64-bit ARM 프로세서 상에서의 블록암호 PIPO 병렬 최적 구현

엄시우*, 권혁동**, 김현준**, 장경배**, 김현지*, 박재훈*, 심민주*, 송경주*, 서화정*
*한성대학교 IT융합공학부
**한성대학교 정보컴퓨터공학과
shuraatum@gmail.com, korlethean@gmail.com, khj930704@gmail.com,
starj1023@gmail.com, khj1594012@gmail.com, p9595jh@gmail.com,
minjoos9797@gmail.com, thdrudwn98@gmail.com, hwajeong84@gmail.com

Optimized implementation of block cipher PIPO in parallel-way on 64-bit ARM Processors

Si-Woo Eum*, Hyeok-Dong Kwon**, Hyun-Jun Kim**, Kyung-Bae Jang**,
Hyun-Ji Kim*, Jae-Hoon Park*, Min-Joo Sim*, Gyeong-Ju Song*,
Hwa-Jeong Seo*
*Dept. of IT Convergence Engineering, Hansung University
**Dept. of Information Computer Engineering, Hansung University

요 약

ICISC'20에서 발표된 경량 블록암호 PIPO는 비트 슬라이스 기법 적용으로 효율적인 구현이 되었으며, 부채널 내성을 지니기에 안전하지 않은 환경에서도 안정적으로 사용 가능한 경량 블록암호이다. 본 논문에서는 ARM 프로세서를 대상으로 PIPO의 병렬 최적 구현을 제안한다. 제안하는 구현물은 8평문, 16평문의 병렬 암호화가 가능하다. 구현에는 최적의 명령어 활용, 레지스터 내부 정렬, 로테이션 연산 최적화 기법을 사용하였다. 구현은 A10x fusion 프로세서를 대상으로 한다. 대상 프로세서상에서, 기존 레퍼런스 PIPO 코드는 64/128, 64/256 규격에서 각각 34.6 cpb, 44.7 cpb의 성능을 가지나, 제안하는 기법은 8평문 64/128, 64/256 규격에서 각각 12.0 cpb, 15.6 cpb, 16평문 64/128, 64/256 규격에서 각각 6.3 cpb, 8.1 cpb의 성능을 보여준다. 이는 기존 대비 각 규격별로 8평문 병렬 구현물은 약 65.3%, 66.4%, 16평문 병렬 구현물은 약 81.8%, 82.1% 더 좋은 성능을 보인다.

1. 서론

경량 암호는 저사양, 제한된 환경(낮은 성능, 작은 메모리, 낮은 전력 등)의 사물인터넷 기기를 위해 개발되었다. 하지만 대부분의 경량 암호는 부채널 공격(Side-Channel Attack)에 취약하다[1]. 부채널 공격은 암호화가 진행되는 동안의 각종 부차적인 정보들을 (소요 시간, 전력, 소리 등) 분석하여 암호화에 사용된 키를 예측하는 물리적 공격 기법이다.

본 논문에서는 경량 프로세서 중 하나인 ARM 프로세서 상에서 부채널 내성을 지니고 있는 경량 블록암호 PIPO의 병렬 암호화를 활용한 최적 구현을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 PIPO 블록암호와 64-bit ARM 프로세서에 대해서 설명한다. 3장에서는 병렬 최적 구현을 위한 최적 구현 기법에 대해 설명한다. 4장에서는 기존 PIPO 구현물과 제안하는 구현물의 성능을 비교한다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

2. 배경

2.1. 경량 블록암호 PIPO

PIPO[2]는 ICISC'20에서 발표된 경량 블록암호이다. PIPO는 다른 블록암호보다 비선형 연산을 적게 사용하여 구현되었고, 효율적인 고차 마스킹 소프트웨어 구현이 가능하며, 특히 8-bit AVR 상에서 부채널 공격 내성을 지니기에 다른 동일한 입·출력 규격을 지닌 블록암호보다 뛰어난 보안성을 제공한다.

