PLU Decomposition(2) 응용

발표자: 양유진

링크:https://youtu.be/kAZICUhYnFU

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

CHAM이란?

- ICISC'17에서 발표되었음.
- stateless 기반 라운드 키를 사용하는 국산 초경량 블록 암호.
- 저사양 사물인터넷 플랫폼 대상으로 함.
- ARX(Addition, Rotation, XOR)연산을 사용함
- ARX 기반의 Feistel 구조로 구성됨.
  - 홀수 라운드

$$X_{i+1}[3] \leftarrow \mathrm{ROL}_8((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_1(X_i[1]) \oplus RK[i \bmod 2k/w])),$$
$$X_{i+1}[j] \leftarrow X_i[j+1] \text{ for } 0 \le j \le 2,$$

  - 짝수 라운드

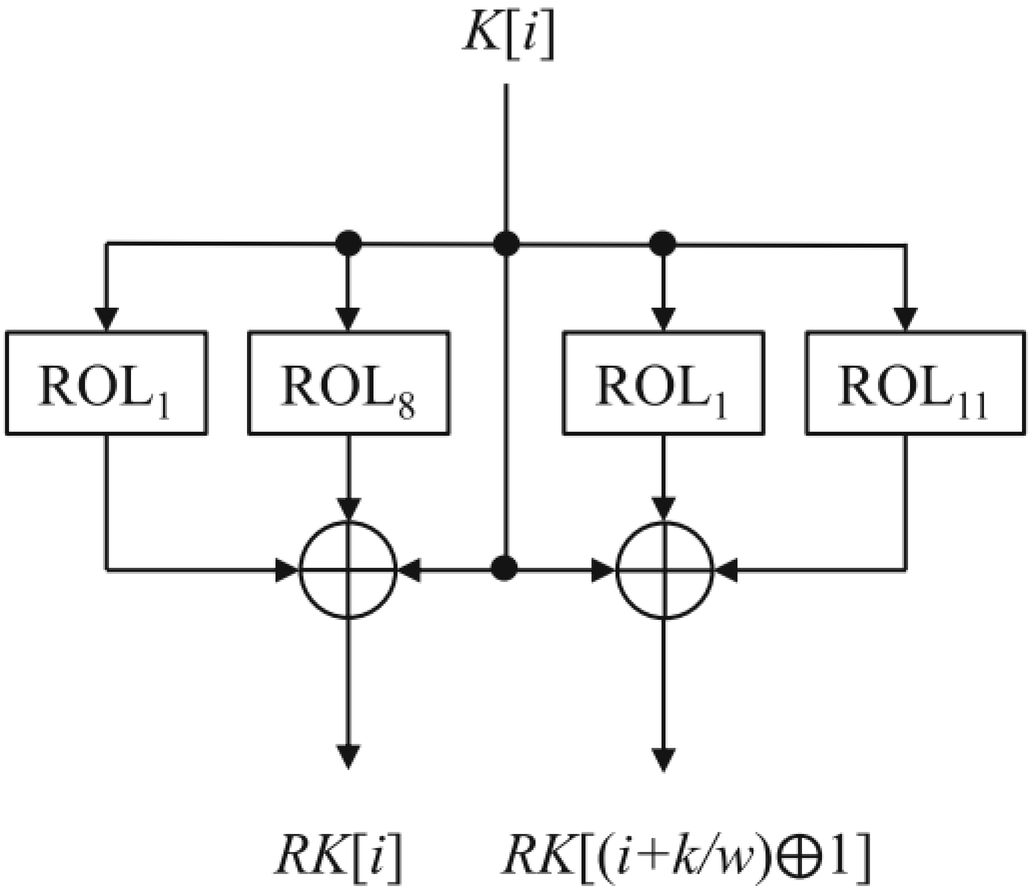
$$X_{i+1}[3] \leftarrow \mathrm{ROL}_1((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_8(X_i[1]) \oplus RK[i \bmod 2k/w])),$$
$$X_{i+1}[j] \leftarrow X_i[j+1] \text{ for } 0 \le j \le 2,$$

$$0 \le i < r$$

Table 3. List of CHAM ciphers and their parameters.

