석사학위논문

ARM 프로세서 상에서의 블록 암호 SEED CTR 모드 최적 구현

2025년

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

송 민 호

석 사 학 위 논 문
지도교수 서화정

ARM 프로세서 상에서의 블록 암호 SEED CTR 모드 최적 구현

Optimized Implementation of block cipher SEED CTR mode on ARM processors

2024년 12월 13일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

송 민 호

석 사 학 위 논 문
지도교수 서화정

ARM 프로세서 상에서의 블록 암호 SEED CTR 모드 최적 구현

Optimized Implementation of block cipher SEED CTR mode on ARM processors

위 논문을 공학 석사학위 논문으로 제출함

2024년 12월 13일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

송 민 호

송민호의 공학 석사학위 논문을 인준함

2024년 12월 13일

심사위원장 박명서 (인)

심 사 위 원 이웅희 (인)

심 사 위 원 서화정 (인)

국 문 초 록

ARM 프로세서 상에서의 블록암호 SEED CTR 모드 최적 구현

한 성 대 학 교 대 학 원
융 합 보 안 학 과
융 합 보 안 전 공
송 민 호

블록 암호는 입력되는 데이터를 고정된 크기의 블록 단위로 암호화하는 암호화 알고리즘이다. 해당 알고리즘은 동일한 키를 사용하는 대칭키 암호화 방식으로 데이터를 암호화하고 복호화하며 다양한 운영 모드를 가지고 있다. 본 논문에서는 국내에서 다방면으로 사용되고 있는 SEED 블록 암호의 CTR 모드를 ARM 프로세서 상에서의 최적 구현을 진행한다.

ARM 프로세서는 모바일 기기 등 다양한 저전력, 고성능 환경에서 활용된다. CTR 모드는 병렬 처리가 가능하여 고속 암호화에 유리하지만 ARM 프로세서의 제한된 자원에서 성능을 보여주기 위해서는 적절한 기법이 필요하다. 본 연구에서는 ARM 프로세서의 NEON SIMD 명령어를 활용하여 병렬 연산을 진행하며 레지스터 활용을 최적화하여 연산 속도를 개선하였다. SEED의 레퍼런스와 비교하였을 때 본 논문의 연구는 15% 정도의 성능을 향상시켰다.

주요어: SEED 블록암호, CTR 모드, 최적화 구현

목 차

표 목 차

그 림 목 차

제 1 장 서 론

현대 정보 사회에서 데이터 보안은 매우 중요한 요소이며 이를 위한 다양한 암호화 기법이 개발되고 있다. 블록 암호는 이러한 보안 기술 중 핵심적인 역할을 하는 방식으로 주로 금융, 통신, 정부 등에서 민감한 데이터를 보호하는 데 사용된다. 그중에서도 SEED 암호는 대한민국에서 개발된 국가 표준 암호화 알고리즘으로 128비트의 대칭 키를 사용하는 블록 암호이다. SEED는 특히 금융권과 공공 기관에서 널리 사용되고 있으며 높은 보안성과 효율성을 제공한다.

암호화 기술의 실제 적용에서 중요한 과제 중 하나는 암호화 알고리즘을 특정 하드웨어 플랫폼에 최적화하여 구현하는 것이다. 하드웨어에 따라 성능 요구 사항과 자원 제약이 달라지기 때문에 이를 고려한 최적화는 매우 중요하다. 최근 모바일 기기와 사물인터넷(IoT) 장치의 보급이 급격히 확산됨에 따라 저전력 및 고효율의 연산 성능을 요구하는 환경에서 SEED와 같은 블록 암호의 효율적인 구현이 필수적이다. 이러한 장치에서 널리 사용되는 프로세서 중 하나가 ARM 프로세서이다.

ARM(Advanced RISC Machines) 프로세서는 저전력과 고효율을 목표로 설계된 RISC(Reduced Instruction Set Computing) 아키텍처를 기반으로 하며 주로 모바일 기기, 임베디드 시스템, IoT 기기에서 많이 사용되고 있다. ARM 프로세서의 경량화된 구조와 낮은 전력 소모 특성은 블록 암호 알고리즘을 실행할 때도 상당한 장점을 제공한다. 그러나 암호화 알고리즘의 효율성을 극대화하기 위해서는 ARM 아키텍처에 맞는 최적화가 필요하다. 특히 ARM 프로세서는 병렬 처리 및 파이프라이닝과 같은 기능을 제공하므로 이러한 특성을 잘 활용하면 암호화 성능을 크게 향상시킬 수 있다.

