HQC PKE의 핵심 연산에 대한 양자회로 최적 구현

임세진*, 장경배*, 오유진*, 서화정*†
* 한성대학교 대학원 IT융합공학과

Ⅰ. 서 론

● 양자컴퓨터의 발전과 Shor 알고리즘에 의해 기존 공개키 암호의 무력화 시기가 다가오고 있음

→ 이를 대비하고자 NIST는 양자내성암호 (PQC) 공모전을 개최함

● 암호를 양자회로로 구현함으로써 양자컴퓨터 환경에서의 보안강도를 확인할 수 있음

● 본 논문에서는 NIST PQC 공모전의 4라운드 후보 알고리즘인 HQC의 PKE 버전에 대한 키생성 및 인코딩 연산에서 핵심 역할을 하는 바이너리 필드 산술과 shortened Reed-Solomon 코드를 양자회로로 최적 구현하고, 필요한 자원을 추정함

Ⅱ. 관련연구

● HQC (Hamming Quasi-Cyclic)

- Hamming 코드와 랜덤한 Quasi-Cyclic (준순환) 코드를 사용하는 코드 기반 암호
- Quasi-Cyclic을 통해 첫번째 행만 저장하여도 전체 행렬을 알 수 있게 함으로써 키크기를 효율적으로 줄임

● HQC의 PKE 구성

- 공개키 $pk = (h, s)$와 비밀키 $sk = (x, y)$를 생성하는 키생성 단계와 메시지 m으로부터 암호문 $ct = (\mathrm{u}, \mathrm{v})$를 생성하는 인코딩 단계, 비밀키를 통해 암호문으로부터 에러 e를 제거하여 메시지를 복구하는 디코딩 단계로 이루어짐
- 바이너리 필드에서 사용되는 기약다항식은 $(\mathrm{x}^n + \mathrm{x}^{n-1} + \cdots + \mathrm{x} + 1)$이며, hqc-128의 경우 $\mathbb{F}_{2^{17668}}/(\mathrm{x}^{17668} + \mathrm{x}^{17667} + \cdots + \mathrm{x} + 1)$을 사용함
- 암호문 v를 구할 때 메시지 m과 Generator 행렬 G를 곱할 때 shortened Reed-Solomon 코드를 사용하는데, 이때도 바이너리 필드 곱셈이 사용되며 $\mathbb{F}_{2^8}/(\mathrm{x}^8 + \mathrm{x}^4 + \mathrm{x}^3 + \mathrm{x}^2 + 1)$의 기약다항식을 사용함

● 바이너리 필드 상에서의 산술 연산

- 덧셈은 캐리가 발생하지 않아 XOR를 사용하여 수행되고 곱셈도 캐리가 발생하지 않아 AND로 수행되는데, 곱셈 결과를 바이너리 필드 크기에 맞게 축소하는 모듈러 reduction 과정이 필요함 ← 주어진 기약다항식에 맞게 XOR를 사용하여 수행

Ⅲ. 구 현

● $\mathbb{F}_{2^{17668}}$의 바이너리 필드는 양자 시뮬레이션이 불가능 → $\mathbb{F}_{2^{12}}$로 축소하여 양자회로 구현 및 자원 추정 수행

● 키 생성은 바이너리 필드 덧셈 및 곱셈 사용

- 덧셈의 경우 XOR는 양자회로의 CNOT 게이트로 구현되므로 depth 1을 가지도록 구현함
- 곱셈은 바이너리 필드 산술 중 높은 계산 복잡도를 가지며, AND와 XOR는 Toffoli 게이트로 구현되는데, Toffoli 게이트는 구현 비용이 높음

● 필드 크기에 상관 없이 Toffoli-depth를 1로 최적화하는 최신 양자 곱셈 기법을 적용하여 구현함

● 인코딩 연산은 shortened Reed-Solomon 코드를 통해 $\mathbb{F}_{2^8}$ 에서의 곱셈 연산이 추가적으로 수행됨

- shortened Reed-Solomon 코드는 공개된 RS-S1 계수 행렬과 메시지 벡터를 곱하는 과정임
- Toffoli-depth와 depth를 최적화하도록 구현함 ➔ 사용된 게이트 수에 비해 낮은 depth를 가짐
- 곱셈을 병렬로 수행하기 위해 보조 큐비트를 통해 값을 복사하여 사용함 (reverse 연산을 통해 재활용 가능)

Algorithm 1 : shortened Reed-Solomon Encode

Input: 8-qubit array msg[K], RS_POLY[$G-1$], cdw[$N1$], gate_value[K], copy[$G-2$], ancilla qubits array ac[30]
Output: cdw

```
for i=0 to G−1
    RS_POLY[i] ← CNOT8(RS_COEFS, RS_POLY[i])
for i=0 to K
    gate_value[i] ← CNOT8(cdw[N1−K−1], gate_value[i])
    gate_value[i] ← CNOT8(msg[K−1−i], gate_value[i])
    for j=0 to G−2
        copy[j] ← Copy_gate_value(gate_value[i], copy[j])
    tmp[0] ← Multiplication(gate_value[i], RS_POLY[0], ac[0])
    for j=1 to G−1
        tmp[j] ← Multiplication(copy[j], RS_POLY[j], ac[j])
    for j=N1−K−1 to 0
        cdw[j] ← CNOT8(cdw[j−1], cdw[j])
        cdw[j] ← CNOT8(tmp[j], cdw[j])
    cdw[0] ← CNOT8(tmp[0], cdw[0])
    for j=0 to G−2
        copy[j] ← Copy_gate_value(gate_value[i], copy[j])
for i=0 to K
    cdw[i] ← CNOT8(msg[i], cdw[i+30])
return cdw
```

$\mathbb{F}_{2^8}/(X^8+X^4+X^3+X^2+1)$과 $\mathbb{F}_{2^{12}}/(X^{12}+X^{11}+\cdots+X+1)$ 산술 양자 회로 구현 결과

Field	Arithmetic	Qubits	CNOT gates	Toffoli gates	Toffoli depth	Full depth
$\mathbb{F}_{2^8}$	Multiplication	81	164	27	1	26
$\mathbb{F}_{2^{12}}$	Addition	24	12	-	-	1
	Multiplication	162	495	54	1	32

shortened Reed-Solomon 코드 인코딩 양자 회로 구현 결과

shortened Reed-Solomon	Qubits	CNOT gates	Toffoli gates	Toffoli depth	Full depth
hqc-128	28,696	94,320	12,960	16	545

Ⅳ. 결 론

● HQC PKE 버전의 키생성 및 인코딩 연산에서 사용되는 바이너리 필드 산술과 shortened Reed-Solomon 코드를 양자회로로 최적화하여 구현함

● 양자 자원의 비용을 절감시키기 위해 바이너리 필드 상에서의 곱셈 연산을 최신 구현 기법을 적용하여 최적화하였으며, shortened Reed-Solomon 코드의 경우 HQC에서 사용된 파라미터 그대로 양자회로로 구현하여 자원을 측정했다는 점에서 의의가 있음

● 제시하는 양자회로를 통해 HQC의 보안강도 분석 연구에 기여할 수 있을 것으로 기대됨

● 향후 Reed-Muller 코드 및 디코딩 단계의 핵심 연산을 구현하고 시뮬레이션이 가능한 범위를 조정하여 바이너리 필드를 최대한 확장할 계획임

https://crypto.modoo.at/

CryptoCraft LAB