SIMON 64/96 in Quantum Computer

https://youtu.be/6WjVYl6C250

장경배

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Key Schedule

```
void Simon6496KeySchedule(u32 K[],u32 rk[])
{
u32 i,c=0xfffffffc;
u64 z=0x7369f885192c0ef5;
rk[0]=K[0]; rk[1]=K[1]; rk[2]=K[2];
for(i=3;i<42;i++){
rk[i]=c^(z&1)^rk[i-3]^ROTR32(rk[i-1],3)^ROTR32(rk[i-1],4);
z>>=1;
}
```

Classic		**Quantum**
c : 연산하는 비트의 최하위 2-비트를 제외하고 모두 NOT 연산	→	최하위 2-비트 제외하고 모두 X gate
z : z 의 최하위 1-비트만 연산 대상과 XOR, (연산 후 Right_Shift)	→	최하위 1-비트를 z의 값에 따라 X gate or 최하위 1-비트에 z의 1-비트 CNOT gate
ROTR32 : 우측으로 Rotate 3비트, 4비트 한 결과를 XOR 연산	→	각각 32개의 CNOT gate로 해결 가능

Key Schedule (Recycle)

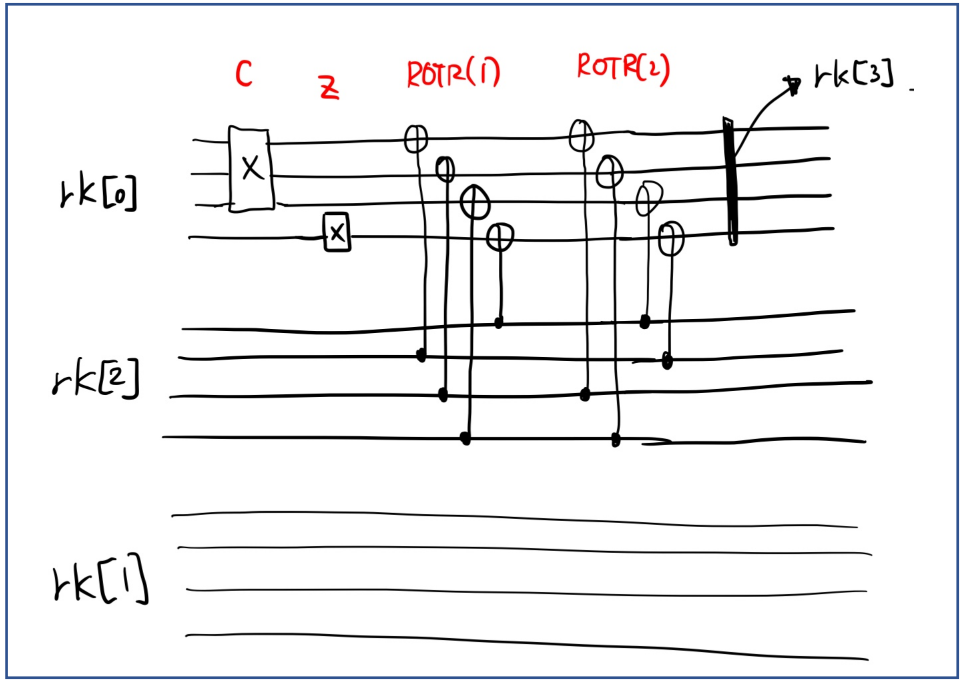
```
void Simon6496KeySchedule(u32 K[],u32 rk[])
{
u32 i,c=0xfffffffc;
u64 z=0x7369f885192c0ef5;
rk[0]=K[0]; rk[1]=K[1]; rk[2]=K[2];
for(i=3;i<42;i++){
rk[i]=c^(z&1)^rk[i-3]^ROTR32(rk[i-1],3)^ROTR32(rk[i-1],4);
z>>=1;
}
```

* Make : rk[i] ,
  Target : rk[i-3]

Example : rk[3]

1. rk[0] 의 최하위 2- 비트 제외하고 모두 NOT 연산
2. rk[0]에 z 의 최하위 1-비트 XOR 연산
3. rk[0]에 rk[2] 의 RotateR(3) 결과 XOR 연산
4. rk[0]에 rk[2] 의 RotateR(4) 결과 XOR 연산
5. rk[3] = rk[0]

