석사학위논문

ARMv8 상에서의 SNOVA 전자 서명 알고리즘 최적 구현

2025년

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

이 민 우

석사학위논문
지도교수 서화정

ARMv8 상에서의 SNOVA 전자 서명 알고리즘 최적 구현

Optimal Implementation of SNOVA on ARMv8 Architecture

2024년 12월 일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

이 민 우

석사학위논문
지도교수 서화정

ARMv8 상에서의 SNOVA 전자 서명 알고리즘 최적 구현

Optimal Implementation of SNOVA on ARMv8 Architecture

위 논문을 공학 석사학위 논문으로 제출함

2024년 12월 일

한 성 대 학 교 대 학 원

융 합 보 안 학 과

융 합 보 안 전 공

이 민 우

이민우의 공학 석사학위 논문을 인준함

2024년 12월 일

심사위원장 박 명 서 (인)

심 사 위 원 이 웅 희 (인)

심 사 위 원 서 화 정 (인)

국 문 초 록

ARMv8 상에서의 SNOVA 전자서명 알고리즘 최적 구현

한 성 대 학 교 대 학 원
융 합 보 안 학 과
융 합 보 안 전 공
이 민 우

양자컴퓨팅 기술의 발전으로 기존 암호 시스템(예: RSA, ECC)이 점차 양자 공격에 취약해지고 있으며, 이에 대응하기 위해 양자 내성 암호(Post-Quantum Cryptography, PQC)의 필요성이 대두되고 있다. SNOVA는 NIST PQC 공모전에서 유망한 전자서명 알고리즘 후보로 제안되었으며, 다변수 공개키 암호 시스템(MPKC) 기반으로 설계되었다. 이 알고리즘은 비가환 행렬을 활용하여 기존 구조적 및 대수적 공격에 대한 내성을 높였으며, 상대적으로 작은 키 크기와 빠른 서명 및 검증 속도를 제공하는 특징이 있다. 특히 SNOVA는 IoT, 모바일 장치와 같은 자원 제한 환경에서도 효율적으로 작동할 수 있도록 설계되었으며, 이러한 특성은 다양한 응용 분야에서 주목받고 있다. 본 연구에서는 SNOVA의 키 생성, 서명 생성, 서명 검증 작업을 최적화하기 위해 어셈블리와 NEON SIMD 명령어를 활용하였다. 주요 최적화는 유한체 연산을 위한 덧셈 및 곱셈 모듈과 AES-128 CTR 모드 가속기 구현을 포함한다. 덧셈과 곱셈 연산은 ARMv8의 벡터 레지스터를 활용하여 병

렬 처리가 가능하도록 설계되었으며, AES 가속기는 하드웨어 명령어를 사용하여 암호화 속도를 극대화하였다. 성능 평가 결과, SNOVA의 주요 파라미터 세트(esk 및 ssk)를 대상으로 키 생성은 최대 68.7%, 서명 생성은 36.7%, 서명 검증은 50.0%의 성능 향상을 달성하였다.

【주요어】 NIST, PQC, SNOVA, ARMv8, GF 연산 최적화, 병렬 구현

목 차

표 목 차

알 고 리 즘 목 차

수 식 목 차

Ⅰ. 서 론

양자 컴퓨팅의 발전으로 인해, RSA와 ECC와 같은 기존의 암호 시스템은 점차 양자 공격에 취약해지고 있다. 특히 Shor 알고리즘은 정수 인수분해와 이산 로그 문제를 다항 시간 내에 해결할 수 있어 이러한 시스템에 큰 위협을 가하고 있다. 이에 따라 양자 컴퓨팅 위협에 대응할 수 있는 양자 내성 암호(PQC) 연구가 활발히 진행되고 있다. 여러 PQC 접근 방식 중에서 다변수 공개키 암호시스템(MPKC)은 다른 양자 이후 암호화 방식에 비해 상대적으로 작은 키 크기와 빠른 속도를 제공할 가능성 덕분에 주목받고 있다.

SNOVA 서명 체계는 다변수 기반 전자서명 알고리즘으로, NIST PQC 추가 전자서명 공모전에서 유망한 후보로 제시되고 있다. 그러나 SNOVA를 포함한 대부분의 PQC 알고리즘은 리소스가 제한된 환경, 예를 들어 사물인터넷(IoT) 장치, 임베디드 시스템, 모바일 플랫폼과 같은 환경에서의 효율적 구현이 요구된다. 이러한 플랫폼들은 제한된 전력과 계산 자원을 갖추고 있기 때문에, 암호화 연산의 빠른 처리 속도와 높은 계산 효율성을 동시에 만족해야 한다. 특히, ARM 기반 프로세서가 널리 사용되는 이러한 환경에서는 전력 효율성과 계산 성능 간의 균형을 맞추는 것이 필수적이다. 따라서 ARM 프로세서에 최적화된 암호 알고리즘 구현은 리소스가 제한된 환경에서의 실제 사용을 위해 필요한 과정이다.

특히 ARMv8-A 아키텍처는 64비트 AArch64 명령어 세트와 NEON 기술을 제공하여, 에너지 효율성 및 고성능 계산이 중요한 환경에서 널리 사용되고 있다. NEON은 ARM의 SIMD(단일 명령 다중 데이터) 확장 기능으로, 행렬 산술과 유한체 연산이 중요한 알고리즘에서 병렬 데이터 처리를 가능하게 한다. NEON 기술을 활용하면 키 생성, 서명, 서명 검증과 같은 주요 암호화 작업에서 성능을 크게 향상시킬 수 있다.

현재 다변수 공개키 암호시스템(MPKC)의 ARMv8 최적화 연구는 상대적으로 부족한 상황이다. 기존 연구들은 주로 격자 기반 또는 코드 기반 방식의

최적화에 중점을 두고 있다. 따라서 본 연구는 이러한 공백을 채우기 위해 NEON 기술을 활용한 SNOVA 서명 체계의 최적 구현을 제시하고자 한다.

본 연구의 주된 목표는 어셈블리 레벨의 최적화를 통해 ARMv8에서 SNOVA의 암호화 작업을 효율적으로 구현하고, 실행 시간을 단축할 수 있음을 입증하는 것이다. 본 논문에서는 NIST PQC 프로젝트의 레퍼런스 코드와 비교하여 최적화된 구현의 성능을 측정하고 성능 향상도를 비교한다.

II. 관련 연구

2.1 NIST PQC 공모전

2016년, NIST는 양자 컴퓨터 기술이 발전함에 따라 RSA, ECC와 같은 기존 암호 시스템이 양자 알고리즘 공격에 취약해질 가능성을 인지하고, 이를 해결하기 위한 새로운 양자내성암호 표준화를 목표로 한 공모전을 발표하였다. 이 공모전은 기존의 이산대수 문제를 기반으로 한 암호화 방식이 아닌, 양자 알고리즘에도 안전한 새로운 수학적 난제를 기반으로 한 암호화 방법을 탐구하고자 시작되었다.

공모전에 제출된 알고리즘은 단순히 높은 보안성을 제공하는 데 그치지 않고, 다양한 컴퓨팅 환경에서도 효율적으로 동작할 수 있어야 했다. 구체적으로, AES-128, AES-192, AES-256과 같은 대칭키 암호화 방식에 대응하는 키 검색 공격에 대해 충분히 안전해야 하며, SHA-256 또는 SHA-384에서 충돌 탐색 공격을 막을 수 있는 내성을 갖추는 것이 요구되었다. 이러한 보안 요건 외에도, 실제 구현 시 발생할 수 있는 메모리 사용량 증가와 연산 속도 저하 문제를 최소화하기 위해 성능과 비용 최적화가 중요한 평가 기준으로 포함되었다. NIST는 이러한 보안성과 성능 최적화를 동시에 만족하는 알고리즘을 찾기 위해, 알고리즘의 확장성과 플랫폼 간 호환성을 중점적으로 평가하였다. 제한된 자원을 가진 환경에서도 안정적으로 실행될 수 있어야 하며, 특정 하드웨어 명령어에 과도하게 의존하는 알고리즘은 일부 플랫폼에서 실행 불가능할 가능성이 있어 부적합하다고 판단되었다. 이에 따라, 다양한 플랫폼에서 효율적으로 구현 가능하고 확장성 높은 알고리즘이 공모전의 주요 후보로 자리 잡았다.

NIST PQC 공모전은 여러 단계의 평가 과정을 통해 진행되었다. 2017년 1차 라운드에서는 69개의 알고리즘이 선정되었으며, 이후 보안성, 성능, 구현 효율성, 확장성 등을 기준으로 심사하여 2019년에는 2차 라운드로 26개의 알

고리즘을 발표하였다. 이 중 17개의 알고리즘은 공개키 기반 암호화 방식이었으며, 나머지 9개는 전자서명 알고리즘으로 분류되었다. 2차 라운드에서 선정된 알고리즘들은 초기 설계에서 발견된 취약점을 보완하여 제출되었고, 각 알고리즘의 성능을 더욱 최적화하였다. 2020년, 3차 라운드에서는 최종적으로 7개의 주요 알고리즘과 8개의 추가 후보 알고리즘이 선정되었다. 이후 2022년 6월, NIST는 양자내성암호 표준으로 CRYSTALS-KYBER를 공개키 암호화 부문에서, CRYSTALS-Dilithium, FALCON, SPHINCS+를 전자서명 부문에서 최종적으로 선정하였다. 이와 동시에, 추가적인 검토가 필요한 BIKE, Classic McEliece, HQC는 4차 라운드의 후보로 발표되었으나, SIKE는 보안상의 문제로 인해 최종적으로 탈락하였다. 2023년 8월 24일, NIST는 양자내성암호 표준 초안을 발표하였다. 이번 초안에는 Crystals-Kyber, Crystals-Dilithium, SPHINCS+가 각각 FIPS-203, FIPS-204, FIPS-205로 지정되었으며, 해당 문서들은 각 알고리즘의 구조, 모듈, 매개변수 등을 상세히 설명하고 있다. 이러한 표준 초안은 다양한 플랫폼에서의 구현을 지원하며, NIST는 이를 기반으로 표준화 작업을 지속적으로 진행하고 있다.

