사전 연산 테이블을 활용한 RISC-V 상에서의 Fixslicing AES CTR 모드 구현

엄시우*, 김현준*, 심민주*, 송경주*, 서화정**

*한성대학교 (대학원생)

**한성대학교 (교수)

Implementation of Fixslice AES CTR mode on RISC-V using Pre-calculation table

Si-Woo Eum*, Hyun-Jun Kim*, Min-Joo Sim*, Gyeong-Ju Song*, Hwa-Jeong Seo**

*Hansung University(Graduate student)

**Hansung University(Professor)

요 약

Fixslicing AES는 Bitsliced AES의 선형 계층의 비용을 최소화하기 위해 Shiftrows 단계를 생략한 기법으로 Bitsliced 기법 대비 30% 성능 향상을 보여준다. 본 논문에서는 사전 연산 테이블 기법을 활용한 RISC-V 상에서의 Fixslicing AES의 CTR 모드 구현을 제안한다. CTR 모드의 특징을 활용하여 2 Round Sbox 연산까지의 사전 연산을 통해 암호화 과정에서 2 Round Sbox까지 생략하여 빠른 암호화가 가능하다. 해당 기법을 활용하여 RISC-V 상에서 블록당 1,283 Cycle의 성능을 가지며, 기존 대비 9%의 성능 향상을 확인할 수 있다.

I. 서론

DES의 암호화 강도가 점점 약해지면서 NIST(National Institute of Standards and Technology)에서 새로운 암호화 알고리즘을 공모하여 채택된 AES[1](Advanced Encryption Standard)는 벨기에 암호학자에 의해 개발된 Rijndael 알고리즘이다. 전 세계적으로 오랫동안 사용되어 온 암호로 연구 또한 활발하게 이루어졌다.

Bitsliced AES[2]는 함수의 계산이 논리 게이트(AND, XOR, OR 등)로 축소되어 CPU의 레지스터 폭만큼 많은 인스턴스를 병렬로 실행할 수 있도록 하는 소프트웨어 구현 기법이다. RISC-V에서 가장 효율적인 구현은 Schwabe와 Stoffelen[3]의 124cpb이다.

Fixslicing AES[4]는 기존 Bitslice로 구현된 AES에서 선형 계층에 필요한 작업량을 이론상으로 52%까지 줄인 기법으로 RISC-V 프로세서 상에서 87cpb의 성능을 보이며 Bitslice 구현보다 30%의 성능 향상을 보여준다.

본 논문에서는 사전 연산 테이블을 활용한 Fixslicing AES-CTR 구현을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 CTR 운용모드, Fixslicing AES, 그리고 RISC-V 프로세서에 관해 설명한다. 3장에서는 구현 기법에 관해 설명한다. 4장에서는 성능 평가를 진행한다. 마지막으로 5장에서는 결론을 내린다.

II. 관련 연구

2.1 CTR 운용모드

CTR(Counter) 모드는 이미 1979년 Diffie와 Hellman[5]에 의해 도입되었다. CTR 모드는 블록 암호를 스트림 암호로 바꾼다. 기존 평문을 암호화 하는 것이 아닌 Nonce와 Count 값을 암호화하여 나온 값을 평문과 XOR을 하는 방식으로 암호화가 진행된다.

CTR 모드의 장점은 소프트웨어와 하드웨어에 효율적인 구현이 가능하며, 스트림 암호의

특성을 활용하여 Count 값을 통해 특정 부분만 복호화를 할 수 있는 장점이 있다. 또한 Nonce 값이 고정된 특징을 활용하여 사전 연산을 통한 최적화가 가능하다.

2.2 Fixslicing AES

Bitsliced AES 기법은 RISC-V 프로세서 상에서 Shiftrows 당 47.5cpb가 발생하며 전체 성능의 38%에 해당한다. Fixslice AES는 이러한 선형 계층의 비용을 최소화하기 위해 Shiftrows 단계를 생략하고 여러 라운드에서 대체 표현을 제안했다. 주로 레지스터 내의 비트를 고정하여 움직이지 않도록 하고 적절한 비트가 Subbytes 작업에 포함되도록 다른 슬라이스를 조정하기 때문에 Fixslice라고 한다. 이러한 조정은 Mixcolumns 단계에서 4가지의 Mixcolumns 로 조정하게끔 구현되었다. Bitslice 기법에 비해 코드량이 증가하는 단점이 존재하지만, 이론적으로 연산량이 56% 감소하는 장점이 있다.

기존 AES와 키 길이(128, 192, 256), 라운드 수(10, 12, 14)는 동일하다. 하지만 블록 길이는 기존 128-bit에서 256-bit로 2개의 블록을 병렬 연산한다. 전체적인 암호화 과정은 [그림 1]과 같다. 마지막 라운드에서는 Mixcolumns 대신 Double_shiftrows 함수가 연산된다.