Cipher	n	k	r	w	k/w
CHAM-64/128	64	128	80	16	8
CHAM-128/128	128	128	80	32	4
CHAM-128/256	128	256	96	32	8

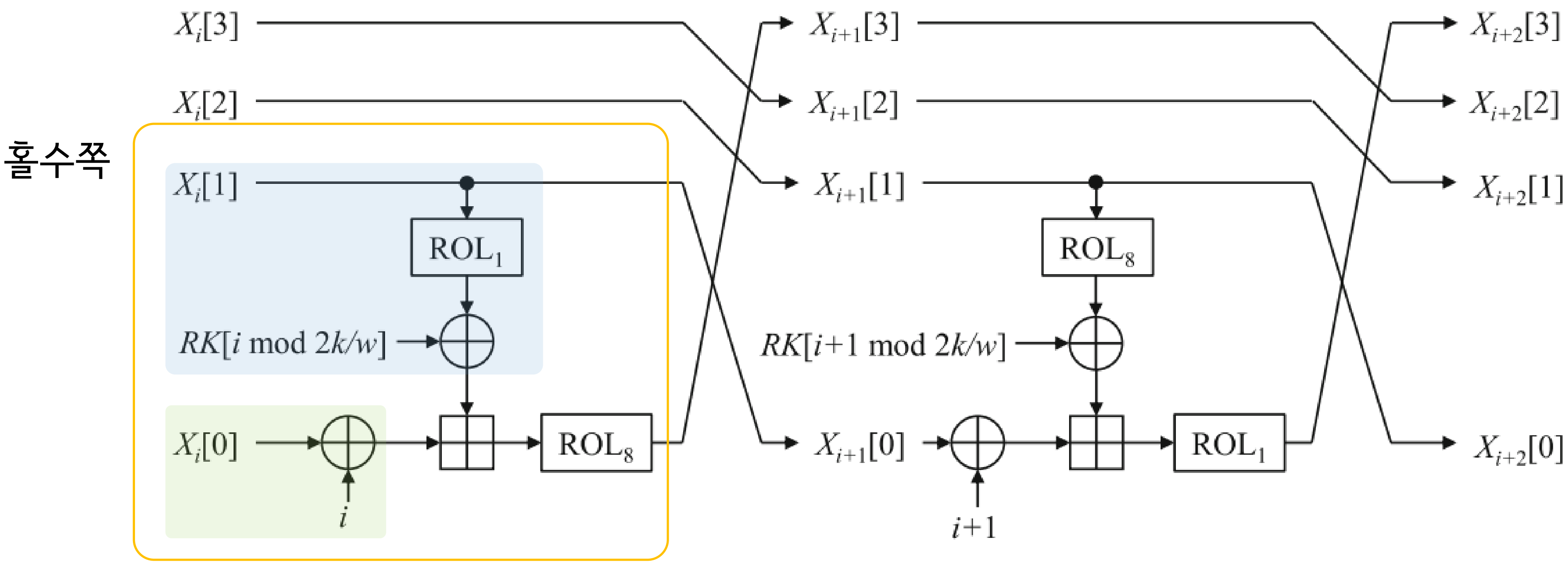
CHAM Key schedule



$$RK[i] \leftarrow K[i] \oplus \mathrm{ROL}_1(K[i]) \oplus \mathrm{ROL}_8(K[i]),$$
$$RK[(i+k/w) \oplus 1] \leftarrow K[i] \oplus \mathrm{ROL}_1(K[i]) \oplus \mathrm{ROL}_{11}(K[i]),$$

CHAM Round 함수 구조

$x \boxplus y$: addition of x and y modulo 2^w



$$X_{i+1}[3] \leftarrow \boxed{\mathrm{ROL}_8((X_i[0] \oplus i) \boxplus (\mathrm{ROL}_1(X_i[1]) \oplus RK[i \bmod 2k/w]))},$$
$$X_{i+1}[j] \leftarrow X_i[j+1] \text{ for } 0 \leq j \leq 2,$$

