ARMv8상에서의 SKINNY Tweakable 블록암호 최적화 구현

엄시우[1], 송경주[1], 강예준[1], 김원웅[1], 서화정[1]
[1]한성대학교 IT융합공학과
shuraatum@gmail.com, thdrudwn98@gmail.com, etus1211@gmail.com,
dnjsdndeee@gmail.com, hwajeong84@gmail.com

Optimization of SKINNY Tweakable Block Cipher on ARMv8

Si-Woo Eum[1], Gyeong-Ju Song[1], Yea-Jun Kang[1], Won-Woong Kim[1],
Hwa-Jeong Seo[1]
[1]Dept. of IT Convergence Engineering, Han-Sung University

요 약

2015년부터 NIST에서는 경량 암호 공모전을 개최하여 저사양 기기에서 활용할 경량 암호 알고리즘을 개발해오고 있다. 본 논문에서는 경량 암호 공모전에서 발표된 Romulus 암호에 활용되는 Tweakey 프레임워크로 설계된 Tweakable 블록암호 Skinny의 최적화 구현을 최신 프로세서 중 하나인 Apple M1 프로세서 상에서 진행하였다. M1 프로세서는 ARMv8 아키텍처로 설계되었으며, ARMv8 벡터 명령어 중 TBL 명령어를 활용한 라운드 함수의 효율적인 구현으로 최적화를 진행하였다. Skinny 블록암호의 블록 길이 128-bit 구현을 진행하였으며, 해당 프로세서에서 구현된 skinny 구현 연구가 없기 때문에 Referenc C코드와 비교를 진행하였다. 성능 측정 결과 128-bit 키 길이에서는 약 19배의 성능 향상을 확인하였으며, 키 길이 384-bit에서는 약 32배의 높은 성능 향상을 확인할 수 있다.

1. 서 론

NIST(National Institute of Standards and Technology)에서 2015년부터 저사양 컴퓨팅 환경에서 사용하기 위한 경량 암호 공모전을 개최하였다. 여러 경량 암호 알고리즘이 발표되었으며, 현재 10개의 암호 알고리즘이 최종라운드를 진행중이다.

최종라운드에 진출한 암호중 Romulus[1]는 Skinny 블록 암호[2]를 활용하는 암호 알고리즘이다. 기존의 블록암호와 다른 특징을 가지고 있는 Skinny 블록암호는 Tweakey 프레임워크를 사용하는 Tweakable 블록 암호이다. 본 논문에서는 ARMv8상에서 Skinny 블록암호 최적화 구현을 진행한다.

본 논문의 구성은 다음과 같다. 2장에서는 Tweakable 블록암호와 Tweakey 프레임워크, Skinny 블록암호, ARMv8에 관하여 설명한다. 3장에서는 구현 기법에 관하여 설명한다. 4장에서는 성능 평가를 진행하고, 마지막으로 5장에서 본 논문의 결론을 내린다.

2. 관련 연구

2.1 Tweakable 블록암호와 Tweakey 프레임워크

일반적인 암호문(C)은 입력되는 평문(P)과 비밀키(K)를 통해서 얻을 수 있다(즉 C=E(P,K)). 하지만 Tweakable 블록암호는 Tweak(T)값을 추가로 받아 암호문을 얻을 수 있다(즉 C=E(P,K,T))[3]. Tweak 값으로 인해 같은 키를 사용하더라도 Tweak 값이 다르면 다른 암호문을 얻을 수 있다. 키를 변경하는 것은 암호화 과정에서 많은 비용이 필요하다. 하지만 Tweak 값을 추가함으로써 키를 변경하지 않고 Tweak 값을 변경시켜 더 적은 비용으로 다른 암호문을 얻을 수 있는 이점이 있다.

Tweakable 블록암호는 디스크 암호화, DB 암호화를 예로 들 수 있다. Tweak 값은 공개된 값을 사용할 수 있기 때문에, 블록 번호, index를 Tweak 값으로 사용하여 암호화를 한다. 이는 같은 키를 사용하여 암호화를 진행할 때, 같은 데이터를 암호화 하더라도 블록 번호와 index가 다르기 때문에 서로 다른 암호문이 나오게 된다.