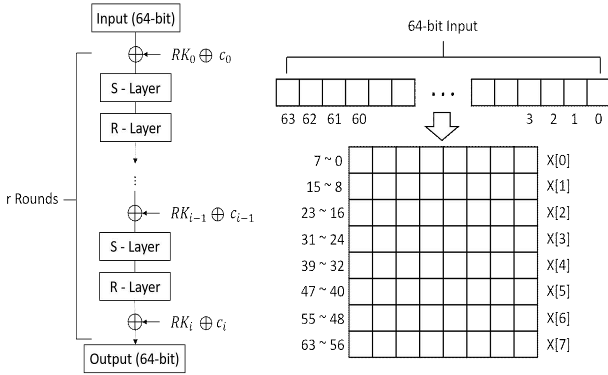
PIPO는 SPN(Substitution-Permutation Network) 구조를 채택하였다. 평문 길이로는 64-bit 사용하며 키 128-bit, 256-bit 크기는 두 종류를 사용한다. PIPO의 매개변수는 표 1에서 확인 가능하다.

<표 1> Parameters of PIPO block cipher

Type	Block size	Key size	Rounds
64/128	64-bit	128-bit	13
64/256	64-bit	256-bit	17

PIPO의 라운드 함수는 비선형 연산이 적용되는 S-layer와 선형 연산이 적용되는 R-layer로 구성된다. 각 라운드 함수의 종료 시, 라운드 키가 XOR된다. 그림 1은 PIPO의 전체적인 구조를 나타낸 것이다.

<그림 1> Structure of PIPO block cipher



라운드 키는 키 길이에 따라 다음 수식과 같이 생성된다.

- Key size = 128

$$K^{128} = K_1^{64} \| K_0^{64}$$

$$RK_i = K_{i \bmod 2} (i = 0, 1, \ldots 13)$$

- Key size = 256

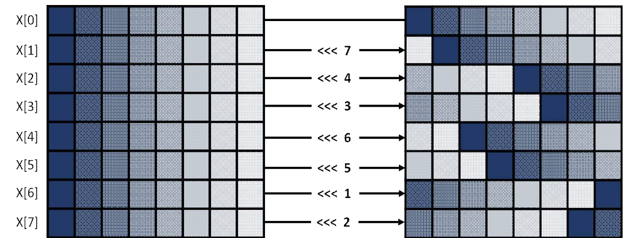
$$K^{256} = K_3^{64} \| K_2^{64} \| K_1^{64} \| K_0^{64}$$

$$RK_i = K_{i \bmod 4} (i = 0, 1, \ldots 17)$$

S-layer는 룩업 테이블 사용과 비트 슬라이스 두 가지로 구현되어 있다. 본 논문에서는 11개의 비선형 비트 연산과 23개의 선형 비트 연산만 포함하는 효율적인 비트 슬라이스 구현을 사용한다.

R-layer는 효율적인 하드웨어 및 소프트웨어 구현을 보장하기 위해, 그림 2와 같이 바이트 단위의 비트 회전만 사용하는 비트 순열로 구현되어 있다.

<그림 2> Flow of R-layer



2.2. 대상 프로세서 64-bit ARM 프로세서

ARM 프로세서는 AVR을 위시한 사물인터넷 환경에서, 고성능을 자랑하는 경량 프로세서 중 하나이다. ARM 프로세서 중, ARMv8-A (ARMv8)은 ARM 프로세서 제품군 중에서 최신 프로세서 중 하나이다. 본 논문에서는 ARMv8의 두 종류 중 하나인, 64-bit AArch64를 대상으로 한다. AArch64 (A64)는 스칼라 레지스터, 벡터 레지스터로 두 종류의 레지스터를 지니며, 각각 32개씩 존재한다. 이 중에서 벡터 레지스터는 데이터의 병렬 처리에 사용할 수 있다[3].

