ARMv8 상에서의 Fault Attack에 안전한 블록 암호 최적 구현 연구 동향

심민주*, 권혁동*, 엄시우*, 서화정*†

*한성대학교 IT융합공학부(대학원생)

*†한성대학교 IT융합공학부 (교수)

Title

Min-Joo SIM*, Hyeok-Dong Kwon*, Si-Woo Eum*, Hwa-Jeong Seo*†

*Dept. of IT Convergence Engineering, Hansung University
(Graduate student)

*†Dept. of IT Convergence Engineering, Hansung University
(Professor)

요 약

대부분의 암호는 부채널 공격 중 하나인 Fault Attack에 취약하다. Fault Attack에 대한 대응 방법은 결함된 데이터를 감지하고, 그 데이터를 복구할 수 있도록 하는 것이다. 이를 위해 중요한 데이터를 중복으로 저장하거나 연산하여 구현이 가능하다. 최근 많은 정보를 포함하고 있는 사물인터넷에 사용되는 저사양 프로세서에 탑재된 블록 암호도 Fault Attack에 취약하다. 이에 따라, 저사양 프로세서 상에서 Fault Attack에 안전한 블록 암호에 대한 최적 구현 연구도 활발히 진행되고 있다. 본 논문에서는 ARMv8 상에서의 Fault Attack에 안전한 블록 암호 최적 구현 동향에 대해서 살펴본다.

I. 서론

사물인터넷이 탑재된 기기 사이의 통신 시스템이 안전하게 구축되어 있다고 하더라도 공격자가 프로그램에 오류를 주입한다면, 사용자의 비밀 정보를 추출할 수 있다. 따라서, 사물인터넷에 사용되는 저사양 프로세서에 탑재된 암호 시스템은 부채널 공격 중 하나인 Fault Attack에 안전해야한다. 이에 따라 Fault Attack에 안전한 블록암호에 대한 연구가 수행되었다[1], [2]. 그리고 국산 블록 암호인 HIGHT에 대한 Cortex-M3, Cortex-M4 상에서의 연구도 수행되었다[3], [4]. 이처럼 Fault Attack에 대응하기 위한 최적 구현 연구가 활발히 진행되고 있다.

따라서, 본 논문에서는 ARMv8 상에서의 Fault Attack에 안전한 블록암호 최적 구현 연구 동향을 살펴본다. 2장에서는 Fault Attack Resistance와 ARMv8 프로세서에 대해 알아본다. 3장에서는 ARMv8 상에서의 Fault Attack Injection에 안전한 블록암호 최적 구현 연구 동향에 대해서 살펴본다. 마지막으로 4장에서 본 논문의 결론을 내린다.

II. 관련 연구

2.1 Fault Attack Resistance

오류 모델(Fault Model)은 암호 시스템에서 오류를 주입하고 연산 오류나 명령어 오류를 조작할 수 있다[1], [2].

연산 오류는 Random Word, Random Byte, Random Bit, Chosen Bit Pair 이렇게 4가지로 나눌 수 있다. 공격자는 특정 값을 목표로 하는데 데이터 변경을 하는데 워드, 바이트, 비트 그리고 선택된 비트 쌍에 대해 알 수 없는 임의의 값으로 변경한다.

명령어 오류는 오류 주입에 의해 명령어가 변경되는 것으로. 일반적으로 특정 명령어에 대해 실행을 하지 않고 nop 명령어로 대체하여 특정

명령어를 건너뛰게끔 변경한다.

Fault Attack에 안전한 암호 구현을 대응책은 Inter-Instruction Redundancy(IRR)를 적용하여 연산 오류를 감지한다. 하지만, 원본 데이터와 중복된 데이터(Redundancy data)에 동일한 오류가 중복된다면, 이를 우회할 수 있다. 따라서, 알려진 평문, 암호문 쌍을 벡터 레지스터에 추가하여 명령어 오류를 감지한다. 그리고 임의의 비트에 따라 랜덤 셔플링을 적용하여 공격자가 프로그램에서 특정 값을 표적으로 삼는 것을 어렵게 한다.

2.2 ARMv8 Processor

ARM(Advanced RISC Machine)은 ISA (Instruction Set Architecture) 고성능 임베디드 프로세서이다. ARM 프로세서에서는 데이터 병렬 컴퓨팅의 한 종류인 SIMD(Single Instruction Multiple Data) 명령어를 NEON이라고 칭한다. 64-bit ARMv8은 31개의 64-bit general 레지스터와 128-bit 32개의 벡터 레지스터(v0~v31)를 지원한다[1]. 그리고 벡터 레지스터 내부에서는 packing unit에 따라 8, 16, 32, 64비트 단위로 병렬 처리가 가능하다.

