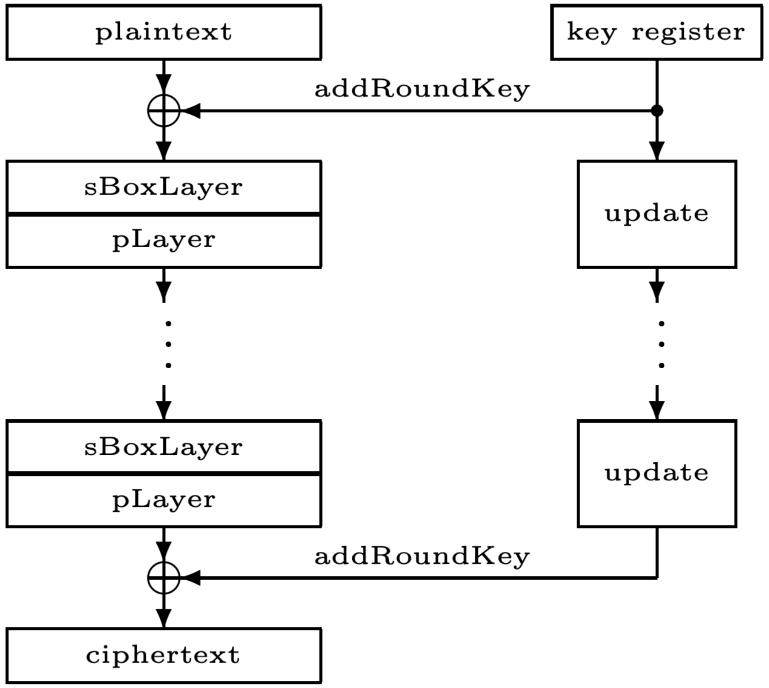
PRESENT CUDA 구현

https://youtu.be/zF7cjr63h8c

Present

64-bit state, 80 or 128 bits key, 31 round



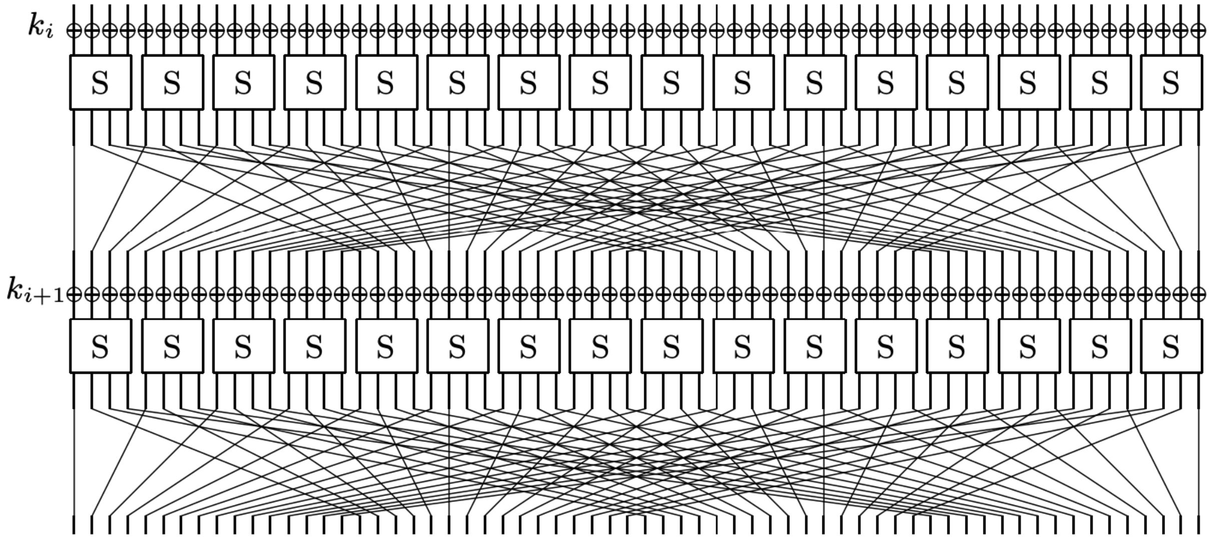
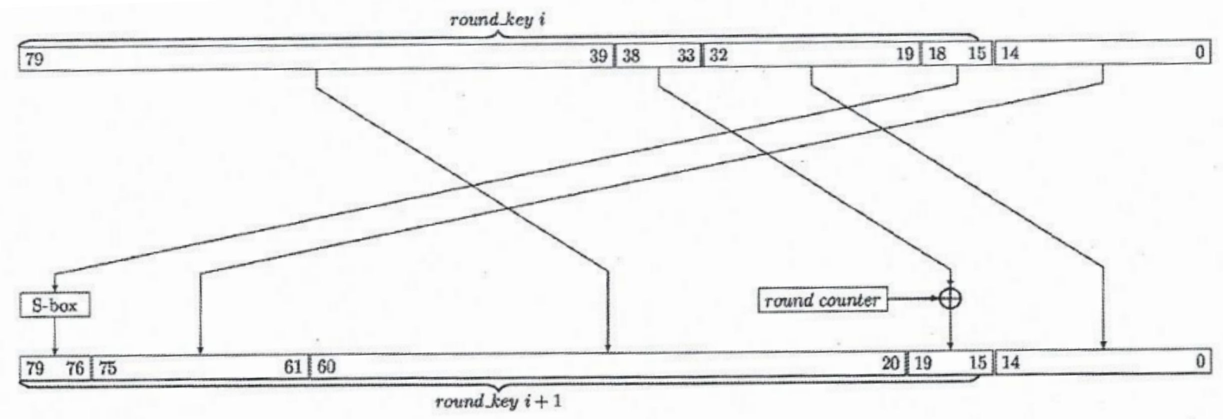


Fig. 2. . The S/P network for PRESENT.