<표 2> Summarized instruction set for optimized PIPO in parallel-way, Vd: Destination vector register, Vn/Vm: Source vector register, Vt: Transferred vector register, Xn: Source scalar register

Instruction	Operand	Description	Operation
AND	Vd, Vn, Vm	Bitwise AND	Vd ← Vn & Vm
EOR	Vd, Vn, Vm	Bitwise Exclusive OR	Vd ← Vn ⊕ Vm
LD1	Vd1-4, (Xn)	Load multiple single-element	Vd1-4 ← (Xn)
MOV	Vt, Vn	Move vector	Vt ← Vn
NOT	Vd, Vn	Bitwise NOT	Vd ← !Vn
ORR	Vd, Vn, Vm	Bitwise inclusive OR	Vd ← Vn | Vm
SLI	Vd, Vn, #shift	Shift Left and Insert immediate	Vd ← Vn << #shift
SRI	Vd, Vn, #shift	Shift Right and Insert immediate	Vd ← Vn >> #shift
ST1	Vt1-4, (Xn)	Store multiple single-element	(Xn) ← Vt1-4
TRN1/2	Vd, Vn, Vm	Transpose vectors primary/secondary	Vd ← Vn[even], Vm[even] Vd ← Vn[odd], Vm[odd]
UZP1/2	Vd, Vn, Vm	Unzip vectors primary/secondary	
ZIP1/2	Vd, Vn, Vm	Zip vectors primary/secondary	

3. 제안기법

제안하는 기법은, 블록암호 PIPO를 ARM 프로세서 상에서 병렬 최적 구현하는 기법을 제시한다. 제안하는 구현물은 명령어 활용, 레지스터 내부 정렬, 로테이션 연산 최적화의 두 가지 최적 기법을 사용하였다.

3.1. 명령어 활용

ARM 프로세서에는 병렬 연산을 위한 Vector instruction이 제공된다. 구현을 위해 최소한의 명령어들을 사용하였으며, 구현에 사용된 명령어는 표 2에서 확인이 가능하다.

3.2. 레지스터 내부 정렬

S-layer는 치환 연산, R-layer는 로테이션 연산을 진행한다. 이때 하나의 레지스터에 동일한 평문 인덱스의 값들만 들어있다면, 명령어 하나로 다수의 평문이 S-layer를 통과하도록 할 수 있다. 벡터 레지스터는 128-bit를 저장할 수 있으므로, 초기 상태에서는 64-bit의 평문(X[0]-X[7]) 두 개를 저장할 수 있다. 하나의 평문 블록 X는 8-bit이므로, 벡터 레지스터 하나에는 동일한 인덱스의 평문 블록을 최대 16개까지 저장할 수 있다. 예를 들어, 하나의 벡터 레지스터에 첫 번째부터 마지막 평문 16개의 X[0] 블록을 저장할 수 있다. 나머지 블록에 대해서도 동일하게 하나의 벡터 레지스터에 저장할 수 있다.

이를 구현하기 위해서 표 3의 코드를 사용한다. 코드의 좌측은 S-layer 진입 전 값을 정렬하는 코드이다. S-layer 이후에는 R-layer가 이어지므로 R-layer 진입 시에는 레지스터 정렬을 진행하지 않는다. R-layer 이후로는 라운드 키를 포함시켜야 하므로 레지스터의 배치를 복구해야 한다. 이는 표 3의 의 우측 코드를 사용한다. 8개 평문 병렬의 경우에는 표 3 코드의 '4s, 8h, 16b'와 같은 arrangement를 변경하는 것으로 하나의 레지스터에 8개의 평문 블록이 저장되도록 한다.

3.3. 로테이션 연산 최적화

R-layer에서는 로테이션 연산이 주로 사용된다. 다른 경량 프로세서인 AVR 프로세서의 경우, 어셈블리 명령어로 ROL, ROR과 같은 로테이션 연산을 자체적으로 지원한다. 하지만 ARM 프로세서의 어셈블리 명령어 중에서는 로테이션 연산을 완벽하게 지원하는 명령어가 존재하지 않는다. 대신 시프트

<표 3> Register value alignment code

Pre alignment code	Post alignment code
UZP1.4s v8, v0, v1	ZIP1.2d v8, v0, v4
UZP1.4s v9, v2, v3	ZIP1.2d v9, v2, v6
UZP2.4s v10, v0, v1	ZIP1.2d v10, v1, v5
UZP2.4s v11, v2, v3	ZIP1.2d v11, v3, v7
UZP1.4s v12, v4, v5	ZIP2.2d v12, v0, v4
UZP1.4s v13, v6, v7	ZIP2.2d v13, v2, v6
UZP2.4s v14, v4, v5	ZIP2.2d v14, v1, v5
UZP2.4s v15, v6, v7	ZIP2.2d v15, v3, v7
UZP1.8h v0, v8, v9	ZIP1.16b v0, v8, v10
UZP1.8h v1, v10, v11	ZIP1.16b v1, v9, v11
UZP2.8h v2, v8, v9	ZIP2.16b v2, v8, v10
UZP2.8h v3, v10, v11	ZIP2.16b v3, v9, v11
UZP1.8h v4, v12, v13	ZIP1.16b v4, v12, v14
UZP1.8h v5, v14, v15	ZIP1.16b v5, v13, v15
UZP2.8h v6, v12, v13	ZIP2.16b v6, v12, v14
UZP2.8h v7, v14, v15	ZIP2.16b v7, v13, v15
UZP1.16b v8, v0, v1	ZIP1.8h v8, v0, v1
UZP1.16b v9, v2, v3	ZIP1.8h v9, v2, v3
UZP2.16b v10, v0, v1	ZIP2.8h v10, v0, v1
UZP2.16b v11, v2, v3	ZIP2.8h v11, v2, v3
UZP1.16b v12, v4, v5	ZIP1.8h v12, v4, v5
UZP1.16b v13, v6, v7	ZIP1.8h v13, v6, v7
UZP2.16b v14, v4, v5	ZIP2.8h v14, v4, v5
UZP2.16b v15, v6, v7	ZIP2.8h v15, v6, v7
TRN1.2d v0, v8, v12	ZIP1.4s v0, v8, v9
TRN1.2d v1, v10, v14	ZIP2.4s v1, v8, v9
TRN1.2d v2, v9, v13	ZIP1.4s v2, v10, v11
TRN1.2d v3, v11, v15	ZIP2.4s v3, v10, v11
TRN2.2d v4, v8, v12	ZIP1.4s v4, v12, v13
TRN2.2d v5, v10, v14	ZIP2.4s v5, v12, v13
TRN2.2d v6, v9, v13	ZIP1.4s v6, v14, v15
TRN2.2d v7, v11, v15	ZIP2.4s v7, v14, v15

명령어를 활용하여 로테이션을 구현할 수 있다.

로테이션 연산 구현에는 표 2에 나열된 명령어 중에서, SLI, SRI 명령어를 활용한다. SLI 명령어는 벡터 레지스터의 값을 왼쪽 방향으로 시프트 한 이후, 레지스터의 빈 공간에는 기존 레지스터가 가지고 있던 값을 채우게 된다. SRI 명령어는 오른쪽으로 시프트하는 차이점이 있다.

로테이션 연산의 구현은 SLI, SRI 명령어를 하나씩 조합하는 것으로 이루어진다. 이때, 원본 값은 저

장할 대상 레지스터와는 다른 레지스터에 존재해야 하므로, 임시 레지스터가 한 개 필요하다.

4. 성능평가

본 장에서는 구현물의 성능평가를 진행한다. 비교 대상으로는 ICISC'20에서 발표하며 배포된 PIPO 레퍼런스 C 코드 구현물을 사용한다.

구현은 Xcode 프레임워크를 사용하여 구현하며, 컴파일 옵션 -O2를 사용하여 컴파일한다. 대상 프로세서는 ARM 프로세서 중 하나인 A10X Fusion 프로세서를 대상으로 구현한다. 성능 비교의 단위로는 cpb(cycles per bytes)를 사용한다. cpb 단위는 1바이트를 연산하는데 소요되는 클록 사이클의 수를 의미한다.

