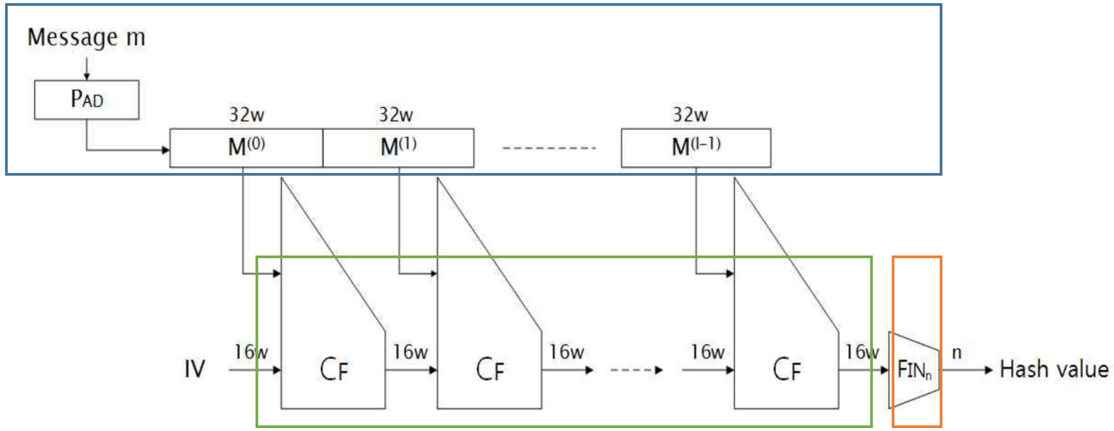
ARM64상에서의 LSH 최적 구현

https://youtu.be/OJjNJTLo1EI

한성대학교
HANSUNG UNIVERSITY

CryptoCraft LAB

LSH



- 2014년에 개발된 국산 해시 함수
- LSH-8w-n(w 비트 워드 단위로 동작, n 비트의 출력값)
  - w : 32, 64 / 8w : 256, 512
  - $1 \leq n \leq 8w$
    - 224, 256, 384, 512
- 3단계를 통해 해시값 출력
  - Initialization(초기화)
    - 입력메시지(m)를 메시지 블록 비트 길이의 배수(1024, 2048)가 되도록 패딩 후, 메시지 블록 단위로 분할
    - 연결 변수를 IV로 초기화
  - Compression(압축)
    - 32w 배열 메시지 블록을 입력으로 사용하여 얻은 출력값을 연결 변수로 사용하여 마지막 블록까지 반복하여 압축 함수 실행
  - Finalization(완료)
    - 연결 변수에 최종 저장된 값으로부터 n 비트 길이의 출력값 생성

ARM64상에서의 LSH 최적구현

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 13, 2019, accepted July 25, 2019, date of publication July 30, 2019, date of current version August 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932129

Fast Implementation of LSH With SIMD

DONGYEONG KIM[1], YOUNGHOON JUNG[2], YOUNGJIN JU[1], AND JUNGHWAN SONG[1]
[1]Department of Mathematics, Research Institute for Natural Sciences, Hanyang University, Seoul 04763, South Korea
[2]The Affiliated Institute of ETRI, Daejeon 341293, South Korea

Corresponding author: Junghwan Song (camp123@hanyang.ac.kr)

This work was supported by the Institute for Information and Communications Technology Planning and Evaluation (IITP) funded by the Korean Government (MSIT) under Grant 2017-0-00267.