2022년 9월, NIST는 기존 PQC 공모전에서 최종 선정된 전자서명 알고리즘들이 격자 기반에 치우쳐 있음을 인지하고, 보다 다양한 암호학적 기반을 탐구하기 위해 추가 전자서명 공모전을 발표하였다. 이번 공모전의 목표는 격자 기반 알고리즘뿐만 아니라 다변수, 코드, 해시 기반 등 다양한 접근 방식을 통해 양자 안전성을 확보하는 서명 알고리즘을 발굴하는 데 있다. 추가 공모전은 NIST가 기존 전자서명 표준으로 선정한 CRYSTALS-Dilithium, FALCON, SPHINCS+를 보완할 수 있는 새로운 알고리즘을 찾는 데 중점을 두었다. 특히, 제한된 자원 환경에서도 동작 가능한 서명 알고리즘을 발굴하고, 기존보다 높은 보안성과 효율성을 제공할 수 있는 방안을 제시하는 것이 핵심 과제였다. 2023년 6월, 공모전에서는 전 세계 연구진으로부터 50개의 서명 알고리즘이 제출되었다. 이 중, 초기 평가를 거쳐 40개의 알고리즘이 1차 라운드에 진출하였다. 1차 라운드에서는 각 알고리즘의 보안성, 성능, 그리고 다양한 플랫폼에서의 구현 가능성이 집중적으로 검토되었다.

2024년 초, NIST는 1차 라운드의 결과를 기반으로 보안 분석과 성능 평가

를 심화하여 14개의 알고리즘을 2차 라운드 후보로 선정하였다. 이 과정에서는 격자 기반 알고리즘뿐만 아니라 다변수 기반과 코드 기반 알고리즘이 포함되어, 다양한 접근 방식이 공평하게 평가되었다. 특히, SNOVA와 같은 다변수 기반 알고리즘은 기존 격자 기반 알고리즘에 비해 상대적으로 낮은 연산 자원을 요구하며, 병렬 연산 최적화를 통해 효율성을 크게 향상시킬 수 있는 점이 강점으로 부각되었다. 2차 라운드에 진출한 알고리즘들은 양자 컴퓨팅 시대의 요구 사항을 충족시키면서도, 실용적인 구현이 가능하도록 설계되었다. 이번 추가 공모전은 단순히 기존 전자서명 알고리즘을 대체하는 것이 아니라, 다양한 응용 환경에 적합한 알고리즘을 발굴하는 데 중요한 역할을 한다. 예를 들어, IoT와 같이 제한된 메모리와 전력을 사용하는 장치에서는 효율성과 확장성이 필수적이다. 이러한 점에서 SNOVA를 포함한 다변수 기반 알고리즘들은 유망한 선택지로 주목받고 있다.

2.2 Unbalanced Oil and Vinegar 알고리즘

UOV(Unbalanced Oil and Vinegar) 알고리즘은 다변수 공개키 암호(Multivariate Public Key Cryptography, MPKC)의 대표적인 전자서명 알고리즘으로, 기존 Oil and Vinegar(OOV) 알고리즘의 확장판으로 설계되었다. Kipnis와 Shamir(1997)이 제안한 UOV는 OOV 알고리즘에서 발견된 구조적 약점을 개선하며, 다변수 암호학 분야에서 중요한 전환점을 마련하였다. UOV는 Oil 변수와 Vinegar 변수로 구분된 다변수 이차 방정식 시스템을 기반으로 서명을 생성하고 검증한다. Vinegar 변수는 무작위로 선택되며, Oil 변수는 공개 방정식을 만족해야 한다는 특징을 가진다. 이러한 구조는 대수적 공격과 구조적 공격에 대한 내성을 제공하며, 특히 Vinegar 변수와 Oil 변수 간의 비율을 조정함으로써 보안성을 강화하였다.

그러나 UOV는 다음과 같은 중요한 한계를 지니고 있었다. 첫째, 공개키의 크기가 매우 크다는 점이다. 다변수 암호의 특성상 공개키는 다차원 행렬 형태를 가지며, 이는 실질적인 구현에서 메모리 사용량을 크게 증가시킨다. 이러한 공개키 크기의 문제는 UOV가 제한된 자원 환경에서 사용되기 어렵게

만드는 주요 요인 중 하나였다. 둘째, 대수적 공격에 취약한 구조적 문제를 완전히 해결하지 못했다는 점이다. 특정 조건에서 Oil 변수와 Vinegar 변수 간의 관계를 분석해 공개키를 역공학할 수 있는 가능성이 여전히 존재했다.

이러한 단점을 해결하기 위해 제안된 것이 바로 SNOVA(Simple Noncommutative Oil and Vinegar) 알고리즘이다. SNOVA는 UOV의 기본 구조를 바탕으로 하면서도 공개키의 크기를 대폭 줄이고 보안성을 강화하기 위한 여러 개선을 도입하였다. SNOVA의 핵심적인 변화는 비가환 행렬(noncommutative matrices)의 도입이다. UOV의 공개키가 대규모 다차원 행렬로 구성되어 메모리와 연산 부담을 초래했던 것과 달리, SNOVA는 비가환 행렬 연산을 통해 보다 압축된 형태의 공개키를 생성한다. 이를 통해 공개키 크기를 효과적으로 줄이는 동시에 구조적 공격과 대수적 공격에 대한 내성을 높였다. 또한, SNOVA는 현대적인 하드웨어 환경에서의 병렬 연산을 최적화하기 위해 설계되었다. ARMv8 아키텍처와 같은 고성능 프로세서에서 NEON SIMD 명령어를 활용하여 UOV보다 훨씬 효율적인 연산 구조를 구현하였으며, 특히 서명 생성과 검증 과정에서의 성능 향상을 달성하였다. 이는 SNOVA가 제한된 자원 환경뿐만 아니라 고성능 플랫폼에서도 실질적인 구현 가능성을 갖춘 알고리즘임을 보여준다.

2.3 SNOVA 알고리즘

SNOVA(Simple Noncommutative Oil and Vinegar)는 비가환 행렬을 활용하는 다변수 공개키 암호 시스템(MPKC)으로, 다변수 이차 방정식(MQ 문제)의 수학적 복잡성을 기반으로 보안성을 제공한다. 이 알고리즘은 기존 Unbalanced Oil and Vinegar(UOV) 서명 체계의 확장된 형태로, 비가환 행렬을 도입함으로써 대수적 공격과 구조적 공격에 대한 내성을 향상시키는 것을 목표로 한다. SNOVA는 주로 메시지 서명과 검증 작업에 사용되며, 다항식 해를 찾는 수학적 문제를 바탕으로 보안을 확보한다. SNOVA 알고리즘은 유한체 GF(16) 상의 비가환 행렬을 주요 연산 구조로 사용한다. 이는 기존의 유한체 산술과 달리 행렬 곱셈이 비가환적(non-commutative)이기 때문에 대

수적 구조 분석이 어렵게 만든다. 다항식의 일반적 형태는 다음과 같이 표현된다.

$$P_{i(X)} = \Sigma_{j=1}^{n} \Sigma_{k=1}^{n} a_{ijk} x_j x_k + \Sigma_{j=1}^{n} b_{ij} x_j + c_i$$

수식 2-1 다변수 이차 다항식

이러한 다항식을 조합하여 개인키와 공개키를 생성하고, 메시지 서명 시 해시 값을 기반으로 다항식의 해를 계산한다. SNOVA는 키 생성, 서명 생성, 서명 검증의 세 단계로 구성된다. 키 생성 과정에서는 난수 행렬을 기반으로 비밀키와 공개키가 생성되며, 주요 비밀키 구성 요소는 다음과 같다.

$$SK = (A_\alpha, B_\alpha, Q_{\alpha 1}, Q_{\alpha 2}, T_{12})$$

수식 2-2 비밀키 구성 요소

이 행렬들은 유한체 상에서 난수를 통해 생성되며, 행렬 곱과 합 연산을 통해 비밀키 연산이 이루어진다. 공개키는 비밀키 요소들을 조합하여 만들어지며, 다음과 같이 나타낼 수 있다.

$$PK = (P_{11}, P_{12}, P_{21}, P_{22})$$

수식 2-3 공개키 생성

SNOVA의 성능은 행렬 연산의 효율성에 의해 결정된다. 주요 연산은 유한체에서 행렬 곱과 덧셈으로 구성되며, 이는 하드웨어 가속을 통해 더욱 효율적으로 수행될 수 있다. 공개키 크기와 서명 크기는 기존 UOV 서명 체계 대비 상당히 작은 편이며, 양자 컴퓨터에 대한 보안성을 확보할 수 있도록 설계되었다. 결론적으로 SNOVA는 다변수 이차 다항식과 비가환 행렬을 활용하여 대수적 공격과 구조적 공격에 대한 내성을 높이고, 효율적인 서명 및 검증 연산을 제공하는 전자서명 알고리즘이다. 이를 통해 기존 MPKC 기반 시스

템의 공개키 크기와 연산 복잡도를 최적화하면서도 강력한 보안성을 유지한다.

알고리즘의 보안 수준과 성능은 몇 가지 주요 파라미터를 조정함으로써 달성할 수 있다. 이러한 파라미터는 Vinegar 변수 수(v), Oil 변수 수(o), 유한체 크기(q), 방정식 차수(d) 등으로 구성되며, 각각은 알고리즘의 서명 생성 및 검증 과정에서 중요한 역할을 수행한다. Vinegar 변수는 서명 생성 과정에서 무작위적으로 선택되며, 다변수 방정식 시스템의 비선형성을 보장한다. Vinegar 변수 수(v)가 증가하면 무작위성이 증가하여 보안성이 강화되지만, 계산 복잡도 또한 증가한다. Oil 변수는 공개 방정식을 만족해야 하는 변수로, 검증 과정에서 주요 역할을 수행한다. Oil 변수 수(o)가 많아지면 서명 검증 단계에서 계산량이 증가하지만, 알고리즘의 강도 역시 강화된다. 유한체 크기(q)는 유한체 $F(q)$ 에서 연산이 수행되므로, 보통 16과 같은 소규모 값으로 설정된다. 유한체 크기는 연산의 복잡도와 직접적으로 관련되며, 큰 값으로 설정할수록 보안성이 높아지지만 계산 비용이 증가한다. 방정식 차수(d)는 다변수 방정식의 비선형 차수를 나타내며, 보통 2로 고정된다. 이는 이차 다항식을 사용하여 서명 생성 및 검증 알고리즘을 구성하는 데 핵심적인 요소이다. SNOVA의 파라미터는 보안성과 성능 간의 균형을 고려하여 조정된다. 높은 보안성을 위해서는 Vinegar 변수 수(v)와 Oil 변수 수(o)를 증가시키고 유한체 크기(q)를 크게 설정할 수 있다. 반대로, 계산 성능을 개선하기 위해서는 이러한 값을 줄이는 방식으로 최적화를 진행할 수 있다. 이처럼 SNOVA 알고리즘은 유연한 파라미터 설정을 통해 다양한 응용 환경에서 사용할 수 있으며, 양자 안전성을 목표로 하는 응용에서 특히 주목받고 있다. 다음은 SNOVA 알고리즘의 주요 파라미터 설정 예시를 나타낸 표이다.

