ARMv8 상에서의 격자 기반 암호 최적 구현 연구 동향

유튜브 주소: https://youtu.be/ztHtzoszAcQ

격자기반 양자내성암호 개요

ARMv8 아키텍처 및 NTT

격자 기반 암호 알고리즘 구현 사례 분석

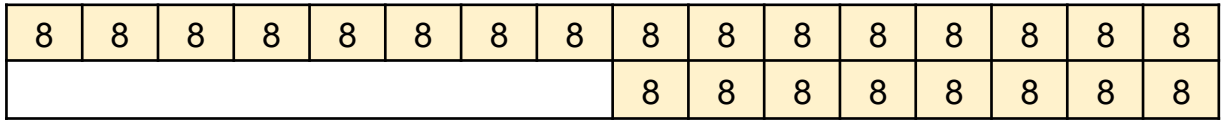
격자기반 양자내성암호 개요

- 양자 컴퓨터의 발전으로 **기존의 공개키 암호체계 무력화 예상**
  - Shor 알고리즘은 이산 대수 문제를 효율적으로 해결 가능

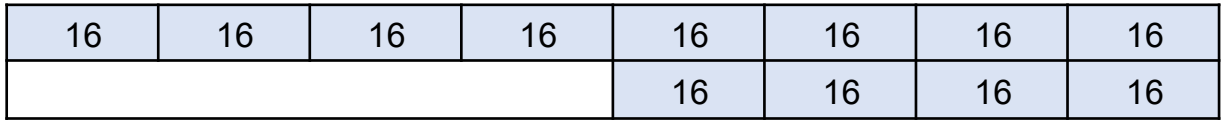
  -> RSA, ECC 등 무력화 가능
- 이에 NIST에서 양자내성암호 공모전 수행
  - **최종 표준은 격자 기반 암호 위주로 선정됨**
- 양자 내성 암호 중 **격자 기반 방식은 가장 효율적인 후보군으로 평가**
  - 수학적 구조 안전성이 높고, 고속 연산이 가능한 이점 보유
  - NTT를 활용한 고속 다항식 곱셈 연산이 주요 연산
- **ARMv8 아키텍처-> PQC 구현에 적합한 플랫폼으로 평가**
  - 64-bit 명령어 및 NEON SIMD 벡터 연산을 통한 병렬 연산 가능
  - 리소스가 제한적인 모바일, 임베디드 환경 등에서 효율적인 구현 가능
- **이에 ARMv8 상에서 격자 기반 암호 최적화 연구 동향 조사**

ARMv8 아키텍처

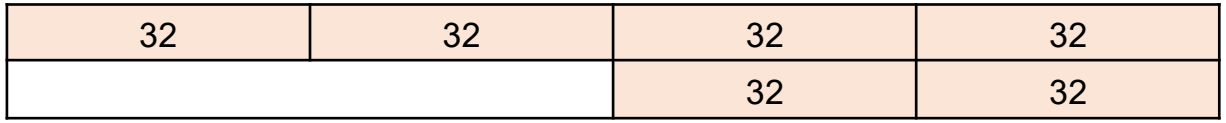
- ARMv8: 2011년에 도입된 64-bit 기반의 RISC 아키텍처
  - 고성능·저전력 설계를 목표로 개발된 최신 ARM ISA(Instruction Set Architecture)
- **64-bit 명령어 및 NEON SIMD 벡터 연산 지원**
  - 64-bit 범용 레지스터 31개 제공
  - **128-bit 벡터 레지스터 32개 제공**
- **SIMD 명령어를 통해 연산 병렬 처리 가능**
  - 덧셈, 곱셈 연산 등 반복 구조 연산의 병렬 처리 가능
  - **데이터 크기에 맞게 레지스터 재구성 가능 -> 데이터 특성에 맞게 병렬 연산 가능**



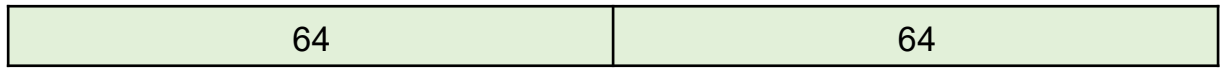
16B / 8B (Byte, 8-bit)



8H / 4H (Half Word, 16-bit)