비교 결과는 표 4 에서 확인 가능하다.

<표 4> Comparison result table (Unit: cpb), [8]: 8-PT in parallel-way [16]: 16-PT in parallel-way

Type	Ref. C	This work[8]	This Work[16]
64/128	34.6	12.0	6.3
64/256	44.7	15.6	8.1

기존 구현물은 64/128은 34.6 cpb, 64/256은 44.7 cpb의 성능을 보인다. 이에 반해, 제안하는 기법의 8평문 병렬 연산은 64/128, 64/256 규격에서 각각 12.0 cpb, 15.6 cpb의 성능을 가진다. 또한 16평문 병렬 연산은 각각의 규격에서 6.3 cpb, 8.1 cpb의 성능을 가진다. 결과적으로 레퍼런스 코드 대비 64/128, 64/256 규격 별로 8평문 병렬 구현물은 각각 65.3%, 66.4%, 16평문 병렬 구현은 각각 81.8%, 82.1% 만큼의 더 높은 성능을 보인다.

이러한 성능 격차의 가장 큰 원인은 병렬 구현에 있다. 기존 구현물은 병렬 연산이 불가하므로 1개의 평문만을 암호화 할 수 있는 반면, 제안하는 구현물은 8개 또는 16개의 평문을 동시에 암호화 할 수 있다. 즉, 단위 시간당 처리 가능한 바이트의 수가 늘어났기 때문에 기존 대비 뛰어난 성능을 지닌다.

또한 어셈블리 명령어를 사용한 최적 구현도 성능 개선에 이바지한다. 특히 R-layer의 로테이션 연산을 단 두 개의 명령어를 사용하여 구현하는 부분에서 동작 속도를 크게 개선시킬 수 있다.

5. 결론

본 논문에서는 경량 블록 암호 PIPO를 ARM 프로세서 상에서 병렬 최적 구현하는 기법을 제시하였다. 제안하는 기법은 벡터 레지스터와 명령어를 활용하여 8평문 또는 16평문을 동시에 암호화하는 병렬 구현을 시도하였다. 또한 최적 구현을 위해 효율적인 명령어를 사용하였고 레지스터 정렬을 통해 병렬 연산이 이루어지게 하였다. 추가로 로테이션 연산 구현을 두 개의 명령어로 처리하여 최적의 로테이션 연산을 구현하였다.

제안하는 기법을 통해 기존 레퍼런스 코드 대비 각각의 규격에서 8평문 병렬 구현은 65.3%, 66.4%, 16평문 병렬 구현은 81.8%, 82.1% 만큼의 더 높은 성능 향상을 확인할 수 있었다. 향후 연구 과제로 병렬 연산 최적 구현 기법을 다양한 경량 암호 알고리즘에 활용한 연구를 제시한다.

6. Acknowledgment

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.2018-0-00264, IoT 융합형 블록체인 플랫폼 보안 원천 기술 연구) 그리고 이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478).

참고문헌

[1] A. Heuse, S. Picek., S. Guilley, and N. MEntens. Side-channel Analysis of Lightweight Ciphers: Does Lightweight Equal Easy?. Lecture Notes in Computer Science, 10155, pp 91-104, 2017.

[2] H.G. Kim, Y.J. Jeon, G.Y. Kim, J.S. Kim, B.Y. Sim, D.G. Han, H.J. Seo, S.G. Kim, S.H. Hong, J.C. Sung, and D.J. Hong. A New Method for Designing Lightweight S-Boxes with High Differential and Linear Branch Numbers, and Its Application. International Conference on Information Security and Cryptology (ICISC 2020), Seoul, Korea, 2020, 62 pages.

[3] H.J. Seo, Z. Liu, P. Longa, and Z. Hu. SIDH on ARM: Faster Modular Multiplications for Faster Post-Quantum Supersingular Isogeny Key Exchange. Conference on Cryptographic Hardware and Embedded Systems (CHES 2018), Amsterdam, Netherlands, 2018, 19 pages.