본 논문에서는 ARM 프로세서 상에서 블록 암호 SEED를 CTR(Counter) 모드로 구현하고 이를 최적화하는 방법에 대해 탐구한다. CTR 모드는 블록 암호를 스트림 암호처럼 사용할 수 있게 하는 운영 모드이며 병렬 처리에 적합한 구조를 가지고 있다. 따라서 ARM의 병렬 처리 기능을 최대한 활용할

수 있다. CTR 모드에서는 각 암호화 블록이 독립적으로 처리될 수 있기 때문에 이는 ARM 프로세서의 성능을 최대로 끌어올리기 위한 좋은 기회가 될 수 있다.

제 2 장 관 련 연 구

제 1 절 SEED 블록 암호

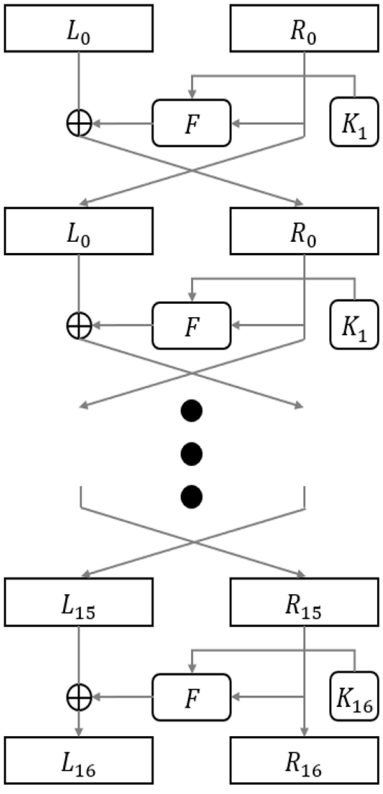
SEED는 한국인터넷진흥원(KISA, Korea Internet & Security Agency)에서 개발한 블록 암호로, 1999년에 대한민국 국가 표준으로 지정되었다. 이 암호 알고리즘은 한국에서 개발된 암호화 알고리즘인 만큼 한국의 주요 금융 시스템과 전자 정부 시스템에서 널리 사용되고 있다. 대표적인 적용 사례로는 인터넷 뱅킹, 공공 기관의 보안 통신, VPN(Virtual Private Network) 등의 네트워크 보안 서비스 등이 있다. 특히 인터넷 뱅킹에서 SEED는 오랜 기간 동안 중요한 역할을 해왔으며 사용자의 금융 데이터를 보호하는 데 기여하고 있다.

비록 SEED가 주로 한국에서 사용되는 암호화 알고리즘이지만 2010년에는 IETF(Internet Engineering Task Force)에서 RFC 4269로 지정되어 국제적으로도 표준으로 인정받았다. 또한 SEED는 TLS(Transport Layer Security) 프로토콜에서 선택 가능한 암호화 알고리즘 중 하나로 포함되어 국제적인 보안 프로토콜에서도 사용될 수 있는 환경을 갖추고 있다.

SEED 알고리즘은 라운드 함수에서 Sbox 연산과 순열을 사용하여 입력 데이터를 복잡하게 변형시킨다. SBox는 입력 값을 비선형적으로 대체하는 역할을 하고 순열은 입력 비트들을 복잡하게 뒤섞어 예측 가능성을 줄인다. 이러한 과정은 SEED의 보안성을 높이는 데 중요한 역할을 한다. 또한 SEED는 다양한 암호 분석 기법에 대해 강력한 저항성을 보인다. 특히 차분 암호 분석 및 선형 암호 분석과 같은 일반적인 공격 방식에 대해 강한 보안성을 갖고 있다. 이로 인해 대한민국 내에서 신뢰받는 표준 암호로 자리 잡았다.

SEED는 128비트 크기의 평문 블록과 128비트 크기의 대칭 키를 사용하여 총 16라운드 연산을 진행하여 128비트 크기의 암호문 블록을 출력한다. 대칭키 암호는 암호화와 복호화 과정에 동일한 키를 사용한다는 특징이 있다. SEED는 Feistel 구조로 이루어져 있어 128비트 크기의 평문 블록을 분할하여