[표 2-1] SNOVA 파라미터

보안 레벨	Vinegar Variables(v)	Oil Variables(o)	Field Size(q)	Degree(1)
I	24	5	16	4
III	43	25	16	2
V	60	10	16	4

2.4 AES-CTR 모드

AES(Advanced Encryption Standard)는 2001년 NIST에 의해 표준으로 지정된 대칭키 블록 암호화 알고리즘으로, Rijndael 알고리즘을 기반으로 설계되었다. AES는 고정된 128비트 블록 크기를 가지며, 키 크기에 따라 128비트, 192비트, 256비트 암호화를 지원한다. 이 알고리즘은 높은 보안성과 효율성을 제공하며, 현대 암호학에서 가장 널리 사용되는 암호화 방식 중 하나로 자리 잡았다. AES는 암호화와 복호화에서 동일한 키를 사용하는 대칭키 암호로, 여러 라운드의 반복 연산을 통해 데이터를 암호화하거나 복호화한다.

AES의 암호화 과정은 SubBytes, ShiftRows, MixColumns, AddRoundKey라는 네 가지 주요 연산으로 구성된다. SubBytes 연산에서는 128비트 블록을 구성하는 각 바이트를 고정된 S-Box를 통해 대체하여 비선형성을 추가한다. 이 S-Box는 비선형 대수적 함수로 설계되어 대수적 공격에 대한 내성을 제공하며, 소프트웨어적으로는 테이블 조회 방식을 사용하거나 bitslice 기법으로 구현하여 성능을 최적화할 수 있다. ShiftRows 연산은 AES 상태 행렬에서 각 행을 왼쪽으로 순환 이동시켜 열 간의 데이터 혼합을 유도한다. 첫 번째 행은 이동하지 않으며, 두 번째 행은 한 칸, 세 번째 행은 두 칸, 네 번째 행은 세 칸 이동된다. 이 과정은 데이터의 공간적 분포를 강화하여 보안성을 높인다. MixColumns 연산은 상태 행렬의 각 열을 고정된 다항식으로 변환하여

열 내 데이터 혼합을 수행한다. 이는 Galois Field(GF(64))에서의 행렬 곱셈으로 구현되며, 데이터의 통계적 성질을 균일하게 만드는 데 기여한다. 소프트웨어적으로는 사전 계산된 테이블을 활용하거나, 병렬화된 bitslice 연산으로 최적화할 수 있다. 마지막으로, AddRoundKey 연산에서는 상태 행렬과 라운드 키를 XOR 연산하여 현재 라운드의 키를 적용한다. 이 과정에서 라운드 키는 키 확장 과정에서 생성되며, 초기 키를 바탕으로 여러 개의 라운드 키가 유도된다.

AES는 키 크기에 따라 10, 12, 또는 14 라운드로 구성되며, 각 라운드에서 위의 네 가지 연산이 순차적으로 수행된다. 마지막 라운드에서는 MixColumns 연산이 생략된다. 복호화는 암호화 과정의 역순으로 이루어지며, SubBytes와 MixColumns 연산의 역변환이 포함된다. AES는 이러한 구조적 설계를 통해 높은 보안성과 효율성을 제공하며, 하드웨어와 소프트웨어에서 모두 최적화가 가능하도록 설계되었다.

AES는 고정된 128비트 블록 단위로 데이터를 처리하는 블록 암호 방식으로, 암호화를 다양한 시나리오에 적합하게 활용할 수 있도록 여러 운영 모드(Operation Mode)가 정의되었다. 대표적인 운영 모드로는 ECB(Electronic Codebook), CBC(Cipher Block Chaining), CFB(Cipher Feedback), OFB(Output Feedback), 그리고 CTR(Counter) 모드가 있다. 이러한 모드들은 AES의 블록 암호 구조를 기반으로 각기 다른 데이터 처리 방식을 제공하며, 암호화 시스템의 유연성과 보안성을 향상시키는 데 사용된다. ECB 모드는 가장 기본적인 블록 암호 운영 방식으로, 각 데이터 블록을 독립적으로 암호화한다. 이 방식은 구현이 간단하고 병렬 처리가 가능하다는 장점이 있지만, 동일한 평문 블록이 동일한 암호문 블록으로 변환되기 때문에 패턴이 노출될 위험이 있다. 이러한 단점은 암호문의 보안성을 약화시키며, ECB 모드가 데이터 패턴을 숨기는 데 적합하지 않게 만든다. CBC 모드는 이러한 문제를 해결하기 위해 각 데이터 블록을 이전 블록의 암호문과 XOR 연산하여 암호화한다. 이 방식은 동일한 평문 블록에 대해 다른 암호문을 생성할 수 있어 보안성이 향상되지만, 암호화 과정이 직렬적이어서 병렬 처리가 어렵다는 단점이 있다. CFB와 OFB 모드는 스트림 암호처럼 동작하도록 설계된 운영

모드로, 각각 암호문과 초기화 벡터(IV)의 출력을 피드백하여 다음 블록의 암호화에 사용하는 방식을 취한다. 이들 모드는 암호화와 복호화에서 동일한 처리 과정을 사용하며, 특정 환경에서 유용하다. 그러나 이들 모드는 블록 암호 운영 방식보다 더 높은 수준의 연산을 요구하며, 특정 응용에서 병렬화가 제한된다.

CTR(Counter) 모드는 AES를 스트림 암호 방식으로 동작하도록 설계된 운영 모드 중 하나로, 블록 암호의 병렬화 가능성을 극대화할 수 있는 구조를 제공한다. CTR 모드에서는 각 데이터 블록이 독립적으로 암호화되며, 고유한 초기화 벡터(IV)와 카운터 값을 결합하여 각 블록에 대해 고유한 입력을 생성한다. AES는 이 입력 값을 암호화하여 키 스트림을 생성하고, 이를 평문과 XOR 연산하여 암호문을 만든다. 복호화 과정에서도 동일한 키 스트림을 생성하여 암호문과 XOR 연산을 수행하면 원래의 평문을 복원할 수 있다. CTR 모드는 다음과 같은 이유로 현대 암호화 시스템에서 널리 사용된다. 첫째, CTR 모드는 블록 간 독립적인 처리가 가능하므로 병렬 처리를 지원한다. 이는 암호화와 복호화 속도를 크게 향상시키며, 대용량 데이터 처리에 적합하다. 둘째, CTR 모드는 초기화 벡터와 카운터 값의 조합을 통해 각 블록에 고유한 입력을 보장하므로, 동일한 키를 사용하더라도 암호문 블록이 중복되지 않는다. 이러한 특성은 재사용 공격(replay attack)을 방지하며, 암호문의 보안성을 유지한다. 셋째, CTR 모드는 간단한 연산 구조를 가지고 있어 하드웨어 및 소프트웨어 구현이 용이하다. 이 방식은 효율적인 키 스트림 생성을 가능하게 하며, 메모리 사용량을 최소화한다.

SNOVA에서는 AES-CTR 모드가 사용되었으며, 이는 SNOVA의 설계 요구 사항과 최적화 목표에 부합하기 때문이다. CTR 모드는 SNOVA의 서명 생성 및 검증 과정에서 병렬 연산의 가능성을 제공하여 ARMv8 아키텍처와 같은 현대 프로세서에서 NEON SIMD 명령어를 활용한 최적화를 가능하게 한다. 또한, CTR 모드는 유한체 연산과 결합하여 추가적인 복잡성을 초래하지 않으면서도 데이터를 효율적으로 변환할 수 있다. 이처럼 CTR 모드의 단순하고 병렬화 가능한 구조는 SNOVA의 구현에서 핵심적인 역할을 하며, 제한된 자원 환경에서도 높은 성능을 발휘하도록 돕는다.

2.5 ARMv8 아키텍처

ARMv8 아키텍처는 ARM Holdings에서 개발한 고성능 저전력 프로세서 구조로, 다양한 응용 분야에서 널리 사용되고 있다. 모바일 장치, 임베디드 시스템, 사물인터넷(IoT) 기기 등 전력 소비가 중요한 환경에서 높은 성능을 제공하기 위해 설계되었다. ARMv8은 이전 세대인 ARMv7과 비교하여 64비트 연산을 지원하는 AArch64 명령어 세트를 도입함으로써 계산 능력과 메모리 접근 범위를 확장하였다.

AArch64 명령어 세트는 31개의 범용 64비트 레지스터와 스택 포인터를 포함하며, 고정된 32비트 명령어 형식을 채택하여 디코딩 효율성을 향상시킨다. 이러한 설계는 대규모 데이터 처리 시 메모리 접근을 유연하게 관리할 수 있도록 지원한다. ARMv8 아키텍처는 64비트 프로세서로 작동하면서도 32비트 모드(AArch32)와의 호환성을 제공해 다양한 소프트웨어 생태계를 지원한다.

ARMv8의 주요 특징 중 하나는 병렬 데이터 처리를 위한 NEON SIMD(Single Instruction, Multiple Data) 기술이다. NEON은 128비트 벡터 레지스터를 사용해 다수의 데이터 요소를 동시에 연산할 수 있어, 특히 멀티미디어 처리, 신호 처리, 암호화 알고리즘에서 유리하다. NEON 연산은 정수 및 부동소수점 데이터를 모두 지원하며, 벡터 연산은 64비트 단위(D 레지스터) 또는 128비트 단위(Q 레지스터)로 수행된다.

암호화 연산 가속은 ARMv8 아키텍처의 또 다른 강점이다. AES, SHA-1, SHA-256과 같은 암호화 알고리즘을 위한 하드웨어 명령어 집합이 포함되어 있어 소프트웨어 기반 암호화 대비 성능이 크게 향상된다. 이는 데이터 보안이 중요한 응용 프로그램에서 전력 소모를 줄이고 처리 시간을 단축하는 데 기여한다. 하드웨어 지원을 통해 블록 암호화, 해시 연산과 같은 연산이 최적화된다.

보안 측면에서는 메모리 보호를 위한 가상 메모리 관리 유닛(MMU)을 갖

추고 있어 운영 체제가 안전한 메모리 관리를 수행할 수 있다. TrustZone 기술은 보안 영역과 비보안 영역을 분리하여 신뢰할 수 있는 실행 환경을 제공한다. 이를 통해 보안이 중요한 금융, 의료, IoT 응용 프로그램 개발 시 보안 위협을 줄일 수 있다.

이러한 특성 덕분에 ARMv8 아키텍처는 암호화 알고리즘 연구에서도 주목받고 있다. 특히 행렬 연산, 벡터 연산 등 병렬 처리가 필요한 계산 집약적 응용 분야에서 NEON 명령어와 벡터 레지스터의 효율적인 활용이 가능하다. 이에 따라 양자 내성 암호 연구, 신호 처리, 머신러닝 등 다양한 계산 환경에서 ARMv8이 연구 및 개발 플랫폼으로 활용되고 있다.

