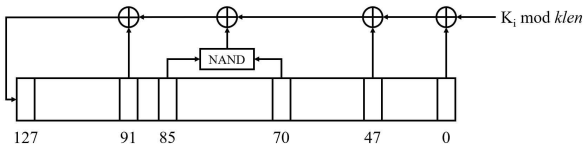

Figure. 1. Structure of Nonlinear Feedback Shift Register.

Keyed permutation은 P_n으로 표현하며, 이때 n은 반복 횟수로 규격과 단계별로 다르다. 이는 표 1에서 정리한다.

Table. 1. Number of 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

Keyed permutation은 상태 변수의 순서를 뒤섞고 중간에 키 값을 삽입하여 상태 변수를 갱신시키는 역할을 한다. 이는 표 2와 같은 코드로 구현된다.

Table. 2. Pseudo code for The keyed permutation P_n.

```
StateUpdate(State, Key, Round):
    for i = 0 to Round
        t1 = (State1>>15)|(State2<<17)
        t2 = (State2>>6)|(State3<<26)
        t3 = (State2>>21)|(State3<<11)
        t4 = (State2>>27)|(State3<<5)
        fb = State0^t1^(~(t2&t3))^t4^key
        State0 = State1
        State1 = State2
        State2 = State3
        State3 = fb
    end
```

제안하는 기법은 TinyJAMBU의 연산중에서 가장 많은 부하를 일으키는 keyed permutation을 최적 구현하는데 집중한다.

III. 제안기법

제안하는 기법은 keyed permutation을 AVR 어셈블리를 사용하여 구현한다. 이때 keyed permutation에 Reverse shift 기법을 사용하여 내부 연산 구조를 바꾸는 것으로 더 좋은 성능을 가지도록 한다.

AVR 프로세서에는 32개의 범용 레지스터가 존재한다. 각 레지스터별로 사용 용도를 지정하여 연산 값의 손실을 방지하고 효과적인 구현이 되도록 한다. 그림 2는 레지스터 할당 계획을 나타낸 것이다.

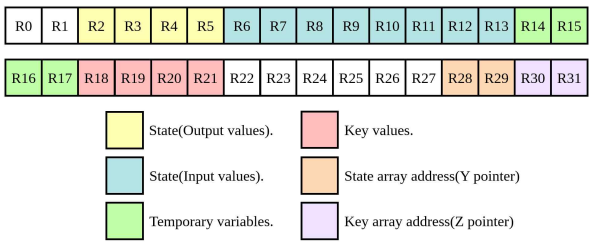

Figure. 2. Registers allocation plan.

keyed permutation 연산은 상태 변수 두 개가 shift된 다음 OR 연산을 통해 하나의 임시 값을 생성하는 구조이다. 이때 t1은 State1, State2를 사용하며 나머지는 State2, State3을 사용한다. 우선 State2, State3의 shift에 주목에 주목한다. 이 값들은 t2, t3, t4 생성에 관여한

다. State2의 값은 t2에서는 6회, t3에서는 21회, t4에서는 27회 right shift된다. 이때 State2의 내부 값을 표현해보면 그림 3과 같이 표현할 수 있다. 하나의 State는 32-bit이므로 4개의 레지스터를 사용한다. 그림에서 색이 없는 부분은 0값으로 비어있음을 뜻한다.

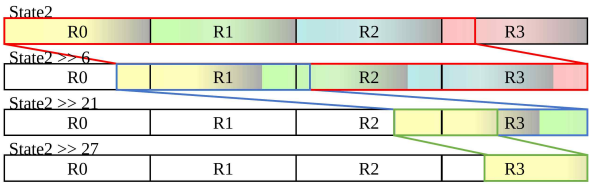

Figure. 3. Computation step of original State2.

값의 이동을 확인해보면 shift 방향은 유지한 채로 최하위 레지스터부터 차례로 값이 소멸해 가는 것을 알 수 있다. 그림 3에서 각각의 단계를 1, 2, 3, 4단계라 표현한다. 2단계에서는 최하위의 6bit가 소멸된다. 3단계는 최상위부터 11bit만 존재하며 4단계는 최상위 레지스터의 5bit만 존재한다.