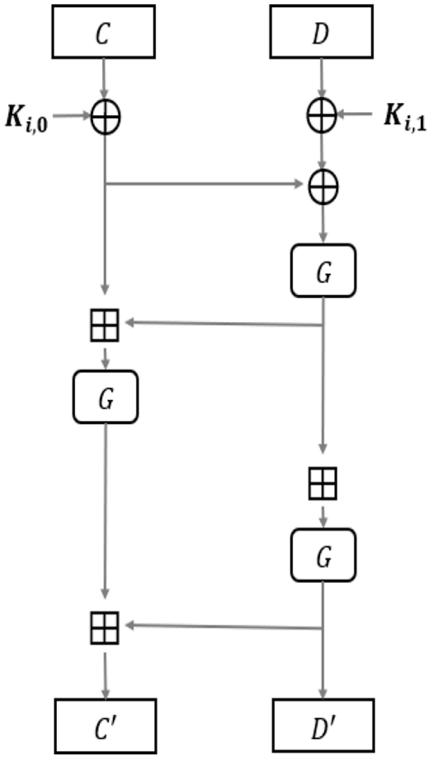
연산을 진행한다. Feistel 구조는 암호화와 복호화 과정이 유사하게 작동하기 때문에 구현이 간편하고 효율적이라는 특징이 있다. SEED의 전체 알고리즘 구조는 [그림 2-1]과 같다.



[그림 2-1] SEED 블록 암호 알고리즘의 전체 구조

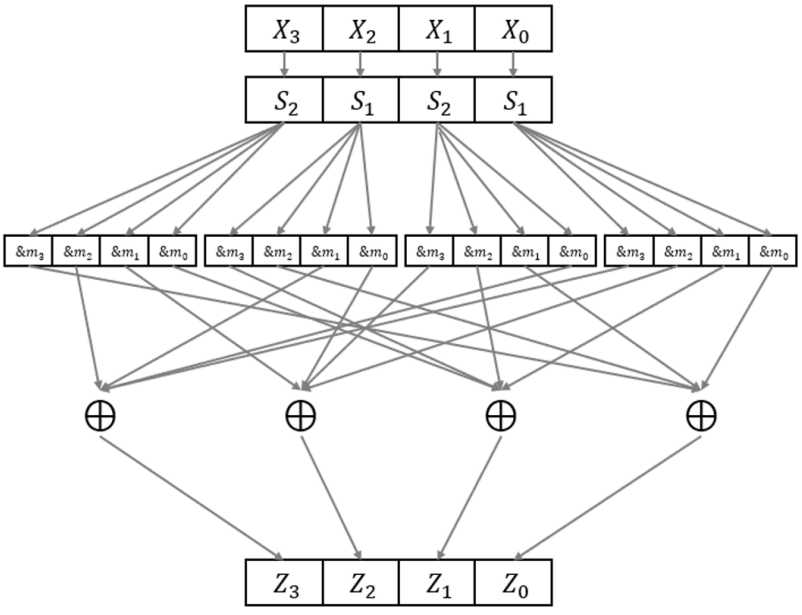
Feistel 구조를 가진 블록 암호 알고리즘은 F 함수의 특성에 따라 분류할 수 있다. SEED의 F 함수는 수정된 64비트 Feistel 형태로 구성되어 있다. F 함수는 두 개의 32비트 블록(C, D)을 입력으로 받는다. 입력받은 두 블록은 라운드 키 연산, XOR 연산, G 함수 연산 등의 과정을 거쳐 두 개의 32비트 블록(C', D')을 출력한다. 즉, 암호화 과정에서 64비트 블록(C, D)과 64비트 라운드 키 $K_i = (K_{i,0};K_{i,1})$이 F 함수의 입력으로 처리된다. F 함수의 전체 구조

는 [그림 2-2]와 같다.



[그림 2-2] F 함수의 전체 구조

F 함수 내부에는 G 함수 연산이 존재한다. G 함수는 32비트 입력 값을 받아 32 비트 출력 값을 생성한다. 입력받은 값은 두 개의 서로 다른 비선형 Sbox S_1, S_2를 통해 변환된다. 이후 변환된 값은 상수 m_0, m_1, m_2, m_3과 연산을 진행하여 출력 값 Z_0, Z_1, Z_2, Z_3을 얻는다. 상수 값과의 연산은 일종의 순열에 해당하며 G 함수의 전체 구조는 [그림 2-3]과 같다.



[그림 2-3] G 함수의 전체 구조

제 2 절 Counter 운영 모드

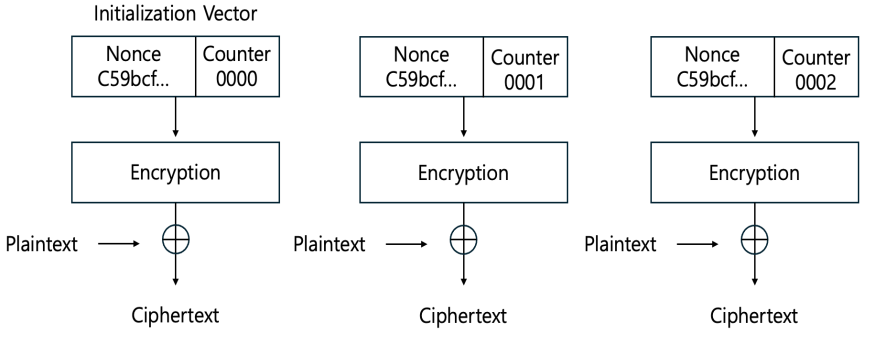
블록 암호 운영 모드는 블록 암호 알고리즘을 다양한 방식으로 적용해 데이터를 암호화하는 방법을 말한다. 블록 암호는 고정된 크기의 데이터 블록을 처리하지만 실제 응용에서 처리할 데이터는 크기가 다양하므로 여러 운영 모드가 필요하다. 대표적인 모드로는 ECB, CBC, CFB, OFB, CTR 등이 있으며 각 모드는 보안성, 처리 속도, 병렬 처리 가능 여부 등에서 차이를 보인다. 운영 모드는 암호화뿐만 아니라 복호화 및 데이터 무결성도 관리할 수 있다. 상황에 따라 적절한 모드를 선택해 보안성을 높일 수 있다.

이 중 Counter 모드(CTR, Counter Mode)는 대칭 키 암호에서 널리 사용되는 블록 암호 운영 모드이다. 이 운영 모드는 블록 암호를 스트림 암호처럼 사용할 수 있게 하며 병렬 처리가 가능해 성능 향상이 가능하다는 점에서 주목받고 있다. 특히 대용량 데이터의 암호화에 적합하고, 네트워크 프로토콜,

저장소 보안, VPN 등에서 많이 사용된다.

CTR 모드에서의 암호화는 Counter라고 불리는 변수 값을 초기화하여 시작된다. 이 Counter는 임의의 초기 값을 가지며 고정된 상수 Nonce값과 조합한 값이 입력 값이다. 조합하여 생성된 값은 Initialization Vector(IV, Counter+Nonce)라고 나타낸다.

평문 데이터는 블록 단위로 처리되며 각각의 블록마다 Counter 값을 암호화한다. 블록 암호의 키와 함께 Counter 값을 암호화하면 암호화된 카운터 값이 생성되며 이를 평문 블록과 XOR 연산을 통해 암호문 블록을 생성한다. 각 블록을 처리할 때마다 카운터 값이 증가한다. 이로 인해 각 블록마다 다른 카운터 값을 사용하므로 동일한 키를 사용해도 서로 다른 암호문이 생성된다. Counter 운용 모드의 구조는 [그림 2-3]과 같다.



[그림 2-4] Counter 운영 모드의 구조

제 3 절 ARM 프로세서

ARM(Advanced RISC Machines) 프로세서는 현재 전 세계에서 가장 널리 사용되는 프로세서 아키텍처 중 하나로 저전력 고효율 설계가 특징이다. ARM 프로세서는 주로 스마트폰, 태블릿, 임베디드 시스템, IoT 기기, 그리고

서버 등 다양한 장치에 사용되며, 그 효율성과 유연성 덕분에 폭넓은 분야에서 중요한 역할을 하고 있다. 이 프로세서는 RISC(Reduced Instruction Set Computing) 아키텍처를 기반으로 하며, 단순하고 효율적인 명령어 집합을 통해 빠른 처리 속도와 낮은 전력 소모를 실현한다.

2011년 10월에 발표된 ARMv8 아키텍처는 ARM 최초의 64비트 임베디드 아키텍처이다. 32비트 모드(AArch32)와 A64 명령어 세트를 갖춘 새로운 64비트 모드(AArch64)를 모두 지원하여 성능과 확장성을 개선했다. 기존 32비트 명령어 세트(A32/T32)와 함께 사용할 수 있으며 더 큰 주소 공간과 향상된 계산 성능을 제공한다. 31개의 범용 레지스터가 있으며, 여기서 x0-x30은 64비트 연산에 사용되고 w0-w30은 32비트 연산에 사용된다. ARMv8에는 32개의 128비트 벡터 레지스터(v0-v31)도 포함된다. 이 레지스터 구조는 데이터 처리 효율성을 크게 개선하고 복잡한 연산을 더 빠르게 수행할 수 있게 한다.

또한 ARMv8 아키텍처는 포인터 인증(PA) 기능을 추가하여 메모리 안전성을 강화한다. 이 기능은 메모리 취약성에 대한 저항력을 높여 시스템 보안을 개선한다. 이러한 다양한 이유로 이 아키텍처는 스마트폰 산업에 상당한 영향을 미쳤으며 다양한 노트북 및 기타 기기에도 널리 채택되었다.

ARM 프로세서에서는 다양한 명령어 세트를 제공하고 있다. 본 논문에서 사용되는 명령어는 [표 2-1]과 같다. [표 2-1]에서 Xn은 소스 스칼라 레지스터를 나타내고, Vd는 대상 벡터 레지스터를 나타내고, Vt는 전송된 벡터 레지스터를 나타내고, Vn 및 Vm은 소스 벡터 레지스터를 나타낸다.

asm	Operands	Description	Operation
EOR	Vd, Vn, Vm	Bitwise exclusive OR	Vd←Vn⊕Vm
SUB	Vd, Vn, Vm	Subtract	Vd←Vn−Vm
ADD	Vd, Vn, Vm	Addition	Vd←Vn+Vm
LD	Vt, (Xn)	Load	Vt←(Xn)
ST	Vt, (Xn)	Store	(Xn)←Vt
MOVI	Vt, #imm	Move immediate	Vt←#imm
TBL	Vd, Vn, Vm	Table vector lookup	Vd←Vn[Vm]
TBX	Vd, Vn, Vm	Table vector lookup extension	Vd←Vn[Vm]
TRN1	Vd, Vn, Vm	Transpose vectors (primary)	Vd←Vn[Vm]
TRN2	Vd, Vn, Vm	Transpose vectors (secondary)	Vd←Vn[Vm]

[표 2-1] ARM 프로세서의 명령어

제 3 장 구 현 기 법

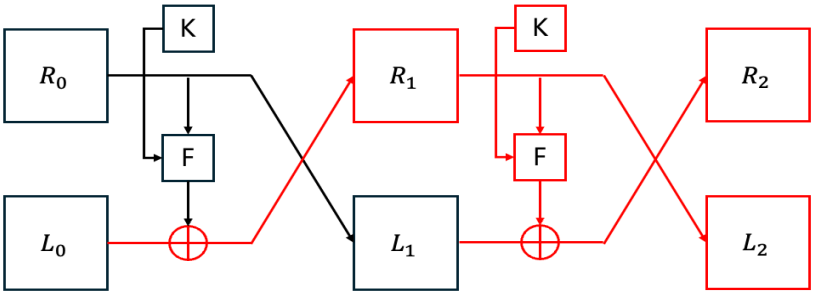
제 1 절 사전 연산 기법

CTR(Counter) 모드에서 사전 연산 기법은 암호화 또는 복호화 과정의 성능을 향상시키기 위해 사용되는 방법 중 하나이다. 이 기법은 암호화 연산의 일부를 미리 계산하여 저장해 둠으로써 실제 암호화 과정에서 계산 시간을 단축하는 방식이다. 해당 방법이 가능한 이유는 CTR 모드에서 고정된 상수 값 Nonce를 사용하기 때문이다. Nonce를 사용하여 다른 고정 값들과 연산을 진행하면 동일한 연산 결과가 나오게 된다. 반면에 Counter 값은 변하기 때문에 다른 값이 나오게 된다.

사전 연산 기법을 활용하기 위해서는 먼저 사용하는 블록 암호의 암호화 진행 과정을 알아봐야 한다. SEED 암호는 128비트의 평문 블록을 입력으로 받는다. CTR모드에서는 평문 블록 대신 128비트의 IV 값을 입력으로 받는데 이 때 IV 값은 96비트의 Nonce 값과 32비트의 Counter 값으로 나눌 수 있다. SEED 암호는 Feistel 구조를 가지고 있어 라운드 연산을 진행할 때 블록을 2개로 나누어 진행한다. 각각의 블록은 64비트이기 때문에 한 블록은 Nonce 값으로만 이루어지고 다른 블록은 Counter 값이 포함된 블록이 된다. 1라운드 연산을 진행할 때 Nonce 값으로 이루어진 블록을 복잡한 F 함수 연산을 수행하는 블록으로 지정한다. 그럼 고정된 값의 연산이라 고정된 값이 나오기 때문에 연산을 할 필요 없이 사전 연산한 값을 저장하고 그 값을 따로 불러서 연산을 진행할 수 있다. 그렇게 되면 1라운드의 F 함수 연산을 생략할 수 있다.

[그림 3-1]에서 색이 칠해진 연산 과정이 Counter에 의해 영향을 받는 부분이다. L_0에는 Nonce 값과 Counter 값이 들어가고 R_0에는 Nonce 값만이 들어간다. L_0는 Counter 값이 존재하여 초기 연산부터 값이 달라 사전 연산을 진행할 수 없다. 하지만 R_0에는 Counter 값이 들어가지 않아 Counter 값에 영향을 받기 전인 F 함수 연산까지의 값을 저장하여 사용할 수 있다. 이

러한 사전 연산 방식을 통해 성능을 향상시킬 수 있다.



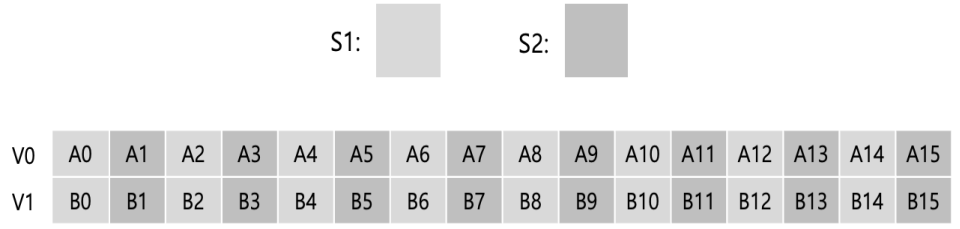
[그림 3-1] Counter 값이 연산에 영향을 주는 과정

제 2 절 벡터 레지스터 내부 정렬

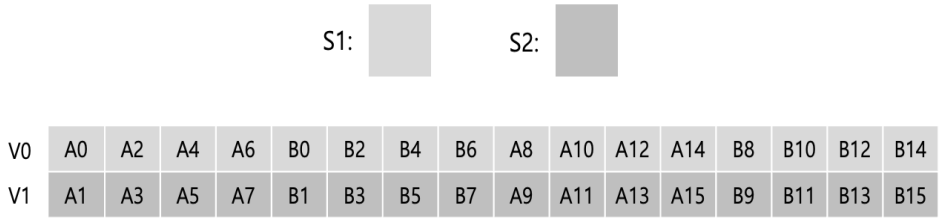
처음 로드를 통해 평문 값을 레지스터에 등록하면 [그림 3-2]와 같은 형태가 된다. 한 평문이 한 레지스터에 등록되어 있는 상태이며 총 2개의 평문 블록을 불러온 상태이다. SEED 블록 암호는 서로 다른 두 개의 Sbox를 사용하는데 이렇게 되면 한 레지스터에서 Sbox S1과 S2 두 개를 사용해서 연산을 진행해야한다. 하지만 TBL 명령어를 통해 효율적으로 Sbox 연산을 진행하려면 해당 레지스터 상태로는 불가능하다. TBL 명령어는 벡터 룩업 테이블을 조회하여 작업을 수행하는 명령어이다. 이 명령어는 특정 인덱스에 따라 데이터를 조회하고 재구성하는데 사용되는데 Sbox를 룩업 테이블에 저장하고 해당 명령어를 사용하면 효율적인 연산이 가능하다. 하지만 TBL 명령어를 사용하기 위해서는 연산에 필요한 인덱스가 동일한 레지스터에 존재해야 한다. 즉 Sbox S1을 사용하는 인덱스들은 모두 같은 한 레지스터에 존재해야하고 Sbox S2를 사용하는 인덱스들도 마찬가지이다.

같은 Sbox를 사용하는 인덱스들은 정렬을 통해 한 레지스터에 배치시킬 수 있다. 이 때 TRN, UZP 등의 명령어를 사용할 수 있다. TRN 명령어는 두 벡터의 데이터를 교차하여 새 벡터를 생성하고 UZP 명령어는 두 벡터의 데

이터를 분리하여 새 벡터를 생성한다. 해당 명령어를 통해 정렬된 레지스터의 상태는 [그림 3-3]과 같다.



[그림 3-2] 초기 레지스터 상태



[그림 3-3] 정렬 후 레지스터 상태

[표 3-1]은 [그림 3-2]의 상태에서 [그림 3-3]의 상태로 만드는 과정의 레지스터 정렬 코드이다. TRN 명령어를 통해 먼저 v0-v3의 레지스터 상태를 바꾸어 비어있는 레지스터인 v28-v31에 저장해둔다. 이후 UZP 명령어를 통해 연산을 진행하면 같은 Sbox를 사용하는 인덱스를 하나의 레지스터에 정렬할 수 있다.

v0-v3 : PT v28-v31 : Temp	
1	trn1.16b v28, v0, v2
2	trn1.16b v29, v1, v3
3	trn2.16b v30, v0, v2
4	trn2.16b v31, v1, v3
5	uzp1.8h v0, v28, v28
6	uzp1.8h v1, v29, v29
7	uzp2.8h v2, v30, v30
8	uzp2.8h v3, v31, v31

[표 3-1] 벡터 레지스터 정렬 코드

제 3 절 TBL 명령어를 통한 Sbox 연산 최적화

Sbox 연산은 TBL 및 TBX 명령어를 사용하여 쉽고 효율적으로 수행할 수 있다. TBL 및 TBX 명령어는 벡터 룩업 테이블을 조회하여 작업을 수행한다. 먼저 Sbox가 저장되어있는 룩업 테이블을 불러와준다. SEED 암호는 서로 다른 8비트 Sbox 2개를 사용하고 각각의 인덱스의 수는 256개이므로 v16-v31 레지스터를 사용하여 저장해준다. 이후 저장된 Sbox를 각각 조회하여 연산을 수행한다.