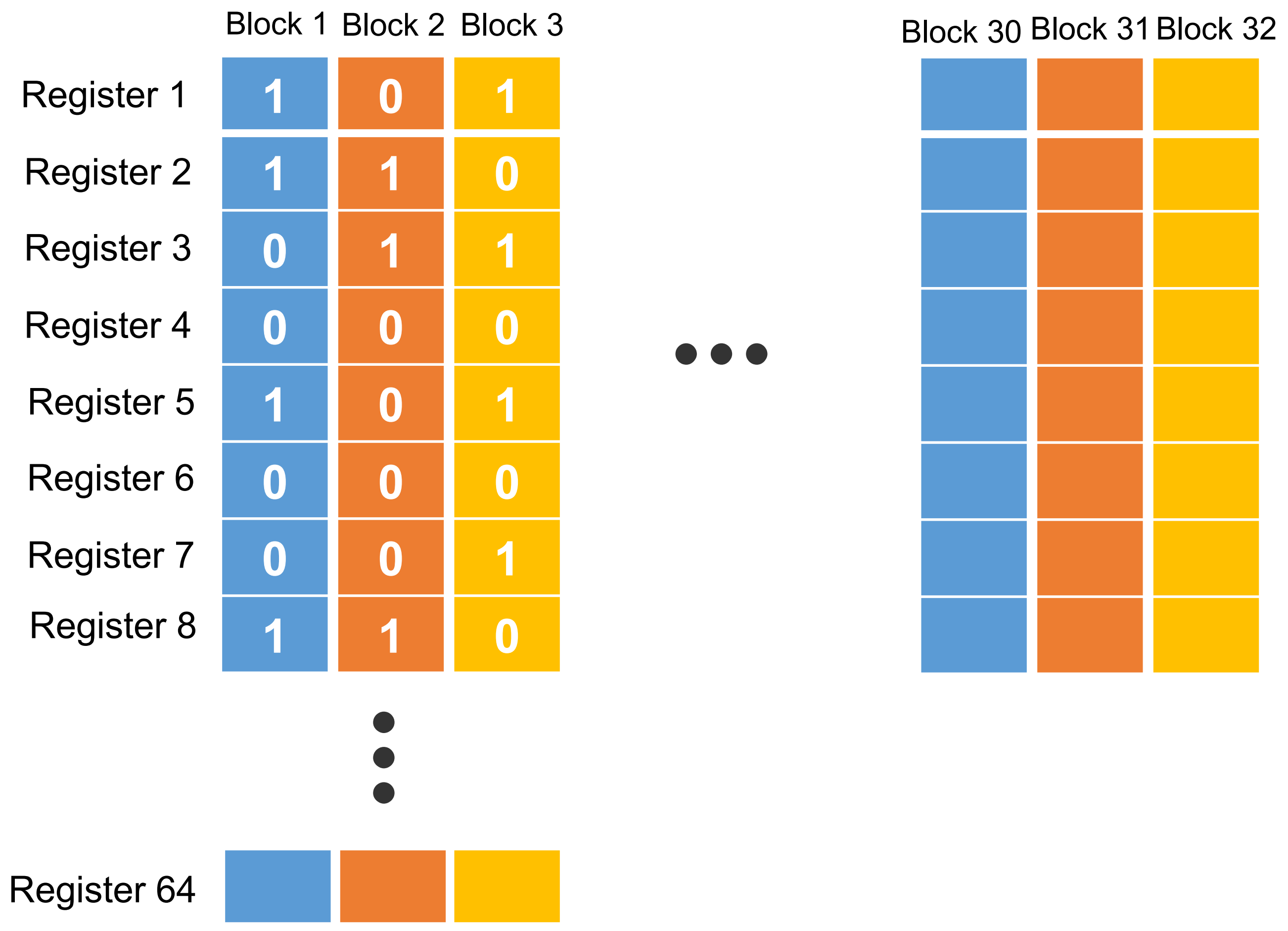
Key schedule algorithm of PRESENT-80 block cipher. Figure is drawn by Florian Delporte and it is publicly available [20] under Creative Commons license CC0.

PRESENTCUDA 구현

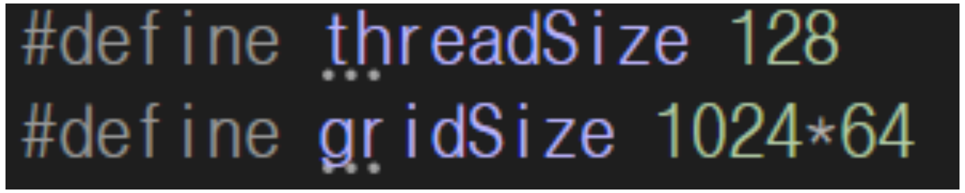
- 키 탐색 구현 – 키 스케줄 + 암호화
- 32블록 병렬 비트슬라이싱
- 암호화의 라운드 연산 과정 중 라운드 키 update 구현
- 공유메모리에 라운드 키 저장

