ARMv8상에서의 SKINNY 블록 암호 병렬 구현

엄시우*, 권혁동*, 김현준*, 장경배*, 서화정**

*한성대학교 (대학원생), **한성대학교 (교수)

Parallel Implementation of SKINNY block cipher on ARMv8

Si-Woo Eum*, Hyeok-Dong Kwon*, Hyun-Jun Kim*,
Kyung-Bae Jang*, Hwa-Jeong Seo**

*Hansung University(Graduate student)
**Hansung University(Professor)

요 약

ARMv8 아키텍처는 SIMD 명령어를 지원하며, 하나의 명령어로 여러 데이터의 병렬 연산이 가능하다. SIMD 명령어를 활용하여 한 번의 암호화로 여러 평문 블록의 효율적인 병렬 암호화가 가능하다. 본 논문에서는 Skinny 블록 암호의 병렬 구현을 진행한다. LD4 명령어를 활용하여 레지스터 정렬 과정을 생략하고, TBL 명령어를 활용한 효율적인 치환 연산과 REV32, TRN 명령어를 활용한 순열 연산을 구현하였다. 결론적으로 ARMv8 아키텍처로 설계된 Apple M1상에서 4개의 평문 블록을 병렬 구현하였으며, 단일 구현 대비 3.6배 높은 성능 향상을 확인할 수 있다.

I. 서 론

ARMv8은 ARM 아키텍처에서 가장 선호하는 아키텍처이다. ARMv8은 하나의 명령어로 여러 데이터의 병렬 산술 연산이 가능한 SIMD(Single Instruction Multiple Data)를 지원한다. 이러한 SIMD를 활용하여 병렬 암호화에 대한 연구가 진행되고 있으며, 최근 연구중 AES의 병렬 암호화의 연구 결과 SIMD 기반 Linux 커널 구현보다 5% 향상된 성능을 보여줍니다[1].

Romulus[2]는 National Institute of Standards and Technology(NIST)에서 개최된 경량 암호 공모전에 발표된 암호로 현재 최종라운드까지 진출한 암호중 하나이며 AES와 비슷한 알고리즘으로 설계된 SKinny 블록 암호를 활용한 암호 알고리즘이다. 본 논문에서는 ARMv8상에서 SIMD 명령어를 활용한 Skinny 블록 암호의 병렬 구현을 진행한다.

본 논문의 구성은 다음과 같다. 2장에서는 Tweakey Framework, Skinny 블록 암호, ARMv8 아키텍처에 관하여 설명한다. 3장에서는 병렬 구현을 위한 제안 기법을 설명한다. 4장에서는 제안 기법을 적용한 구현의 성능 평가를 진행하고, 마지막으로 5장에서 결론을 내린다.

II. 관련 연구

2.1 Tweakey Framework

Tweakey Framework[4]는 Tweak과 Key가 합쳐진 Tweakey값을 사용하여 암호화를 진행한다. Tweakey의 구성은 T-bit의 Tweak, K-bit의 Key로 구성된다. 이러한 n-bit(n:T+K) Tweakey를 사용한 암호화 과정은 다음 수식과 같다.

- $TweaKey_{i+1} = H(TweaKey_i)$ (1)
- $State_{i+1} = Func(State_i \oplus G(TweakKey_i)$ (2)

수식-(1)은 다음 라운드에서 사용할 Tweakey

가 함수 H에 의해 연산되고, 수식-(2)는 State와 Tweakey의 XOR된 값이 라운드 함수 Func에 의해 다음 라운드 State가 연산된다.

Tweakey Framework는 Tweakable 블록 암호와 Related-key 공격에 저항하는 암호를 설계를 목적을 가지고 제안된 Framework이다.

2.2 SKINNY 블록 암호

AES와 유사하게 설계된 Skinny 블록 암호는 Tweakey Framework를 사용하는 Tweakable 블록 암호이다[3]. 블록 길이는 64-bit, 128-bit를 지원하며, Tweakey 길이는 블록 길이의 3배까지 지원한다. 라운드 함수중 AddTweakey 함수에서는 2.1장에서의 수식-(1)에 H함수가 포함되어 있다. 전체적인 암호 알고리즘은 (그림 1)과 같다.

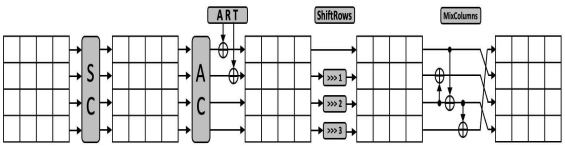

Fig 1. Algorithm of The Skinny Block Cipher

라운드 함수는 사전 연산된 Sbox 테이블의 값으로 치환 연산을 하는 Subcell(SC), 사전 정의된 상수값이 State값에 혼합되는 AddConstant(AC), Tweakey가 State에 혼합되는 AddTweakey(ART), Permutation을 진행하는 Shiftrows, 행렬 곱을 통해 확산 연산을 하는 Mixcolumns로 이루어져 있다.