Instruction	Operands	Operation
ADD	V_d, V_m, V_n	Addition
SRI	V_m, V_n, $\#n$	Logical shift to right direction with insertion
SHL	V_m, V_n, $\#n$	Logical shift to left direction
EOR	V_d, V_m, V_n	Exclusive-OR
UZP1	V_d, V_m, V_n	Unzip vectors (primary)
UZP2	V_d, V_m, V_n	Unzip vectors (secondary)
REV	V_m, V_n	Reverse vector
DUP	V_m, V_n	Duplication

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

[표 1]은 ARMv8에 정의된 명령어셋 중 암호 구현에 주로 사용되는 명령어의 일부 정리한 것이다.

III. ARMv8 상에서의 Fault Attack에 안전한 블록 암호 최적 구현 동향

본 장에서는 ARMv8 상에서의 Fault Attack에 안전한 블록 암호 최적 구현 동향에 대해서 살펴본다.

3.1 HIGHT Block Cipher

HIGHT는 2005년 한국인터넷진흥원(KISA), ETRI, 고려대가 공동으로 개발한 경량 블록 암호로, 변형된 Feistel ARX(Addition, Rotation, XOR) 구조이다[6].

[7]은 Fault Attack에 대한 대응책에서 많은 연산이 필요한 기존 랜덤 셔플링 과정[2]에 대해 ARM NEON을 사용한 랜덤 셔플링 최적화 기법을 제안하였다. 랜덤 테이블을 효율적으로 조회하는 TBL 명령어를 사용하여 랜덤 셔플링을 구현하였다. 이에 따라, 필요한 레지스터 수가 기존보다 적고, 기존보다 높은 성능을 보였다.

그리고 ARMv7과 ARMv8 명령어 사이에 차이가 있어 [3]에서 제안한 오류 검사 방식 대신 새로운 오류 검사 방식을 제안하였다. 새로운 오류 검사 방식은 다음과 같다. UZP1 명령어를 통해 암호화된 데이터만 추출 후, UZP2 명령어를 통해 중복 데이터를 추출한다. 그리고 중복 데이터와 암호화된 데이터를 XOR 연산을 하여 연산 오류를 확인한다. 그리고 암호화된 데이터를 저장하는 레지스터에 위치한 KC(Known Cipher text)을 추출하고, 추출된 KC를 TBL 명령어를 사용하여 올바르게 재정렬한다. 그리고 KC와 알려진 암호문을 xor 연산하여 비교한다. 마지막으로, KC와 알려진 암호문에 대해 xor 연산한 결과와 중복데이터와 실제 암호화된 데이터를 xor 연산 결과를 다시 xor 연산해주어 0이 결과 값으로 나온다면, 오류가 주입되지 않았다고 판단하여 암호문을 반환한다.

[그림 1]은 HIGHT에 대해 Fault Attack에 안전하게 구현하기 위한 벡터 레지스터 구성이다. PT는 평문이고, RT는 평문에 대한 중복된 데이터를 의미한다. 그리고 KC는 명령어 오류를

감지하기 위해 알려진 데이터에 대한 암호문 값을 각각 배치한 것이다.

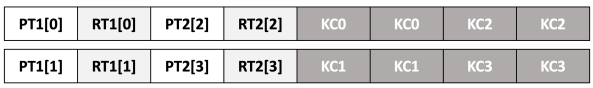
[그림 1] Vector register setting for revised HIGHT Fault attack countermeasure(PT : Plain text, RT : Redundancy plaintext, KC : Known Ciphertext).

HIGHT에서 Fault Attack Resistance 최적 구현은 다음과 같다. 평문과 알려진 평문을 벡터 레지스터에 로드한다. IRR 적용을 위해 MOV 명령어와 TBL 명령어를 사용하여 [그림 1]과 같이 레지스터 내부정렬을 수행한다. 메시지와 라운드 키에는 랜덤 셔플링이 적용된다. 이때, HIGHT는 32개의 라운드를 수행하기 때문에 32비트 임시 비트를 사용한다. HIGHT 32번의 라운드를 모두 수행 후, IRR 적용하기 전 레지스터 정렬과 동일하게 다시 내부 정렬을 수행한다. 마지막으로, 오류 검사를 통해 오류 여부를 검사한다.

이와 같은 기법이 적용된 성능은 레퍼런스 보다 100% 성능 향상을 보였다.

3.2 Revised CHAM Block Cipher

CHAM은 ICISC'17에서 발표된 Feistel 구조로 되어 있는 경량 블록 암호 알고리즘이다. Revised CHAM은 기존 CHAM과 동일하며, 라운드 수의 차이만 있다[8].



[그림 2] Vector register setting for revised CHAM-64/128 attack countermeasure(PT : Plain text, RT : Redundancy plaintext, KC : Known Ciphertext).