비트슬라이싱

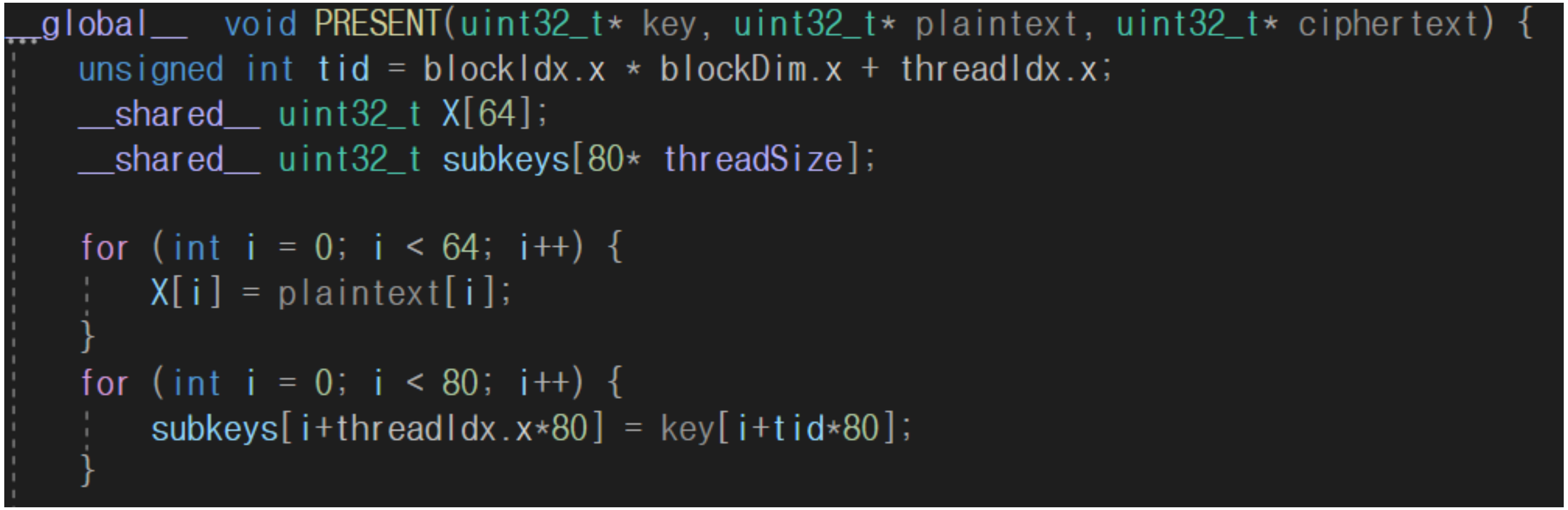
- GPU의 32비트 코어에 맞추어 32비트 블록 병렬 연산



공유메모리에 라운드 키 저장



```
#define threadSize 128
#define gridSize 1024*64
```

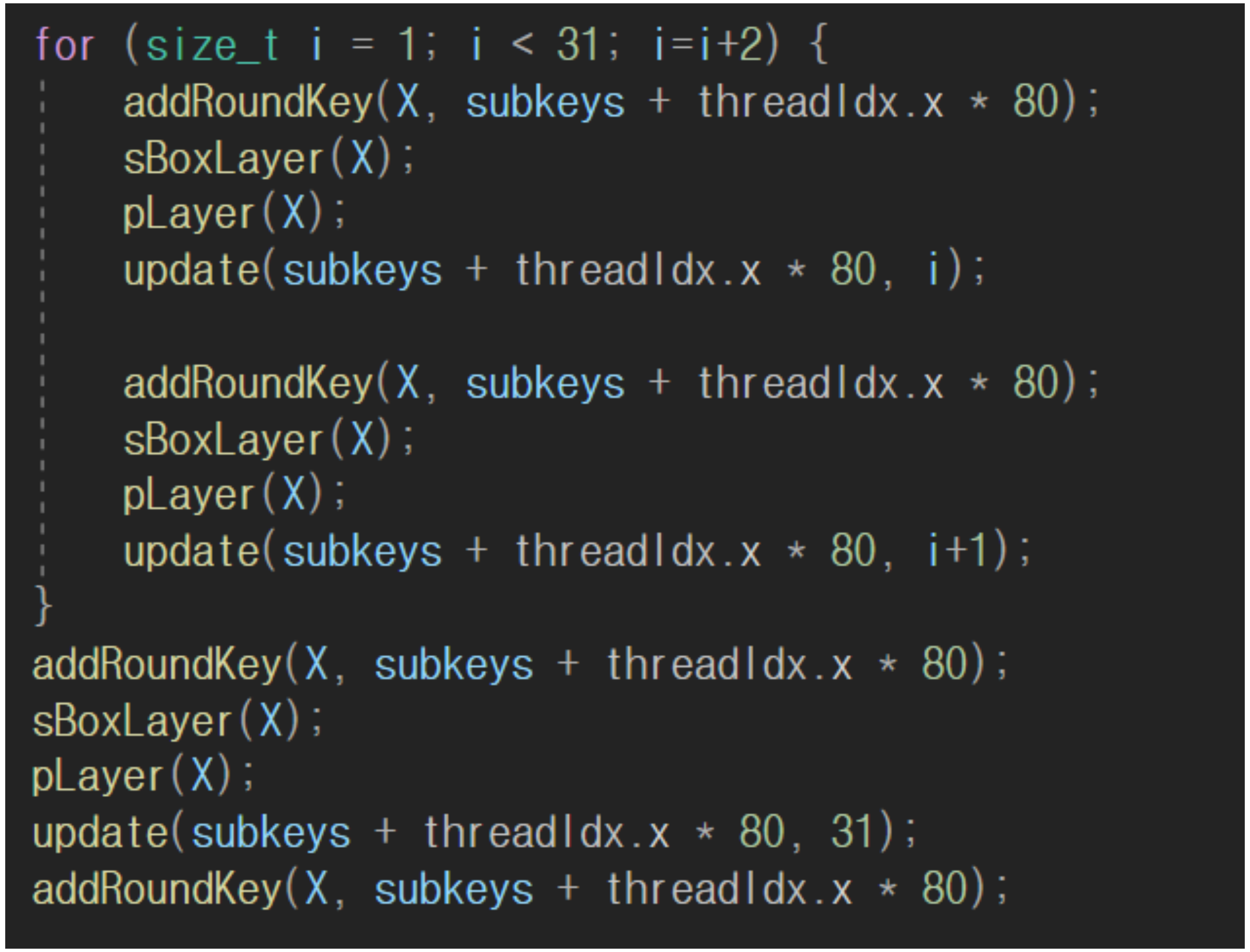


```
__global__ void PRESENT(uint32_t* key, uint32_t* plaintext, uint32_t* ciphertext) {
    unsigned int tid = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ uint32_t X[64];
    __shared__ uint32_t subkeys[80* threadSize];

    for (int i = 0; i < 64; i++) {
        X[i] = plaintext[i];
    }
    for (int i = 0; i < 80; i++) {
        subkeys[i+threadIdx.x*80] = key[i+tid*80];
    }
```

연산 과정 중 라운드 키 update

- 비트슬라이싱 구현으로 많은 레지스터가 사용
- 루프 언롤링



```
for (size_t i = 1; i < 31; i=i+2) {
    addRoundKey(X, subkeys + threadIdx.x * 80);
    sBoxLayer(X);
    pLayer(X);
    update(subkeys + threadIdx.x * 80, i);

    addRoundKey(X, subkeys + threadIdx.x * 80);
    sBoxLayer(X);
    pLayer(X);
    update(subkeys + threadIdx.x * 80, i+1);
}
addRoundKey(X, subkeys + threadIdx.x * 80);
sBoxLayer(X);
pLayer(X);
update(subkeys + threadIdx.x * 80, 31);
addRoundKey(X, subkeys + threadIdx.x * 80);
```

ADD Round key

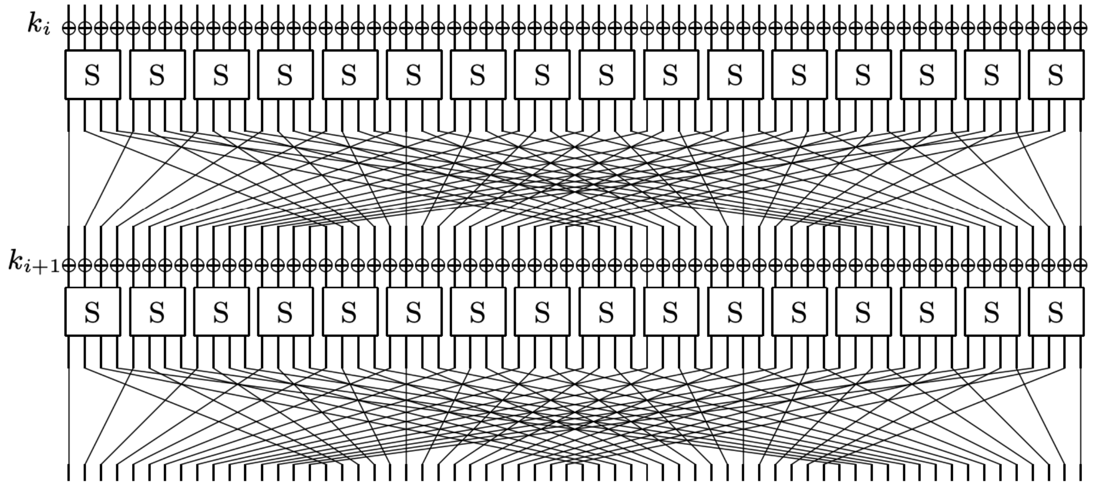
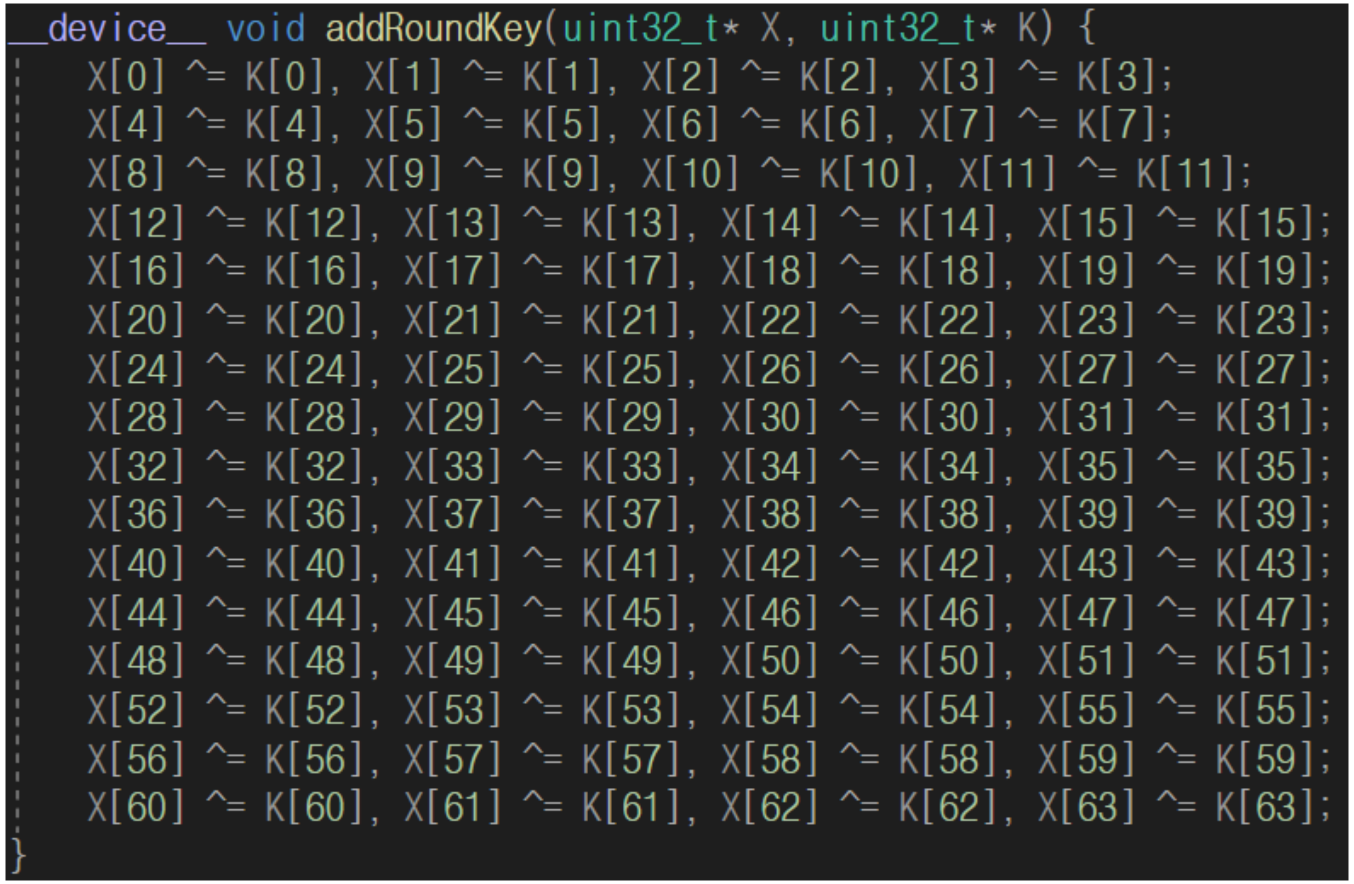


Fig. 2. . The S/P network for PRESENT.



```
__device__ void addRoundKey(uint32_t* X, uint32_t* K) {
    X[0] ^= K[0], X[1] ^= K[1], X[2] ^= K[2], X[3] ^= K[3];
    X[4] ^= K[4], X[5] ^= K[5], X[6] ^= K[6], X[7] ^= K[7];
    X[8] ^= K[8], X[9] ^= K[9], X[10] ^= K[10], X[11] ^= K[11];
    X[12] ^= K[12], X[13] ^= K[13], X[14] ^= K[14], X[15] ^= K[15];
    X[16] ^= K[16], X[17] ^= K[17], X[18] ^= K[18], X[19] ^= K[19];
    X[20] ^= K[20], X[21] ^= K[21], X[22] ^= K[22], X[23] ^= K[23];
    X[24] ^= K[24], X[25] ^= K[25], X[26] ^= K[26], X[27] ^= K[27];
    X[28] ^= K[28], X[29] ^= K[29], X[30] ^= K[30], X[31] ^= K[31];
    X[32] ^= K[32], X[33] ^= K[33], X[34] ^= K[34], X[35] ^= K[35];
    X[36] ^= K[36], X[37] ^= K[37], X[38] ^= K[38], X[39] ^= K[39];
    X[40] ^= K[40], X[41] ^= K[41], X[42] ^= K[42], X[43] ^= K[43];
    X[44] ^= K[44], X[45] ^= K[45], X[46] ^= K[46], X[47] ^= K[47];
    X[48] ^= K[48], X[49] ^= K[49], X[50] ^= K[50], X[51] ^= K[51];
    X[52] ^= K[52], X[53] ^= K[53], X[54] ^= K[54], X[55] ^= K[55];
    X[56] ^= K[56], X[57] ^= K[57], X[58] ^= K[58], X[59] ^= K[59];
    X[60] ^= K[60], X[61] ^= K[61], X[62] ^= K[62], X[63] ^= K[63];
}
```

SBOX Layer

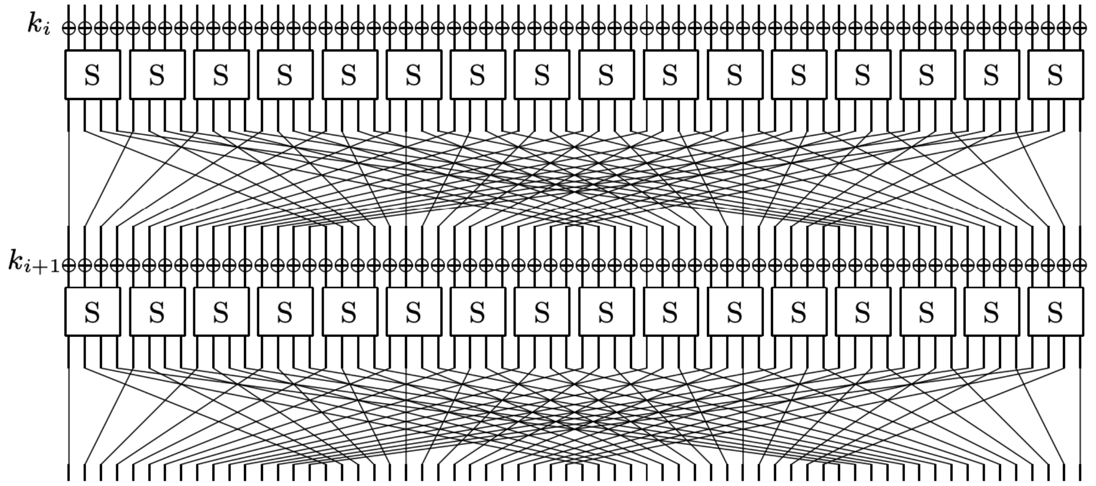
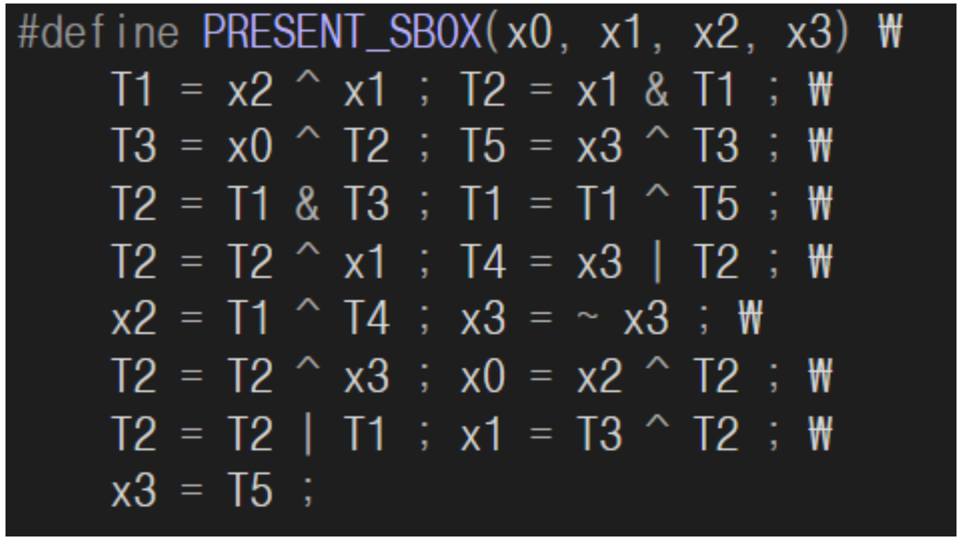
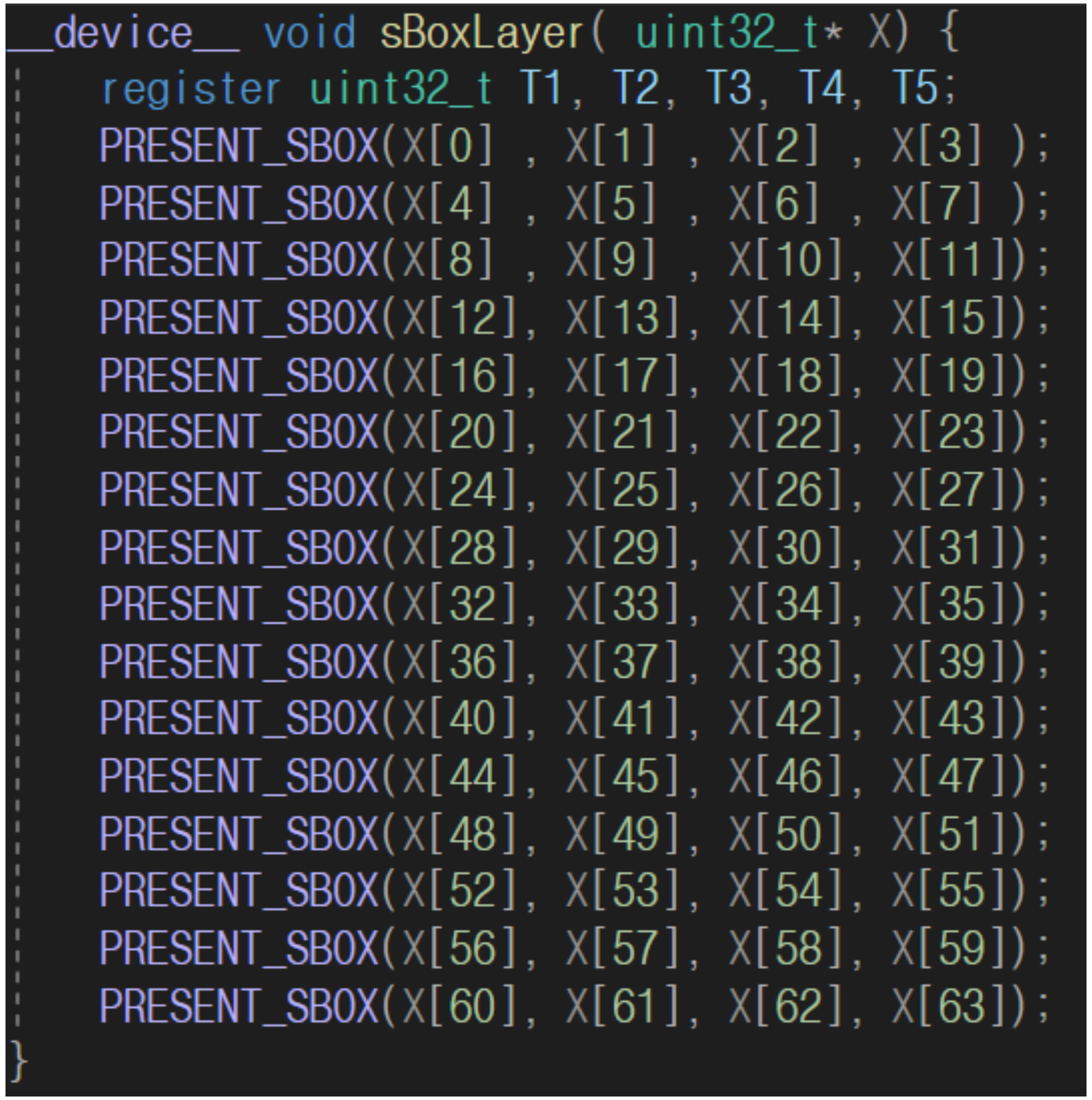


Fig. 2. . The S/P network for PRESENT.

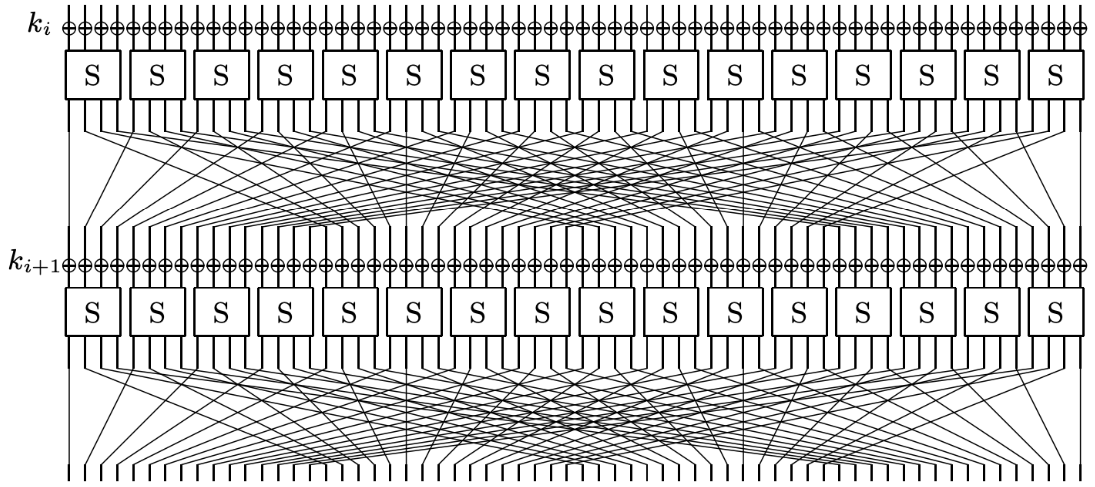


```
#define PRESENT_SBOX(x0, x1, x2, x3) ₩
    T1 = x2 ^ x1 ; T2 = x1 & T1 ; ₩
    T3 = x0 ^ T2 ; T5 = x3 ^ T3 ; ₩
    T2 = T1 & T3 ; T1 = T1 ^ T5 ; ₩
    T2 = T2 ^ x1 ; T4 = x3 | T2 ; ₩
    x2 = T1 ^ T4 ; x3 = ~ x3 ; ₩
    T2 = T2 ^ x3 ; x0 = x2 ^ T2 ; ₩
    T2 = T2 | T1 ; x1 = T3 ^ T2 ; ₩
    x3 = T5 ;
```



