SM3 양자회로 구현

https://youtu.be/_5UbHZ86Z6o

IT융합공학부 송경주

한성대학교
HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

SM3

SM3는 중국 국가 표준에서 사용되는 암호화 해시함수이다.

32word 단위로 동작하며 최종적으로 256비트의 해시값을 출력한다.

메시지 패딩, 메시지 확장, 압축, 해시 값 출력 순서로 동작한다.

SM3 메시지 확장

a. Split message block $B^{(i)}$ into 16 words $W_0, W_1, \cdots, W_{15}$.

b.
$$\begin{aligned}&\text{FOR } j = 16 \text{ TO } 67\\ &\quad W_j \leftarrow P_1\left(W_{j-16} \oplus W_{j-9} \oplus \left(W_{j-3} <<< 15\right)\right) \oplus \left(W_{j-13} <<< 7\right) \oplus W_{j-6}\\ &\text{ENDFOR}\end{aligned}$$

c.
$$\begin{aligned}&\text{FOR } j = 0 \text{ TO } 63\\ &\quad W_j' = W_j \oplus W_{j+4}\\ &\text{ENDFOR}\end{aligned}$$

패딩 된 입력 메시지를 16개의 워드 단위(w=32)로 나눈 후, 16개의 워드 단위 메시지를 통해 메시지를 확장 시킴.

확장시킨 메시지를 W(i), W’(i) (i=0,..,63) 를 한 쌍으로 압축 함수를 진행함.

SM3 메시지 압축

$$ABCDEFGH \leftarrow V(i)$$

$$\text{FOR } j = 0 \text{ TO } 63$$

$$SS1 \leftarrow \left(\left(A <<< 12\right) + E + \left(T_j <<< \left(j \bmod 32\right)\right)\right) <<< 7$$

$$SS2 \leftarrow SS1 \oplus \left(A <<< 12\right)$$

$$TT1 \leftarrow FF_j\left(A, B, C\right) + D + SS2 + W_j'$$

$$TT2 \leftarrow GG_j\left(E, F, G\right) + H + SS1 + W_j$$

$$D \leftarrow C$$

$$C \leftarrow B <<< 9$$

$$B \leftarrow A$$

$$A \leftarrow TT1$$

$$H \leftarrow G$$

$$G \leftarrow F <<< 19$$

$$F \leftarrow E$$

$$E \leftarrow P_0\left(TT2\right)$$

$$\text{ENDFOR}$$

$$V(i+1) \leftarrow ABCDEFGH \oplus V(i)$$

확장시킨 메시지를 이용하여 32비트 레지스터 A,B,C,D,E,F,G,H 를 업데이트 한다.

SS1, SS2, TT1, TT2 는 값을 담는 변수이다.

FF와 GG 함수는 AND와 OR을 이용한 계산을 수행한다.

P0은 자기자신을 shift시키며 permutation연산을 수행한다.

메시지 압축을 진행한 뒤 최종 레지스터 값 (32*8 = 256 bit)이 해시 값이 된다.

SM3 양자회로 구현 특징1

기존 SM3에서는 메시지 확장 진행 후 메시지 압축을 수행하여 레지스터를 업데이트 한다.

(문제점) 확장된 메시지 전체를 저장할 큐빗 필요

→ 이러한 문제를 해결하기 위해 메시지 확장과 압축을 섞어서 진행하였다.

