# 64-bit ARMv8 프로세서 상에서의 KpqC 후보 알고리즘 SMAUG의 고속 구현

**권혁동**[†], 송경주[†], 심민주[†], 이민우[‡], 서화정[‡]

[†]한성대학교 정보컴퓨터공학과

[‡]한성대학교 융합보안학과

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

KpqC 공모전

양자내성암호 SMAUG

제안 기법

성능 평가

결론

# KpqC 공모전

- 미국의 국립표준기술연구소에서는 양자내성암호 표준을 선정
  - 공개키 암호화 1종, 전자서명 3종
  - Round 4 진행 중
- 국내 표준을 선정하기 위한 KpqC 공모전이 개최
  - 22년 12월 공개키 암호화 7종, 전자서명 9종이 Round 1에 진출

| KpqC 일정 | |
|---|---|
| 2021.11.25 | 공모전 공지 |
| 2022.02.18 | 제출 마감 |
| 2022.03.15 | 1차 평가 |
| 2023.12 | Round 1 결과 발표 |
| 2024.03 | Round 2 결과 발표 |
| 2024.09 | KpqC 표준 발표 |

| Base | Code | Lattice | MQ | Hash | ZK |
|---|---|---|---|---|---|
| PKE-KEM | IPCC | NTRU+ | - | - | - |
| | ROLLO | SMAUG | | | |
| | PALOMA | TiGER | | | |
| | REDOG | | | | |
| Digital Signature | pqsigRM | GCKSign | - | FIBS | AIMer |
| | | HAETAE | | | |
| | | NCC-Sign | | | |
| | | MQ-Sign | | | |
| | | Peregrine | | | |
| | | SOLMAE | | | |

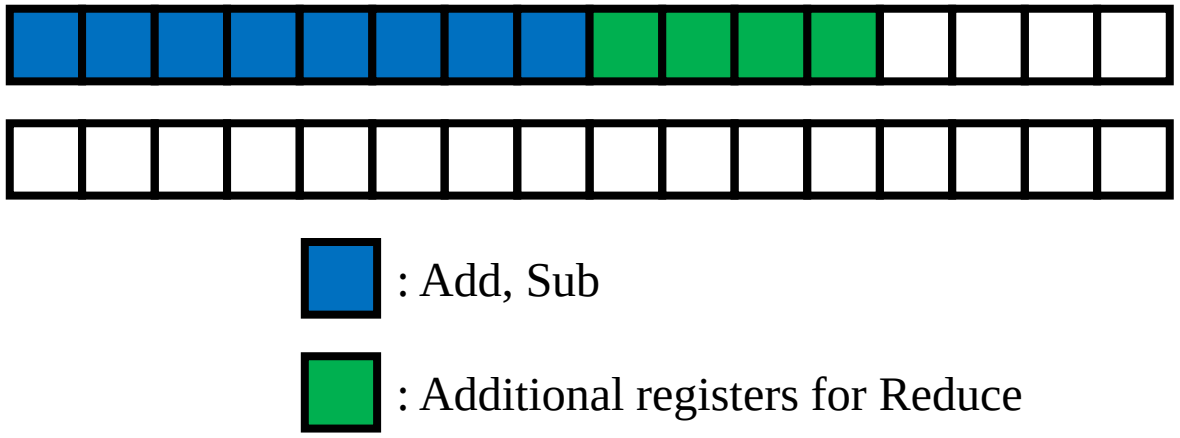
# 양자내성암호 SMAUG

- 격자 기반 암호 알고리즘
  - Module-Learning with Errors(M-LWE)
  - Module-Learning with Rounding(M-LWR)
  - Lizard, Ring-Lizard의 구조를 추종
  - 해시함수 약간 사용: XOF로 SHAKE-128, KDF로 SHAKE-256
- **NTT를 사용하지 않음**
  - **Modulus를 작게하여 곱셈 부하를 줄임**
  - **Moduli를 2의 배수로하여 modular reduction을 시프트로만 연산**
- 격자 기반 알고리즘 특성상 복호화 실패 가능성이 존재
  - 매개변수 조절을 통해 오류 발생 확률을 낮춤

| Scheme | $n$ | $k$ | $(p, q)$ | Secret key $(byte)$ | Public key $(byte)$ | Ciphertext $(byte)$ |
|---|---|---|---|---|---|---|
| SMUAG128 | 256 | 2 | (256, 1024) | 174 | 672 | 768 |
| SMUAG192 | 256 | 3 | (256, 1024) | 185 | 992 | 1024 |
| SMUAG256 | 256 | 5 | (256, 1024) | 182 | 1632 | 1536 |

# 제안 기법

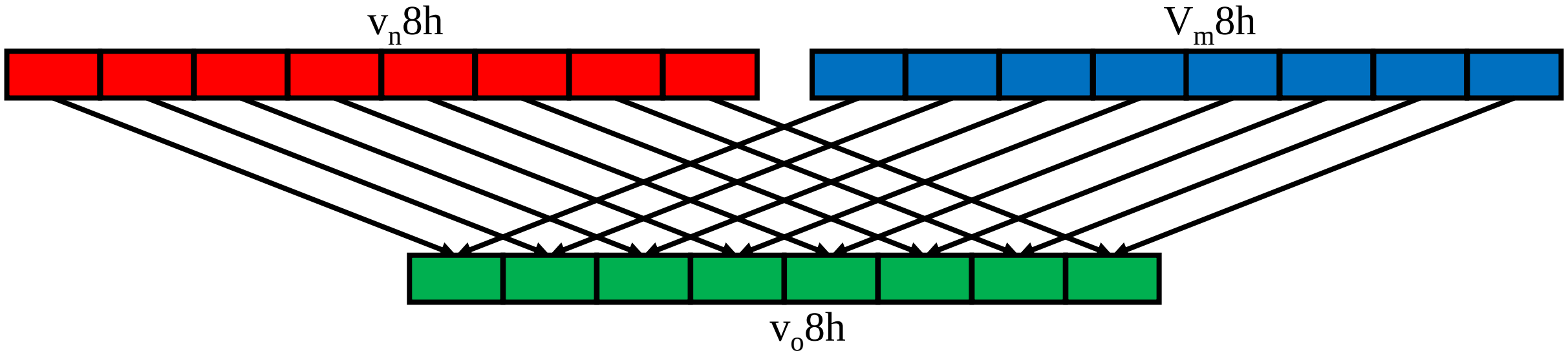
- SMAUG의 곱셈기 중에서 **최하위 모듈을 병렬**적으로 구현
- 대상 프로세서: ARMv8 프로세서
  - Vector register를 사용하여 연산을 병렬로 진행
- 레지스터 사용
  - Add, Sub: v0~v7 사용
  - Reduce: v0~v11 사용




| Instruction | Description |
|---|---|
| LSL | Logical shift left. |
| ADD | Addition. |
| LD1 | Load multiple single-element structures. |
| ST1 | Store multiple single-element structures. |
| SUB | Subtraction. |
| RET | Return from subroutine. |

# 제안 기법

- Vector register는 **병렬적으로 연산**을 할 수 있음
  - 레지스터의 값을 어떤 크기로 취급할지 지정 가능 → **Arrangement Specifier**
  - SMAUG에서는 16-bit 단위로 연산 → Half-word를 사용
- 연산 순서
  - 메모리에서 레지스터로 필요한 값 로드
  - 필요한 명령어를 사용하여 원래 연산에 맞도록 구현
  - 완성된 값을 다시 레지스터에서 메모리로 저장



# 제안 기법

- Add, Sub 구현 코드
  - 필요한 값을 호출하도록 메모리 주소 값 조정
  - 사용할 데이터를 로드
  - 덧셈(또는 뺄셈) 진행
  - 연산이 완료된 값을 저장
- 다음 메모리 주소로 이동하도록 Post-index 사용
  - #64로 명시하여 연산 종료 후 자동으로 주소 값 조정

```
LSL x2, x2, #1
ADD x0, x0, x2

LD1.8h {v0, v1, v2, v3}, [x0]
LD1.8h {v4, v5, v6, v7}, [x1], #64
ADD.8h v0, v0, v4
ADD.8h v1, v1, v5
ADD.8h v2, v2, v6
ADD.8h v3, v3, v7
ST1.8h {v0, v1, v2, v3}, [x0], #64
```

# 제안 기법

- Reduce 구현 코드
  - 기본적인 부분은 Add, Sub과 비슷함
  - 하지만 배열 값을 불러오는데 **Post-index를 사용할 수 없음**

    → Post-index는 정해진 범위 내에서만 가능
  - 따라서 **수동으로 주소 값을 조정함**
  - 그 외는 기존과 동일

| T | imm |
|---|---|
| 8B | #32 |
| 16B | #64 |
| 4H | #32 |
| 8H | #64 |
| 2S | #32 |
| 4S | #64 |
| 1D | #32 |
| 2D | #64 |

```
LD1.8h {v0, v1, v2, v3}, [x0]
LD1.8h {v4, v5, v6, v7}, [x1]
ADD x1, x1, #512
LD1.8h {v8, v9, v10, v11}, [x1]
SUB x1, x1, #448
SUB.8h v4, v4, v8
SUB.8h v5, v5, v9
SUB.8h v6, v6, v10
SUB.8h v7, v7, v11
ADD.8h v0, v0, v4
ADD.8h v1, v1, v5
ADD.8h v2, v2, v6
ADD.8h v3, v3, v7
ST1.8h {v0, v1, v2, v3}, [x0], #64
```

# 성능 평가

- 구현 환경: Xcode 14.3 Framework
- 대상 프로세서: Apple M1 Pro (3.2Ghz)
- 곱셈기 성능 비교
  - 반복횟수: 1,000,000

| Algorithm | Reference | | This Work | | Diff |
|---|---|---|---|---|---|
| | ms | Clock count | ms | Clock count | |
| Add | 0.089 | 0.2848 | 0.01 | 0.032 | 8.9 |
| Sub | 0.089 | 0.2848 | 0.01 | 0.032 | 8.9 |
| Reduce | 0.32 | 1.024 | 0.013 | 0.0416 | 24.61 |

# 성능 평가

- 알고리즘 성능 비교
  - 작성한 곱셈기 알고리즘을 적용
  - 반복횟수: 10,000

| Scheme | | Reference | | This Work | | Diff |
|---|---|---|---|---|---|---|
| | | ms | Clock count | ms | Clock count | |
| 128 | K | 47.30 | 15,136.0 | 24.39 | 7,804.8 | 1.93 |
| | E | 53.72 | 17,190.4 | 21.28 | 6,809.6 | 2.52 |
| | D | 61.52 | 19,686.4 | 17.55 | 5,616.0 | 3.51 |
| 192 | K | 68.38 | 21,881.6 | 32.04 | 10,252.8 | 2.13 |
| | E | 77.71 | 24,867.2 | 29.41 | 9,411.2 | 2.62 |
| | D | 86.80 | 27,776.0 | 35.53 | 11,369.6 | 2.44 |
| 256 | K | 111.33 | 35,625.6 | 51.72 | 16,550.4 | 2.15 |
| | E | 119.75 | 38,320.0 | 47.86 | 15,315.2 | 2.50 |
| | D | 129.20 | 41,344.0 | 44.44 | 14,220.8 | 2.90 |

# 결론

- 본 논문에서는 KpqC Round 1 후보 알고리즘 SMAUG를 최적 구현
  - ARMv8 프로세서의 **vector register**를 활용
  - **병렬 연산을 통해 곱셈 연산을 최적화**
  - 곱셈기의 성능 개선 **최대 24.62배**
  - 알고리즘에 적용 시 **최대 3.51배** 성능 개선
- SMAUG의 추가적인 부분에 최적 구현을 시도
  - 난수 생성기에 AES 가속기 적용
  - 다른 외적인 부분에 병렬 구현

# Q & A