→



Encrypt

```
#define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
#define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)
```

```
void Simon6496Encrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
u32 i;
Ct[1]=Pt[1]; Ct[0]=Pt[0];
for(i=0;i<42;) R32x2(Ct[1],Ct[0],rk[i++],rk[i++]);

}
```

rk[0], rk[1], rk[2] 는 기본 key

for 문

Frist → Ct[0] 에 f32(Ct[1]) 를 XOR, 그리고 Key[0] 을 XOR

바뀐 Ct[1]에 변한 값의 f32(Ct[0]), 그리고 Key[1] 을 XOR

Second → Ct[0] 에 f32(Ct[1]) 를 XOR, 그리고 Key[2] 을 XOR

→ 바뀐 Ct[1]에 변한 값의 f32(Ct[0]), 그리고 Key[3] 을 XOR → 여기서 Key Schedule

Encrypt (Quantum)

```python
def Enc(eng):

    k0 = eng.allocate_qureg(32)
    k1 = eng.allocate_qureg(32)
    k2 = eng.allocate_qureg(32)

    z = eng.allocate_qubit()

    text0 = eng.allocate_qureg(32)
    text1 = eng.allocate_qureg(32)

######################################################################

#Encrypt (key[0~2])
    #key[0])
    for i in range (32): #0
        Toffoli | (text1[(31+i)%32], text1[(24+i)%32], text0[i]) #  ROTL32(x,1) & ROTL32(x,8)
        CNOT | (text1[(30+i)%32], text0[i])                      #  ^ ROTL32(x,2)
        CNOT | (k0[i], text0[i])                                 #  ^ key

    #key[1]
    for i in range(32): #1
       Toffoli | (text0[(31 + i) % 32], text0[(24 + i) % 32], text1[i])
       CNOT | (text0[(30 + i)% 32] , text1[i])
       CNOT | (k1[i], text1[i])

    #key[2]
    for i in range(32):
       Toffoli | (text1[(31 + i) % 32], text1[(24 + i) % 32], text0[i])
       CNOT | (text1[(30 + i) % 32], text0[i])
       CNOT | (k2[i], text0[i])
```

First

Second

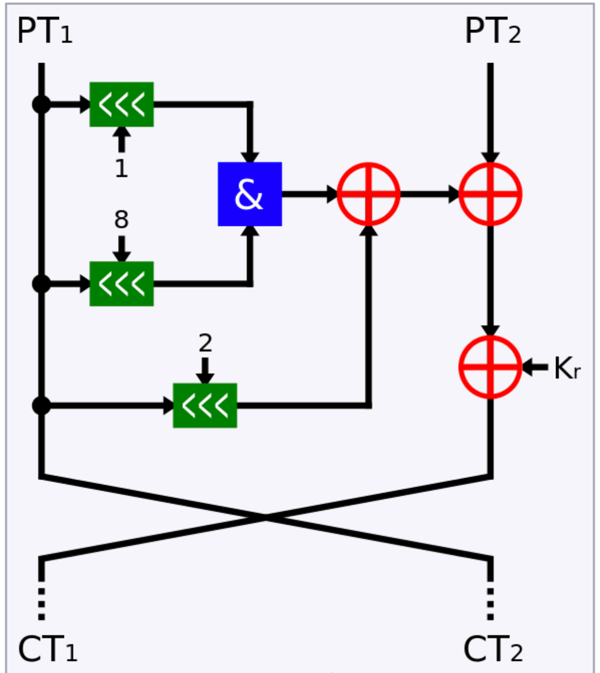
```python
#key[2]
for i in range(32):
    Toffoli | (text1[(31 + i) % 32], text1[(24 + i) % 32], text0[i])
    CNOT | (text1[(30 + i) % 32], text0[i])
    CNOT | (k2[i], text0[i])

# KeyExpansion (key[3]) (0,2)
    for i in range (30):
        X | k0[i+2]

    CNOT | (z, k0[0]) # X | (k0[0])

    #Rotate_right (3,4)
    for i in range (32):
        CNOT | (k2[((i+3) % 32)] , k0[i])
        CNOT | (k2[((i + 4) % 32)], k0[i])

#key[3]
for i in range(32):
    Toffoli | (text0[(31 + i) % 32], text0[(24 + i) % 32], text1[i])
    CNOT | (text0[(30 + i)% 32] , text1[i])
    CNOT | (k0[i], text1[i])
```

.
.
.
.

Encrypt 하면서 Key Schedule 하면 효율적

Simon 64/96 (2019)

Simon

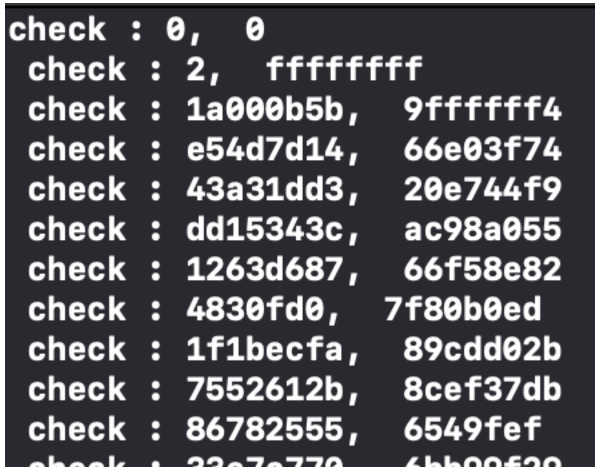


One round of Simon

General

Designers	Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers NSA
First published	2013[1]
Related to	Speck

≠ ?