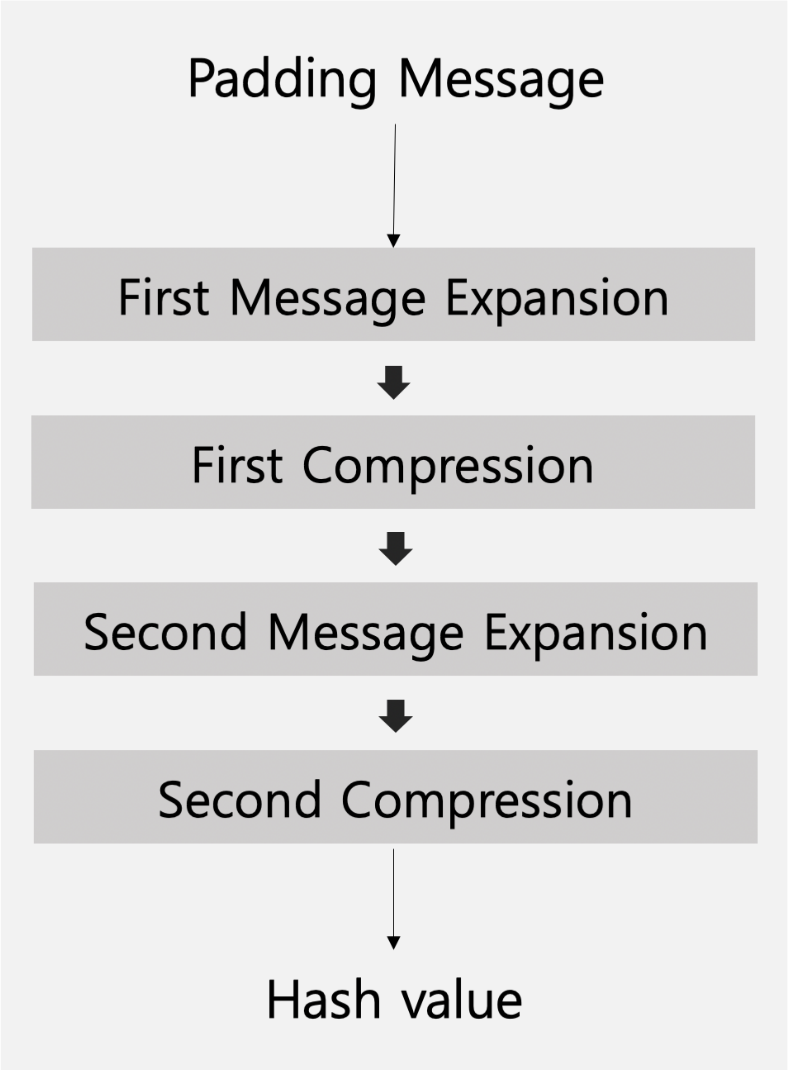
SM3 양자회로 구현 특징2

두개의 permutation연산 중 permutation0에서는큐빗을 재사용 할 수 없기 때문에 매 라운드마다 32개의 큐빗을 할당해야 하는 문제가 있었다.

(기존 값을 보관할 큐빗이 필요했음.)

→CNOT연산을 통해 이전 값을 찾아 사용하여 문제점을 해결하였다.

SM3 양자회로 진행 과정



메시지 확장과 압축을 섞어 메시지 저장에 사용할 큐빗을 줄이고 효율적인 양자회로를 구성함.

First Message Expansion

Algorithm 2 First message expansion quantum circuit algorithm.

Input: $W_0, W_1, ..., W_{15}$.
Output: $W_{16}, W_{17}, ..., W_{67}$.

Update:
    for $i = 0$ **to** 31 **do**
        $W_{j-16}[i] \leftarrow \mathrm{CNOT}(W_{j-9}[i], W_{j-16}[i]),\ \ j = 16, ..., 67$
        $W_{j-16}[i] \leftarrow \mathrm{CNOT}(W_{j-3}[(i+15)\%32], W_{j-16}[i]),\ \ j = 16, ..., 67$
    end for
    $Permutation_{p1}(W_{j-16})$
for $i = 0$ **to** 31 **do**
    $W_j[i] \leftarrow \mathrm{CNOT}(W_{j-16}[i], W_j[i]),\ j = 16, ..., 67$
    $W_j[i] \leftarrow \mathrm{CNOT}(W_{j-13}[(i+15)\%32], W_j[i]),\ \ j = 16, ..., 67$
    $W_j[i] \leftarrow \mathrm{CNOT}(W_{j-6}[(i+15)\%32], W_j[i]),\ \ j = 16, ..., 67$
end for
Update(reverse)
return $W_{16}, W_{17}, ..., W_{67}$

```
def init_MSG_Exp(eng, update_W, W0, W1, W2,W3,W4,p):
    with Compute(eng):
        for i in range(32):
            CNOT | (W1[i],W0[i])
        for i in range(32):
            CNOT | (W2[(i+15)%32],W0[i])
        permutation_P1(eng, W0, p)

    for i in range(32):
        CNOT | (W0[i],update_W[i])
    for i in range(32):
        CNOT | (W3[(i+7)%32], update_W[i])
    for i in range(32):
        CNOT | (W4[i], update_W[i])

    Uncompute(eng)
```

First Compression

Algorithm 4 First compression quantum circuit algorithm.

Input: 32-qubits-register A, B, C, D, E, F, G, H $, W_0, ..., W_{63}$.

Output: 32-qubits-register A, B, C, D, E, F, G, H after the first compression.