2.3 ARMv8 아키텍처

ARMv8-A는 고성능 임베디드 64-bit 아키텍처로, 32-bit(AArch32), 64-bit(AArch64)아키텍처 모두를 지원한다. 64-bit로 사용 가능한 x0-x30의 범용 레지스터 31개를 제공하고, 이 범용 레지스터는 w0-w30으로 32-bit로도 사용 가능하다. 벡터 레지스터는 128-bit 크기를 가지며, v0-v31로 32개를 제공하고 있다. 본 논문에서 사용되는 주요 명령어는 <표 1>와 같다[5].

Instruction	Description
LD4	Load single 4-element structures to one lane of four registers
SUB	Subtract
EOR	Bitwise Exclusive OR
TBL	Table vector Lookup
REV32	Reverse elements in 32-bit words
TRN2	Transpose vectors(secondary)

Table 1. Instruction of ARMv8

III. 제안 기법

본 논문에서는 ARMv8상에서 Skinny 블록 암호의 병렬 구현을 진행한다. 병렬 구현을 통해 4개의 평문 블록(블록 길이 128-bit(16byte)기준 64byte)이 병렬로 연산되어 한 번의 암호화 과정을 통하여 4개의 평문 블록이 암호화가 가능하다. 최적화 병렬 구현을 위하여 LD4 명령어를 활용한 레지스터 정렬 생략과 TBL 명령어를 활용한 Subcell 함수의 최적화, REV32, TRN 명령어를 활용한 Shiftrows 함수의 최적화 구현을 진행하였다. 본 장에서는 위에서 진행한 기법에 관하여 나누어 설명한다.

3.1 레지스터 정렬 생략

병렬 구현을 위해서는 병렬 연산이 가능하도록 레지스터로 평문 값을 올린 후에 레지스터 정렬을 진행하여야 한다. 즉, 암호화 과정을 진행하기 전에 레지스터 정렬 과정이 추가가 되어야 한다.

하지만 ARMv8의 SIMD 명령어에서는 다양한 Load 명령어를 지원하고 있다. 다양한 Load 명령어중 LD4 명령어를 활용하여 레지스터 정렬 과정 추가 없이 평문 블록을 Load와 동시에 정렬을 진행할 수 있다. 또한 암호화 후의 병렬 구현을 위해 정렬되어 있는 결과 값을 ST4 명령어를 통하여 Load와 마찬가지로 정렬을 생략한 메모리 저장이 가능하다. LD4 명령어로 평문 블록을 불러왔을 때, (그림 2)와 같다.

	Index : 3				Index : 2				Index : 1				Index : 0			
V0	D3	D2	D1	D0	C3	C2	C1	C0	B3	B2	B1	B0	A3	A2	A1	A0
V1	D7	D6	D5	D4	C7	C6	C5	C4	B7	B6	B5	B4	A7	A6	A5	A4
V2	D11	D10	D9	D8	C11	C10	C9	C8	B11	B10	B8	B09	A11	A10	A9	A8
V3	D15	D14	D13	D12	C15	C14	C13	C12	B15	B14	B13	B12	A15	A14	A13	A12

Fig. 2. Result of LD4.S {v0-v3}[\index], [x0], #16 (x0: plaintext address, \index: 0-3)

\index를 0~3까지 4번 수행하게 되면 (그림 2)와 같이 Load와 동시에 레지스터 정렬이 가능하다. #16은 LD4 명령어 동작 후 x0의 값을 16 증가시켜 평문 블록의 주소값을 저장하고 있는 x0의 값을 변경한다.

3.2 Subcell 함수 최적화 기법

TBL 명령어는 벡터 레지스터에 저장된 Table 값을 활용하여 치환된다. 기존의 치환 연산은 테이블이 저장된 주소 값을 변경하여 메모리에 저장된 테이블 값을 가져오는 식으로 동작한다. 하지만 TBL 명령어는 메모리에 저장된 테이블 값을 벡터 레지스터에 올리고 치환 시 벡터 레지스터에 저장된 테이블 값을 활용하여 치환 연산된다. 레지스터간의 연산으로 동작하기 때문에 메모리 접근을 통한 치환 연산보다 효율적인 치환이 가능하다. TBL 명령어의 동작 과정은 (그림 3)과 같다[1].

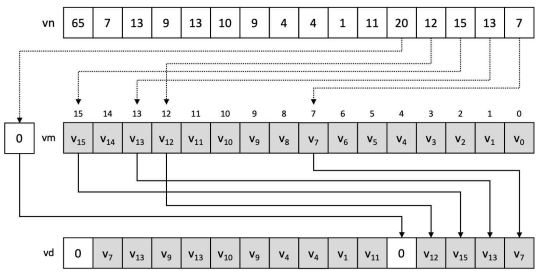

Fig. 3. Process of TBL16b vd, {vm}, vn

vn레지스터에 저장되어 있는 값은 vm에서 인덱스로 사용되며, 해당 인덱스에 저장되어 있는 값이 vd에 저장되게 된다. 이때 vn에서 인덱스를 초과하는 값은 vd에 0으로 저장된다. 이후에 치환되지 못한 값은 TBL 명령어가 아닌 TBX 명령어를 사용하여 치환하지 못한 인덱스만 치환이 가능하다.