Encrypt 부분 (2019_Ref)

SIMON and SPECK
Implementation Guide

Ray Beaulieu
Douglas Shors
Jason Smith
Stefan Treatman-Clark
Bryan Weeks
Louis Wingers

National Security Agency
9800 Savage Road, Fort Meade, MD, 20755, USA

{rabeaul,djshors,jksmit,sgtreat,beweeks,lrwinge}@tycho.ncsc.mil

January 15, 2019

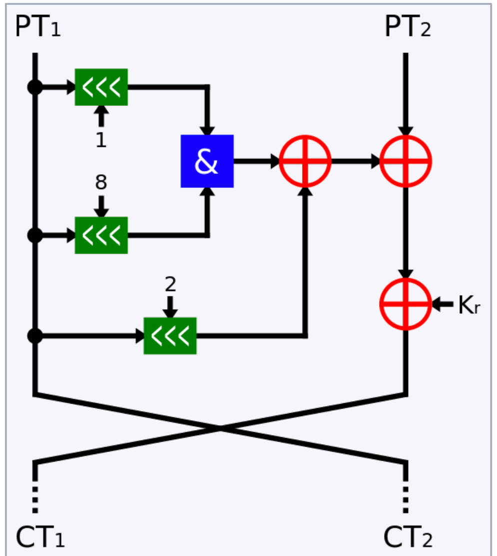
Reference code for SIMON64/96, and SIMON64/128. For Feistel ciphers like SIMON, it is a common trick to encrypt two rounds at a time so as to avoid swapping words. The same sort of trick can be applied to the various key schedules, which we do.

Simon 32/64 (2015)

```c
void Encrypt ( u16 text[], u16 crypt[], u16 key[] )
{
    u8 i;
    u16 tmp;
    crypt[0] = text[0];
    crypt[1] = text[1];

    for ( i=0 ; i<32 ; i++ )
    {
        tmp = crypt[0];
        crypt[0] = crypt[1] ^ ((ROTATE_LEFT_16(crypt[0],1)) & (ROTATE_LEFT_16(crypt[0],8))) ^ (ROTATE_LEFT_16(crypt[0],2)) ^ key[i];
        crypt[1] = tmp;
        printf("Check :%x, %x\n", crypt[0],crypt[1]);
    }
}
```

```
Check :0, ffff
Check :0, 0
Check :ffff, 0
Check :ffff, ffff
Check :fffd, ffff
Check :9df1, fffd
Check :d3b8, 9df1
Check :18a2, d3b8
Check :50e, 18a2
Check :fda7, 50e
Check :bbe6, fda7
Check :cd5c, bbe6
Check :67d8, cd5c
```

Simon



One round of Simon

Decrypt

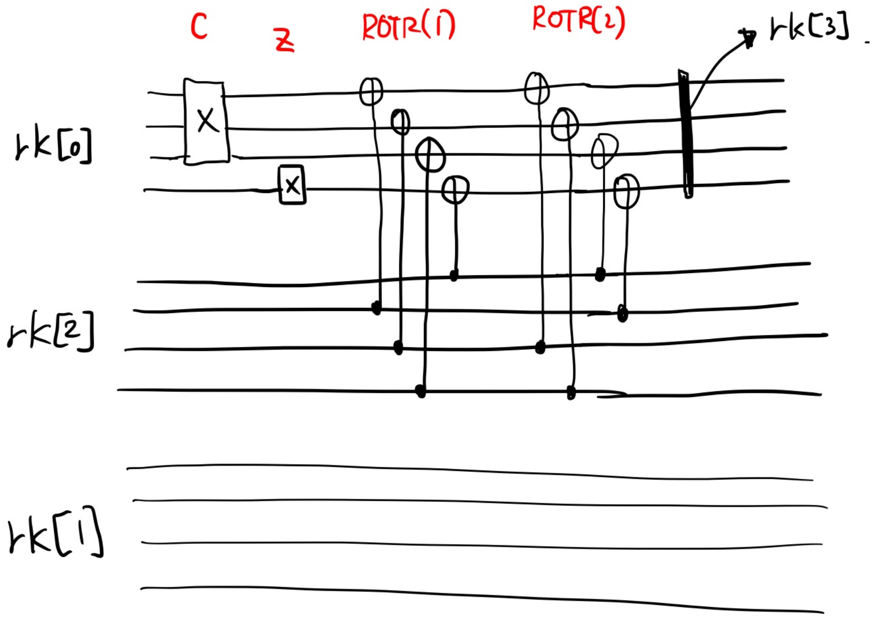
```
void Simon6496Decrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
int i;
Pt[1]=Ct[1]; Pt[0]=Ct[0];
for(i=41;i>=0;) R32x2(Pt[0],Pt[1],rk[i--],rk[i--]);
}
```

똑같은데 , rk[i] 사용이 내림차순

뒤까지 만든 후, 역으로 접근
→ Qubit 절약 가능, CNOT 증가

다 만들고 Decrypt
→ Qubit 증가, CNOT 감소

다른 방법 ??



[3] [41]

→ [2] [40]

[1] [39]

Q & A