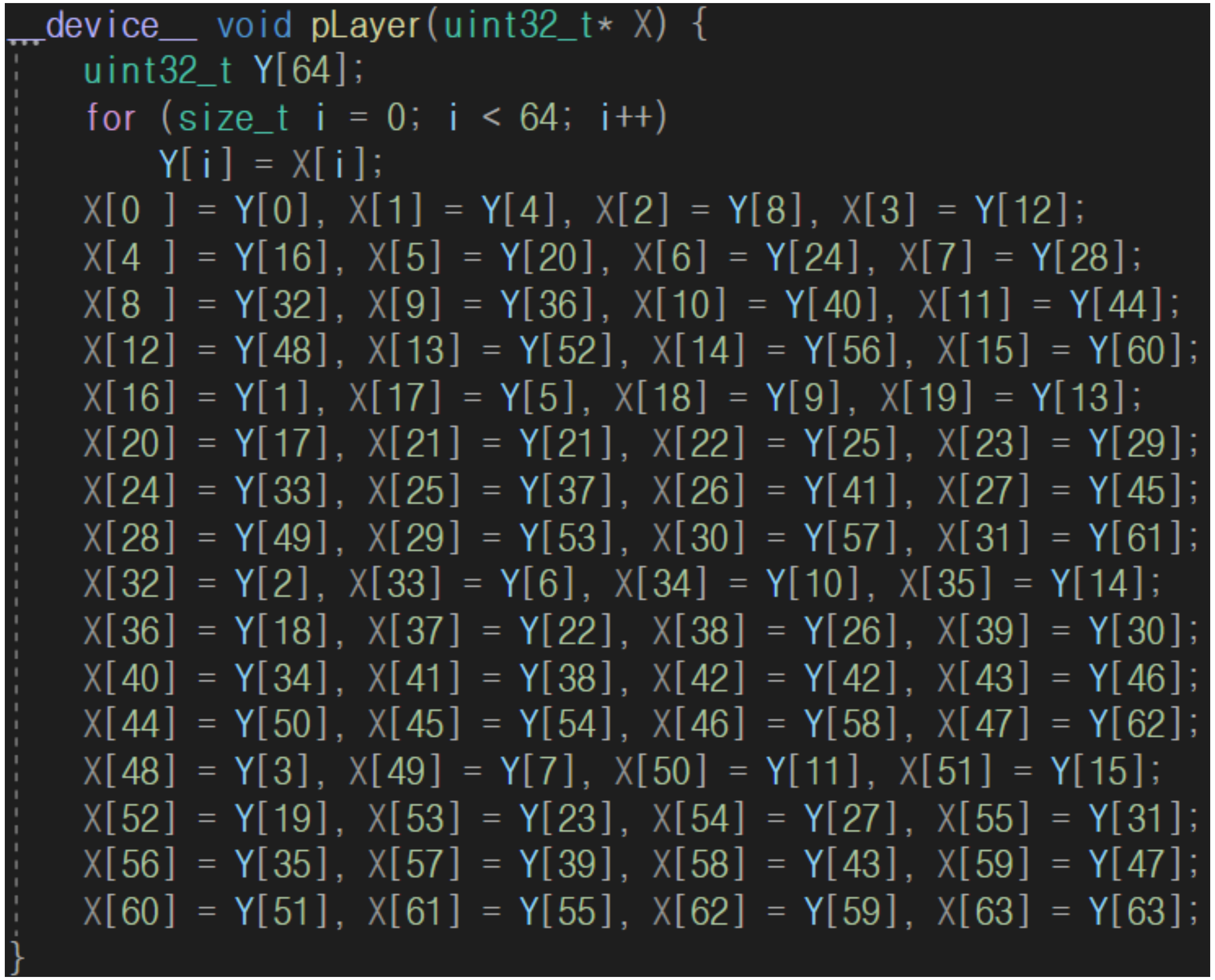
```
__device__ void sBoxLayer( uint32_t* X) {
    register uint32_t T1, T2, T3, T4, T5;
    PRESENT_SBOX(X[0] , X[1] , X[2] , X[3] );
    PRESENT_SBOX(X[4] , X[5] , X[6] , X[7] );
    PRESENT_SBOX(X[8] , X[9] , X[10], X[11]);
    PRESENT_SBOX(X[12], X[13], X[14], X[15]);
    PRESENT_SBOX(X[16], X[17], X[18], X[19]);
    PRESENT_SBOX(X[20], X[21], X[22], X[23]);
    PRESENT_SBOX(X[24], X[25], X[26], X[27]);
    PRESENT_SBOX(X[28], X[29], X[30], X[31]);
    PRESENT_SBOX(X[32], X[33], X[34], X[35]);
    PRESENT_SBOX(X[36], X[37], X[38], X[39]);
    PRESENT_SBOX(X[40], X[41], X[42], X[43]);
    PRESENT_SBOX(X[44], X[45], X[46], X[47]);
    PRESENT_SBOX(X[48], X[49], X[50], X[51]);
    PRESENT_SBOX(X[52], X[53], X[54], X[55]);
    PRESENT_SBOX(X[56], X[57], X[58], X[59]);
    PRESENT_SBOX(X[60], X[61], X[62], X[63]);
}
```

Permutation Layer



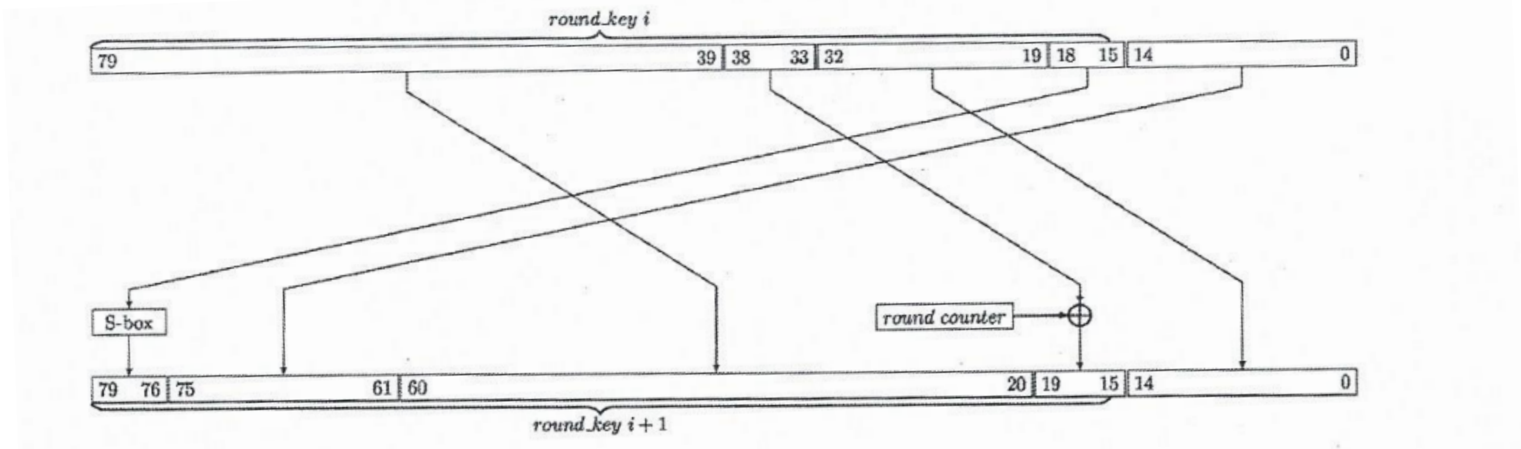
k_i
S S S S S S S S S S S S S S S S
k_{i+1}
S S S S S S S S S S S S S S S S

Fig. 2. . The S/P network for PRESENT.

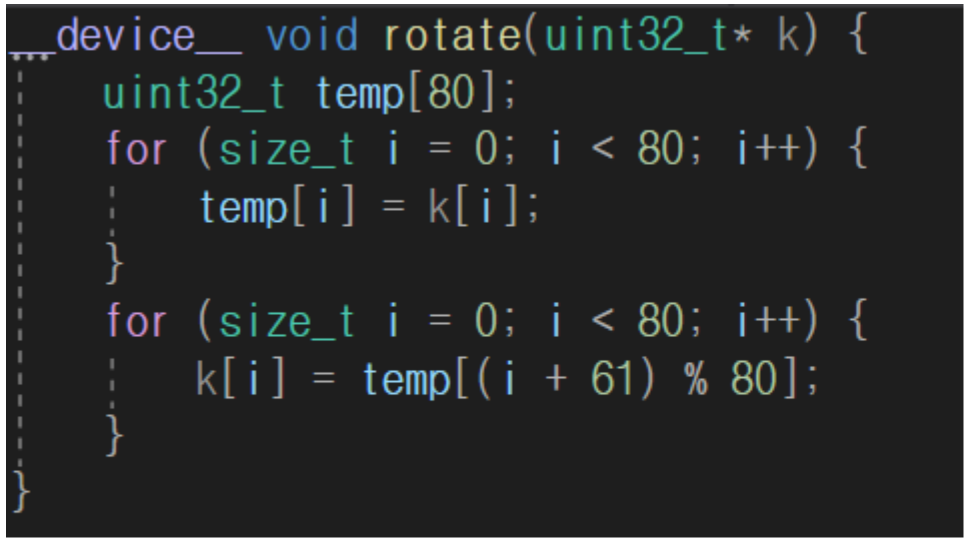


```
__device__ void pLayer(uint32_t* X) {
    uint32_t Y[64];
    for (size_t i = 0; i < 64; i++)
        Y[i] = X[i];
    X[0 ] = Y[0], X[1] = Y[4], X[2] = Y[8], X[3] = Y[12];
    X[4 ] = Y[16], X[5] = Y[20], X[6] = Y[24], X[7] = Y[28];
    X[8 ] = Y[32], X[9] = Y[36], X[10] = Y[40], X[11] = Y[44];
    X[12] = Y[48], X[13] = Y[52], X[14] = Y[56], X[15] = Y[60];
    X[16] = Y[1], X[17] = Y[5], X[18] = Y[9], X[19] = Y[13];
    X[20] = Y[17], X[21] = Y[21], X[22] = Y[25], X[23] = Y[29];
    X[24] = Y[33], X[25] = Y[37], X[26] = Y[41], X[27] = Y[45];
    X[28] = Y[49], X[29] = Y[53], X[30] = Y[57], X[31] = Y[61];
    X[32] = Y[2], X[33] = Y[6], X[34] = Y[10], X[35] = Y[14];
    X[36] = Y[18], X[37] = Y[22], X[38] = Y[26], X[39] = Y[30];
    X[40] = Y[34], X[41] = Y[38], X[42] = Y[42], X[43] = Y[46];
    X[44] = Y[50], X[45] = Y[54], X[46] = Y[58], X[47] = Y[62];
    X[48] = Y[3], X[49] = Y[7], X[50] = Y[11], X[51] = Y[15];
    X[52] = Y[19], X[53] = Y[23], X[54] = Y[27], X[55] = Y[31];
    X[56] = Y[35], X[57] = Y[39], X[58] = Y[43], X[59] = Y[47];
    X[60] = Y[51], X[61] = Y[55], X[62] = Y[59], X[63] = Y[63];
}
```

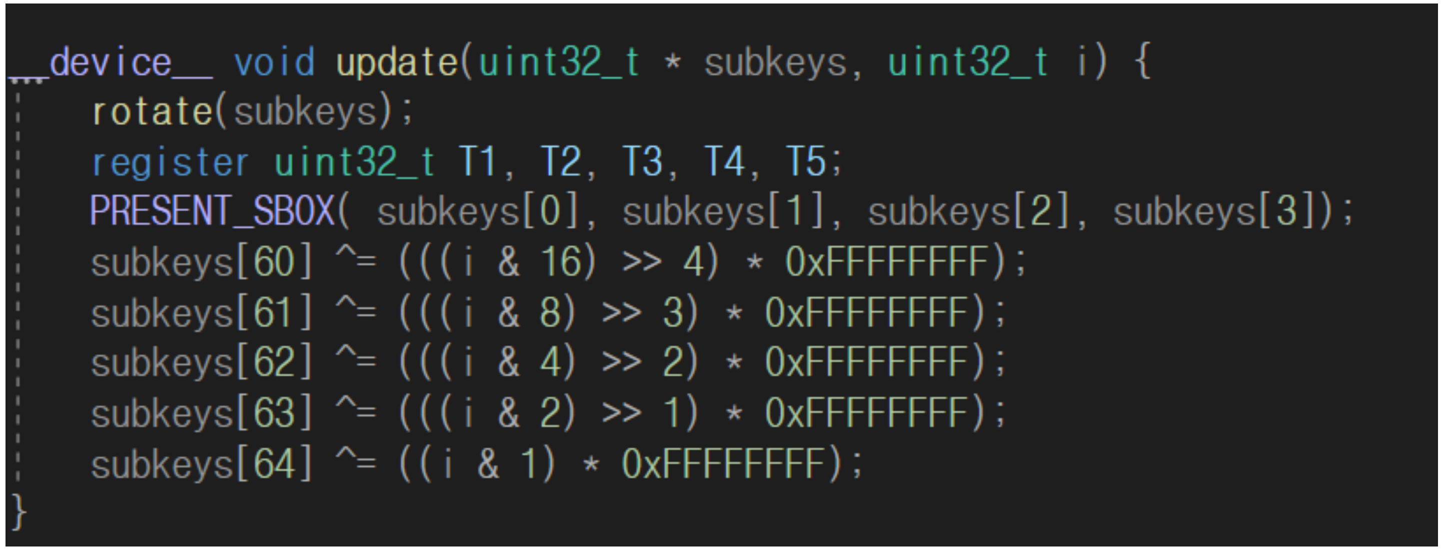
Key update



Key schedule algorithm of PRESENT-80 block cipher. Figure is drawn by Florian Delporte and it is publicly available [20] under Creative Commons license CC0.



```
__device__ void rotate(uint32_t* k) {
    uint32_t temp[80];
    for (size_t i = 0; i < 80; i++) {
        temp[i] = k[i];
    }
    for (size_t i = 0; i < 80; i++) {
        k[i] = temp[(i + 61) % 80];
    }
}
```



```
__device__ void update(uint32_t * subkeys, uint32_t i) {
    rotate(subkeys);
    register uint32_t T1, T2, T3, T4, T5;
    PRESENT_SBOX( subkeys[0], subkeys[1], subkeys[2], subkeys[3]);
    subkeys[60] ^= (((i & 16) >> 4) * 0xFFFFFFFF);
    subkeys[61] ^= (((i & 8) >> 3) * 0xFFFFFFFF);
    subkeys[62] ^= (((i & 4) >> 2) * 0xFFFFFFFF);
    subkeys[63] ^= (((i & 2) >> 1) * 0xFFFFFFFF);
    subkeys[64] ^= ((i & 1) * 0xFFFFFFFF);
}
```

성능 비교

- Table-based implementation
  (Cihangir Tezcan , “Key lengths revisited: GPU-based brute force cryptanalysis of DES, 3DES, and PRESENT” (2022))