2.6 ARMv8 NEON SIMD

ARMv8 아키텍처는 데이터 병렬 처리를 위한 도구로 NEON SIMD(Single Instruction Multiple Data) 명령어 집합을 제공한다. NEON은 다수의 데이터 요소를 동시에 처리할 수 있는 128비트 벡터 레지스터를 활용하여 암호학, 디지털 신호 처리(DSP), 컴퓨터 비전 등 연산 집약적인 애플리케이션에서 성능 향상을 가능하게 한다. 이를 통해 ARMv8 기반 시스템은 제한된 전력 소비로도 높은 계산 효율을 달성할 수 있다.

NEON SIMD는 최대 16개의 8비트 정수, 8개의 16비트 정수, 4개의 32비트 정수 또는 2개의 64비트 정수를 단일 레지스터에서 병렬로 처리할 수 있다. 이러한 구조는 암호 알고리즘의 핵심 연산인 벡터 곱셈과 덧셈, 비트 이동 및 XOR 연산에서 성능을 극대화한다. 예를 들어, 암호 알고리즘의 주요 연산인 GF(2^n) 상의 곱셈은 NEON 명령어를 사용하여 다수의 다항식을 병렬로 계산할 수 있어 데이터 처리량을 증가시킨다. 또한, NEON 명령어는 데이터 로드와 저장을 효율적으로 관리하는 명령어도 포함하고 있다. 벡터 로드(LD1)와 저장(ST1) 명령어는 연속된 메모리 블록을 한 번에 로드하거나 저장하여 메모리 접근 오버헤드를 줄인다. 이는 다중 블록 암호화, 데이터 직렬화 및 복호화 과정에서 처리 지연(latency)을 줄이는 데 특히 유용하다.

NEON의 또 다른 특징은 산술 연산과 논리 연산의 병렬 처리 능력이다.

예를 들어, XOR(EOR), AND(BIC), OR(ORR)와 같은 비트 연산 명령어는 대량의 데이터를 동시에 처리함으로써 암호학적 연산 속도를 비약적으로 향상시킨다. 이러한 연산들은 대칭키 암호 알고리즘이나 메시지 인증 코드(MAC) 계산에 필수적이다.

ARMv8 아키텍처에서 NEON 명령어를 사용하는 병렬 구현 사례로는 다항식 곱셈, 유한체 연산, 벡터 변환 등이 있으며, 이러한 작업들은 비트-단위 수준에서 최적화된다. 특히, NEON 명령어는 매트릭스 곱셈과 같은 연산 작업에서 상당한 성능 향상을 제공한다.

2.7 ARMv8 상에서의 최적 구현에 대한 이전 연구

ARMv8 아키텍처에서 PQC(Post-Quantum Cryptography) 알고리즘의 최적 구현에 대한 연구는 활발히 진행되고 있다. 특히, ARMv8의 64비트 환경과 NEON SIMD 명령어를 활용하여 암호화 알고리즘의 성능을 향상시키는 시도가 주목받고 있다.

ARMv8 아키텍처에서의 대표적인 최적화 연구 사례로는 NIST PQC 후보 알고리즘인 SABER의 고속 구현이 있다. SABER는 핵심 연산으로 다항식 곱셈을 사용하며, ARMv8에서 NEON 명령어를 통해 평가(evaluation)와 보간(interpolation) 단계를 가속화하는 연구가 수행되었다. 이 연구에서는 Toom-Cook 곱셈 알고리즘을 기반으로 평가 과정의 데이터를 효율적으로 병렬 처리하기 위해 데이터 인터리빙(interleaving) 기법이 적용되었다. 보간 단계에서는 ARMv8의 SIMD 연산을 통해 고속 덧셈과 곱셈을 수행하여, 레퍼런스 구현 대비 평가 단계에서 약 3.5배, 보간 단계에서는 약 5배의 성능 향상을 달성하였다.

또한, 국내 연구진들은 KpqC 알고리즘 후보인 SMAUG의 ARMv8 상에서의 고속 구현 사례를 발표하였다. 이 연구에서는 $GF(2^n)$에서의 비트-단위 병렬 곱셈을 NEON 명령어로 구현하여 고속 다항식 곱셈을 수행하였으며, 이를 통해 연산 지연(latency)을 크게 줄였다. 곱셈 모듈은 ARMv8의 128비트 벡터 레지스터를 적극적으로 활용하여 네 개의 32비트 값을 동시에 처리할

수 있도록 설계되었다. 이러한 접근은 PQC 알고리즘의 서명 생성과 검증 속도를 크게 향상시켰으며, 곱셈 연산에서는 최대 24.62배, 암호화 연산에서는 최대 3.51배의 성능 향상을 이루었다.

더불어, Lattice 기반 알고리즘인 Kyber, NTRU 및 Dilithium과 같은 주요 PQC 후보들이 ARMv8 상에서 벡터화(vectorization)와 명령어 스케줄링(instruction scheduling)을 통한 연산 최적화를 수행한 사례도 존재한다. 예를 들어, Kyber의 경우 몽고메리 곱셈(Montgomery multiplication)과 NTT(Negacyclic Number Theoretic Transform) 기반 다항식 연산을 ARMv8에서 최적화하여 데이터 로드 및 저장 오버헤드를 줄였다.

III. SNOVA 알고리즘 최적 구현 기법

3.1 덧셈기 최적화

SNOVA 알고리즘에서 사용되는 GF(16)는 16개의 원소를 가지는 유한체(Galois Field)로, 다항식 GF(2)[x]에서 생성 다항식 $f(x) = x^4 + x + 1$에 의해 정의된다. GF(16)의 원소는 $\{0,1,\alpha,\alpha^2,\alpha^3,\cdots,\alpha^{14}\}$로 표현되며, 이때 α α는 생성 원소로 $f(\alpha)$ = 0을 만족한다. 이러한 원소들은 다항식의 계수로 나타낼 수 있고, GF(16)에서의 연산은 이 계수를 기반으로 정의된다. GF(16)에서 행렬 덧셈은 유한체의 덧셈 규칙을 따르며, 행렬의 대응 원소 간의 연산으로 정의된다. 수학적으로 GF(16)의 두 원소 a , b의 덧셈은 $a + b = a \oplus b$와 같이 나타낼 수 있다. 여기서 $\oplus$는 비트 단위 XOR 연산을 의미한다. 예를 들어, $a = \alpha^2 + 1$이고 $b = \alpha^{3+}\alpha + 1$이라면, 이들의 덧셈은 다음과 같이 계산된다.

$$a + b = (\alpha^2) + 1 \oplus (\alpha^3 + \alpha + 1) = \alpha^3 + \alpha + \alpha^2$$

수식 3-1 행렬 덧셈

GF(16)에서의 덧셈은 실제로는 XOR 연산으로 효율적으로 계산된다. GF(16)의 각 원소는 최대 x^3까지의 계수를 가지는 다항식으로 표현되며, 이는 이진수로 변환할 수 있다. 예를 들어, 원소 $\alpha^2 + 1$은 이진수로 0101로 나타낼 수 있다. GF(16)의 덧셈은 두 다항식의 각 계수끼리 더하는 연산으로 정의되며, 이 연산은 모듈러 2 연산과 동일하다. 이를 수식으로 표현하면 다음과 같다.

$$(a3x^3 + a2x2 + a1x + a0) + (b3x^3 + b2x^2 + b1x + b0) =$$
$$((a3 \oplus b3)x^3 + (a2 \oplus b2)x^2 + (a1 \oplus b1)x + (a0 \oplus b0))$$

수식 3-2 행렬 덧셈 다항식 연산

GF(16)에서의 XOR 연산은 다음과 같은 이유로 효율적이다. 첫째, XOR 연산은 하드웨어에서 단일 게이트로 구현 가능하며, 매우 빠르고 간단하다. 둘째, XOR 연산은 비트 단위로 독립적으로 계산되므로 병렬 처리에 적합하다. 셋째, XOR 연산은 메모리 접근 없이 레지스터 내에서 수행되므로 메모리 대역폭 사용을 줄일 수 있다. GF(16)에서의 곱셈과 비교했을 때, 덧셈 연산은 훨씬 간단하다. GF(16)에서의 곱셈은 두 다항식을 곱한 뒤 생성 다항식으로 나누는 과정을 포함하며, 이는 모듈러 연산과 자리 이동을 필요로 한다. 이와 달리 덧셈은 단순히 XOR 연산만 수행하면 되므로 연산 비용이 훨씬 낮다. 이러한 특성은 덧셈 연산이 어셈블리 코드에서 최적화하기에 적합한 이유를 설명한다.

ARMv8 아키텍처에서 덧셈 연산은 단일 명령어로 구현 가능하다. 예를 들어, ARMv8의 EOR 명령어는 XOR 연산을 수행하며, 메모리와 레지스터 간의 간단한 연산으로 처리가 가능하다. 또한, XOR 연산은 데이터 간 종속성이 낮아 파이프라이닝(pipelining)에 최적화되어 있으며, 현대 프로세서에서 매우 효율적으로 실행된다.

Algorithm : SNOVA 덧셈기 최적 구현

Input: $A = a0, a1, \ldots, an, B = b0, b1, \ldots, bn$
Output: $C = A \oplus B$
#Inialization
1: for $i = 0$ *to* n *do*
2: $C[i] \leftarrow A[i] \oplus B[i]$
#Addition
3: $A[i] \oplus B[i]$ for *each* i

[알고리즘 3-1] SNOVA 덧셈기 최적 구현

Rank=2의 경우 행렬의 크기는 2×2로 총 4개의 요소를 포함하며, 각 요소는 1바이트 크기의 데이터를 가진다. 따라서 전체 행렬의 데이터 크기는 4바이트로 제한된다. ARMv8 아키텍처를 활용한 구현에서는 일반 레지스터를 사용하여 이 데이터를 효율적으로 처리하였다. 행렬 덧셈 연산은 GF(16) 상에서 정의된 덧셈 규칙을 따르며, 이는 비트 단위 XOR 연산으로 구현된다. 우선 입력 행렬 a와 b의 메모리 주소는 각각 입력 레지스터 $x0$와 $x1$에 전달되며, 결과를 저장할 메모리 주소는 $x2$에 전달된다. 먼저 a 행렬의 4바이트 데이터를 메모리로부터 로드하여 레지스터에 저장하고, 이어서 b행렬의 4바이트 데이터도 동일하게 로드한다. 이 과정은 메모리 접근을 최소화하기 위해 단일 로드 명령어를 사용하여 수행된다. 이후 두 레지스터에 저장된 데이터를 XOR 연산하여 결과를 새로운 레지스터에 저장한다. 이 XOR 연산은 GF(16)에서의 덧셈 연산을 정확히 표현하며, 두 다항식의 계수를 모듈러 2 방식으로 더하는 것과 동일한 결과를 생성한다. 예를 들어, a=0x0F이고 b=0x03일 때, XOR 연산을 통해 c = 0x0C가 계산된다. 계산된 결과는 다시 메모리에 저장되어 출력 행렬 c로 반환된다. 마지막으로 함수는 호출자에게 제어를 반환하며 실행을 종료한다. Rank=2에서의 구현은 단순한 데이터 구조와 ARMv8 아키텍처의 강점을 최대한 활용하여 높은 효율성을 제공하였다. 총 데이터 크기가 4바이트로 작기 때문에 단일 메모리 로드와 저장 명령으로 모든 연산을 처리할 수 있었다. 이는 메모리 접근 횟수를 줄이고 캐시 효율성을 극대화하여 전반적인 성능을 향상시켰다. 또한, XOR 연산은 ARMv8에서 단일 명령어로 처리 가능하여 파이프라이닝에서 높은 성능을 발휘하였다. 이와 같은 Rank=2의 행렬 덧셈 구현은 작은 데이터셋에서 매우 효율적으로 작동하지만, 병렬 처리가 불가능한 구조로 인해 더 큰 데이터셋에서는 비효율적일 수 있다. 따라서 Rank=3 및 Rank=4의 구현에서는 병렬 처리 기법을 추가적으로 적용하여 성능을 개선하였다.

Rank=3의 경우 행렬의 크기는 3×3으로, 총 9개의 요소를 포함하며 각 요소는 1바이트 크기의 데이터를 가진다. 따라서 전체 행렬의 데이터 크기는 9바이트에 이른다. Rank=3 구현에서는 ARMv8 아키텍처의 NEON SIMD 레

지스터와 일반 레지스터를 혼합하여 사용함으로써 효율성을 극대화하였다. 특히, 병렬화가 가능한 데이터는 NEON SIMD 레지스터를 통해 처리하고, 남은 데이터를 일반 레지스터로 처리하여 최적화된 연산을 수행하였다. 우선 입력 행렬 a와 b의 메모리 주소는 각각 $x0$와 $x1$에 전달되며, 결과를 저장할 메모리 주소는 $x2$에 전달된다. 구현에서는 먼저 a와 b 행렬의 처음 8바이트를 NEON SIMD 레지스터에 로드하여 병렬 처리를 수행한다. NEON SIMD는 16바이트 단위의 병렬 연산을 지원하지만, Rank=3에서는 9바이트만 처리해야 하므로 NEON 레지스터를 활용한 병렬 연산과 일반 레지스터를 활용한 단일 연산을 조합하였다. 처음 8바이트는 NEON 레지스터에서 한 번에 XOR 연산을 수행한다. 이 과정에서 a와 b의 데이터를 각각 SIMD 레지스터에 로드하고, XOR 명령어를 사용하여 두 레지스터의 데이터를 병렬로 연산한다. 연산된 결과는 다시 메모리에 저장되어 결과 행렬의 첫 8바이트를 구성한다. 이후 남은 1바이트는 일반 레지스터를 활용하여 처리된다. 메모리로부터 a와 b의 마지막 바이트를 각각 로드하여 XOR 연산을 수행한 뒤, 결과를 메모리에 저장하여 마지막 1바이트를 완성한다. 이와 같은 구현 방식은 NEON SIMD를 활용하여 데이터의 대부분(8바이트)을 병렬로 처리함으로써 처리 속도를 크게 향상시켰다. 동시에 일반 레지스터를 사용하여 잔여 데이터를 처리함으로써 Rank=3에서 발생하는 비정렬 데이터를 효율적으로 처리하였다. 예를 들어, 행렬 a=[0x0F,0x1E,0x2D]와 b=[0x03,0x07,0x09]가 주어졌을 때, 처음 8바이트는 병렬로 XOR 연산을 수행하여 결과를 생성하고, 나머지 1바이트는 일반 레지스터에서 별도로 처리하여 완전한 결과를 도출하였다. Rank=3 구현의 주요 장점은 NEON SIMD의 병렬 처리 능력을 활용하여 데이터의 대부분을 고속으로 처리할 수 있다는 점이다. 특히, 메모리 접근 횟수를 최소화하고 병렬화된 데이터 처리 구조를 사용하여 성능을 최적화하였다. 그러나 남은 1바이트에 대해서는 일반 레지스터를 사용해야 하므로 NEON SIMD의 효율성이 완전히 발휘되지 않는 한계가 존재한다. 이러한 한계를 보완하기 위해 Rank=4와 같은 더 큰 데이터셋에서는 NEON SIMD를 활용한 완전 병렬 처리가 적용되었다.

Rank=4의 경우 행렬의 크기는 4×4로, 총 16개의 요소를 포함하며 각 요

소는 1바이트 크기의 데이터를 가진다. 따라서 전체 행렬의 데이터 크기는 16바이트에 해당한다. Rank=4 구현에서는 ARMv8 아키텍처의 NEON SIMD 레지스터를 활용하여 전체 데이터를 단일 명령어로 처리함으로써 완전 병렬화된 연산을 실현하였다. 이는 이전 Rank=2와 Rank=3 구현에서 사용된 일반 레지스터와 혼합 방식과는 차별화된 접근법이다. 입력 행렬 a 와 b 의 메모리 주소는 각각 $x0$와 $x1$에 전달되며, 결과 행렬 c를 저장할 메모리 주소는 $x2$에 전달된다. NEON SIMD는 128비트 레지스터를 지원하며, 이는 한 번의 로드-저장 명령어로 16바이트 데이터를 처리할 수 있음을 의미한다. Rank=4 구현에서는 이를 활용하여 모든 데이터를 단일 NEON SIMD 명령어 집합으로 처리하였다. 먼저, 입력 행렬 a와 b의 데이터를 NEON 레지스터에 로드한다. 이 과정에서 a의 16바이트 데이터가 SIMD 레지스터 v0에 로드되고, b의 16바이트 데이터가 v1에 로드된다. 이후 XOR 명령어를 사용하여 두 레지스터의 데이터를 병렬로 연산한다. 이 XOR 연산은 GF(16) 상에서의 덧셈 연산을 수행하며, 각각의 대응 원소를 모듈러 2 방식으로 더하는 결과를 생성한다. 예를 들어, a=[0x0F,0x1E,…,0xFF]와 b=[0x03,0x07,…,0xAA]일 때, XOR 연산을 통해 c=[0x0C,0x19,…,0x55]가 계산된다. 연산이 완료된 결과는 NEON 레지스터 v0에 저장되며, 이를 메모리에 다시 저장하여 결과 행렬 c를 완성한다. 이 과정은 단일 NEON 명령어 집합으로 수행되며, 메모리 접근 횟수를 최소화하고 병렬 처리를 통해 실행 속도를 최적화하였다. Rank=4 구현의 주요 특징은 완전 병렬화된 연산 구조에 있다. ARMv8의 NEON SIMD 명령어는 16바이트 데이터를 한 번에 처리할 수 있으므로, Rank=4 행렬 덧셈은 단일 명령어 집합으로 구현 가능하다. 이는 메모리 접근의 효율성을 극대화하며, 명령어 파이프라이닝과 병렬 처리의 이점을 극대화한다. 또한, 추가적인 일반 레지스터 연산이 필요하지 않아 코드의 간결성과 실행 효율성 모두에서 탁월한 성능을 제공한다. 이와 같은 Rank=4 구현은 병렬화와 데이터 처리 효율성이 가장 잘 결합된 사례로, SNOVA 알고리즘의 성능 최적화에 크게 기여하였다. 특히, 전체 데이터를 단일 NEON 레지스터로 처리함으로써 메모리 대역폭 사용량을 최소화하였으며, Rank=3과 같은 혼합 방식에서 발생할 수 있는 오버헤드를 제거하였다.

3.2 곱셈기 최적화

SNOVA 알고리즘의 주요 연산 중 하나는 유한체 GF(16) 상의 행렬 곱셈이다. 이 연산은 공개키 생성, 서명 생성, 검증 과정에서 가장 큰 계산 자원을 요구하는 핵심 연산으로, 전체 알고리즘 성능을 좌우한다. ARMv8 아키텍처에서는 이 곱셈 연산을 최적화하기 위해 NEON SIMD(Single Instruction Multiple Data) 벡터 연산이 활용되었다. NEON은 여러 데이터를 병렬로 처리할 수 있는 128비트 벡터 레지스터를 제공하므로, GF(16) 행렬 곱셈을 병렬화하여 계산량을 줄일 수 있다.

행렬 곱셈은 두 개의 입력 행렬 A와 B의 행과 열을 순차적으로 처리하여 결과 행렬 C를 생성하는 방식으로 수행된다. 기본적인 연산은 다음과 같다.

$$C[i][j] = \Sigma_k A[i][k] \cdot B[k][j]$$

수식 3-3 행렬 곱셈

여기서 ci, j는 결과 행렬 C의 i-번째 행과 j-번째 열에 해당하는 원소이며, ai k와 bk, j는 각각 입력 행렬 A와 B의 원소를 나타낸다. 이 연산은 각 행렬 원소가 GF(16) 상에서 정의된 곱셈과 덧셈 규칙을 따르는 점에서 일반적인 행렬 곱셈과 차별화된다. GF(16) 상에서의 행렬 곱셈은 다항식 기반 연산을 포함하며, 이는 계산 복잡도를 증가시키는 요인으로 작용한다. 두 다항식을 곱한 결과를 생성 다항식 $f(x)$로 나눈 나머지를 계산해야 하므로, 스칼라 연산 방식으로 처리할 경우 상당한 계산 자원이 요구된다. 그러나 SNOVA에서는 이러한 계산량을 줄이고 성능을 최적화하기 위해 ARMv8 아키텍처의 NEON SIMD(Single Instruction Multiple Data) 벡터 연산을 활용하였다.

NEON SIMD는 128비트 벡터 레지스터를 제공하며, 이는 최대 16개의 8비트 데이터를 병렬로 처리할 수 있다. SNOVA에서는 이러한 벡터 연산의 이점을 활용하여 행렬 곱셈의 병렬화를 구현하였다. 예를 들어, 행렬의 한 행

과 다른 행렬의 열 간의 곱셈 및 누적 연산을 NEON SIMD 명령어를 사용하여 병렬로 수행함으로써 계산 속도를 크게 향상시켰다. 특히, NEON SIMD의 vdupq_n_u8 명령어를 사용하여 개별 행렬 원소를 128비트 벡터로 확장하고, veorq_u8 명령어를 통해 XOR 연산을 수행하는 방식으로 효율적인 곱셈 연산이 가능하도록 하였다. SNOVA의 행렬 곱셈은 행렬의 차원, 즉 rank에 따라 다른 차원의 행렬을 처리하며, 각 차원에 최적화된 방법으로 구현되었다. Rank=2의 경우 2×2 행렬의 총 4개의 요소를 처리하며, NEON SIMD를 활용하지 않고도 효율적으로 계산할 수 있다. 반면, Rank=3 및 Rank=4와 같은 더 큰 차원의 행렬에서는 NEON SIMD의 병렬화 능력이 필수적이다. Rank=4에서는 16개의 행렬 요소를 단일 NEON SIMD 레지스터로 한 번에 처리할 수 있어, NEON SIMD의 성능이 가장 극대화된다.