1: **Update:**

2: $\quad T_j \leftarrow (T_j <<< j \ mod \ 32) <<< 7, \quad j = 0, .., 63$

3: $\quad value0 \leftarrow GG$

4: $\quad value1 \leftarrow FF$

5: $\quad E \leftarrow SS1$

6: $\quad A \leftarrow SS2$

7: $H \leftarrow TT2$

8: **return** A, B, C, D, E, F, G, H

```
with Compute(eng):
    T_0(eng, T, j)
    GG(eng, j, E, F, G, AND_value, AND_value0, OR_value0) #GG = OR_value0
    FF(eng, j, A, B, C, AND_value1, AND_value2, OR_value1, OR_value2) #FF = OR_value2
    SS_1(eng, A, E, T, j, c0) #SS1 = E
    SS_2(eng, E, A) #SS2 = A


TT_2(eng, OR_value0, H, E, W_low, j, c0)  # TT2 = H
```

Second Message Expansion

Algorithm 3 Second message expansion quantum circuit algorithm.

Input: $W_k, W_{k+4}, \quad k = 0, ..., 63.$
Output: $W'_t, \quad t = 0, ..., 63.$

1: **for** $i = 0$ **to** 31 **do**
2: $\quad W'j_{[}i] \leftarrow \text{CNOT}(Wj_{[}i], W_{j+4}[i]), \quad j = 0, ..., 63$
3: **end for**
4: **return** $W'_t, \quad t = 0, ..., 63$

```
def MSG_Exp_0(eng, W_low, W_high):
    for i in range(32):
        CNOT | (W_high[i], W_low[i])
```

Second Compression

Algorithm 5 Second compression quantum circuit algorithm.

Input: 32-qubits-register A, B, C, D, E, F, G, H ,$W'_0, ..., W'_{63}$.

Output: 32-qubits-register A, B, C, D, E, F, G, H after the second compression.

1: $D \leftarrow TT1$

2: **Update of first compression (reverse)**

3: $H \leftarrow Permutation_{p0}$

4: $Swap(D, H)$

5: $B \leftarrow B <<< 9$

6: $F \leftarrow F <<< 19$

7: $Swap(A, H)$

8: $Swap(B, H)$

9: $Swap(C, H)$

10: $Swap(D, H)$

11: $Swap(E, H)$

12: $Swap(F, H)$

13: $Swap(G, H)$

14: **return** A, B, C, D, E, F, G, H

```
TT_1(eng, OR_value2, D, A, W_low, j, c0) #TT1 = D
Uncompute(eng)

permutation_P0(eng, H)

for i in range(32):
    Swap | (D[i], H[i])

for k in range(9):
    for i in range(31):
        Swap | (B[i], B[i + 1])

for k in range(19):
    for i in range(31):
        Swap | (F[i], F[i + 1])

for i in range(32):
    Swap | (A[i], H[i])
for i in range(32):
    Swap | (B[i], H[i])
for i in range(32):
    Swap | (C[i], H[i])
for i in range(32):
    Swap | (D[i], H[i])
for i in range(32):
    Swap | (E[i], H[i])
for i in range(32):
    Swap | (F[i], H[i])
for i in range(32):
    Swap | (G[i], H[i])
```

Premutation0 연산의 일부

```
CNOT | (x[7], x[16])
CNOT | (x[31], x[16])
CNOT | (x[22], x[16])
CNOT | (x[14], x[16])
CNOT | (x[13], x[16])
CNOT | (x[5], x[16])
```

bits	state
a_{16}	$a_{16} \oplus a_7$
a_{16}	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{22} \oplus a_{14}$
a_{16}	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{14} \oplus a_{13} \oplus a_5$
a_{16}	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_{13} \oplus a_5$
a_{16}	$a_{16} \oplus a_7 \oplus a_{31} \oplus a_5$
a_{16}	$a_{16} \oplus a_7 \oplus a_{31}$

최종 출력 해시값

```
# 입력: 616263
X | M0[1]
X | M0[2]

X | M0[7]

X | M0[9]
X | M0[10]

X | M0[14]

X | M0[17]
X | M0[18]

X | M0[22]
X | M0[23]
```

```
Hash value
01100110110001111111100001111010 0
```

그루버 알고리즘 적용 자원 측정 결과

```
Gate counts:
    Allocate : 2721
    CCX : 43328
    CX : 134144
    Deallocate : 2721
    Swap : 43866
    X : 3826

Depth (number of qubits) : 128129.
```

Q & A