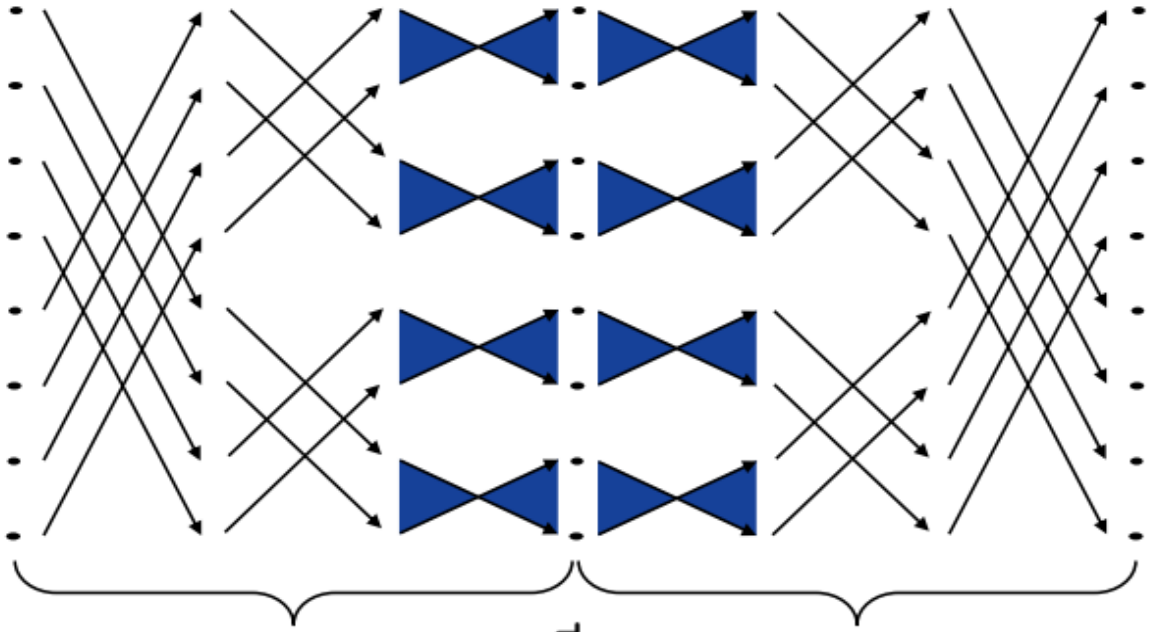
4S / 2S (Word, 32-bit)



2D (Double Word, 64-bit)

NTT(Number-Theoretic Transform)

- NTT: 이산 푸리에 변환을 유한체 위에서 수행한 형태
  - **주로 격자 기반 암호의 고속 다항식 곱셈 연산에 활용됨**
- **NTT는 다항식 곱셈을 $O(n^2) \rightarrow O(n \log n)$으로 가속화 가능**



버터플라이 연산

NTT 기본 연산 단위

Forward: Cooley-Tukey(CT) 알고리즘

- 입력 다항식(시간 영역)을 주파수 영역의 데이터로 변환

Inverse: Gentleman-Sande(GS) 알고리즘

- 주파수 영역으로 변환된 데이터를 시간 영역의 다항식으로 재 변환

Crystals-Dilithium on ARMv8

- ARMv8 플랫폼 상에서 **Crystals-Dilithium**을 최적화한 연구 사례
  - Dilithium: NIST PQC 공모전 표준으로 선정된 격자 기반 전자서명 알고리즘
  - Module-LWE 문제 기반
- 대상 플랫폼: Jetson Xavier(ARMv8.2)
- 주요 최적화 전략
  - **Merging:** 불필요한 메모리 접근 감소
  - **Register Holding:** 레지스터 내 값 보존
  - **Interleaving:** ARM core + NEON 병렬 사용

Dilithium-3	기존 (Cycle)	최적화 (Cycle)	향상률
KeyGen	253,395	176,174	**+43.83%**
Sign	1,257,304	589,567	**+113.25%**
Verify	255,421	179,971	**+41.92%**

Crystals-Dilithium on ARMv8 최적화 기법

- Dilithium의 핵심 연산인 **NTT/InvNTT/Point-wise에 대해 최적화 적용**

핵심 연산	최적화 기법 설명
NTT (정방향)	- NEON 병렬화를 위한 Butterfly task-parallelism 적용 - Depth 0–2는 Merging 방식으로 메모리 접근 최소화 - Depth 3–7는 vector register holding 방식으로 처리
Inverse NTT (역변환)	- NTT와 유사한 방식의 병렬화 적용 - Register holding 및 interleaving 방식 적용
Point-wise 곱셈	- 다항식 4개 계수 병렬 처리 - ARM core는 2계수 처리(interleaving) - Montgomery reduction에 barrel shifter, SMLSL 명령 활용

HAETAE on ARMv8

- ARMv8 플랫폼 상에서 **HAETAE 알고리즘**을 최적화한 연구 사례
  - HAETAE: KpqC 표준으로 선정된 격자 기반 전자서명 알고리즘
  - LWE 및 SIS 문제 기반, Fiat-Shamir with Aborts 구조
- 대상 플랫폼: Apple M1 Processor
- 주요 최적화 전략
  - **NEON 명령어(SQDMULH, ZIP, TRN 등) 활용한 SIMD 최적화**
  - 데이터 재정렬과 사전 계산을 통한 메모리 접근 최소화

HAETAE-180	기존 (Cycle)	최적화 (Cycle)	향상률
KeyGen	1,502	1,295	**+13.78%**
Sign	4,198	3,721	**+11.36%**
Verify	694	545	**+21.47%**