Algorithm : SNOVA 곱셈기 최적 구현

Input: $A = a0, a1, \ldots, an, B = b0, b1, \ldots, bn$
Output: $C = A \times B$
#Inialization
1: for $i = 0$ *to* n *do*
2: $C[i] \leftarrow 0$
#Multiplication
3: for $i = 0$ *to* n *do*
4: for $i = 0$ *to* n *do*
5. *Multiply* $A[i] \times B[j]$
6. $C[i] \leftarrow C[i] \oplus (A[i] \times B[j])$
#Storing
7: *Store* $C[i] \in$ *memory* for *each* i

[알고리즘 3-2] SNOVA 곱셈기 최적 구현

Rank=2 행렬 곱셈의 최적 구현은 ARMv8 아키텍처의 NEON SIMD 인트린직 명령어를 활용하여 효율적인 연산을 수행하도록 설계되었다. Rank=2의 경우, 행렬 크기가 2×2로 작아 총 4개의 요소를 처리하며, 각 행렬 요소는 GF(16) 상에서 정의된 규칙에 따라 곱셈 및 XOR 연산을 수행한다. 구현

과정은 데이터의 병렬 처리와 메모리 접근 최적화에 중점을 두었다. 입력 행렬 a와 b의 각 요소는 8비트 데이터로 이루어져 있으며, NEON SIMD 명령어를 통해 128비트 벡터 레지스터로 확장된다. Rank=2의 경우 전체 데이터를 단일 벡터 레지스터로 처리하지는 못하지만, 특정 연산 단계에서 병렬화의 이점을 제공한다. 먼저, 각 행렬의 첫 번째 행과 열의 요소가 NEON 레지스터로 로드되며, 이 과정에서 vdupq_n_u8 명령어를 사용하여 단일 8비트 값을 128비트로 확장한다. 이렇게 확장된 벡터는 이후 병렬 곱셈 연산에 활용된다. 곱셈 연산은 vdupq_n_u8 명령어를 통해 확장된 값과 다른 행렬의 벡터 값 간에 병렬로 수행된다. 곱셈 결과는 GF(16) 상에서 정의된 덧셈 규칙에 따라 누적 XOR 연산으로 결합된다. 누적 연산은 veorq_u8 명령어를 사용하여 수행되며, 이 과정에서 각 요소의 독립성이 활용되어 병렬 처리 성능이 극대화된다. 이후 계산된 결과는 vgetq_lane_u8 명령어를 사용해 스칼라 값으로 추출되며, 결과 행렬 c의 대응 위치에 저장된다. 이렇게 계산된 결과는 메모리에 기록되어 Rank=2의 행렬 곱셈이 완료된다. Rank=2 구현에서 주요 최적화는 병렬화된 곱셈, 메모리 접근 최소화, 단순화된 구조를 기반으로 이루어졌다. NEON SIMD를 활용하여 각 행과 열 간의 곱셈을 병렬로 처리함으로써 단일 명령어로 다중 연산을 수행하였다. 이는 실행 시간을 단축하고 처리 효율을 높이는 데 기여하였다. 또한, 데이터가 레지스터에 저장된 상태에서 최대한의 연산이 이루어지도록 설계하여 메모리 접근 횟수를 줄였다. 이는 메모리 병목 현상을 완화하고 실행 성능을 최적화하였다. 더불어, Rank=2의 데이터 크기가 작아 불필요한 복잡성을 줄이고, 간결한 명령어 집합으로 연산을 처리하여 코드 유지보수성을 높이고 실행 시간을 단축하였다. Rank=2 구현은 NEON SIMD의 병렬화 능력을 적절히 활용하여 GF(16) 상에서의 행렬 곱셈을 효율적으로 처리하였다. 병렬화된 곱셈과 XOR 연산은 스칼라 연산 방식에 비해 월등히 높은 성능을 제공하였으나, 데이터 크기가 NEON 레지스터의 최대 용량인 16바이트에 미치지 못하므로 레지스터 활용도가 제한되는 단점이 존재한다. 이러한 한계는 Rank=3 및 Rank=4 구현에서 병렬화 구조를 확장하여 해결되었다.

Rank=3 행렬 곱셈의 최적 구현은 ARMv8 아키텍처의 NEON SIMD 인

트린직 명령어를 활용하여 효율성을 극대화하도록 설계되었다. Rank=3의 경우, 행렬 크기는 3×3으로 총 9개의 요소를 포함하며, 이 행렬 곱셈은 유한체 GF(16) 상에서 정의된 규칙에 따라 수행된다. Rank=2와 비교했을 때 데이터 크기가 증가함에 따라 병렬 처리의 중요성이 더욱 부각되며, NEON SIMD의 활용도 역시 확대된다. Rank=3 구현에서는 3×3 행렬의 각 행과 열 간의 곱셈 및 누적 연산을 효율적으로 처리하기 위해 NEON SIMD 명령어를 활용하였다. 먼저, 입력 행렬 a와 b의 데이터를 로드하여 NEON SIMD 레지스터에 저장한다. vdupq_n_u8 명령어를 사용하여 단일 8비트 데이터를 128비트로 확장하고, 이를 통해 병렬 곱셈 연산을 준비한다. 초기화된 각 행과 열의 요소는 벡터 연산을 통해 병렬로 계산되며, 이 과정에서 veorq_u8 명령어를 활용하여 GF(16) 상에서의 XOR 연산을 수행한다. Rank=3의 경우, 총 9바이트 데이터를 처리해야 하므로, NEON SIMD 레지스터(16바이트) 용량 대비 일부 공간이 남게 된다. 따라서 초기 8바이트는 완전 병렬로 처리되며, 나머지 1바이트는 추가적인 연산을 통해 계산된다. 이를 위해 행렬의 마지막 열 또는 행에 해당하는 데이터를 일반 연산 명령어와 결합하여 처리한다. 이 과정에서 병렬 처리의 장점과 단일 연산의 유연성을 동시에 활용하여 Rank=3에 특화된 최적화를 구현하였다. NEON SIMD를 활용한 Rank=3 구현의 주요 장점은 병렬 처리와 메모리 접근 최적화에 있다. 벡터 연산을 통해 동일한 연산을 병렬로 수행하여 실행 시간을 단축하였으며, 데이터가 레지스터에 저장된 상태에서 연산을 최대한 수행함으로써 메모리 접근 횟수를 줄였다. 특히, vdupq_n_u8과 veorq_u8 명령어를 사용하여 각 행렬 원소 간의 곱셈과 누적 연산을 처리하는 과정에서 병렬화의 효율성을 극대화하였다. 다만, Rank=3 구현은 Rank=4와 비교했을 때 일부 비효율성이 존재한다. NEON SIMD의 16바이트 레지스터 용량에 비해 데이터 크기(9바이트)가 작아 레지스터의 활용도가 완전하지 않으며, 잔여 데이터(1바이트)를 별도의 연산으로 처리해야 하는 추가 비용이 발생한다. 이러한 점은 Rank=4에서 완전 병렬화를 통해 해결되며, Rank=3 구현은 중간 데이터 크기를 처리하기 위한 타협적 접근으로 평가할 수 있다.

Rank=4 행렬 곱셈의 최적 구현은 ARMv8 아키텍처의 NEON SIMD인트

린직 명령어를 활용하여 완전 병렬화된 연산을 실현하였다. Rank=4의 경우, 행렬 크기는 4×4로 총 16개의 요소를 포함하며, 데이터 크기가 NEON SIMD의 128비트 레지스터 용량(16바이트)과 정확히 일치한다. 이를 통해 NEON SIMD의 성능을 최대한 활용할 수 있으며, 연산 효율성과 메모리 접근 최적화가 극대화되었다. Rank=4 구현에서는 입력 행렬 a와 b의 각 행과 열 간의 곱셈 및 누적 연산이 NEON SIMD를 통해 병렬로 수행된다. 먼저, 각 행렬의 데이터를 NEON SIMD 레지스터로 로드하는 과정이 수행된다. 이를 위해 vdupq_n_u8 명령어를 사용하여 단일 8비트 데이터를 128비트 벡터로 확장하고, 이러한 확장된 벡터는 병렬 연산을 수행하는 데 사용된다. a의 행과 b의 열이 대응되며, 각 원소 간의 곱셈은 GF(16) 상에서 정의된 연산 규칙에 따라 계산된다. 곱셈 연산은 NEON SIMD 명령어를 통해 병렬로 수행된다. a와 b의 각 요소는 vdupq_n_u8 명령어로 확장된 후, veorq_u8 명령어를 사용하여 병렬 XOR 연산으로 누적된다. GF(16) 상에서의 XOR 연산은 덧셈 연산에 해당하므로, 누적 XOR 연산을 통해 각 행과 열 간의 곱셈 결과를 계산한다. Rank=4에서는 모든 데이터를 단일 NEON SIMD 명령어 집합으로 처리할 수 있으므로, 별도의 단일 연산 처리나 추가적인 데이터 관리가 필요하지 않다. 계산된 결과는 NEON SIMD 레지스터에 저장되며, vst1q_u8 명령어를 사용하여 메모리에 기록된다. 각 행렬의 계산 결과는 메모리에 저장된 이후 Rank=4 행렬 곱셈이 완료된다. 이와 같은 병렬화된 연산 구조는 Rank=4 데이터 크기가 NEON SIMD 레지스터 용량과 일치하기 때문에, NEON SIMD의 성능을 최대한 활용할 수 있도록 설계되었다. Rank=4 구현의 주요 최적화는 완전 병렬화, 메모리 접근 최소화, 그리고 데이터 크기와 NEON SIMD 용량 간의 완전한 일치를 기반으로 한다. NEON SIMD를 활용하여 전체 데이터를 한 번에 로드하고 저장할 수 있어 메모리 접근 횟수가 최소화되었으며, 이는 실행 속도를 크게 향상시키는 결과를 낳았다. 또한, 128비트 레지스터를 사용하여 각 행렬 원소 간의 병렬 연산을 수행함으로써 실행 효율성을 극대화하였다. Rank=4 구현은 NEON SIMD의 모든 병렬 처리 능력을 활용한 사례로, SNOVA 알고리즘의 행렬 곱셈 최적화를 완성하는 핵심 단계로 자리 잡는다. 특히, Rank=3 및 Rank=2와 비교했

을 때 추가적인 연산 비용이 없으며, 병렬화 성능이 가장 극대화된 구현으로 평가된다.