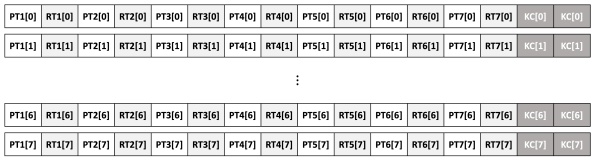
[7]은 CHAM에 대한 구현도 HIGHT와 유사하게 구현하였다. [그림 2]는 CHAM에 대해 Fault Attack에 안전하게 구현하기 위한 벡터 레지스터 구성이다. 레지스터 내부 정렬은 HIGHT도 동일하게 구현되었다. HIGHT는 각 라운드를 랜덤 셔플링으로 수행되었지만, CHAM은 4라운드마다 수행하게 구현하여 랜덤 셔플링이 4라운드마다 한 번씩 수행되게 구현되었다. 따라서, 12비트의 랜덤 비트가 필요하다. 이외의 나머지 과정은 HIGHT와 동일하여 설명을 생략한다.

이와 같은 기법이 적용된 성능은 레퍼런스 코드 대비 CHAM-64/128, CHAM-128/128, CHAM-128/256에 대해 각각 130%, 230%, 190% 성능 향상을 보였다.

3.3 PIPO Block Cipher

PIPO는 ICISC'20에서 발표된 SPN (Substitution, Permutation Network) 구조를 가진 경량 블록 암호이다[9].

[그림 3]은 PIPO에 대해 Fault Attack에 안전하게 구현하기 위한 벡터 레지스터 구성이다.



[그림 3] Vector register setting for PIPO attack countermeasure(PT : Plain text, RT : Redundancy plaintext, KC : Known Ciphertext).

[10]은 랜덤 셔플링을 ARM/NEON 프로세서의 각 명령어를 모두를 사용하는 Interleaved 방식을 사용하여 구현 기법을 제안하였다. [7]의 랜덤 셔플링 구현 실행 시간은 ARM과 NEON 명령어의 개수의 합이다. 하지만, Interleaved 방식을 적용하면, ARM/NEON 프로세서의 각 명령어를 인터리빙하여 ARM 연산에 대한 Latency가 NEON의 오버헤드에 효과적으로 숨겨지기 때문에, 전체적인 연산 오버헤드를 효과적으로 줄일 수 있다. 그리고 랜덤 셔플링을 제외한 나머지 방법은 ARM은 [3]을 활용하였고 NEON은 [7]을 활용하여 구현되었다.

이와 같은 기법이 적용된 성능은 레퍼런스 보다 3배 빠른 성능을 보였다.

IV. 결론

본 논문에서는 ARMv8 상에서의 Fault Attack에 안전한 블록 암호 최적 구현 연구 동향에 대해 살펴보았다. TBL를 사용하여 효율적인 랜덤 셔플링 기법, 새로운 오류 검사 방식을 제안된 것을 확인하였다. ARM/NEON Interleaved 방식을 적용한 효율적인 랜덤 셔플링 최적 구현 기법도 확인하였다. 따라서, 향후 연구로, 본 논문에서 살펴본 최적기법을 적용하여 HIGHT, CHAM, PIPO 이외의 블록 암호에 대해서도 ARMv8 상에서 Fault Attack에 안전한 최적 구현을 제안한다.

[참고문헌]

[1] Patrick, Conor, et al. "Lightweight fault attack resistance in software using intra-instruction redundancy." International Conference on Selected Areas in Cryptography. Springer, Cham, 2016.

[2] Seo, Hwajeong, et al. "Lightweight fault attack resistance in software using intra-instruction redundancy, revisited." International Workshop on Information Security Applications. Springer, Cham, 2017.

[3] Seo, Hwajeong, and Zhe Liu. "All the hight you need on cortex-M4." International Conference on Information Security and Cryptology. Springer, Cham, 2019.

[4] Seo, H., Kim, H., Jang, K., Kwon, H., Sim, M., Song, G., ... & Kim, H. (2021). Secure HIGHT Implementation on ARM Processors. Mathematics, 9(9), 1044.

[5] 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).

[6] KISA: HIGHT Algorithm Specification. Accessed: Aug. 20, 2020. [Online]. Available: https://seed.kisa.or.kr/kisa/algorithm/-EgovHight- Info.do

[7] Song, Jingyo, and Seog Chung Seo. "Secure and fast implementation of ARX-based block ciphers using ASIMD instructions in ARMv8 platforms." IEEE Access 8 (2020): 193138-193153.

[8] D.Roh, B.Koo, Y.Jung, I.W.Jeong, D.-G.Lee, and D.Kwon,'Revised version of block cipher CHAM,' in Information Security and Cryptology. Seoul, South Korea: Springer, 2019, pp. 1-19.

[9] H.Kim, Y.Jeon, G.Kim, J.Kim, B.-Y.Sim, D.-G.Han, H.Seo, S.Kim, S. Hong, J. Sung, and D. Hong, "Pipo: A lightweight block cipher with efficient higher-order masking software implementations," in Information Security and Cryptology - ICISC 2020 - 23rd International Conference, Seoul, South Korea, December 2-4, 2020, Proceedings, pp. 99-122, Springer, 2020.

[10] Song, Jingyo, Youngbeom Kim, and Seog Chung Seo. "High-Speed Fault Attack Resistant Implementation of PIPO Block Cipher on ARM Cortex-A." IEEE Access 9 (2021): 162893-162908.