HAETAE on ARMv8 최적화 기법

- **NTT, Inverse-NTT, Montgomery** 연산에 대해 최적화 적용

핵심 연산	최적화 기법
NTT	- NEON 벡터 레지스터를 이용한 병렬화 - ZIP1/ZIP2, TRN1을 이용해 데이터 정렬 - SQDMULH를 활용하여 모듈러 곱셈 고속화
Inverse-NTT	- NTT와 유사한 방식의 병렬화 적용 - 사전 계산된 역 zeta 값 활용 기법 적용 - 레지스터 재배열 및 조건 분기 최소화 기법 적용
Pointwise Montgomery 곱	- SQDMULH 및 SHSUB 명령어를 통한 곱셈+모듈러 연산 통합 - 4개 계수를 한 번에 처리(병렬화) 기법 적용 - 고정 레지스터 내 q, q^{-1} 관리를 통해 반복 최소화

Fast Falcon Signature Generation and Verification Using ARMv8 NEON Instructions

- ARMv8 플랫폼 상에서 **Falcon 알고리즘**을 최적화한 연구 사례
  - FALCON: NIST PQC 공모전 표준으로 선정된 격자 기반 전자서명 알고리즘
  - Ring-LWE 문제 기반, Fast Fourier sampling과 NTRU 구조 결합
- 대상 플랫폼: Apple M1 Processor
- 주요 최적화 전략
  - 실수 기반 **FFT와 Gaussian sampling을 NEON으로 병렬화**
  - **계수 재배열과 루프 unrolling**을 통해 캐시 접근 최적화
  - floating-point 연산에서 **VLD1/VST1 활용**해 데이터 대역폭 개선

falcon-512	기존 구현 (kc)	최적 구현 (kc)	향상률
Sign	654.0	459.2	**+42%**
Verify	43.5	22.7	**+92%**

Falcon on ARMv8 최적화 기법

- **FFT, NTT, Gaussian Sampling**에 대해 최적화 적용

핵심 연산	최적화 기법
FFT (Fast Fourier Transform)	- Iterative 방식의 Forward/Inverse FFT 구현 - NEON SIMD 벡터화 적용 - 루프 언롤링 및 버터플라이 연산 묶기로 병렬성 증가
Twiddle Factor Table	- 대칭성, 회전성, 켤레 복소수 활용하여 4배 압축 저장(병렬화) -> 메모리 접근 최적화 및 캐시 효율 증가
NTT	- Signed Barrett 곱셈 방식 적용하여 연산 범위 축소 - 각 레벨마다 Barrett 연산을 최소화하도록 값 범위 분석
Gaussian Sampling	- Serial Gaussian 샘플링을 vectorized FFT와 결합 - 즉, 연속 샘플링 성능 향상
Floating-Point 연산	- ARMv8.3의 fcmla, fcadd 명령 활용 - 정확성 보장 위해 fmul, fadd 방식 선택 가능 (rounding 보완)

Optimizing HAWK Signature Scheme Performance on ARMv8

- ARMv8 플랫폼 상에서 **HAWK 알고리즘**을 최적화한 연구 사례
  - 격자 동형 문제(Lattice Isomorphism Problem, LIP)기반 전자서명 알고리즘
- 대상 플랫폼: Apple M1 Processor
- 주요 최적화 전략 요약
  - 프로파일링 분석 기반 병목 연산 탐색 및 집중 최적화
  - **ARMv8 특화 명령어 활용**(sbfx, csel, madd)으로 명령어 수 감소
  - **NTT 병렬 구현**: log n 값에 따라 선택적으로 적용하여 성능 개선 극대화

HAWK-512	기존 (Cycle)	최적화 (Cycle)	향상률
KeyGen	8,352,475 cycles	8,125,339 cycles	**+2.72%**
Sign	180,094 cycles	179,441 cycles	**+0.36%**
Verify	158,098 cycles	157,985 cycles	**+0.07%**

HAWK on ARMv8 최적화 기법

- **NTRUSolve, Rebuild_CRT, NTT**에 대해 최적화 적용

핵심 연산	적용된 최적화 기법
Modular reduction	- sbfx 명령어를 이용해 LSB 추출 및 sign-extension을 단일 명령어로 처리 - csel 명령어를 활용해 분기 없이 조건별 결과 선택 → branchless constant-time 구현
Polynomial multiplication	- madd 명령어를 이용해 곱셈과 덧셈을 하나의 명령으로 실행 → 명령어 수 감소 및 클럭 사이클 단축
NTT	- AArch64 SIMD 벡터 레지스터를 활용한 병렬 처리 구조 설계 - log n ≥ 8 이상부터 병렬화 적용 - 작은 n에 대해선 reference NTT 유지로 성능 저하 방지
Gaussian sampling & SHAKE-based randomness	- 서명 연산의 Gaussian 샘플링 및 SHAKE 함수가 성능 병목 원인으로 확인 - SHAKE 최적화는 향후 성능 개선 여지로 제시됨

Q & A