3.3 AES 가속기

AES-128 CTR 모드는 SNOVA 전자서명 알고리즘에서 난수 생성을 위한 주요 암호화 구성 요소로 사용된다. ARMv8 아키텍처는 AES 연산을 가속하기 위한 하드웨어 명령어 집합을 지원하여 고속 암호화를 가능하게 하지만, 이를 효율적으로 사용하기 위해서는 올바른 AES 키 스케줄 구현이 필수적이다. 본 연구에서는 ARMv8 하드웨어 명령어와 직접 호환되는 표준 AES-128 키 스케줄을 어셈블리 코드로 구현하였다. 이는 SNOVA 알고리즘의 성능을 최적화하고 기존의 BearSSL 기반 키 스케줄 구현에서 발생할 수 있는 비호환성 문제를 해결하기 위한 것이다.

BearSSL은 소프트웨어 이식성과 보안성을 목표로 설계된 라이브러리로, AES 암호화를 비트슬라이스(bit-sliced) 방식으로 처리한다. 이 방식은 복수의 입력 블록을 병렬로 처리할 수 있는 장점을 제공하지만, ARMv8 하드웨어 명령어 AESE 및 AESMC와 직접 호환되지 않는다. BearSSL에서 생성되는 AES 라운드 키는 비트슬라이스 형식으로 저장되며, 이는 전통적인 AES 라운드 키 배열과 구조적으로 다르다. 따라서 하드웨어 가속 명령어와 결합하여 사용할 수 없다.

ARMv8 AES 명령어를 활용하려면 총 11개의 16바이트 라운드 키가 표준 AES 키 스케줄 알고리즘을 통해 생성되어야 한다. 이를 위해 본 연구에서는 AES-128 키 스케줄을 어셈블리 언어로 직접 구현하였다. 초기 입력 키를 첫 번째 라운드 키로 설정한 후, 표준 AES 키 확장 과정을 반복 수행하여 남은 10개의 라운드 키를 생성한다. 주요 연산은 키 워드의 회전(RotWord), S-box 치환(SubWord), 라운드 상수(Rcon) 적용, XOR 연산 등으로 구성되며, 각 라운드 키는 이러한 연산을 거쳐 계산된다.

Algorithm : AES-128 CTR모드 가속기 구현

Input: $P = \{p_0, p_1, \ldots, p_n\}$ (Plaintext), K (AES Key), N (Nonce)
Output: $C = \{c_0, c_1, \ldots, c_n\}$ (Ciphertext)

Initialization:
Load the AES round keys $R[i]$ using the standard AES key schedule.
Initialize the counter block Ctr with the given nonce N.
Encryption Loop:
for $i = 0$ **to** $n - 1$ **do**
    Load the counter block Ctr$[i]$ into a NEON vector register.
    Encrypt Ctr$[i]$ using the AES rounds:
    for $j = 0$ **to** 9 **do**
        Apply AESE$(R[j])$ and AESMC$(R[j])$.
    end for
    Apply the final round AESE$(R[10])$.
    XOR the encrypted counter block with the plaintext block:
    $C[i] = P[i] \oplus$ Encrypted Ctr$[i]$.
end for
Counter Update:
Increment the counter: Ctr$[i + 1]$ = Ctr$[i] + 1$.
Output:
Store each ciphertext block $C[i]$ in memory.

[알고리즘 3-3] AES-128 CTR모드 가속기 구현

AES 가속기 구현은 ARMv8의 벡터 레지스터와 AES 전용 명령어를 결합하여 최대 성능을 발휘할 수 있도록 설계되었다. AESE 명령어는 AES 라운드 암호화를 수행하고, AESMC 명령어는 AES 믹스 컬럼(MixColumns) 연산을 적용한다. 16바이트 블록이 처리 단위이며, 초기값은 넌스(nonce)로 설정되며 이후 암호화 루프를 통해 블록 단위로 증분된다.

암호화 루프는 넌스 값을 벡터 레지스터에 로드하고, 이를 AES 라운드 키와 결합하여 AESE 및 AESMC 명령어로 반복 처리한다. 각 라운드는 라운드 키와 XOR 연산을 수행한 후 S-box와 믹스 컬럼 연산을 적용한다. 마지막 라운드에서는 믹스 컬럼을 제외하고 암호문이 생성된다. 결과는 출력 버퍼에 저장되며, 카운터 값은 넌스 값에 증분되어 다음 블록 암호화를 준비한다.

이러한 구현은 메모리 접근 수를 최소화하고 암호화 루프 내에서 모든 연산이 벡터 레지스터에서 수행되도록 최적화되었다. AES 라운드 키는 초기화

시 미리 계산되어 레지스터에 로드되므로, 매 암호화 라운드마다 불필요한 메모리 접근이 제거된다. 이는 기존의 소프트웨어 AES 구현과 비교해 큰 성능 향상을 가능하게 한다.

IV. 성능 평가

4.1 성능 측정 환경

SNOVA 알고리즘의 ARMv8 아키텍처 상에서의 최적화 구현 성능을 평가하기 위해 벤치마크를 수행하였다. 성능 평가는 ARM 플랫폼에서 PQC 알고리즘의 성능을 테스트하고 분석하기 위해 설계된 mupq 프레임워크를 기반으로 진행되었다. mupq를 선택한 이유는 타 알고리즘과의 형평성을 유지하기 위함이다.

mupq는 PQC 알고리즘을 ARM Cortex-M 및 ARMv8과 같은 저전력 임베디드 환경에서 구현하고 최적화할 수 있는 프레임워크로, 알고리즘의 실행 시간 및 자원 사용량 분석을 위한 기능을 제공한다.

NIST에서 제공하는 참조 코드는 OpenSSL과 같은 특정 라이브러리를 사용하며, 이는 ARMv8 환경과 최적화 수준이 반드시 일치하지 않을 수 있다., 이는 추후 타 알고리즘과 성능 비교 시 공정성을 해칠 가능성이 있다. 반면, mupq는 ARM 아키텍처에 특화된 벤치마크 환경을 제공하므로, 동일 조건에서 타 알고리즘과의 성능 비교를 할 수 있다.

벤치마크 대상은 주요 암호화 작업인 키 생성(crypto_sign_keypair), 서명(crypto_sign), 그리고 **서명 검증(crypto_sign_verify)**으로, 해당 작업들의 실행 사이클을 측정하였다. 성능 측정 환경은 Apple M2 칩(8GB RAM) 하드웨어를 기반으로 구성되었으며, GCC 14.0.3 컴파일러를 사용하여 O3 최적화 레벨로 빌드하였다. 소프트웨어 개발 환경으로는 Visual Studio Code(VSCode)를 사용하였다. 추가적으로, 최적화된 덧셈 및 곱셈 모듈의 성능을 측정하기 위해 사용자 정의 벤치마크 스크립트를 개발하였다.

4.2 성능 측정 결과

최적화된 SNOVA 구현의 성능 평가는 esk(Extended Secret Key)와 ssk(Secret Signing Key) 두 가지 파라미터 세트에 대해 이루어졌다. SNOVA 알고리즘은 총 18개의 파라미터 세트를 포함하며, 이 중 9개는 esk 파라미터 세트, 나머지 9개는 ssk 파라미터 세트로 구성된다. 두 파라미터 세트는 각각의 특성과 목적에 따라 설계되었다.

esk는 확장된 비밀 키로, ssk에서 추가 연산을 통해 생성된 고급 형태이다. 이는 보조 데이터나 사전 계산된 값 등을 포함하며, 성능 최적화를 위해 서명 생성 시 빠르게 접근할 수 있는 구조를 제공한다. 그러나 추가적인 연산과 데이터 구조로 인해 계산이 더 복잡하고 느릴 수 있다. 반면, ssk는 서명 생성에 직접 사용하는 기본 비밀 키로, 연산 과정이 단순하고 가볍게 설계되어 있다. ssk는 주로 서명 생성의 핵심 키로 사용되며, esk 생성의 입력값으로 활용된다.

[표 4-1] esk 파라미터 최적 구현 성능 측정 결과

파라미터	Work	레퍼런스	최적 구현
snova-24-5-16-4-esk	키 생성	3,475,085	1,398,483
	서명 생성	15,753,137	10,707,076
	서명 검증	10,629,480	5,898,839
snova-25-8-16-3-esk	키 생성	3,463,081	2,050,620
	서명 생성	6,484,644	6,007,548
	서명 검증	3,158,913	2,843,025
snova-28-17-16-2-esk	키 생성	5,557,774	4,370,283
	서명 생성	2,144,271	2,127,047
	서명 검증	2,282,003	1,330,599
snova-37-8-16-4-esk	키 생성	18,112,642	8,641,329
	서명 생성	59,263,911	41,794,416
	서명 검증	40,590,714	22,410,705
snova-43-25-16-2-esk	키 생성	24,795,480	21,714,931
	서명 생성	7,117,074	7,062,414
	서명 검증	7,609,277	4,451,052
snova-49-11-16-3-esk	키 생성	17,887,340	13,837,133
	서명 생성	23,711,616	23,494,859
	서명 검증	14,493,366	11,888,327
snova-60-10-16-4-esk	키 생성	69,097,561	33,384,333
	서명 생성	168,794,828	115,369,891
	서명 검증	122,557,890	67,624,592
snova-61-33-16-2-esk	키 생성	83,359,185	73,577,806
	서명 생성	17,722,885	17,651,531
	서명 검증	19,220,693	11,221,976
snova-66-15-16-3-esk	키 생성	57,863,043	46,400,037
	서명 생성	60,121,205	60,103,122
	서명 검증	35,164,753	31,463,332

esk 파라미터 세트의 성능 측정 결과는 위 표와 같다. Rank 4에서 모든 작업에서 가장 높은 성능 향상이 관찰되었다. 키 생성 작업에서는 약 51.7~65.2%의 성능 향상을 기록하였고, 서명 생성 작업에서는 최대 36.7%, 서명 검증 작업에서는 44.8~50.0%의 성능 향상이 있었다. 이는 Rank 4가 ARMv8의 NEON 벡터 연산을 완전히 병렬로 활용할 수 있기 때문이다. 반면, Rank 2와 Rank 3의 경우 성능 향상 폭은 상대적으로 낮았다. Rank 2는 연산 요소가 적어 NEON 벡터 연산의 병렬 처리 장점을 충분히 활용하지 못

하며, Rank 3는 9개의 행렬 요소를 처리해야 하는 특성상 NEON 레지스터의 8바이트 단위 병렬 처리 구조와 비효율적인 정렬로 인해 성능이 제한되었다. 이에 따라 Rank 2와 3에서는 키 생성에서 약 20~30%, 서명 검증에서는 약 10~25%의 성능 향상이 나타났다. 특히, 서명 생성 작업에서는 Rank 4 대비 성능 개선이 거의 없거나 미미한 수준으로 관찰되었다.