TBL 명령어는 먼저 Vn 레지스터에 저장된 벡터 값을 읽고 해당 값을 Vm 레지스터의 인덱스로 사용한다. 그런 다음 Vm의 해당 인덱스에 저장된 값을 Vd에 저장한다. Sbox는 Vm에 저장되므로 해당 명령어를 사용하면 효율적인 연산이 가능하다. TBL 명령어를 사용하여 진행한 Sbox 연산 과정은 [표 3-2]와 같다.

v16-v31 : Sbox
v15 : 0x40
v12-v14 : Temp

```
        .macro sbox reg, address
1       ld1.16b {v16-v19}, [\address], #64
2       ld1.16b {v20-v23}, [\address], #64
3       ld1.16b {v24-v27}, [\address], #64
4       ld1.16b {v28-v31}, [\address]
5       sub \address, \address, #192
6       sub.16b v14, ₩reg, v15
7       tbl.16b ₩reg, {v16-v19}, ₩reg
8       sub.16b v13, v14, v15
9       tbx.16b ₩reg, {v20-v23}, v14
10      sub.16b v12, v13, v15
11      tbx.16b ₩reg, {v24-v27}, v13
12      tbx.16b ₩reg, {v28-v31}, v12
        .endm
```

[표 3-2] TBL 명령어를 사용한 Sbox 연산

제 4 절 G 함수 최적화

S-box 연산을 수행한 후에는 상수 m_0, m_1, m_2, m_3과 연산을 진행한다. 상수 값을 레지스터에 따로 저장하여 쓰지 않고 지속적으로 로드해서 사용하면 많은 성능 저하를 일으킬 수 있다. 이에 빈 레지스터에 상수 값을 저장하여 사용한다.

저장된 상수 값은 바로 사용할 수 없다. 앞서 평문 블록이 들어가는 벡터 레지스터를 정렬했듯이 이에 맞춰 상수 값도 정렬을 해줘야 한다. 평문 레지스터와 정렬을 맞추려면 v8 레지스터에 $m0$, $m2$ 값이 있어야 하고, v9 레지스터에 $m1$, $m3$ 값이 있어야 하고, v10 레지스터에 $m2$, $m0$ 값이 있어야 하고, v11 레지스터에 $m3$, $m1$ 값이 있어야 한다. m_0, m_1, m_2, m_3를 각각의 빈 레지스터에 할당하고 TRN 명령어를 사용하여 정렬하여 이후의 계산을 용이하게 한다. 상수 값을 저장하고 정렬하는 과정은 [표 3-3]에서 확인할 수 있다.

v8-v11 : m_0, m_1, m_2, m_3	
v12 : Temp	
m_0, m_1, m_2, m_3 : 0xFC, 0xF3, 0xCF, 0x3F	
1	movi v8.16b, #0xFC
2	movi v9.16b, #0xF3
3	movi v10.16b, #0xCF
4	movi v11.16b, #0x3F
5	trn1.16b v12, v8, v10
6	mov v8.16b, v12.16b
7	trn1.16b v12, v9, v11
8	mov v9.16b, v12.16b
9	trn1.16b v12, v10, v8
10	mov v10.16b, v12.16b
11	trn1.16b v12, v11, v9
12	mov v11.16b, v12.16b

[표 3-3] 상수 값 저장 및 정렬

제 4 장 성 능 평 가

본 논문에서는 ARM 프로세서 상에서 성능 측정을 진행한다. Apple M2 칩이 장착된 ARMv8 아키텍처 상에서 진행되며 최적화 옵션은 -O3로 설정되어있다. 본 연구와 레퍼런스 코드 구현을 같은 환경에서 성능을 측정해보고 비교한다.

성능 측정은 암호화 연산이 동작할 때의 Cycle을 측정하여 비교한다. 측정의 기준은 암호화 함수를 10,000번 반복시켜 측정된 Cycle의 평균 값을 사용한다. 성능 측정 결과는 [표 4-1]과 같다.

Imple.	parallel	Cycles
Reference C	1-PT	896
This work	2-PT	1,550

[표 4-1] 본 연구와 레퍼런스 구현의 성능 측정 결과

기존의 레퍼런스 구현의 경우 896 Cycle의 성능을 보여주고 있고 본 논문의 연구는 1,550 Cycle의 성능을 보여주고 있다. 본 논문에서 제안하는 기법은 2개의 평문 블록에 대한 병렬 구현을 진행하여 레퍼런스 구현과 비교하여 약 15%의 성능 향상을 보여주고 있다.