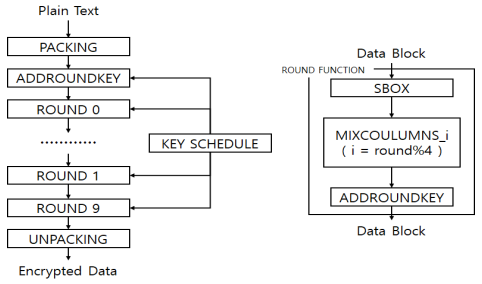

Fig 1. Fixslicing AES Algorithm

2.3 RISC-V 프로세서

본 논문에서 대상으로한 RISC-V 프로세서중 RV32I는 32-bit 구조로 32개의 32-bit 레지스터를 제공한다. RISC-V 프로세서의 레지스터 용도는 [표 1]과 같다[6].

zero 레지스터는 항상 0의 값을 갖는 레지스터이다. stack pointer(sp) 레지스터는 스택 주소를 갖고 있으며, 스택을 활용할 때 사용하는 레지스터이다. Callee saved 레지스터(sp, s0~s11)는 사용하기 이전의 값을 보존해야 하는 레지스터 이다.

Description	Register	Saver
zero register	zero(x0)	
return address	ra(x1)	
stack pointer	sp(x2)	callee
global pointer	gp(x3)	
thread pointer	tp(x4)	
saved registers	s0~s11	callee
function arguments and return value	a0~a7	
temporal registers	t0~t6	

Table 1. Register in RISC-V

III. 구현 기법

3.1 사전 연산 테이블

사전 연산 테이블 생성 과정은 [그림 2]과 같다. IV(Initalization Vextor)가 다음과 같을 때, Fixslice AES 구현에서 2round sbox 까지 진행 후 결과를 확인해보면, 4-byte의 counter를 1-byte 단위로 볼 경우 각 byte마다 영향을 주는 state가 다르다는 것을 알 수 있다. 예를 들어 s[3]의 값이 1이 증가하였을 때, 2 round sbox 진행 후의 결과에서 s[3]과 같은 색이 아닌 state의 값은 증가하기 전의 값과 동일하다.

1-byte의 각 counter(S0~S3)는 0~255의 값을 가질 수 있으며, 해당 값을 Index로 사용하여 Pre-table에 값을 저장하거나 불러올 때 사용한다. 이때 Pre-table에 저장하기 위해 Inv_Shiftrows를 진행하여 일렬로 정렬한 후 table에 저장한다. 또한 테이블에 저장된 값을 불러와 state를 생성할 때는 값을 불러온 후 Shiftrows를 한번 진행하고 이후 2 round Mixcolumns를 진행한다.

결과적으로 Pre-table은 4KB의 메모리를 사용하며, Inv_Shiftrows와 Shiftrows의 연산이 추가되지만 Mixcolumns 1번, Addroundkey 2번, Sbox 2번의 연산을 생략할 수 있다.