그림 3의 각 단계별로 색상으로 연결된 부분은 shift 이후 잔존하게 되는 값들을 표현한 것이다. 이 값들은 최하위 레지스터 쪽으로 일괄적인 이동을 보인다. 하지만 연산 결과 값을 반드시 최하위 레지스터에 위치시킬 필요는 없다. 3단계의 값은 1단계의 값 중 R0의 8bit와 R2의 3bit로 구성된다. 이 값을 완성시키기 위해서는 2단계에서 15회 shift를 진행하였다. 하지만 반대 방향으로 shift하게 되면 단 1번의 shift로 원하는 값을 획득할 수 있다. 마찬가지로 4단계의 값은 1단계 기준으로 R0의 5bit만 있다. 이 값은 3단계에서 반대 방향으로 2번만 shift하면 획득할 수 있다.

State2에 대한 reverse shift 구현은 그림 4와 같다. 여기서 2단계 값은 기존과 동일하게 생성한다. 3단계는 2단계의 값을 15회 shift하는 대신, 반대 방향으로 1회 shift하는 것으로 완성할 수 있다. 이때 필요한 값은 R0, R1, R2에 있기 때문에 R3는 shift 시킬 필요가 없다.

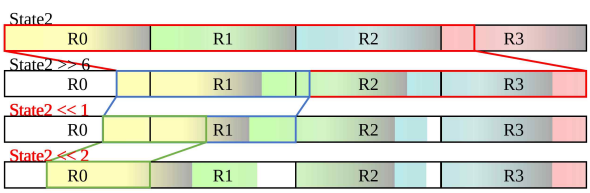

Figure. 4. Reverse shift technique for State2.

마찬가지로 4단계 값의 완성은 3단계 값을 shift를 하는 대신, 반대 방향으로 2회 shift를 진행한다. 이때는 R0, R1의 값만 필요하고 R2는 더 이상 shift하지 않는다.

t2, t3, t4의 나머지 부분은 State3에서 유도된다. State2와 동일하게 중간부터 반대 방향으로 shift하는 것으로 연산 횟수를 줄일 수 있다. 2단계에서는 기존과 동일하게 5회 좌측 shift를 진행하고, 3단계는 좌측 6회 추가 shift 대신 우측 2회 shift로, 4단계는 좌측 15회 추가 shift 대신 우측 1회 shift로 대체할 수 있다. State3의 reverse shift 연산 흐름은 그림 5에서 확인할 수 있다.

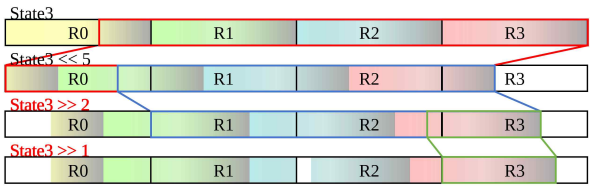

Figure. 5. Reverse shift technique for State3.

주의할 점으로 t2에는 State3<<26이, t3에는 State3<<11이, 그리고 t4에는 State3<<5가 필요하다. 하지만 Reverse shift 연산 특성상 State3<<5가 가장 먼저 연산되며 이는 기존 연산 순서의 역순이다. 때문에 연산 순서를 반대로 진행하되, 연산 결과물을 스택에 저장하고 필요할 때 가져오는 형식으로 구현한다.

t1에서는 State1>>15와 State2<<17이 필요하다. 이때 State1에는 reverse shift를 적용하여 State1<<1로 변경한다. State2는 17회 좌측 shift 대신, 좌측 1회 shift로 변경한다. 이는 17회 shift 결과물이 하위 15bit인 점을 감안하면, 하위 16bit만 레지스터에 로드한 다음 이를 1회 shift하여 동일한 값을 얻을 수 있기 때문이다.