3.3 Shiftrows 함수 최적화 기법

Shiftrows 함수에서는 State의 2번째 행은 오른쪽 로테이션 1번(RotRight^1(S)), 3번째 행은 오른쪽 로테이션 2번(RotRight^2(S)). 4번째 행은 오른쪽 로테이션 3번(RotRight^3(S))으로 동작한다. RotRight^2(S)는 RotLeft^2(S)와 같고, RotRight^3(S)은 RotLeft^1(S)과 같다. 로테이션 구현은 REV32, TRN2 명령어를 활용하면 서로 다른 평문 블록에 영향을 주지 않는 간단한 구현이 가능하다. REV32와 TRN2 명령어의 동작 과정은 (그림 4)와 같다.

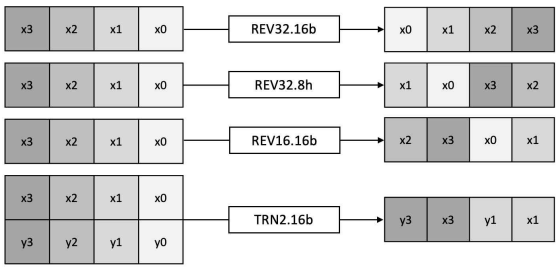

Fig. 4. Result of REV32 and TRN2 instruction

해당 명령어를 활용하여 각 로테이션은 다음 수식과 같이 구현될 수 있다[1].

- RotRight^1(S) = REV32.8H(RotLeft^1(S))
- RotLeft^2(S) = REV32.8H(S)
- RotLeft^1(S) = S` = TRN2.16B(S, T)
  T = REV32.16B(S)

16B와 8H는 벡터 레지스터의 배열 지정자로 벡터 레지스터의 인덱스의 단위를 의미한다. 즉 B는 8-bit 단위로 128-bit가 나뉘어 인덱스를 구분하고 H는 16-bit 단위로 나누어 인덱스를 구분하여 연산을 진행한다. 다음 수식을 구현된 Assembly 코드로 보면 <표 2>와 같다.

IV. 성능 평가

성능 평가는 ARMv8 아키텍처로 설계된 Apple M1을 사용하는 2020년형 Apple Macbook Pro13상에서 진행한다. 기존 연구 결과가 없어 단일 평문 구현과 제안 기법을 적용한 병렬 구현의 Cycle per Byte(cpb)를 측정해

```
.macro Shiftrows
//RotRight[1](S)
REV32.16B       v8, v1
TRN2.16B        v1, v1, v8
REV32.8H        v1, v1

//RotLeft[2](S)
REV32 8H        v2, v2

//RotLeft[1](S)
REV32.16B       v8, v3
TRN2.16B        v3, v3, v8
.endm
```

Table 2. Assambly code of Shiftrows layer(v1-v3 : state vector register, v8 : temp vector register)

비교를 진행한다. 블록 길이 128-bit, Tweakey 길이 128-bit만을 구현 후 비교하며, 측정 결과는 <표 3>과 같다.

Type	Performance
Single Implementation	9.29
Parallel Implementation*	2.57

Table 3. Performance Result(unit: cpb)

단일 구현과 병렬 구현의 성능 측정 결과 단일 구현 대비 병렬 구현에서 3.6배 성능 향상을 확인할 수 있다.

V. 결 론

본 논문에서는 Skinny 블록 암호의 병렬 최적화 구현을 진행하였다. 병렬 구현을 통해 4개의 평문 블록이 병렬 암호화가 가능하다. 본 논문에서는 ARMv8에서 지원하는 SIMD 명령어를 적절히 사용하여 최적화 구현을 진행하였다. 해당 명령어를 활용하면 비슷한 알고리즘으로 설계된 다른 블록 암호에서도 효율적인 병렬 최적화 구현이 가능할 것으로 보인다. 결과적으로 병렬 구현을 통해 단일 구현 대비 3.6배 높은 성능 향상을 확인하였으며. 추후 연구로 병렬 구현을 활용한 CTR 운용모드 구현을 제안한다.

VI.Acknowledgement

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.2018-0-00264, IoT 융합형 블록체인 플랫폼 보안 원천 기술 연구, 50%) 그리고 이 성과는 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 50%)

[참고문헌]

[1] H. Fujii, F. C. Rodrigues, and J. L´opez, “Fast AES implementation using ARMv8 ASIMD without cryptography extension,” in International Conference on Information Security and Cryptology, pp. 84 - 101, Springer, 2019.

[2] Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: New Results on Romulus. In: NIST LWC Workshop (2020)

[3] Beierle C. et al. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Robshaw M., Katz J. Advances in Cryptology – CRYPTO 2016. Lecture Notes in Computer Science, vol 9815.

[4] J Jean, I Nikolić, T Peyrin. “Tweaks and Keys for Block Cipher: The TWEAKEY Framework“. Advances in Cryptology – ASIACRYPT 2014. Lecture Notes in omputer Science, vol 8874.

[5] Armv8-A Instruction Set Architecture. [online] available: https://documentation-service.arm.com/static/613a2c38674a052ae36ca307?token=.