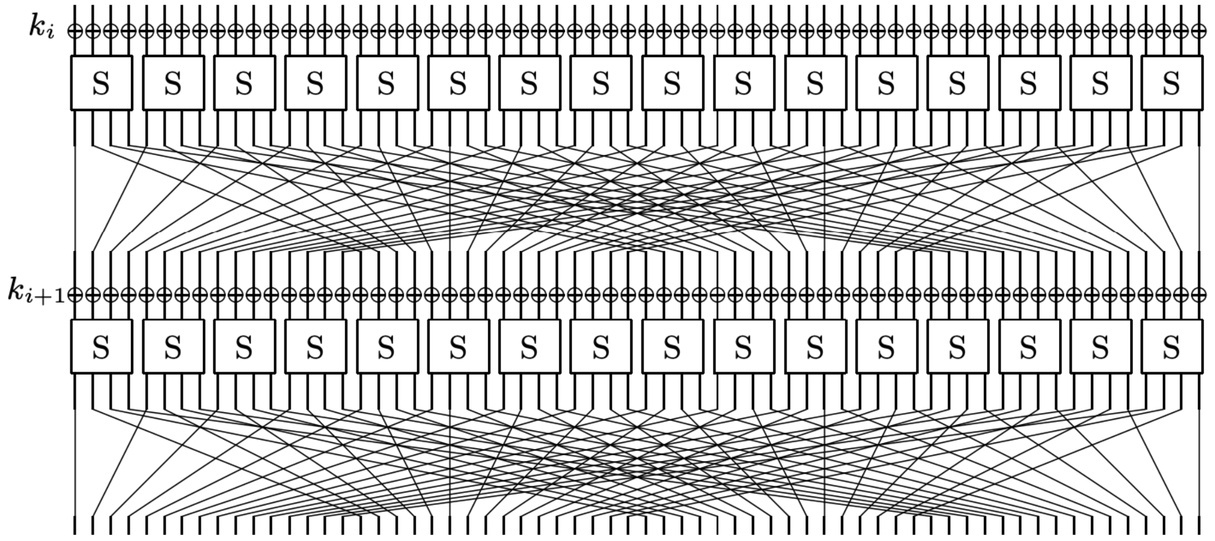
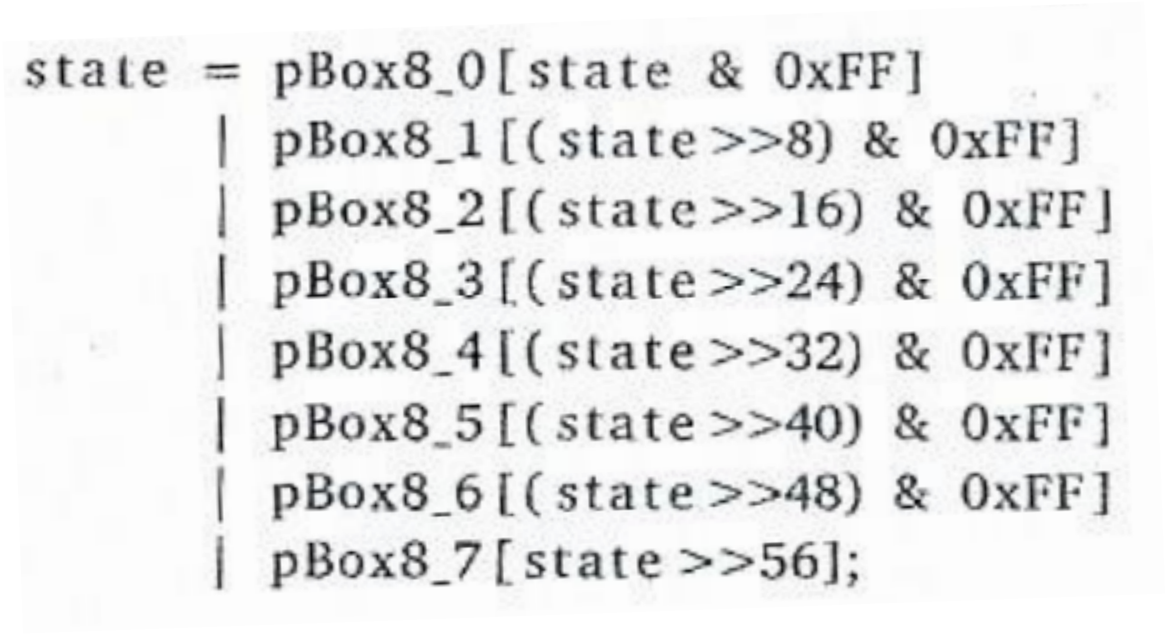


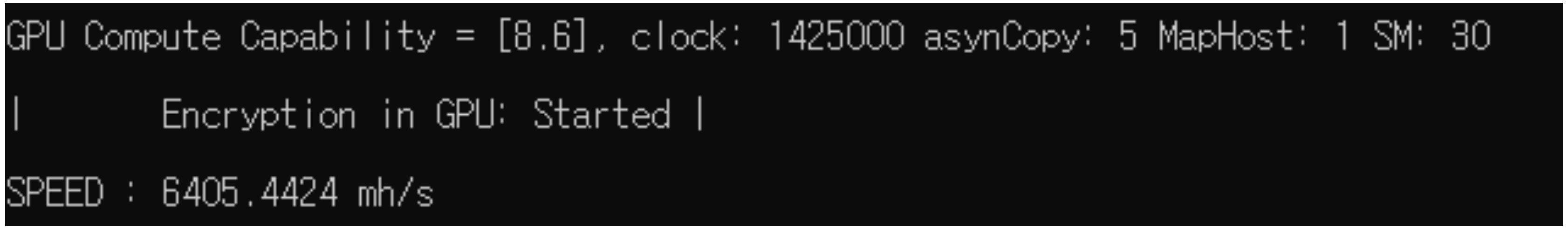
Fig. 2. . The S/P network for PRESENT.



```
state = pBox8_0[state & 0xFF]
      | pBox8_1[(state>>8) & 0xFF]
      | pBox8_2[(state>>16) & 0xFF]
      | pBox8_3[(state>>24) & 0xFF]
      | pBox8_4[(state>>32) & 0xFF]
      | pBox8_5[(state>>40) & 0xFF]
      | pBox8_6[(state>>48) & 0xFF]
      | pBox8_7[state>>56];
```

성능 비교

- RTX 3060 환경, Visual studio
- RTX 3070 과 비교 3.39 x



```
GPU Compute Capability = [8.6], clock: 1425000 asynCopy: 5 MapHost: 1 SM: 30

|       Encryption in GPU: Started |

SPEED : 6405.4424 mh/s
```

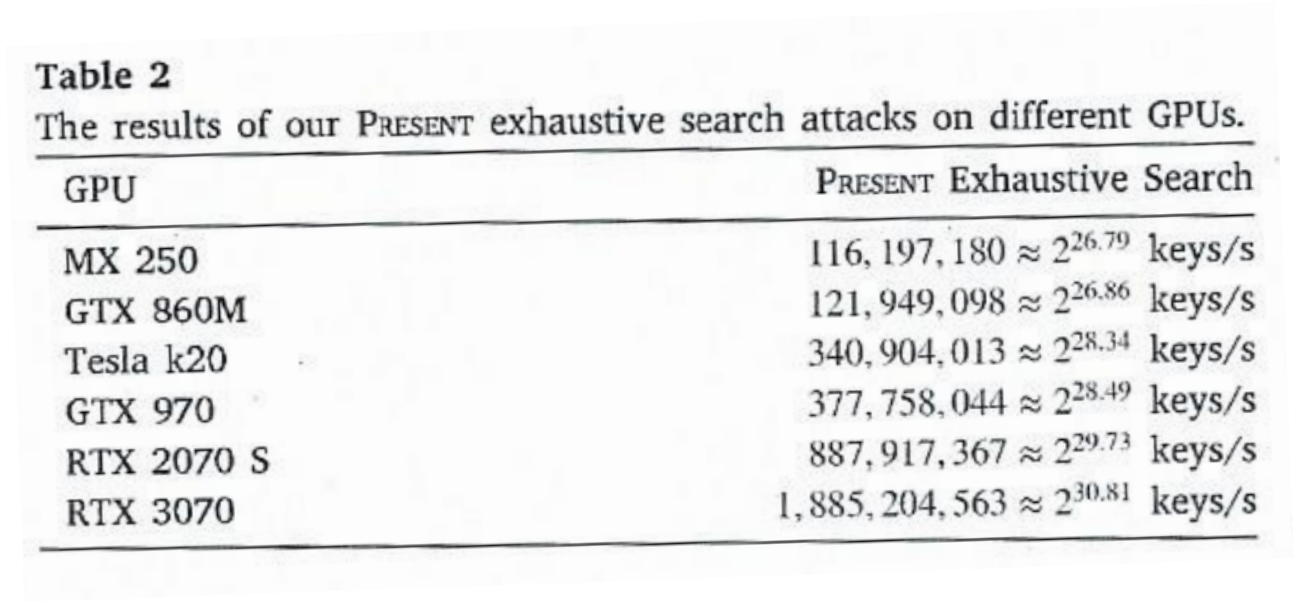


Table 2
The results of our PRESENT exhaustive search attacks on different GPUs.

GPU	PRESENT Exhaustive Search
MX 250	$116,197,180 \approx 2^{26.79}$ keys/s
GTX 860M	$121,949,098 \approx 2^{26.86}$ keys/s
Tesla k20	$340,904,013 \approx 2^{28.34}$ keys/s
GTX 970	$377,758,044 \approx 2^{28.49}$ keys/s
RTX 2070 S	$887,917,367 \approx 2^{29.73}$ keys/s
RTX 3070	$1,885,204,563 \approx 2^{30.81}$ keys/s

Q & A