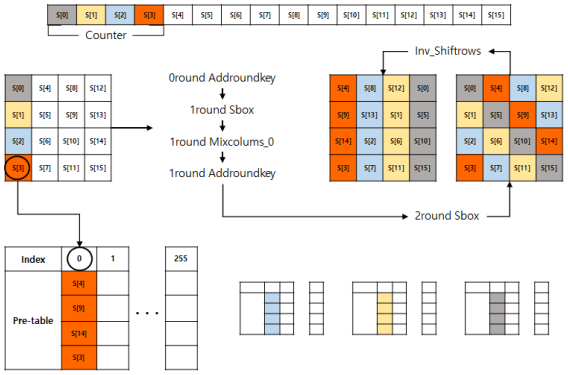

Fig 2. Pre-Calculate Table Structure

3.2 구현 코드

Input : a3 (Counter value),
a4 (LTU address)
Output : s0, s1, s2, s3, s4, s5, s6, s7
(intermediate value)

```
1. li t1, 0xff00
2. andi t0, a3, 0xff
3. and t1, t1, a3
4. srli 1t1, t1, 8
5. srli t3, a3, 16
6. andi t2, t3, 0xff
7. srli t3, t3, 8
8. add t0, t0, a4
9. add t1, t1, a4
10. add t2, t2, a2
11. add t3, t3, a2
12. lbu s0, 1024(t3)
13. lbu a3, 0(t2)
14. slli a3, a3, 8
15. xor s0, s0, a3
16. lbu a3, 1024(t1)
17. slli a3, a3, 16
18. xor s0, s0, a3
19. lbu a3, 0(t0)
20. slli a3, a3, 24
21. xor s0, s0, a3
22. lbu s2, 1(t0)
23. lbu a3, 1025(t3)
24. slli a3, a3, 8
25. xor s2, s2, a3
26. lbu a3, 1(t2)
27. slli a3, a3, 16
28. xor s2, s2, a3
29. lbu a3, 1025(t1)
30. slli a3, a3, 24
31. xor s2, s2, a3
32. lbu s4, 1026(t1)
33. lbu a3, 2(t0)
34. slli a3, a3, 8
35. xor s4, s4, a3
36. lbu a3, 1026(t3)
37. slli a3, a3, 16
38. xor s4, s4, a3
39. lbu a3, 2(t2)
40. slli a3, a3, 24
41. xor s4, s4, a3
39. lbu s6, 3(t2)
40. lbu a3, 1027(t1)
41. slli a3, a3, 8
42. xor s6, s6, a3
43. lbu a3, 3(t0)
44. slli a3, a3, 16
45. xor s6, s6, a3
46. lbu a3, 1027(t3)
47. slli a3, a3, 24
48. xor s6, s6, a3
49. mv s1, s0
50. mv s3, s2
51. mv s5, s4
52. mv s7, s6
53. li a3, 0x00ffffff
54. and s1, s1, a3
55. li a3, 0xffffff00
56. and s3, s3, a3
57. li a3, 0xffff00ff
58. and s5, s5, a3
59. li a3, 0xff00ffff
60. and s7, s7, a3
61. lbu a3, 4(t0)
62. slli a3, a3, 24
63. xor s1, s1, a3
64. lbu a3, 5(t0)
65. xor s3, s3, a3
66. lbu a3, 6(t0)
67. slli a3, a3, 8
68. xor s5, s5, a3
69. lbu a3, 7(t0)
70. slli a3, a3, 16
71. xor s7, s7, a3
```

Table 3. Code to call Pre-table values on RISC-V

사전 연산 테이블에 저장된 값을 가져와 State를 생성하는 코드이다. State를 생성할 때 레지스터에 Load 후 Shiftrows 하는 방식이 아닌, LBU 명령어를 활용하여 Byte 단위로 Load 후 SLLI, SRLI, XOR 명령어를 활용하여 위치를 조정한다.

Fixslicing AES는 2개의 블록이 병렬 연산되기 때문에 두 번째 블록은 첫 번째 블록에서 count가 1증가한 차이만 존재한다. 즉 [그림 2]에서 S[3]의 값만 1 증가하기 때문에 모든 값을 불러오지 않고 MV 명령어를 통해 첫 번째 블록의 값을 복사하여 해당 state가 영향을 주는 state의 값만 테이블에서 불러와 변경해 준다. 이에 해당하는 코드는 [표 3]의 49줄 이후에 해당한다.

IV. 성능 평가

RISC-V는 SiFive사의 HiFive1 Rev B 플랫폼이다. 4MB의 Quad SPI 플레시 메모리, E31 코어가 탑재되어 있어 320MHz로 연산이 수행된다. 성능 비교는 CTR모드로 동작한 128비트 2블록을 암호화 할 때의 평균 사이클을 측정하였다.

[표 4]는 제안하는 기법을 활용한 Fixslicing AES을 CTR모드로 동작시켰을 때의 cpb(Cycle Per Block)를 측정하여 기존의 Fixslicing의 cpb와 비교한 결과이다. 결과적으로 RISC-V 상에서 9%의 성능 향상을 확인하였다.

	[4]	Ours
cpb(Cycle per block)	1,398	1,283

Table 4. Performance result

V. 결론

본 논문에서는 사전 연산 테이블을 활용하여 Fixslicing AES의 CTR모드 구현을 제안하였다. 사전 연산은 2라운드의 Sbox까지 진행하기 때문에 암호화 과정에서 해당 연산을 생략하는 기법을 시도한다.

제안하는 기법을 통해 기존 성능 대비

RISC-V상에서 9%의 성능 향상을 확인할 수 있었다. 향후 과제로 다양한 운용모드의 효율적인 최적화 구현을 제안한다.

VI. 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, 50%) 그리고 이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 25%) 그리고 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00540, GPU/ASIC 기반 암호알고리즘 고속화 설계 및 구현 기술개발, 25%).

[참고문헌]

[1] Joan Daemen and Vincent Rijmen. “The Design of Rijndael: AES - The Advanced Encryption Standard.” Information Security and Cryptography. Springer, 2002.

[2] Chester RebeiroDavid SelvakumarA. S. L. Devi. “Bitslice Implementation of AES”. CANS 2006: Cryptology and Network Security pp 203-212. 2006.

[3] Peter Schwabe and Ko Stoffelen. All the AES You Need on Cortex-M3 and M4. In Selected Areas in Cryptography - SAC 2016, pages 180-194, 2016.

[4] Adomnicai, A., and Peyrin, T. “Fixslicing AES-like Ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V”. IACR Transactions on Cryptographic Hardware and Embedded Systems, 402-425. 2021. https://doi.org/10.46586/tches.v2021.i1.402-425

[5] WHITFIELD DIFFIE and MARTIN HELLMAN. Privacy and Authentication: An Introduction to Cryptography. Proceedings of the IEEE, 67 (1979), pp. 397-427.

[6] SiFive, Inc. “The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2”. May 7, 2017. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf