Scrypt 양자회로 구현

https://youtu.be/C4ZOhT2HEzI

정보컴퓨터공학과 송경주

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

Scrypt

- 2009년 Colin Percival이 제안한 비밀번호 기반 키 파생 함수
- 계산이 집약적이고 메모리를 많이 사용하도록 설계되어 하드웨어 공격, 특히 ASIC(Application-Specific Integrated Circuit) 및 GPU를 사용하는 공격에 강함
- 따라서 Scrypt는 비밀번호 해싱 및 암호화폐와 같은 매우 안전한 애플리케이션에 이상적

Scrypt

- $PBKDF2_{SHA256}$
  - 128 rp 바이트 문자열 생성
  - 생성된 문자열은 p개의 동일한 길이의 블록으로 나눠지고 각각에 대해 SMix 함수 호출
  - SMix 함수의 결과는 최종 $PBKDF2_{SHA256}$ 호출에서 salt로 사용됨 (해당 과정에서 original password와 new salt 를 사용하여 최종 dk Len 바이트 output key 생성
- **SMix**
  - scrypt 알고리즘의 핵심이며 scrypt의 메모리 하드 구성 요소를 구성함
  - scrypt RFC [1]에서는 블록 크기 매개변수를 r = 8로 권장

  → Smix에 대한 초기 입력 블록의 크기가 1kB이므로 캐시에 쉽게 들어갈 수 있음
  - SMix 기능은 이 1kB 블록을 N 블록 배열로 확장하고 이전에 액세스한 블록의 내용을 기반으로 의사 난수 순서로 블록을 반복적으로 액세스
  - 따라서 N이 크면 SMix 함수는 scrypt 실행에 적지 않은 비용을 차지

Parameter	Meaning
p	• scrypt에서 SMix가 호출되는 횟수 결정(line 3-4) • SMix 호출은 서로 독립적(병렬)으므로 해당 파라미터가 결국 병렬화 매개변수
N	• SMix 함수에 전달되는 cost 매개변수 • SMix 함수가 N개의 서로 다른 BlockMix 해시를 계산, 저장 및 의사 무작위로 액세스하도록 요구하여 scrypt의 CPU 및 메모리 사용량을 제어(line 5~12)
r	• r은 BlockMix 함수가 작동하는 블록의 크기를 결정하는 블록 크기 매개변수 (line 13-14)

Algorithm 1 scrypt algorithm

1: **function** SCRYPT($P, S, p, N, r, dkLen$)
2: $(B_0||B_1||...||B_{p-1}) \leftarrow \text{PBKDF2}_{SHA256}(P, S, 1, 128rp)$
3: **for** $i = 0$ to $p - 1$ **do**
4: $B_i \leftarrow \text{SMIX}_r(B_i, N)$
return $\text{PBKDF2}_{SHA256}(P, B_0||B_1||...||B_{p-1}, 1, dkLen)$
5: **function** $\text{SMIX}_r(B, N)$
6: $X \leftarrow B$
7: **for** $i = 0$ to $N - 1$ **do**
8: $V_i \leftarrow X$
9: $X \leftarrow \text{BLOCKMIX}_{Salsa20/8,r}(X)$
10: **for** $i = 0$ to $N - 1$ **do**
11: $j \leftarrow \text{INTEGERIFY}(X) \bmod N$
12: $X \leftarrow \text{BLOCKMIX}_{Salsa20/8,r}(X \oplus V_j)$
return X
13: **function** $\text{BLOCKMIX}_{Salsa20/8,r}(B_0||B_1||...||B_{2r-1})$
14: ▷ each B_i must be 64-bytes (enforced by Salsa20/8 definition)
15: $X \leftarrow B_{2r-1}$
16: **for** $i = 0$ to $2r - 1$ **do**
17: $X \leftarrow \text{SALSA20/8}(X \oplus B_i)$
18: $Y_i \leftarrow X$
return $Y_0||Y_2||...||Y_{2r-2}||Y_1||Y_3||...||Y_{2r-1}$

[1]C. Percival and S. Josefsson. 2016. The scrypt Password-Based Key Derivation Function. RFC 7914. RFC Editor

Scrypt quantum circuit

- **Inverse operation을 통한 ancilla 큐비트 재사용**
  - Clean ancilla 큐비트를 연산에서 사용한 후, dirty ancilla를 inverse 연산으로 다시 clean 상태로 만들고 반복적으로 사용
  - **SHA256**: 사용한 dirty ancilla를 역연산을 통해 clean 상태로 리셋하여 모든 loop에서 재사용, 해당 방법으로 SHA-256 in $\boldsymbol{PBKDF2_{SHA256}}$ 에서 큐비트 수를 8128개 줄이고 depth 약간 증가(depth 약 6)
  - **Salsa**: Salsa 내부 연산 'Operation on columns' 및 'Operate on rows'에서 중간 값을 ancilla 에 저장하지 않고 입력을 직접 업데이트 시킴
    - ex) $\mathrm{c} = a \oplus b$ (out-of-place) → $\mathrm{b} = a \oplus b$ (in-place)
  - 이후 후속연산에서 업데이트 된 큐비트의 업데이트 전 값이 필요할 때, 업데이트된 값을 사용을 마친 후 inverse 연산을 적용하여 업데이트 전 값을 복원해서 사용

Scrypt quantum circuit

- **SHA-256 및 Salsa20/8의 병렬구조**
  - 양자회로 depth를 줄이기 위해 내부 연산들을 가능한한 병렬로 설계
  - SHA-256에서 inverse를 통한 ancilla 0 reset → inverse 연산을 후속 연산들과 병렬로 설계
  - 마찬가지로 Salsa 20/8 함수 내에서도 업데이트된 큐비트를 업데이트 전 상태로 복원하는 inverse 연산과 후속연산을 일부 병렬로 처리함

- **따라서 큐비트를 줄이기 위해 inverse 연산이 추가되었지만 full depth에는 큰 영향을 주지 않음**

SHA-256 in $PBKDF2_{SHA256}$

- **SHA-256 내부동작은: Ch, Maj, SIR, ROT, S0, S1, s0, s1로 구성됨**
- **Inverse 연산을 병렬로 설계하여 약간의 depth 증가로 큐비트를 크게 줄임**
- S0, S1, Maj → RND function
- s1, s0 → MSCH function
- SHR → s1, s0 에서 사용
- ROTR → S0, S1, s0, s1 에서 사용

Algorithm 3 $S1$ quantum circuit in SHA-256

Input: x, h, $S1_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S1_{anc}[i] \leftarrow CNOT(x[(i+6)\%32], S1_{anc}[i])$
4: $S1_{anc}[i] \leftarrow CNOT(x[(i+11)\%32], S1_{anc}[i])$
5: $S1_{anc}[i] \leftarrow CNOT(x[(i+25)\%32], S1_{anc}[i])$
6: ◇**Point 2: End inverse operation**
7: $h \leftarrow Add(S1_{anc}, h)$

8: // Reset $S1_{anc}$ to clean $|0\rangle$
9: ◇**Start inverse operation from Point 1 to Point 2.**

Ch(x, y, z) $= ((x \,\&\, (y \hat{} z)) \hat{} z)$
Maj(x, y, z) $= ((x \& (y \mid z)) \mid (y \& z))$
SHR(x, n) $= (x \gg n)$
ROTR(x, n) $= ((x \gg n) \mid (x \ll (32-n)))$
S0(x) $= (ROTR(x,2) \hat{} ROTR(x,13) \hat{} ROTR(x,22))$
S1(x) $= (ROTR(x,6) \hat{} ROTR(x,11) \hat{} ROTR(x,25))$
s0(x) $= (ROTR(x,7) \hat{} ROTR(x,18) \hat{} SHR(x,3))$
s1(x) $= (ROTR(x,17) \hat{} ROTR(x,19) \hat{} SHR(x,10))$

Algorithm 2 $S0$ quantum circuit in SHA-256

Input: x, h, $S0_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S0_{anc}[i] \leftarrow CNOT(x[(i+2)\%32], S0_{anc}[i])$
4: $S0_{anc}[i] \leftarrow CNOT(x[(i+13)\%32], S0_{anc}[i])$
5: $S0_{anc}[i] \leftarrow CNOT(x[(i+22)\%32], S0_{anc}[i])$
6: ◇**Point 2: End inverse operation**

7: $h \leftarrow Add(S0_{anc}, h)$
8: ◇**Start inverse operation from Point 1 to Point 2.**

Algorithm 4 Maj quantum circuit in SHA-256

Input: x, h, Maj_{anc1}, Maj_{anc2}, Maj_{anc3}

1: **for** (i=0 **to** 32) :
2: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$

3: //OR-gate operation
4: **for** (i=0 **to** 32) :
5: $y[i] \leftarrow X(y[i])$
6: $z[i] \leftarrow X(z[i])$
7: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
8: $y[i] \leftarrow X(y[i])$
9: $z[i] \leftarrow X(z[i])$
10: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

11: **for** (i=0 **to** 32) :
12: $Maj_{anc3}[i] \leftarrow Toffoli(x[i], Maj_{anc2}[i], Maj_{anc3}[i])$

13: //OR-gate operation (Reset Maj_{anc2})
14: **for** (i=0 **to** 32) :
15: $y[i] \leftarrow X(y[i])$
16: $z[i] \leftarrow X(z[i])$
17: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
18: $y[i] \leftarrow X(y[i])$
19: $z[i] \leftarrow X(z[i])$
20: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

21: //OR-gate operation.
22: **for** (i=0 **to** 32) :
23: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
24: $Maj_{anc3}[i] \leftarrow X(z[i])$
25: $Maj_{anc2}[i] \leftarrow Toffoli(Maj_{anc1}[i], Maj_{anc3}[i], Maj_{anc2}[i])$
26: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
27: $Maj_{anc3}[i] \leftarrow X(Maj_{anc3}[i])$
28: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

29: $h \leftarrow Add(Maj_{anc2}, h)$

30: **for** (i=0 **to** 32) :
31: $Maj_{anc3}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc3}[i])$ //Reset Maj_{anc3}
32: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$ //Reset Maj_{anc1}

SHA-256 in $PBKDF2_{SHA256}$

- **SHA-256 내부동작은: Ch, Maj, SIR, ROT, S0, S1, s0, s1로 구성됨**
- **Inverse 연산을 병렬로 설계하여 약간의 depth 증가로 큐비트를 크게 줄임**
- **S0, S1, Maj → RND function**
- s1, s0 → MSCH function
- SHR → s1, s0 에서 사용
- ROTR → S0, S1, s0, s1 에서 사용

Algorithm 3 $S1$ quantum circuit in SHA-256

Input: x, h, $S1_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S1_{anc}[i] \leftarrow CNOT(x[(i+6)\%32], S1_{anc}[i])$
4: $S1_{anc}[i] \leftarrow CNOT(x[(i+11)\%32], S1_{anc}[i])$
5: $S1_{anc}[i] \leftarrow CNOT(x[(i+25)\%32], S1_{anc}[i])$
6: ◇**Point 2: End inverse operation**
7: $h \leftarrow Add(S1_{anc}, h)$

8: // Reset $S1_{anc}$ to clean $|0\rangle$
9: ◇**Start inverse operation from Point 1 to Point 2.**

Ch(x, y, z) $= ((x \ \& \ (y \hat{} z)) \hat{} z)$
Maj(x, y, z) $= ((x \& (y \mid z)) \mid (y \& z))$
SHR(x, n) $= (x \gg n)$
ROTR(x, n) $= ((x \gg n) \mid (x \ll (32-n)))$
S0(x) $= (ROTR(x,2) \hat{} ROTR(x,13) \hat{} ROTR(x,22))$
S1(x) $= (ROTR(x,6) \hat{} ROTR(x,11) \hat{} ROTR(x,25))$
s0(x) $= (ROTR(x,7) \hat{} ROTR(x,18) \hat{} SHR(x,3))$
s1(x) $= (ROTR(x,17) \hat{} ROTR(x,19) \hat{} SHR(x,10))$

Algorithm 2 $S0$ quantum circuit in SHA-256

Input: x, h, $S0_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S0_{anc}[i] \leftarrow CNOT(x[(i+2)\%32], S0_{anc}[i])$
4: $S0_{anc}[i] \leftarrow CNOT(x[(i+13)\%32], S0_{anc}[i])$
5: $S0_{anc}[i] \leftarrow CNOT(x[(i+22)\%32], S0_{anc}[i])$
6: ◇**Point 2: End inverse operation**

7: $h \leftarrow Add(S0_{anc}, h)$
8: ◇**Start inverse operation from Point 1 to Point 2.**

Algorithm 4 Maj quantum circuit in SHA-256

Input: x, h, Maj_{anc1}, Maj_{anc2}, Maj_{anc3}

1: **for** (i=0 **to** 32) :
2: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$

3: //OR-gate operation
4: **for** (i=0 **to** 32) :
5: $y[i] \leftarrow X(y[i])$
6: $z[i] \leftarrow X(z[i])$
7: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
8: $y[i] \leftarrow X(y[i])$
9: $z[i] \leftarrow X(z[i])$
10: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

11: **for** (i=0 **to** 32) :
12: $Maj_{anc3}[i] \leftarrow Toffoli(x[i], Maj_{anc2}[i], Maj_{anc3}[i])$

13: //OR-gate operation (Reset Maj_{anc2})
14: **for** (i=0 **to** 32) :
15: $y[i] \leftarrow X(y[i])$
16: $z[i] \leftarrow X(z[i])$
17: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
18: $y[i] \leftarrow X(y[i])$
19: $z[i] \leftarrow X(z[i])$
20: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

21: //OR-gate operation.
22: **for** (i=0 **to** 32) :
23: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
24: $Maj_{anc3}[i] \leftarrow X(z[i])$
25: $Maj_{anc2}[i] \leftarrow Toffoli(Maj_{anc1}[i], Maj_{anc3}[i], Maj_{anc2}[i])$
26: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
27: $Maj_{anc3}[i] \leftarrow X(Maj_{anc3}[i])$
28: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

29: $h \leftarrow Add(Maj_{anc2}, h)$

30: **for** (i=0 **to** 32) :
31: $Maj_{anc3}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc3}[i])$ //Reset Maj_{anc3}
32: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$ //Reset Maj_{anc1}

SHA-256 in $PBKDF2_{SHA256}$

- RND 함수 동작에서 $S1^{\dagger}$, $S0^{\dagger}$, $Maj^{\dagger}$ 연산이 후속연산$(Ch, Maj, Add + S1)$과 병렬로 동작하도록 하여 양자회로 Full-depth를 줄임
- 해당 방법을 통해 SHA-256 1번 동작 기준 depth 6 증가로 8,128개의 큐비트를 줄임
- S0, S1 에서 ROTR은 n-bit right rotation을 수행하는데, 비트단위 연산을 수행하는 양자컴퓨터에서 control 큐비트의 index 조정으로 이를 생략함

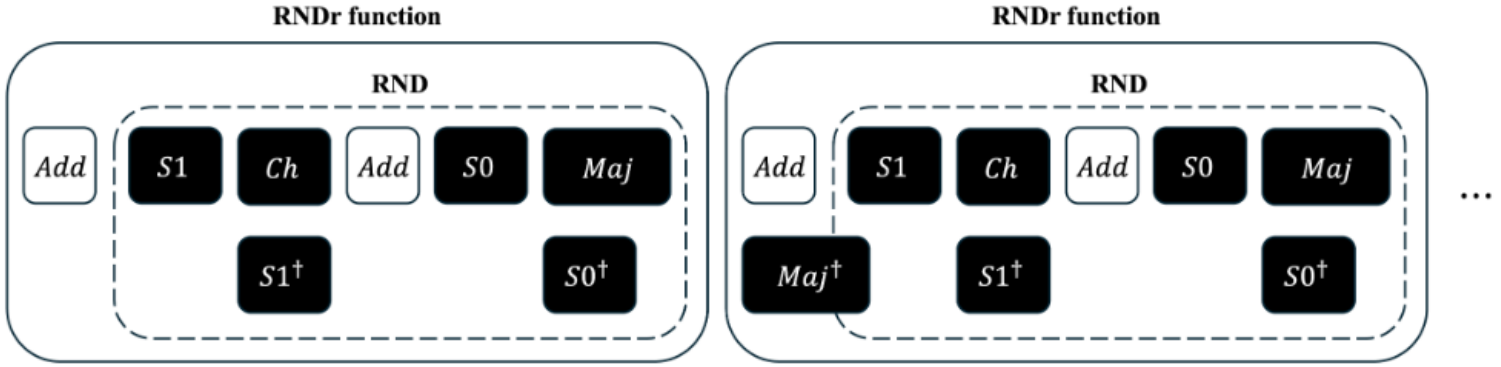
```
def RND(eng, a, b, c, d, e, f, g, h, k):
    RND_carry = eng.allocate_qubit()

    S1(eng, e, h)

    Ch(eng, e, f, g, h)

    CDKM(eng, k, h, RND_carry, len(h))
    CDKM(eng, h, d, RND_carry, len(d))
    S0(eng, a, h)
    Maj(eng, a, b, c, h)
```

$\mathbf{Ch(x, y, z)} = ((x \,\&\, (y \hat{} z)) \hat{} z)$
$\mathbf{Maj(x, y, z)} = ((x \& (y \mid z)) \mid (y \& z))$
$\mathbf{SHR(x, n)} = (x \gg n)$
$\mathbf{ROTR(x, n)} = ((x \gg n) \mid (x \ll (32 - n)))$
$\mathbf{S0(x)} = (ROTR(x, 2) \hat{} ROTR(x, 13) \hat{} ROTR(x, 22))$
$\mathbf{S1(x)} = (ROTR(x, 6) \hat{} ROTR(x, 11) \hat{} ROTR(x, 25))$



SHA-256 in $PBKDF2_{SHA256}$

- **SHA-256 내부동작은: Ch, Maj, SIR, ROT, S0, S1, s0, s1로 구성됨**
- **Inverse 연산을 병렬로 설계하여 약간의 depth 증가로 큐비트를 크게 줄임**
- S0, S1, Maj → RND function
- **s1, s0 → MSCH function**
- SHR → s1, s0 에서 사용
- ROTR → S0, S1, s0, s1 에서 사용

Algorithm 3 $S1$ quantum circuit in SHA-256

Input: x, h, $S1_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S1_{anc}[i] \leftarrow CNOT(x[(i+6)\%32], S1_{anc}[i])$
4: $S1_{anc}[i] \leftarrow CNOT(x[(i+11)\%32], S1_{anc}[i])$
5: $S1_{anc}[i] \leftarrow CNOT(x[(i+25)\%32], S1_{anc}[i])$
6: ◇**Point 2: End inverse operation**
7: $h \leftarrow Add(S1_{anc}, h)$

8: // Reset $S1_{anc}$ to clean $|0\rangle$
9: ◇**Start inverse operation from Point 1 to Point 2.**

Ch(x, y, z) $= ((x \ \& \ (y \hat{} z)) \hat{} z)$
Maj(x, y, z) $= ((x \& (y \mid z)) \mid (y \& z))$
SHR(x, n) $= (x \gg n)$
ROTR(x, n) $= ((x \gg n) \mid (x \ll (32-n)))$
S0(x) $= (ROTR(x,2) \hat{} ROTR(x,13) \hat{} ROTR(x,22))$
S1(x) $= (ROTR(x,6) \hat{} ROTR(x,11) \hat{} ROTR(x,25))$
s0(x) $= (ROTR(x,7) \hat{} ROTR(x,18) \hat{} SHR(x,3))$
s1(x) $= (ROTR(x,17) \hat{} ROTR(x,19) \hat{} SHR(x,10))$

Algorithm 2 $S0$ quantum circuit in SHA-256

Input: x, h, $S0_{anc}$

1: ◇**Point 1: Start inverse operation**
2: **for** (i=0 **to** 32) :
3: $S0_{anc}[i] \leftarrow CNOT(x[(i+2)\%32], S0_{anc}[i])$
4: $S0_{anc}[i] \leftarrow CNOT(x[(i+13)\%32], S0_{anc}[i])$
5: $S0_{anc}[i] \leftarrow CNOT(x[(i+22)\%32], S0_{anc}[i])$
6: ◇**Point 2: End inverse operation**

7: $h \leftarrow Add(S0_{anc}, h)$
8: ◇**Start inverse operation from Point 1 to Point 2.**

Algorithm 4 Maj quantum circuit in SHA-256

Input: x, h, Maj_{anc1}, Maj_{anc2}, Maj_{anc3}

1: **for** (i=0 **to** 32) :
2: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$

3: //OR-gate operation
4: **for** (i=0 **to** 32) :
5: $y[i] \leftarrow X(y[i])$
6: $z[i] \leftarrow X(z[i])$
7: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
8: $y[i] \leftarrow X(y[i])$
9: $z[i] \leftarrow X(z[i])$
10: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

11: **for** (i=0 **to** 32) :
12: $Maj_{anc3}[i] \leftarrow Toffoli(x[i], Maj_{anc2}[i], Maj_{anc3}[i])$

13: //OR-gate operation (Reset Maj_{anc2})
14: **for** (i=0 **to** 32) :
15: $y[i] \leftarrow X(y[i])$
16: $z[i] \leftarrow X(z[i])$
17: $Maj_{anc2}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc2}[i])$
18: $y[i] \leftarrow X(y[i])$
19: $z[i] \leftarrow X(z[i])$
20: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

21: //OR-gate operation.
22: **for** (i=0 **to** 32) :
23: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
24: $Maj_{anc3}[i] \leftarrow X(z[i])$
25: $Maj_{anc2}[i] \leftarrow Toffoli(Maj_{anc1}[i], Maj_{anc3}[i], Maj_{anc2}[i])$
26: $Maj_{anc1}[i] \leftarrow X(Maj_{anc1}[i])$
27: $Maj_{anc3}[i] \leftarrow X(Maj_{anc3}[i])$
28: $Maj_{anc2}[i] \leftarrow X(Maj_{anc2}[i])$

29: $h \leftarrow Add(Maj_{anc2}, h)$

30: **for** (i=0 **to** 32) :
31: $Maj_{anc3}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc3}[i])$ //Reset Maj_{anc3}
32: $Maj_{anc1}[i] \leftarrow Toffoli(y[i], z[i], Maj_{anc1}[i])$ //Reset Maj_{anc1}

SHA-256 in $PBKDF2_{SHA256}$

- s0, s1에서 사용되는 SHR은 n-bit right shift를 수행
  - n크기만큼 0으로 채워지므로 해당 n크기 부분은 CNOT이 동작하지 않음
  - 결과적으로 n크기만큼 CNOT을 적용하지 않도록 하여 shift연산 생략
- 알고리즘의 5, 6 line 4의 loop 문의 반복 수를 32-n (n=3, 10)로 설정하여 CNOT 동작을 n크기만큼 제외
- m-XOR[1] 방식을 통해 CNOT2(ROTR(x, 17), ROTR(x, 18), ancilla)의 결과를 이후 사용되지 않는 h에 대해 CNOT(ROTR(x, 17), h), CNOT(ROTR(x, 19), h)로 변경

Algorithm 5 *s0* quantum circuit in SHA-256

Input: x, h

1: **for** (i=0 **to** 32) :
2: $h[i] \leftarrow CNOT(x[(i+7)\%32], h[i])$
3: $h[i] \leftarrow CNOT(x[(i+18)\%32], h[i])$
4: **for** (i=0 **to** 32 - 3) :
5: $h[i] \leftarrow CNOT(x[i+3], h[i])$

6: // *h* update

Algorithm 6 *s1* quantum circuit in SHA-256

Input: x, h

1: **for** (i=0 **to** 32) :
2: $h[i] \leftarrow CNOT(x[(i+17)\%32], h[i])$
3: $h[i] \leftarrow CNOT(x[(i+19)\%32], h[i])$
4: **for** (i=0 **to** 32 - 10) :
5: $h[i] \leftarrow CNOT(x[i+10], h[i])$

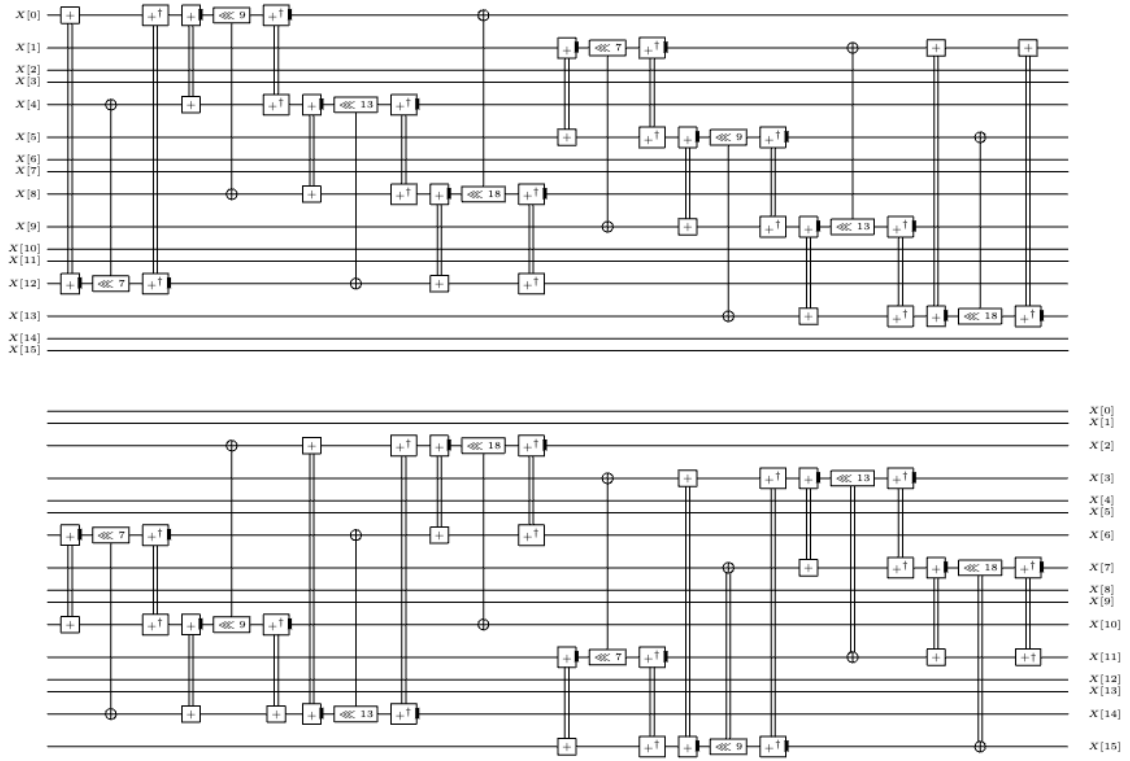
6: // *h* update

[1]

Salsa20/8 in SMIX

- Salsa20/8 는 8라운드 버전의 Salsa20
- CNOT 연산할 때, mixing-XOR이 사용됨
- Salsa 내부 연산 ‘Operation on columns’ 및 ‘Operate on rows’에서 중간 값을 ancilla에 저장하지 않고 입력을 직접 업데이트 시킴

  - Add는 두 큐비트의 in-place 덧셈, $Add^{\dagger}$ 는 inverse Add 수행
  - X는 16x32 array 큐비트를 나타냄
  - <<< n : n만큼 왼쪽으로 이동 후 XOR
  - >>> n : n만큼 오른쪽으로 이동 후 XOR



Operation on columns in Salsa20/8

Evaluation

Quantum resource in Scrypt

Qubit	Qubit	Toffoli	CNOT	X	Full depth
SHA-256 Transform	17100	58448	137888	63231	19088
Salsa20/8	1040	4880	13776	4924	1969

*Loop가 굉장히 많음..

Q & A