Parallel CHAM Round 함수 구조

i=0~2 에 대해서만 병렬 덧셈 수행

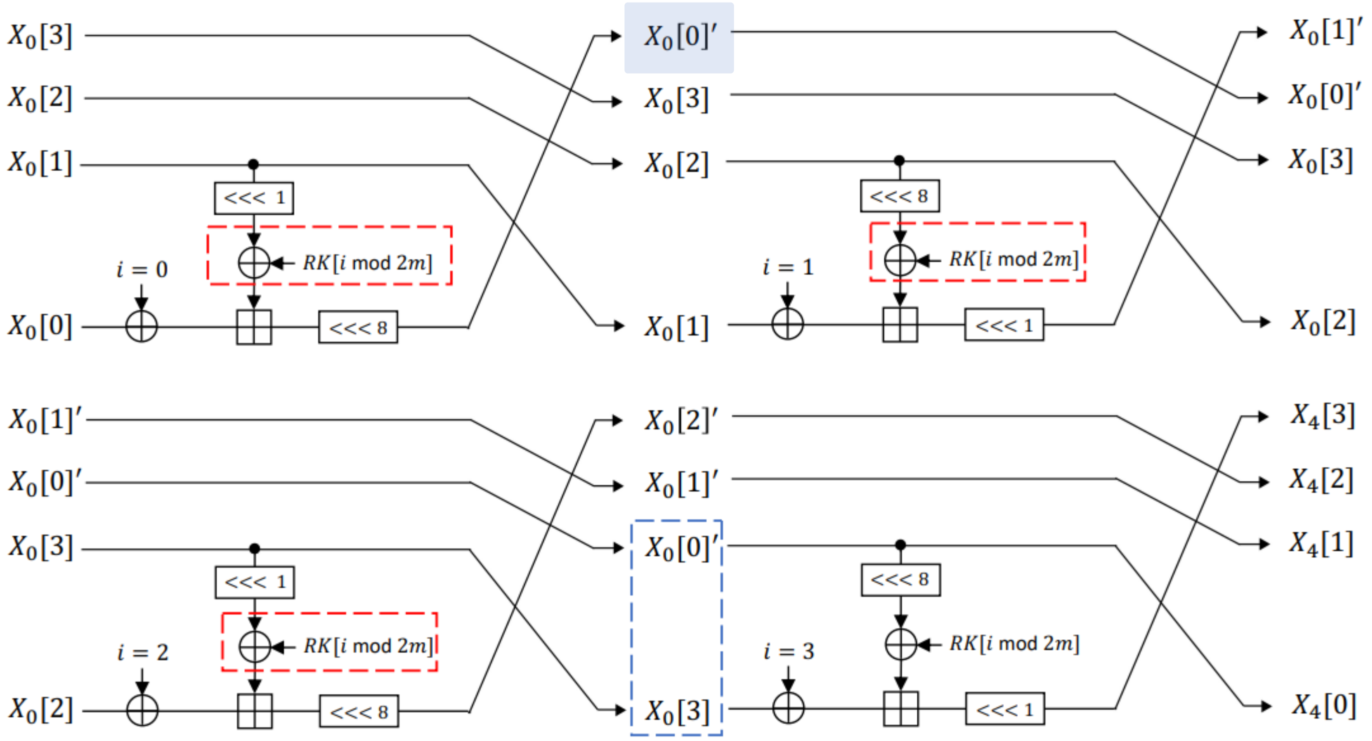


Fig. 2: Four round functions of CHAM ($i = 0, 1, 2, 3$).

i=3일 때, $X_0[0]'$ (i=0일 때 round 함수 덧셈 결과)이 필요함

PLU 분해를 적용한 CHAM Key schedule

PLU분해를 이용하여 병렬 구조로 만들어진 CHAM의 Key Schedule을 개선함.

$$RK[i] = K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 8),$$

$$RK[(i + k/w) \oplus 1] = K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 11) \cdots (1)$$

라운드키 생성식

연립선형방정식은 선형행렬로 표현할 수 있음

sage math를 활용하여 생성

```
[1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1]
[1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
[0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1]
[1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1]
```

16 x 16

[1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1]