Tweakey 프레임워크는 Tweakable 블록암호와 Related-key 공격에 저항하는 블록암호의 설계를 통합하는 목적을 가지고 제안된 프레임워크이다[4].

Tweakey 프레임워크에서는 Key와 Tweak값이 합쳐진 값을 Tweakey로 사용하여 암호화를 진행한다. Tweakey의 구성은 t-bit의 Tweak 값과 k-bit의 Key값을 구성되어 있다. 이렇게 구성된 (t+k)는 n-bit의 내부 상태 S와 연산된다. 암호화 과정을 수식으로 보면 다음과 같다.

- $State_{i+1} = F(State_i \oplus G(TK_i)$ (1)
- $TK_{i+1} = H(TK_i)$ (2)

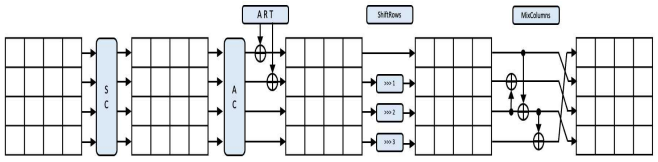
수식(1)은 Tweakey와 State의 XOR 연산된 값이 라운드함수(F)에 의해 다음 라운드 State를 연산한다.

수식(2)는 함수 H에 의해 다음 라운드에서 사용할 Tweakey가 연산된다.

2.2 Skinny 블록암호

Skinny 블록암호는 CRYPTO16에 소개된 Tweakey 프레임워크를 사용하는 Tweakable 블록암호이다[2]. 64-bit, 128-bit 블록 길이를 지원하며, Tweakey는 블록 길이의 3배까지 지원한다. 즉 64-bit의 경우 64-bit, 128-bit, 192-bit를 지원한다.

Skinny 블록암호의 라운드 함수는 AES 블록암호화 유사하게 동작한다. 라운드함수는 SubCell, AddConstant, AddTweakey, Shiftrows, Mixcolumns로 구성되어 있다. AddTweakey 함수에는 위의 수식(2)에서 다음 라운드에 사용할 Tweakey를 연산하는 H 함수가 포함되어 있다. 전체적인 암호화 과정은 (그림 1)과 같다.



(그림 1) Skinny 블록암호 알고리즘. (SC: SubCell, AC: AddConstant, ART: Add Round Tweakey)

SubCell(SC)은 사전 연산된 Sbox 테이블의 값으로 치환하는 연산을 진행한다. AddConstant(AC)는 사전에 정의된 상수값이 State값에 혼합되며, AddTweakey(ART)는 Tweakey가 State와 혼합된다. ShiftRows는 Permutation을 진행하며, MixColumns는 행렬 곱을 통해 State의 각 열간의 확산이 이루어진다.

2.3 ARMv8

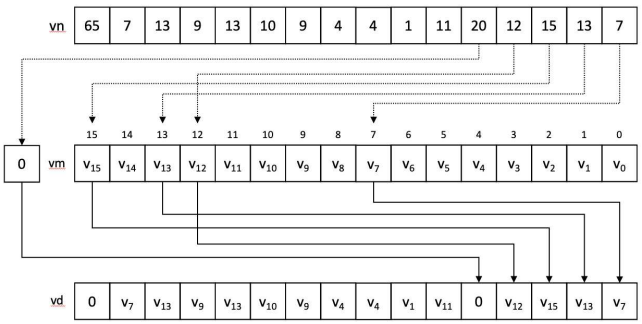
ARMv8-A는 고성능 임베디드 64-bit 아키텍처이며, 64-bit(AArch64) 아키텍처와 32-bit(AArch32) 아키텍처를 모두 지원한다. 31개의 범용 레지스터를 제공하며, x0-x30는 64-bit로 사용가능하며, w0-w30는 32-bit로 사용가능하다. 또한 32개의 벡터 레지스터 v0-v31을 제공하며, 128-bit 크기로 사용가능하다. ARMv8 프로세서는 2011년 출시 후 스마트폰 시장에서 큰 영향력을 보여주며, 현재 다양한 노트북과 스마트폰에서 많이 사용되고 있다[5].

3. 구현 기법

본 장에서는 최적화 구현에 관하여 설명한다. 먼저 최적화 구현을 위해 사용된 TBL 명령어에 관하여 설명하고, Skinny 블록암호의 라운드함수중 SubCell, MixColumns에 대하여 설명한다.

3.1 TBL 명령어

TBL 명령어는 테이블 벡터 조회 명령어이다. TBL 명령어는 소스 레지스터의 벡터 요소에서 각 값을 읽고, 읽은 결과를 인덱스로 사용하여 룩업 테이블 레지스터에서 인덱스에 해당 하는 값을 목적지 레지스터에 쓴다. 이 과정을 그림으로 도식화하면 (그림 2)와 같다.



(그림 2) TBL vd, {vm}, vn. (vm: Lookup 테이블 레지스터)

(그림 2)는 vn 레지스터에 벡터 요소들이 인덱스로 사용되어 vm 레지스터에서 해당 인덱스에 저장된 값이 vd로 저장되는 과정을 나타낸다. 이때 하나의 요소는 8-bit의 값을 저장할 수 있기 때문에 0~255까지의 값이 저장될 수 있다. 하지만 벡터레지스터의 인덱스는 0~15까지 있기 때문에 해당 인덱스에 해당되지 않는(즉, (그림 2)에서의 20, 65) 값은 vd 레지스터에 0으로 저장된다. 이렇게 치환되지 않은 값은

인덱스 범위의 값으로 조정한 후, TBX 명령어를 활용하여 (그림 2)와 같은 과정을 진행할 수 있게 된다.

TBL, TBX 명령어를 활용하면 SubCell 함수에서의 SBOX 테이블을 활용한 치환과 ShiftRows 함수에서의 Permutation 연산을 효율적으로 구현이 가능하다.

3.2 SubCell 함수

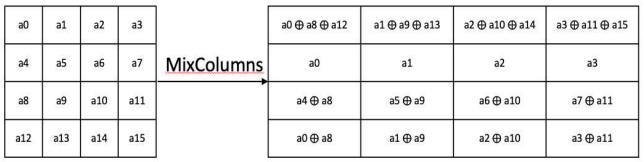
SubCell 함수에서는 사전 연산된 SBOX 테이블을 통해 값을 치환하는 연산을 진행한다. SubCell 함수를 구현하기 위해 TBL 명령어와 TBX 명령어를 활용한다. 일반적으로 SBOX 테이블의 주소를 변경해가며 값을 읽고 쓰는 구현 방법이 있다. 하지만 TBL, TBX 명령어를 활용한 구현에서는 SBOX 테이블의 주소값을 활용하는 것이 아니라, SBOX 테이블의 값을 벡터 레지스터에 저장하여 사용한다. SubCell 함수의 Assembly 코드는 <표 1>과 같다.

```
.macro Subcell
sub v7.16b, v1.16b, v15.16b
tbl v1.16b, { v16.16b - v19.16b }, v1.16b
sub v6.16b, v7.16b, v15.16b
tbx v1.16b, { v20.16b - v23.16b }, v7.16b
sub v5.16b, v6.16b, v15.16b
tbx v1.16b, { v24.16b - v27.16b }, v6.16b
tbx v1.16b, { v28.16b - v31.16b }, v5.16b
.endm
```

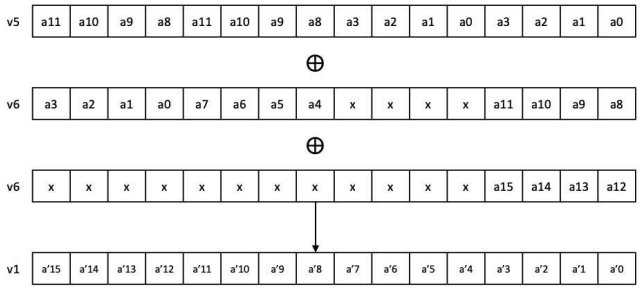
<표 1> SubCell 함수 Assembly 구현 코드.(v1: plain text, v16-v31: Sbox table(256-byte), v15 각 벡터 인덱스에 0x40.)