제 5 장 결 론

본 논문에서는 ARM 프로세서 상에서 SEED 블록 암호의 CTR 운영 모드의 최적 구현을 연구하였다. 최적 구현을 위해 NEON SIMD 명령어를 활용하였고 사전 연산 기법 등 다양한 기법을 적용하였다. Nonce 값이 고정되어 사전 연산이 가능하다는 CTR 운영 모드의 특성을 통해 1 라운드의 F 함수까지 사전 연산이 가능하다는 것을 확인하였다. 해당 함수의 연산 생략을 통해 성능 향상을 달성하였다. Sbox 연산을 룩업 테이블을 통해 최적화 구현을 진행하였지만 레지스터의 절반을 Sbox 연산의 룩업 테이블로 사용해야 한다는 단점도 존재하였다. 하지만 더 빠른 암호화 연산이 가능하기 때문에 적은 레지스터를 사용하여 구현할 수 있는 블록 암호에 대해서는 의미 있는 성능 향상이 있을 것으로 보인다. 본 연구에서는 이러한 기법들을 적용하여 약 15%의 성능 향상을 보여주었다.

본 논문에서 제시하는 기법은 SEED 블록 암호에만 해당하는 것이 아닌 다른 블록 암호에 대해서도 적용할 수 있는 기법으로 보고 있다. 그래서 향후 연구로는 이러한 연산 기법이 적용 가능한 다른 블록 암호 알고리즘에 대해 조사하고 제안한 기법을 적용하여 최적화 구현 연구를 진행한다.

참 고 문 헌

1. 국내문헌

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.

Eum, Siwoo, et al. "Parallel Implementations of ARIA on ARM Processors and Graphics Processing Unit." Applied Sciences 12.23 (2022): 12246.

Yi, Jaeyoung, et al. "Fully pipelined hardware implementation of 128-bit SEED block cipher algorithm." Reconfigurable Computing: Architectures, Tools and Applications: 5th International Workshop, ARC 2009, Karlsruhe, Germany, March 16-18, 2009. Proceedings 5. Springer Berlin Heidelberg, 2009.

Seo, Young-Ho, Jong-Hyeon Kim, and Dong-Wook Kim. "Hardware implementation of 128-bit symmetric cipher SEED." Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No. 00EX434). IEEE, 2000.

Lee, H. J., et al. The SEED encryption algorithm. No. rfc4269. 2005..

2. 국외문헌

Fujii, Hayato, Félix Carvalho Rodrigues, and Julio López. "Fast AES implementation using ARMv8 ASIMD without cryptography extension." Information Security and Cryptology–ICISC 2019: 22nd International Conference, Seoul, South Korea, December 4–6, 2019, Revised Selected Papers 22. Springer International Publishing, 2020.

Wang, K. C. "ARMv8 Architecture and Programming." Embedded and Real-Time Operating Systems. Cham: Springer International Publishing, 2023. 505–792.

ARMv8-A Instruction Set Architecture. Available online: https://documentation-service.arm.com/static/613a2c38674a052ae36ca307 (accessed on 26 June 2019

Song, JinGyo, and Seog Chung Seo. "Efficient parallel implementation of CTR mode of ARX-based block ciphers on ARMv8 microcontrollers." Applied Sciences 11.6 (2021): 2548.

Lipmaa, Helger, Phillip Rogaway, and David Wagner. "CTR-mode encryption." First NIST Workshop on Modes of Operation. Vol. 39. 2000.

ABSTRACT

Optimized Implementation of block cipher SEED CTR mode on ARM processors

Song, Min-Ho

Major in Convergence Security

Dept. of Convergence Security

The Graduate School

Hansung University

The block cipher is an encryption algorithm that encrypts data in fixed size block units. It is a symmetric key encryption method that encrypts and decrypts data using the same key and has various operation modes. In this paper, we optimally implement the CTR mode of the SEED block cipher algorithm, which is widely used in Korea, on an ARM processor. ARM processors are utilized in various low-power and high-performance environments such as mobile devices and embedded systems. The CTR mode is advantageous for high-speed encryption because it allows parallel processing, but an appropriate technique is required to demonstrate performance with the limited resources of the ARM processor. In this study, we performed parallel operations using the NEON SIMD instructions of the ARM processor and optimized register utilization to improve the operation speed. Compared to the SEED reference, the study in this paper improved the performance by about 15%.

keyword: SEED block cipher, CTR mode, optimized implementation