ARMv8 상에서의 블록 암호 최적 구현 동향

심민주[1], 권혁동[1], 김현준[1], 서화정[1]
[1]한성대학교 it융합공학부
minjoos9797@gmail.com, korlethean@gmail.com, khj930704@gmail.com,
hwajeong84@gmail.com

Block Cipher Optimal Implementation Trend on ARMv8

Min-Joo Sim[1], Hyeok-Dong Kwon[1], Hyun-Jun Kim[1], Hwa-Jeong Seo[1]
[1]Dept. of IT Convergence Engineering, Hansung University

요 약

이동통신 산업이 급속도로 발전됨과 동시에 사물인터넷도 빠르게 발전하고 있다. 사물인터넷의 성능이 향상되면서 무선 네트워크에 포함된 많은 데이터를 포함하고 있는 사물인터넷이 증가하였다. 사물인터넷에 사용되는 저사양 프로세서들은 일반 컴퓨터에 비해 제한적이다. 그러므로, 사물인터넷에서 효율적으로 동작되는 암호 알고리즘에 대한 연구는 필수적이다. 따라서, 본 논문에서는 많은 분야에서 널리 사용되고 있는 마이크로 컨트롤러인 ARMv8 프로세서 상에서의 블록 암호 최적 구현에 대한 연구 동향에 대해 알아본다.

1. 서론

빠른 속도로 발전되고 있는 사물인터넷은 많은 개인 정보를 포함하고 있다. 이에 따라, 사물인터넷에 사용되는 저사양 프로세서 보안에 대한 연구는 활발히 진행되어야 한다. 저사양 프로세서는 일반 컴퓨터에 비해 메모리 등이 제한적이기 때문에 저사양 프로세서 상에서의 암호 알고리즘이 효율적으로 동작하기 위한 최적 연구가 필수적이다. 이에 따라, 사물인터넷에서 효율적으로 동작하는 암호 알고리즘들이 제안되고 있으며, 기존에 널리 사용되는 암호 알고리즘들을 최적 구현하여 저사양 프로세서에 탑재하는 연구가 활발히 진행되고 있다.

본 논문에서는 널리 사용되고 있는 64-bit ARMv8 프로세서에서의 블록 암호 최적 구현 동향에 대해 알아보고자 한다. 2장에서는 ARMv8 프로세서에 대해 알아본다. 3장에서는 ARMv8 상에서의 블록 암호 최적 구현 동향에 대해서 살펴본다. 마지막으로 4장에서는 본 논문의 결론을 내린다.

2. ARMv8 Processor

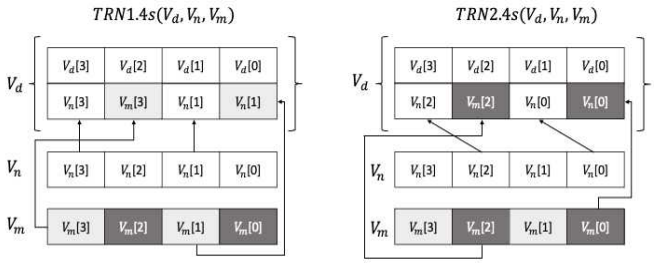
ARM(Advanced RISC Machine)은 ISA (Instruction Set Architecure) 고성능 임베디드 프로세서이다. ARM 프로세서에서는 데이터 병렬 컴퓨팅의 한 종류인 SIMD(Single Instruction Multiple Data) 명령어를 NEON이라고 칭한다. ARMv7에서는 128-bit 16개의 백터 레지스터(q0~q15)를 지원하였다. 하지만, 64-bit ARMv8은 128-bit 32개의 벡터 레지스터(v0~v31)를 지원한다[1].

Instruction	Operands	Operation
ADD	V_d, V_m, V_n	Addition
SHL	V_m, V_n, #n	Logical shift to left direction
SRI	V_m, V_n, #n	Logical shift to right direction with insertion
EOR	V_d, V_m, V_n	Exclusive-OR
TRN1	V_d, V_m, V_n	Transpose vector (primary)
TRN2	V_d, V_m, V_n	Transpose vector (secondary)
REV	V_m, V_n	Reverse vector
DUP	V_m, V_n	Duplication

[표 1] Summary of instruction set for ARMv8 NEON architecture.

그리고 벡터 레지스터 내부에서는 8, 16, 32, 64-bit 단위로 병렬 처리가 가능하다.

[표 1]은 ARMv8에 정의된 명령어셋 중 암호 구현에 주로 사용되는 명령어의 일부 정리한 것이다. ADD는 덧셈 연산을 수행한다. SHL는 레지스터를 왼쪽으로 이동시킨다. SRI는 레지스터에 값을 삽입하여 오른쪽으로 이동시킨다. EOR은 exclusive-or 연산을 수행한다. [그림 1]은 TRN1과 TRN2의 Transpose 과정의 연산을 나타낸 것이다. TRN1 명령어는 두 레지스터의 하위 값을 묶어 출력해주고, TRN2 명령어는 두 레지스터의 상위 값을 묶어 출력해준다.



[그림 1] Transpose process of TRN1.4s and TRN2.4s instructions.

REV는 halfword 혹은 word 단위로 벡터 레지스터 내부의 값들의 순서를 역방향으로 바꾼다. ROL8(Rotation left 8)이나 ROR8 (Rotation Right 8)을 구현을 할 때, SHL과 SRI를 사용하는 것보다 REV16을 사용하는 것이 더 효율적이다. 마지막으로, DUP는 General 레지스터에 저장된 값을 벡터 레지스터에 복사한다.

3. ARMv8 상에서의 블록암호 최적 구현

본 장에서는 ARMv8 프로세서 상에서 구현한 다양한 블록 암호 최적 구현 연구 동향에 대해서 살펴본다.

[2]는 WISA'13에서 발표된 ARX 기반 블록 암호 LEA를 ARMv8 상에서 24개의 평문에 대해 최적 구현하였다. 한 번의 암호화로 24개의 평문의 데이터에 동일한 연산이 수행되도록 구현하기 위해 병렬 구현을 하였다. 병렬 구현을 위해 암호화 시작 전, 벡터 레지스터를 활용하여 24개의 128-bit 평문을 24개의 벡터 레지스터에 로드하였다. 이때, 효율적으로 메모리 접근을 하기 위해 post-incremental 기법이 적용된 명령어셋을 사용하였다. 메모리에서 평문을 불러온 후, 불러온 평문과 같은 크기만큼 자동으로 증가하기 때문에 그 다음 평문을 불러오기 위한 주소를 계산하기 위해 소모되는 시간을 줄였다. 이후 벡터 레지스터에 24개의 평문에 대해 벡터 형식으로 다음과 같이 정렬하였다. TRN1과 TRN2 명령어를 각각 4번 사용하여 PT1~PT4에 해당되는 첫 번째 인자를 모아주고, 두 번째, 세 번째, 네 번째 인자를 모아주어 각각 4개의 레지스터에 대해 내부 정렬을 해주었다. 평문은 24개이기 때문에 이와 같은 레지스터 내부 정렬을 총 6번 반복하여 진행하였다. 이후 LEA 암호화가 모두 끝난 후에 얻은 암호문은 벡터화 형식으로 저장되어 있는 상태이다. 따라서, 다시 레지스터 내부 정렬하기 이전의 레지스터 정렬로 되돌려놓는 과정을 진행하였다. 위와 같은 기법을 사용하여 성능 측정을 한 결과는 다음과 같다. ARMv8 플랫폼 중에 속하는 Apple A7에서는 2.4 cpb(cycles/byte), Apple A9에서는 2.2 cpb의 성능을 달성하였다. 그리고 [2]에서 총 31개의 벡터 레지스터 중 평문에 사용되지 않은 7개의 벡터 레지스터는 중간값 저장을 하기 위한 임시 레지스터로 사용하여 메모리 접근 횟수를 최소화 하였다.

[3]은 2005년에 KISA, ETRI, 고려대학교에서 개발한 ARX 기반 블록 암호 HIGHT와 2017년 국가보안기술연구소에서 개발한 블록 암호 CHAM을 ARMv8 상에서 최적 구현하였다. [3]도 [2]와 동일하게 post-incremental 기법이 적용된 명령어셋을 사용하여 평문 블록을 메모리로부터 로드 시키거나 다시 메모리로 저장시킬 때 오버헤드 없이 병렬 처리가 자동을 이뤄질 수 있게 구현하였다. HIGHT는 8개의 평문을 병렬 구현하고, CHAM은 64/128, 128/128, 128/256 규격에 대해 각각 4개, 2개, 2개의 평문에 대해서 병렬 구현하였다. 여러 데이터 블록을 동시처리 하기 위해서는 라운드 키를 복제해야한다. 하지만, [3]은 라운드 키를 복제하는 비용을 최소화하기 위해 최적의 접근 방식과 DUP과 TRN1 명령어를 사용하여 병렬 처리하여 효율적인 키 스케줄링을 제안하였다. 이외에도 부채널 공격 중 하나인 Fault Attack에 대한 내성을 지닌 모델을 제시하였다. 해당 모델은 이미 알려진 평문과 암호문 쌍을 벡터 레지스터에 추가하여 명령어 오류를 감지한다. 그리고 임의의 비트에 따라 새로운 Random shuffling method를 적용하였다. 제안한 Random shuffling method는 TBL 명령어를 사용하여 효율적으로 Random Table을 조회한다. 그리고 랜덤 비트가 1일 경우, 스왑이 적용되고 0일 경우 스왑이 적용되지 않게 구현하였다. [3]

의 구현물에 대한 성능은 다음과 같다. Fault Attack에 대한 내성을 지니지 않은 모델은 래퍼런스 코드와 비교하였을 때, HIGHT는 100% 의 성능 향상을 보였다. 그리고 CHAM은 64/128, 128/128, 128/256에 대해서 각각 130%, 230%, 190%의 성능향상을 보였다. 그리고 Fault Attack에 대한 내성을 지닌 모델은 내성이 없는 구현물과 비교하였을 때, HIGHT에 대한 구현물은 50%의 성능 향상을 보였다. CHAM은 64/128, 128/128, 128/256에 대해서 각각 30%, 80%, 70%의 성능 향상을 보였다.

[4]는 HIGHT, CHAM 그리고 LEA에 대해 모두 카운터 운용 모드를 적용하여 ARMv8 상에서 최적 구현하였다. 32-bit 단위로 블록 연산이 진행되는 CHAM과 LEA는 4개의 벡터 레지스터에 평문을 로드하고 암호화 결과값을 다시 메모리에 로드할 때, LD4와 ST4 명령어를 사용하여 데이터 병렬화를 하였다. 하지만, 8-bit 단위로 연산 처리가 되는 HIGHT는 벡터 레지스터 단위를 16-bit로 설정하여 효율적으로 8개의 벡터 레지스터에서 16개의 평문에 대해 병렬 구현을 하였다. 카운터 운용모드는 고정된 논스를 사용하기 때문에 사전 연산 테이블을 통해 HIGHT는 5라운드, CHAM은 7라운드 그리고 LEA에서는 4라운드까지 최적화 하였다. 제안된 기법이 적용된 HIGHT, CHAM, LEA는 각각 8.62%, 15.87%, 8.76%의 성능 향상을 보였다.

[5]는 ICISC'20에서 발표된 블록암호 PIPO를 ARMv8 상에서 8평문과 16개의 평문에 대해 최적 구현하였다. 8개, 16개의 평문에 대한 병렬 구현은 각각 4개, 8개의의 벡터 레지스터를 사용하였다. 라운드 키는 한 번 사용할 때 마다 64-bit씩 사용하여 하나의 벡터 레지스터를 사용하였다. 평문에 대한 레지스터 내부 정렬은 UZP1, UZP2, TRN1, TRN2 명령어를 사용하여 정렬하였다. 명령어는 4개를 사용하지만 레지스터의 16b, 4s 등과 같은 arrangement를 변경해주어 [2~4]과 같이 동일 연산이 진행되는 블록을 하나의 레지스터에 내부 정렬을 해주었다. 레지스터 내부 정렬은 기존 S-Layer 연산이 진행되기 전에 정렬해주고 R-Layer 이후에 다시 레지스터 배치를 복구해주어 하나의 라운드 당 2번의 레지스터 내부 정렬이 필요하다. 하지만, 이렇게 매 라운드 마다 레지스터 내부 정렬을 하는 것에 소요되는 시간을 줄이기 위해 레지스터 내부 정렬 최소화 기법을 제안하였다. PIPO의 라운드 함수는 AddRoundkey, S-Layer, R-Layer, AddroundKey로 구성된다. 따라서, 해당 기법은 S-Layer 연산을 하기 전 단계인 AddRoundKey의 데이터를 레지스터 내부 정렬에 맞도록 변경해줌으로써, 내부 레지스터 정렬을 매 라운드 별로 2번씩 진행해야했던 내부 정렬을 전체 라운드에서 2번만 진행하게 최소화하였다. [5]의 구현물에 대한 성능은 다음과 같다. 8개의 평문에 대한 병렬 구현은 64/128, 128/128, 128/256에 대해서 각각 65.3%, 66.4%, 76.3%, 77.2%, 그리고 16개의 평문에 대한 병렬 구현은 각각 81.8%, 82.1%, 88.7%, 89.3% 만큼의 성능 향상을 보였다.

4. 결론

본 논문에서는 ARMv8 상에서의 블록 암호 최적 구현 연구 동향에 대해 살펴보았다. 최근 다양한 서비스가 제공됨에 따라, 사물인터넷은 많은 데이터를 포함하고 있다. 그렇기 때문에, 사물인터넷에 사용되는 저사양 프로세서의 보안에 대한 관심은 필수적이다. 저사양 프로세서에 암호 알고리즘을 탑재하기 위해 기존 블록 암호부터 최근 발표된 경량 블록 암호까지 다양한 블록 암호를 대상으로 최적 구현 연구가 활발히 진행되고 있다. 앞으로도 이러한 암호 최적 구현 연구가 지속적으로 진행되어야 할 것으로 생각된다.

5. 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, 100%).

참고문헌

[1] Arm® A64 Instruction Set Architecture: Armv8, for Armv8-A Architecture Profile. Available online: https://developer.arm.com/docs/ddi0596/c/simd-and-floating-point-instructions-alphabetic-order (accessed on 2 February 2021).

[2] H. J Seo, "High speed implementation of LEA on ARMv8," Journal of the Korea Institute of Information and Communication Engineering Vol.21, No.10, pp.1929-1934, 2017.

[3] J. G. Song and S. C. Seo, "Secure and fast implementation of ARX-based block ciphers using ASIMD instructions in ARMv8 platforms,"

IEEE Access 8 : 193138-193153, 2020.
[4] J. G. Song, and S. C. Seo, "Efficient parallel implementation of CTR mode of ARX-based block ciphers on ARMv8 microcontrollers," Applied Sciences 11.6, 2548, 2021.
[5] S. W. Eum, et al., "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, 2021.