TBL 명령어는 한번에 4개의 레지스터에 저장된 테이블을 조회할 수 있다. 즉, SBOX 테이블의 크기는 256-byte이므로 이를 벡터레지스터에 나누어 저장하게 되면, 16개에 나누어 저장되게 되고 한번에 4개의 레지스터씩(64-byte) 조회가 가능하므로 4번의 조회가 필요하다. 따라서 1번의 tbl, 3번의 tbx 명령어가 사용되며, 한번의 조회가 끝나면 64-byte 인덱스 범위까지 조회를 진행하였기 때문에 치환되지 못한 값을 인덱스 범위로 조정하기 위해 0x40을 빼준 후 tbx 명령어로 치환된 부분을 제외한 나머지를 다시 조회하여 치환한다. tbx 명령어는 tbl 명령어를 만나기 전까지는 이전에 변환된 값을 제외한 값만을 조회하여 변환한다.

3.3 MixColumns 함수



(그림 3) MixColumns 함수



(그림 4) MixColumns 함수 연산 과정

(그림 3)은 MixColumns 함수 연산 후 각 인덱스에서 필요한 연산을 도식화한 그림이다. 해당 함수를 구현하기 위해 각각의 값을 벡터레지스터에 정렬한 후 값을 연산한다. (그림 4)는 해당 과정을 도식화하여 나타낸 그림이다. (그림 4)의 과정을 Assembly 코드는 <표 2>와 같다.

```
.macro Mixcolumns
trn1.4s v5, v1, v1
tbl.16b v6, {v1}, v13
dup.2d v7, x31
ins.s v7[0], v1[3]
eor.16b v1, v5, v6
eor.16b v1, v1, v7
.endm
```

<표 2> MixColumns 함수 Assembly 구현 코드

4. 성능 평가

본 논문에서는 Apple 2020년형 MacBook Pro 13인치 상에서 성능 평가 측정을 진행한다. 해당 기기는 Apple에서 설계한 프로세서 M1 프로세서를 사용한다. 최신 ARM 프로세서 중 하나인 Apple M1 프로세서는 DSP, CPU, 신경망 엔진, GPU 등을 하나의 칩안에 집적한 일종의 SoC(System on Chip)이다[6].

M1 프로세서 상에서 구현된 Skinny 블록암호의 기존 연구가 없기 때문에 레퍼런스 C코드와 성능 비교를 진행한다.

본 논문에서는 Skinny 블록암호의 128-bit 블록길이의 구현을 진행하였으며, 자세한 성능 측정 결과는 <표 3>과 같다.

Type	128	256	384
reference	2774.11	4821.68	7932.54
This work	146.42	191.82	242.50
rate	18.94x	25.14x	32.71x

<표 3> 레퍼런스 C코드와 Assembly 구현 성능 측정 결과(단위: cycle)

<표 3>에서 볼 수 있듯이 기존 레퍼런스 C코드 대비 128-bit 키 길이의 경우 약 19배의 성능 향상을 확인할 수 있었다. 384-bit 키 길이의 경우 약 32배의 높은 성능 향상을 확인할 수 있었다.

5. 결 론

본 논문에서는 Tweakable 블록암호 중 하나인 Skinny 블록암호의 ARMv8 프로세서 상에서의 최적화 구현을 진행하였다. Tweakey를 활용하는 Skinny 블록암호는 AES와 유사한 라운드 함수로 설계되어 있다. 각 라운드 함수를 구현하기 위해 TBL 명령어를 적극 활용하였으며 라운드 함수의 효율적인 구현을 통해 레퍼런스 C코드의 성능 대비 128-bit 키 길이에서는 약 19배의 성능 향상을 확인할 수 있으며, 384-bit 키 길이의 경우 약 32배의 높은 성능 향상을 확인할 수 있었다. 추후 연구로 해당 구현 기법을 활용하여 벡터레지스터를 더 활용한 병렬 구현을 제안한다.

6. 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 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2021-0-00540, GPU/ASIC 기반 암호알고리즘 고속화 설계 및 구현 기술개발, 50%).

참고문헌

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

[2] 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.

[3] M Liskov, RL Rivextm D Wagner. “Tweakable block ciphers”. Advances in Cryptology-CRYPTO 2002. pp 31-46.

[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] Seo, Hwajeong, et al. "No Silver Bullet: Optimized Montgomery Multiplication on Various 64-Bit ARM Platforms." International Conference on Information Security Applications. Springer, Cham, 2021.

[6] Kwon, Hyeokdong, et al. "Look-up the Rainbow: Efficient Table-based Parallel Implementation of Rainbow Signature on 64-bit ARMv8 Processors." Cryptology ePrint Archive (2021).