[표 4-2] ssk 파라미터 최적 구현 성능 측정 결과

파라미터	Work	레퍼런스	최적 구현
snova-24-5-16-4-ssk	키 생성	3,162,043	1,398,307
	서명 생성	15,532,906	10,813,358
	서명 검증	10,665,857	5,911,721
snova-25-8-16-3-ssk	키 생성	2,964,322	2,047,839
	서명 생성	5,934,865	5,904,548
	서명 검증	3,777,688	2,895,751
snova-28-17-16-2-ssk	키 생성	5,839,063	4,363,678
	서명 생성	2,134,444	2,126,962
	서명 검증	2,270,131	1,335,159
snova-37-8-16-4-ssk	키 생성	20,797,176	8,641,214
	서명 생성	59,903,432	41,767,647
	서명 검증	40,680,194	22,403,252
snova-43-25-16-2-ssk	키 생성	26,863,060	21,603,320
	서명 생성	7,011,836	7,001,149
	서명 검증	7,617,343	4,451,764
snova-49-11-16-3-ssk	키 생성	19,166,980	13,781,456
	서명 생성	23,715,980	23,580,833
	서명 검증	14,569,760	12,809,535
snova-60-10-16-4-ssk	키 생성	69,064,963	33,406,701
	서명 생성	168,849,382	115,361,855
	서명 검증	122,586,802	67,590,588
snova-61-33-16-2-ssk	키 생성	82,685,800	73,589,368
	서명 생성	17,421,722	17,250,398
	서명 검증	19,222,942	11,213,860
snova-66-15-16-3-ssk	키 생성	57,911,107	46,343,915
	서명 생성	60,674,453	59,117,498
	서명 검증	35,254,524	31,426,092

ssk 파라미터 세트는 상대적으로 간단한 구조를 가지며, 성능 측정 결과가

esk와 비슷한 경향을 보였으나 조금 더 빠른 성능을 보이는 것으로 확인되었다. esk와 마찬가지로 Rank 4에서 가장 높은 성능 향상이 나타났으며, 키 생성 작업에서는 약 53.8~68.7%, 서명 생성 작업에서는 29.5%, 서명 검증 작업에서는 42.8~47.2% 정도의 성능 향상이 있었다. Rank 2와 Rank 3에서도 성능 향상이 있었지만, Rank 4만큼의 개선은 이루어지지 않았다. Rank 2는 병렬화 효율이 제한되었고, Rank 3는 구조적 제약으로 인해 성능 개선이 비교적 낮았다. 키 생성에서 약 25%, 서명 검증에서 약 15~20%의 성능 향상이 관찰되었으며, 서명 생성에서는 Rank 4 대비 개선 폭이 크지 않았다.

SNOVA의 esk와 ssk 파라미터 세트는 모두 최적화된 구현을 통해 성능 향상을 이루었으며, 특히 Rank 4에서 가장 높은 성능 향상이 관찰되었다. 이는 NEON 병렬 처리 기능을 최대한 활용한 결과로 분석된다. 반면, Rank 2와 Rank 3는 병렬화 효율의 제약으로 인해 성능 향상 폭이 상대적으로 낮았다. esk와 ssk의 성능 차이가 미미한 이유는 ssk의 단순한 구조와 esk의 추가 연산이 성능에 미치는 영향을 상쇄하기 때문으로 보인다.

V. 결 론

본 연구에서는 ARMv8 아키텍처 상에서 SNOVA 전자 서명 알고리즘의 최적 구현을 제안하고, 성능 최적화를 통해 주요 작업인 키 생성, 서명 생성, 서명 검증에서의 실행 시간을 크게 단축할 수 있음을 입증하였다. 연구의 주요 초점은 ARMv8의 하드웨어 가속 기능과 NEON SIMD 명령어를 활용하여 SNOVA 알고리즘의 핵심 연산인 행렬 덧셈, 곱셈, 그리고 AES-CTR 모드를 효과적으로 병렬화하는 데 있었다. 이를 통해 기존의 참조 구현 대비 모든 파라미터 세트에서 높은 성능 향상을 달성하였다.

특히, Rank 4의 파라미터 세트에서 가장 두드러진 성능 향상이 관찰되었다. 이는 NEON 벡터 연산이 완전히 병렬로 활용되어 병렬 처리 효율이 극대화된 결과로 분석된다. 반면, Rank 2와 Rank 3에서는 구조적 제약과 병렬화 한계로 인해 성능 향상이 상대적으로 낮게 나타났다. 또한, esk와 ssk 파라미터 세트 간의 성능 차이는 미미하였으며, 이는 esk의 복잡성이 성능에 미치는 영향을 상쇄하기 때문으로 해석된다.

SNOVA의 최적 구현은 기존 PQC(Post-Quantum Cryptography) 알고리즘 최적화 연구에서 부족했던 ARMv8 아키텍처 기반 다변수 공개키 암호(MPKC) 시스템의 연구 공백을 채우는 중요한 기여를 하였다. 본 연구 결과는 SNOVA 알고리즘이 제한된 자원 환경에서 실행 가능하며, 모바일 및 IoT 디바이스와 같은 저전력 고효율 플랫폼에 적합한 암호화 시스템으로 자리 잡을 가능성을 제시한다.

참 고 문 헌

1. 국외문헌

National Institute of Standards and Technology (NIST), “Post-Quantum Cryptography Standardization,” https://csrc.nist.gov/projects/pqc-dig-sig/round-1-additional-signatures, 2024.

Kipnis, A., and Shamir, A., “Cryptanalysis of the Oil & Vinegar Signature Scheme,”

Lecture Notes in Computer Science, vol. 1294, pp. 257-266, 1997.

SNOVA, “Proposal for NISTPQC: Digital Signature Schemes project,” May 25, 2023.

ARM Holdings, “ARM Architecture Reference Manual, ARMv8,” ARM Limited, 2021.

Apple Inc., “Apple M2 Chip,” https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities, 2022.

Bernstein, D. J., Buchmann, J., & Dahmen, E. (Eds.). (2009). Post-Quantum Cryptography. Springer Science & Business Media.

Goedhart, L. D., & Pollard, D. J. (2017). Efficient cryptographic algorithms for ARMv8 architecture. Journal of Cryptographic Engineering, 7(2), 143-153.

Hyukdong Kwon, Kyungjoo Song, Minju Sim, Minwoo Lee, & Hwajung Seo (2023). "High-Performance Implementation of SMAUG on 64-bit ARMv8 Processors." Proceedings of the Korea Information Processing Society Conference, 113-115.

Donghyun Kim, Dongwoo Kim, Sangjin Lee, & Seokjong Seo (2021). "Optimized Implementation of Toom-Cook Algorithm in NIST PQC SABER Using ARM/NEON Processors." Journal of Information Security and Cryptography, 31(5), 1057-1068.

Langlois, A., & Stehlé, D. “Worst-case to Average-case Reductions for Module Lattices.” Journal of Cryptographic Engineering, vol. 10, no. 2, pp. 155-174, 2020.

Seo, H., & Kim, M. “Optimized Implementation of PQC Schemes on ARM Platforms.” IEEE Transactions on Information Forensics and Security, vol. 15, no. 6, pp. 980-994, 2022.

Courtois, N. T., & Goubin, L. “Efficient Multiplications in GF(2^n) with Applications to Cryptographic Protocols.” Advances in Cryptology - EUROCRYPT, pp. 392-405, 2018.

Duong, L., & Nguyen, P. Q. “Lattice-based Post-Quantum Cryptography: Implementations and Security Analysis.” Proceedings of IEEE Symposium on Security and Privacy, pp. 1-12, 2021.

Beullens, J., & Vercauteren, F. “Efficient Arithmetic in Finite Fields for Post-Quantum Cryptography.” Journal of Cryptology, vol. 14, no. 3, pp. 233-256, 2020.

Arm Limited, “Optimizing Cryptographic Algorithms Using NEON on ARMv8,” Technical Report, 2022.

Kim, S., & Lee, J. “Efficient Modular Arithmetic for Post-Quantum Cryptography in ARMv8-based Systems.” IEEE Transactions on Computers, vol. 69, no. 4, pp. 731-743, 2022.

NIST PQC Team, “Finalists in Post-Quantum Cryptography Standardization.” NIST Official Report, 2023.

Seo, M., & Kim, J. “Advanced Key Management for Quantum-Resistant Cryptographic Algorithms.” International Journal of Applied Cryptography, vol. 19, no. 4, pp. 512-534, 2023.

PQClean. "Clean, Portable, Tested Implementations of Post-Quantum Cryptography.", 2024.

mupq. "Post-Quantum Cryptographic Implementations for Microcontrollers.", 2024.

Kannwischer, M. J., Rijneveld, J., Schwabe, P., & Stoffelen, K. "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4." IACR Cryptology ePrint Archive, 2019, 844.

Kipnis, A., & Shamir, A. "Cryptanalysis of the Oil & Vinegar Signature Scheme." In Lecture Notes in Co ㅇmputer Science, Vol. 1294, pp. 257-266. 1997.

ABSTRACT

Optimal Implementation of SNOVA on ARMv8 Architecture

Lee, Min-Woo

Major in Convergence Security

Dept. of Convergence Security

The Graduate School

Hansung University

This thesis focuses on the optimal implementation of the SNOVA digital signature algorithm on the ARMv8 architecture. SNOVA, a post-quantum cryptographic system, ensures high security based on multivariate quadratic equations. In this study, assembly language and NEON SIMD instructions were employed to optimize key generation, signature creation, and signature verification operations in SNOVA. The main optimizations include the implementation of addition and multiplication modules for finite field operations, along with an AES-128 CTR mode accelerator. The addition and multiplication operations were designed for parallel processing using ARMv8 vector registers, while the AES accelerator maximized encryption speed through hardware instructions. Performance evaluations revealed significant improvements in key cryptographic tasks. The optimized implementation achieved up to

68.7% improvement in key generation, 36.7% in signature creation, and 50.0% in signature verification for major SNOVA parameter sets (esk and ssk). This research demonstrates the feasibility of implementing multivariate-based digital signature algorithms on ARMv8 and provides a foundational reference for future development of efficient post-quantum cryptographic systems.

KEYWORD: NIST, PQC, SNOVA, ARMv8, GF Operations Optimization, Parallel Implementation