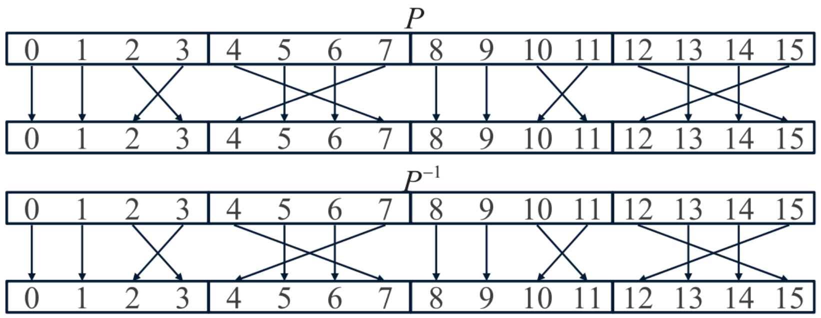
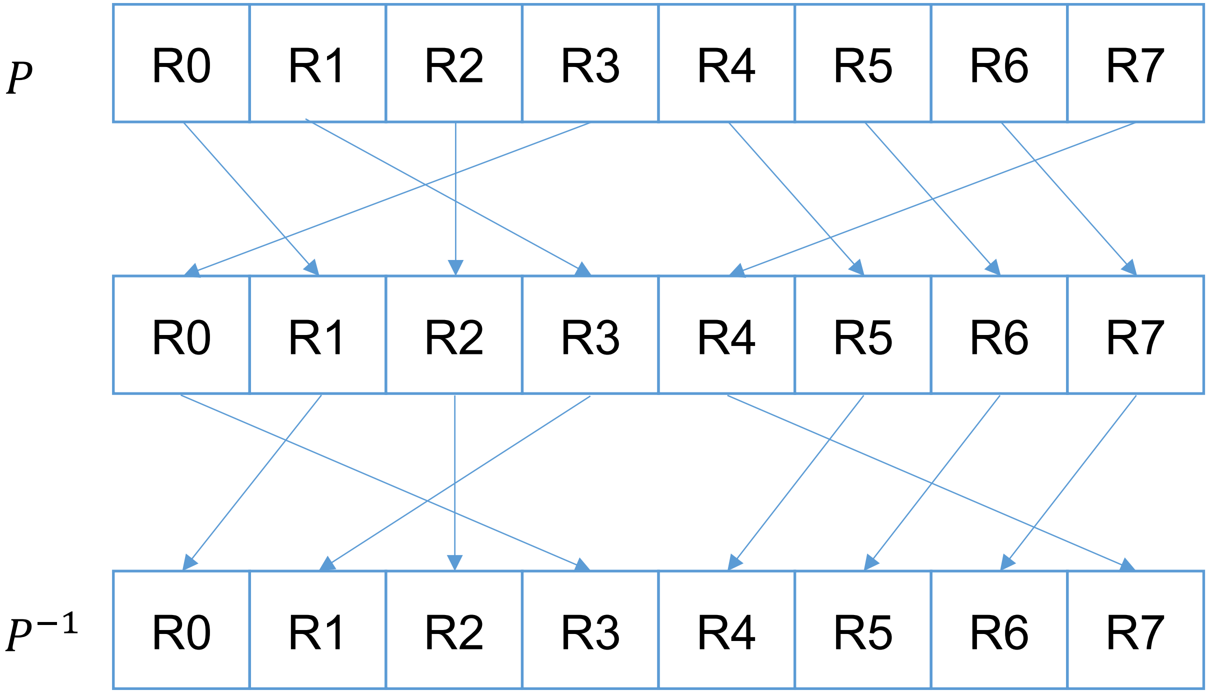


FIGURE 9. **The permutation P and P^{-1} used for LSH-256 with AVX2.**



ARM64상에서의 LSH 최적구현

```
v0 : 0x44444444, 0x11111111, 0x22222222, 0x33333333
v1 : 0x55555555, 0x66666666, 0x77777777, 0x88888888

zip1.4s : 44444444 55555555 11111111 66666666
zip2.4s : 22222222 77777777 33333333 88888888

zip1.2d : 44444444 11111111 55555555 66666666
zip2.2d : 22222222 33333333 77777777 88888888

uzp1.4s : 44444444 22222222 55555555 77777777
uzp2.4s : 11111111 33333333 66666666 88888888

uzp1.2d : 44444444 11111111 55555555 66666666
uzp2.2d : 22222222 33333333 77777777 88888888

trn1.2d : 44444444 11111111 55555555 66666666
trn2.2d : 22222222 33333333 77777777 88888888

trn1.4s : 44444444 55555555 22222222 77777777
trn2.4s : 11111111 66666666 33333333 88888888
```

```
0x44444444, 0x11111111  0x33333333, 0x22222222
0x55555555, 0x66666666 0x88888888, 0x77777777

trn1.2d     v8, v0, v1
trn2.2d     v9, v0, v1
rev64.4s    v9, v9
trn1.2d     v0, v8, v9
trn2.2d     v1, v8, v9
```

TRN1.2d
TRN2.2d

UZP1.2d
UZP2.2d

ZIP1.2d
ZIP2.2d

0	1		
2	3		

ZIP1.4s
ZIP2.4s

0		1	
2		3	

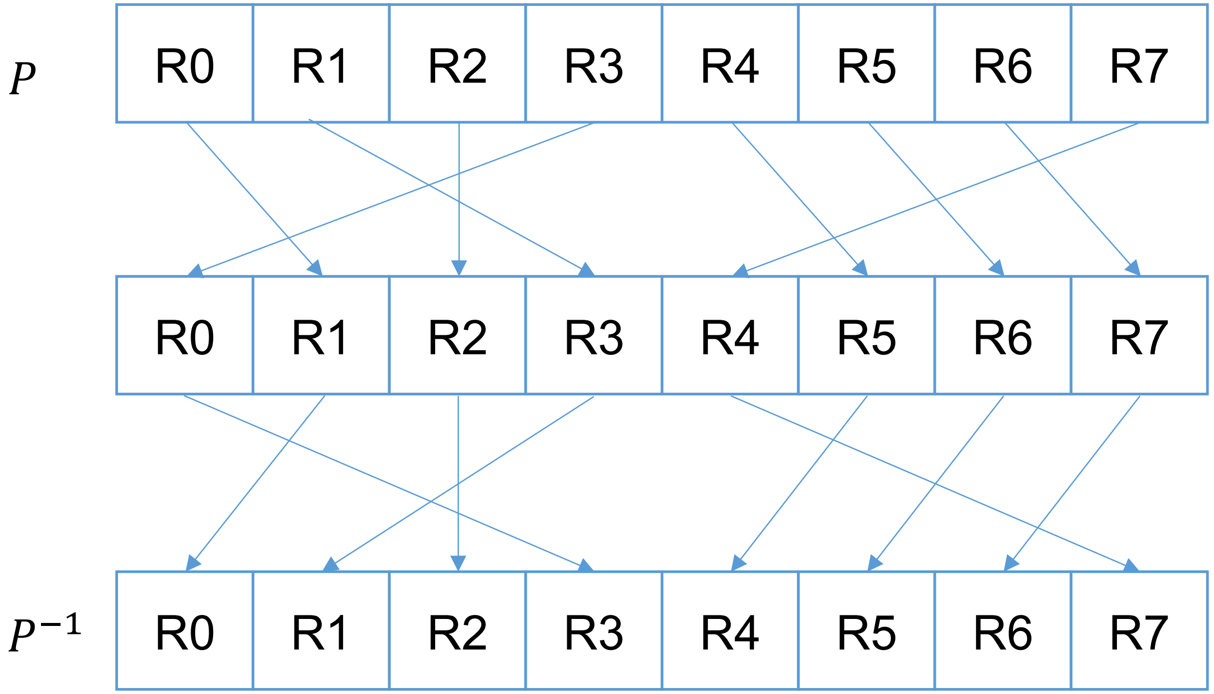
UZP1.4s
UZP2.4s

0	2		
1	3		

TRN1.4s
TRN2.4s

0		2	
1		3	

ARM64상에서의 LSH 최적구현




```
.macro func_P s0, s1, s2, s3
//first ->v5, v6, v7, v8
    uzp1.2d     v20, \s0, \s2
    uzp2.2d     v21, \s0, \s2
    rev64.4s    v21, v21
    zip1.4s     \s0, v21, v20
    zip2.4s     \s2, v21, v20

    uzp1.2d     v20, \s1, \s3
    uzp2.2d     v21, \s1, \s3
    rev64.4s    v20, v20
    trn1.4s     v9, v20, v21
    trn2.4s     v10, v20, v21
    rev64.4s    v10, v10
    zip1.2d     \s1, v10, v9
    zip2.2d     \s3, v10, v9
.endm
```

```
.macro inv_func_P s0, s1, s2, s3
//first ->v5, v6, v7, v8

    uzp1.4s     v20, \s0, \s2
    uzp2.4s     v21, \s0, \s2
    rev64.4s    v20, v20
    uzp1.2d     \s0, v21, v20
    zip2.2d     \s2, v21, v20

    uzp1.2d     v20, \s1, \s3
    uzp2.2d     v21, \s1, \s3
    rev64.4s    v20, v20
    trn1.4s     v9, v20, v21
    trn2.4s     v10, v20, v21
    rev64.4s    v10, v10
    zip1.2d     \s1, v9, v10
    zip2.2d     \s3, v9, v10
.endm
```

```
0x33333333 0x44444444 0x22222222 0x11111111
0x88888888 0x55555555 0x77777777 0x66666666
```

```
v0 : 0x44444444, 0x11111111, 0x22222222, 0x33333333
v1 : 0x55555555, 0x66666666, 0x77777777, 0x88888888
```

```
    mov.8h      v5, v14                 //    memset(ctx->cv_l, 0, 8 * sizeof(uint32_t));
    mov.8h      v6, v14
    mov.8h      v7, v14                 //     memset(ctx->cv_r, 0, 8 * sizeof(uint32_t));
    mov.8h      v8, v14

    mov         v5.s[0], v3.s[0]        //cv_l[0] = 32
    mov         v5.s[1], v9.s[0]        //cv_l[1] = 256

/////////////////////////////////////////////////////////////////////
    func_P v5, v6, v7, v8
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
    init_for // for문 1회
    init_for // for문 2회
    init_for // for문 3회
```

```
//update
    sri.4s      v11, v1, #3                //databytelen = databitlen >> 3; 오른쪽으로 3비트 v11 = 128(0x80)
    and         v2.16b, v1.16b, v10.16b    //pos2 = databitlen & 0x7; 0x0 = 0x400 and 0x7

    sri.4s      v12, v4, #3                //remain_msg_byte = ctx->remain_databitlen >> 3; //remain_msg_byte = 0
    and         v13.16b, v4.16b, v23.16b   //remain_msg_bit = ctx->remain_databitlen & 7;//remain_msg_bit = 0

    memcpy
//////////////////////////////////////////////////////////////////////////////
    func_P v24, v25, v26, v27 //update (last_block)
    func_P v28, v29, v30, v31 //update (last block)
```

```
//fin(ctx)
    eor.16b     v5, v5, v7
    eor.16b     v6, v6, v8
/////////////////////////////////////////////////

    inv_func_P v5, v6, v7, v8
```