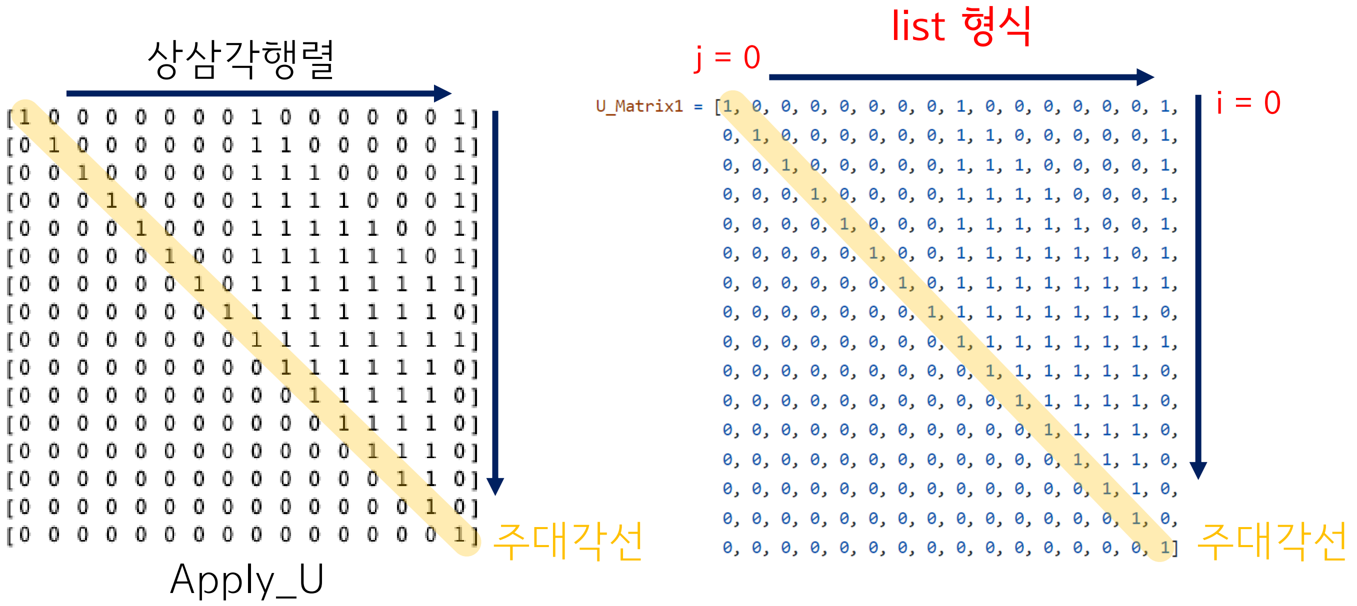
$K[i]$ $\quad$ $K[i] \lll 8$ $\quad$ $K[i] \lll 1$

PLU 분해를 적용한 CHAM Key schedule

Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key $K_{0\sim7}$, word size n, upper triangular matrix $U_{1\sim2}$, lower triangular matrix $L_{1\sim2}$
Output: Round key $RK_{0\sim7}$

```
Transform K_{0~7} :
  // Apply_U
  for i = 0 to n - 2
    for j = 0 to n - i - 2
      if U_{1~2}[(i * n) + 1 + i + j] == 1
        CNOT (K[i + j + 1], K[i])
  // Apply_L
  for i = 0 to n - 2
    for j = n - 1 to i + 1 step -1
      if L_{1~2}[n * (n - 1 - i) + n - 1 - j] == 1
        CNOT (K[n - 1 - j], K[n - 1 - i])
  // Apply_P
  if RK_{0~7} == True
    SWAP (K[12], K[11]), SWAP (K[11], K[13])
    SWAP (K[11], K[10]), SWAP (K[10], K[14])
    SWAP (K[10], K[9]), SWAP (K[9], K[15])
    return RK_{0~7}
  else
    SWAP (K[11], K[13]), SWAP (K[10], K[9])
    SWAP (K[7], K[8]), SWAP (K[5], K[6])
    return RK_{8~15}
Reverse(transform K_{0~7}) // Reverse operation
```

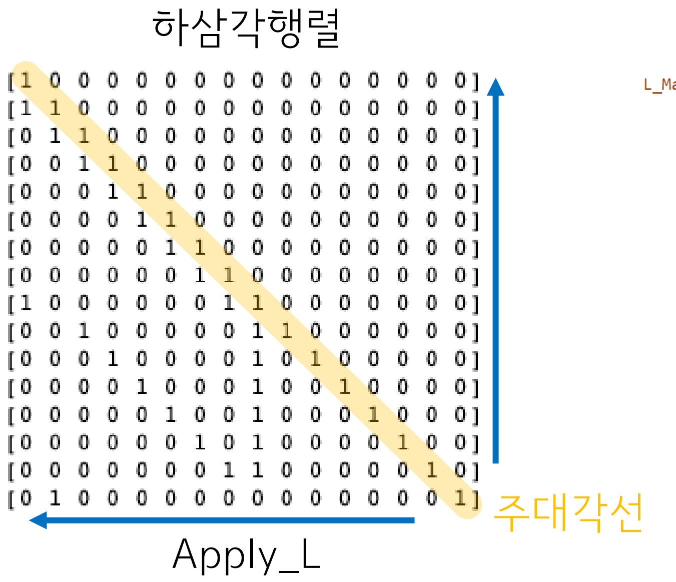
PLU 분해는 U, L, P 순서대로 진행됨.

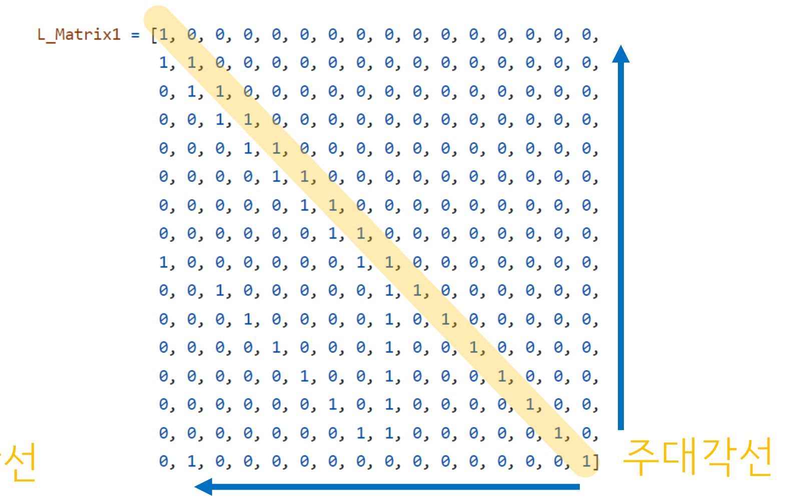


PLU 분해를 적용한 CHAM Key schedule

Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key $K_{0\sim7}$, word size n, upper triangular matrix $U_{1\sim2}$, lower triangular matrix $L_{1\sim2}$
Output: Round key $RK_{0\sim7}$

Transform $K_{0\sim7}$:
  // Apply_U
  for i = 0 **to** n - 2
    for j = 0 **to** n - i - 2
      if $U_{1\sim2}$[(i ∗ n) + 1 + i + j] == 1
        CNOT (K[i + j + 1], K[i])
  // Apply_L
  for i = 0 **to** n - 2
    for j = n - 1 **to** i + 1 **step** -1
      if $L_{1\sim2}$[n ∗ (n − 1 - i) + n − 1 - j] == 1
        CNOT (K[n − 1 - j], K[n − 1 − i])
  // Apply_P
  if $RK_{0\sim7}$ == True
    SWAP (K[12], K[11]), **SWAP** (K[11], K[13])
    SWAP (K[11], K[10]), **SWAP** (K[10], K[14])
    SWAP (K[10], K[9]), **SWAP** (K[9], K[15])
  return $RK_{0\sim7}$
  else
    SWAP (K[11], K[13]), **SWAP** (K[10], K[9])
    SWAP (K[7], K[8]), **SWAP** (K[5], K[6])
  return $RK_{8\sim15}$
Reverse(**transform** $K_{0\sim7}$) // Reverse operation





PLU 분해를 적용한 CHAM Key schedule

Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key $K_{0\sim7}$, word size n, upper triangular matrix $U_{1\sim2}$, lower triangular matrix $L_{1\sim2}$
Output: Round key $RK_{0\sim7}$

```
Transform K0~7 :
  // Apply_U
  for i = 0 to n - 2
    for j = 0 to n - i - 2
      if U1~2[(i * n) + 1 + i + j] == 1
        CNOT (K[i + j + 1], K[i])
  // Apply_L
  for i = 0 to n - 2
    for j = n - 1 to i + 1 step -1
      if L1~2[n * (n - 1 - i) + n - 1 - j] == 1
        CNOT (K[n - 1 - j], K[n - 1 - i])
  // Apply_P
  if RK0~7 == True
    SWAP (K[12], K[11]), SWAP (K[11], K[13])
    SWAP (K[11], K[10]), SWAP (K[10], K[14])
    SWAP (K[10], K[9]), SWAP (K[9], K[15])
  return RK0~7
  else
    SWAP (K[11], K[13]), SWAP (K[10], K[9])
    SWAP (K[7], K[8]), SWAP (K[5], K[6])
  return RK8~15
Reverse(transform K0~7) // Reverse operation
```

SWAP 게이트를 통해 치환행렬(P) 수행

$RK_{0\sim7}$은 U_1, L_1 사용

$RK_{8\sim15}$은 U_2, L_2 사용

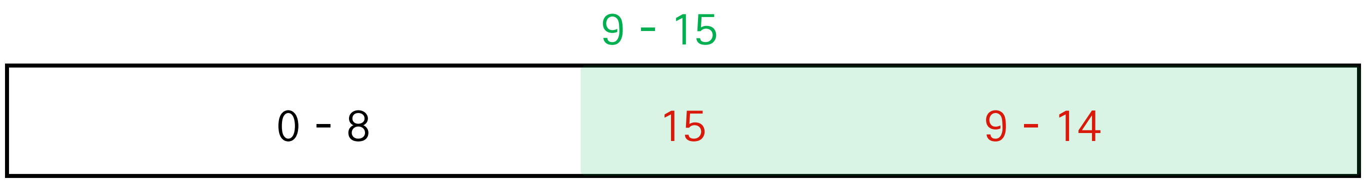
PLU 분해를 적용한 CHAM Key schedule

```
Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key K0~7, word size n, upper triangular matrix U1~2, lower triangular matrix L1~2
Output: Round key RK0~7

Transform K0~7 :
  // Apply_U
  for i = 0 to n - 2
    for j = 0 to n - i - 2
      if U1~2[(i * n) + 1 + i + j] == 1
        CNOT (K[i + j + 1], K[i])
  // Apply_L
  for i = 0 to n - 2
    for j = n - 1 to i + 1 step -1
      if L1~2[n * (n - 1 - i) + n - 1 - j] == 1
        CNOT (K[n - 1 - j], K[n - 1 - i])
  // Apply_P
  if RK0~7 == True
    SWAP (K[12], K[11]), SWAP (K[11], K[13])
    SWAP (K[11], K[10]), SWAP (K[10], K[14])
    SWAP (K[10], K[9]), SWAP (K[9], K[15])
  return RK0~7
  else
    SWAP (K[11], K[13]), SWAP (K[10], K[9])
    SWAP (K[7], K[8]), SWAP (K[5], K[6])
  return RK8~15
Reverse(transform K0~7) // Reverse operation
```

logical swap 이용한 방법 → 직관적

```python
def Apply_P_16_0to7(eng, key):
    out = []
    for i in range(9):
        out.append(key[i])
    out.append(key[15])
    for i in range(6):
        out.append(key[i + 9])
    return out
```



PLU 분해를 적용한 CHAM Key schedule

Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key $K_{0\sim7}$, word size n, upper triangular matrix $U_{1\sim2}$, lower triangular matrix $L_{1\sim2}$
Output: Round key $RK_{0\sim7}$

```
1: Transform K0~7 :
   // Apply_U
2:   for i = 0 to n - 2
3:     for j = 0 to n - i - 2
4:       if U1~2[(i * n) + 1 + i + j] == 1
5:         CNOT (K[i + j + 1], K[i])
   // Apply_L
6:   for i = 0 to n - 2
7:     for j = n - 1 to i + 1 step -1
8:       if L1~2[n * (n - 1 - i) + n - 1 - j] == 1
9:         CNOT (K[n - 1 - j], K[n - 1 - i])
   // Apply_P
10:  if RK0~7 == True
11:    SWAP (K[12], K[11]), SWAP (K[11], K[13])
12:    SWAP (K[11], K[10]), SWAP (K[10], K[14])
13:    SWAP (K[10], K[9]), SWAP (K[9], K[15])
14:  return RK0~7
15:  else
16:    SWAP (K[11], K[13]), SWAP (K[10], K[9])
17:    SWAP (K[7], K[8]), SWAP (K[5], K[6])
18:  return RK8~15
19: Reverse(transform K0~7) // Reverse operation
```

```
with Compute(eng):
    ## RK0
    Apply_U(eng, k0, n, U_Matrix1)
    Apply_L(eng, k0, n, L_Matrix1)
              ⋮
    ## RK7
    Apply_U(eng, k7, n, U_Matrix1)
    Apply_L(eng, k7, n, L_Matrix1)
```

U, L 과정

```
    k0 = Apply_P_16_0to7(eng, k0)
              ⋮
    k7 = Apply_P_16_0to7(eng, k7)
```

P 과정

Encrypting 과정

```
Uncompute(eng)
```

reverse 연산 수행 → K 재활용

PLU 분해를 적용한 CHAM Key schedule

Algorithm 1: Quantum circuit implementation of Apply_PLU
Input: key $K_{0\sim7}$, word size n, upper triangular matrix $U_{1\sim2}$, lower triangular matrix $L_{1\sim2}$
Output: Round key $RK_{0\sim7}$

```
Transform K0~7 :
  // Apply_U
  for i = 0 to n - 2
    for j = 0 to n - i - 2
      if U1~2[(i * n) + 1 + i + j] == 1
        CNOT (K[i + j + 1], K[i])
  // Apply_L
  for i = 0 to n - 2
    for j = n - 1 to i + 1 step -1
      if L1~2[n * (n - 1 - i) + n - 1 - j] == 1
        CNOT (K[n - 1 - j], K[n - 1 - i])
  // Apply_P
  if RK0~7 == True
    SWAP (K[12], K[11]), SWAP (K[11], K[13])
    SWAP (K[11], K[10]), SWAP (K[10], K[14])
    SWAP (K[10], K[9]), SWAP (K[9], K[15])
  return RK0~7
  else
    SWAP (K[11], K[13]), SWAP (K[10], K[9])
    SWAP (K[7], K[8]), SWAP (K[5], K[6])
  return RK8~15
Reverse(transform K0~7) // Reverse operation
```

```python
with Compute(eng):
    ## RK0
    Apply_U(eng, k0, n, U_Matrix1)
    Apply_L(eng, k0, n, L_Matrix1)
              ⋮
    ## RK7
    Apply_U(eng, k7, n, U_Matrix1)
    Apply_L(eng, k7, n, L_Matrix1)
```

U, L 과정

```python
    k0 = Apply_P_16_0to7(eng, k0)
              ⋮
    k7 = Apply_P_16_0to7(eng, k7)
```

P 과정

Encrypting 과정

```python
Uncompute(eng)
```

reverse 연산 수행 → K 재활용

```python
def rev_Apply_P_16_0to7(eng, key):
    out = []
    for i in range(9):
        out.append(key[i])
    for i in range(6):
        out.append(key[i + 10])
    out.append(key[9])
    return out
```

PLU 분해를 적용한 CHAM Key schedule 양자 회로 구현 비용

※다른 게이트 비용은 동일하기에 비교를 위해 값이 다른 항목들만 가져왔음.

<표 1> CHAM Quantum Resources Comparison by Methods Applied to Key-schedule

method	CHAM	qubits	CNOT	Depth
prev[4]	64/128	204	27,120	17,035
	128/128	292	58,040	37,766
	128/256	420	70,080	45,252
PLU	64/128	195	35,280	17,092
	128/128	259	81,280	37,878
	128/256	387	95,536	45,014

PLU는 in-place로 회로가 구성되어있음.

→ 보조 큐비트 사용하지 않아, **큐비트 수 절약**

감사합니다