IV. 성능평가

제안하는 기법은 TinyJAMBU 레퍼런스 코드를 기반으로 작성한다. 레퍼런스 코드는 레퍼런스 버전과 최적화 버전 두 가지가 존재한다. 최적화 버전은 레퍼런스 버전과는 다르게 State 이동 단계를 제거하여 조금 더 빠른 keyed permutation을 진행한다. 본 논문에서 제안하는 reverse shift 구현물은 최적화 버전을 기반으로 구현하였다.

구현은 Microchip Studio 프레임워크를 사용하여 진행하며, ATmega128 프로세서를 대상으로 한다. 컴파일 옵션으로는 -O3(fastest)를 사용하고 평문 길이는 8byte를 사용한다. 성능 평가 단위는 clock cycles이며 keyed permutation 속도와 각 TinyJAMBU별 암호화, 복호화 속도를 비교한다. keyed permutation의 성능평가는 그림 6과 같다.

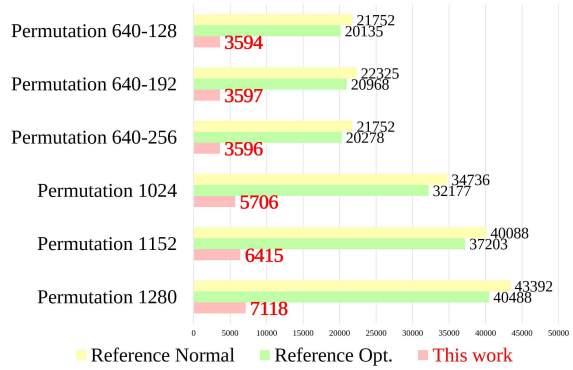

Figure. 6. Performance evaluation of keyed permutation.

Keyed permutation 비교 결과 제안하는 기법이 최소 560%에서 최대 624%더 좋은 성능을 지님을 알 수 있다. 제안하는 기법이 더 적은 shift 횟수를 사용하여 빠른 속도로 permutation을 종료하는 것을 알 수 있다.

이어서 reverse shift 기법을 적용한 TinyJAMBU와 기존 구현물의 성능 비교를 진행한다. 결과는 그림 7과 같다.

제안하는 기법이 최소 550%에서 최대 587% 더 좋은 성능을 지니며, 블록암호 구조 특성상 암·복호화의 성능이 비슷한 것을 알 수 있다.

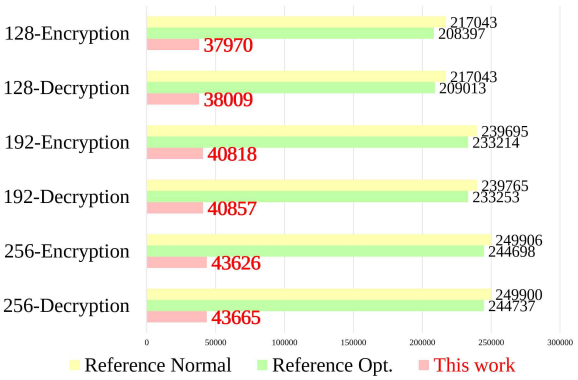

Figure. 7. Evaluation result of TinyJAMBU algorithms.

V. 결론

본 논문에서는 Reverse shift 기법을 사용한 TinyJAMBU에 대해 제안하였다. 제안하는 기법은 결과 값을 생성하기 위해 shift를 반대 방향으로 진행하는 것으로 shift 횟수를 줄이는 기법이다. 성능평가 결과 TinyJAMBU의 성능을 최대 587% 향상시킬 수 있었다.

VI. Acknowledgement

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (<Q|Crypton>, No.2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, 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%).

[참고문헌]

[1] H.Wu, and T.Huang. “JAMBU lightweight authenticated encryption mode and AES-JAMBU,” *CAESAR competition proposal*, 2014.

[2] S.Banik, A.Chakraborti, A.Inoue, T.Iwata, K.Minematsu, M.Nandi, T.Peyrin, Y.Sasaki, S.M.Sim, and Y.Todo, “GIFT-COFB”, *Cryptology ePrint Archive.*, 2020.

[3] H.Wu, and T.Huang. “TinyJAMBU: A family of lightweight authenticated encryption algorithms (version 2),” *Submission to the NIST Lightweight Cryptography Standardization Process*, 2021.