ARM64상에서의 LSH 최적구현

```
.macro lsh256_init  //init_for
    ld1.4s      {v18,v19},[x3], #32
    mix_even
    word_perm
    ld1.4s      {v18,v19},[x3], #32
    mix_odd
    word_perm
.endm
```

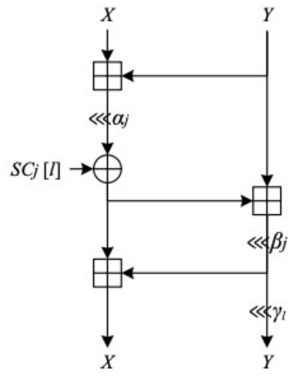
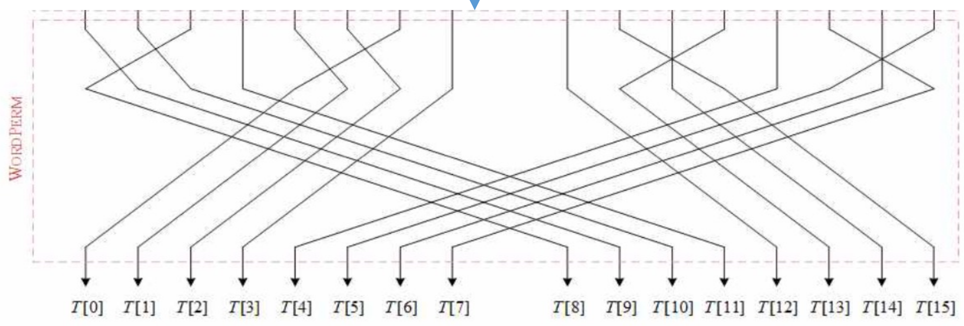


Fig. 1: Two-word mix function $\mathrm{Mix}_{j,l}(X,Y)$

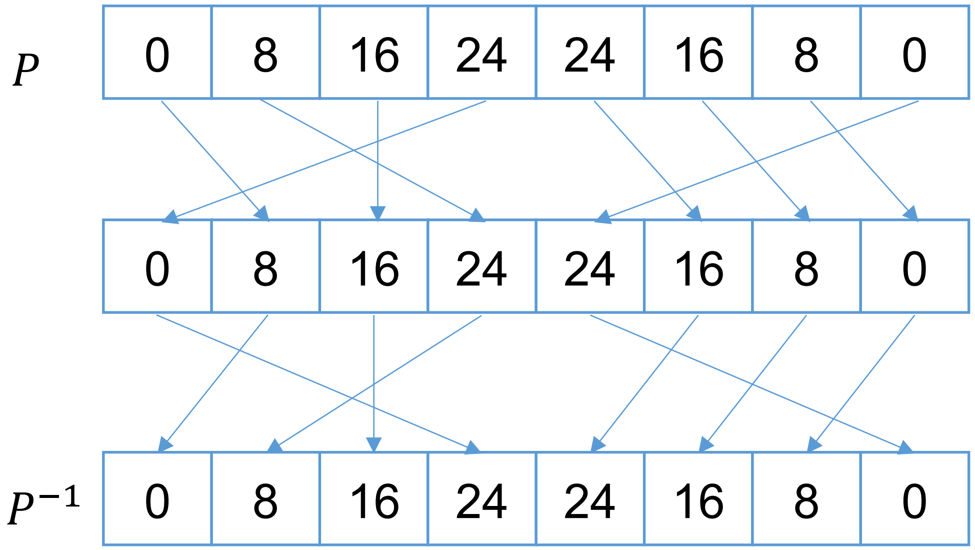
$$X \leftarrow X \boxplus Y,$$
$$X \leftarrow X^{\lll \alpha_j},$$
$$X \leftarrow X \oplus SC_j[l],$$
$$Y \leftarrow X \boxplus Y,$$
$$Y \leftarrow Y^{\lll \beta_j},$$
$$X \leftarrow X \boxplus Y,$$
$$Y \leftarrow Y^{\lll \gamma_l}.$$



w	j	α_j	β_j	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7
32	짝수	29	1	0	8	16	24	24	16	8	0
	홀수	5	17								

ARM64상에서의 LSH 최적구현

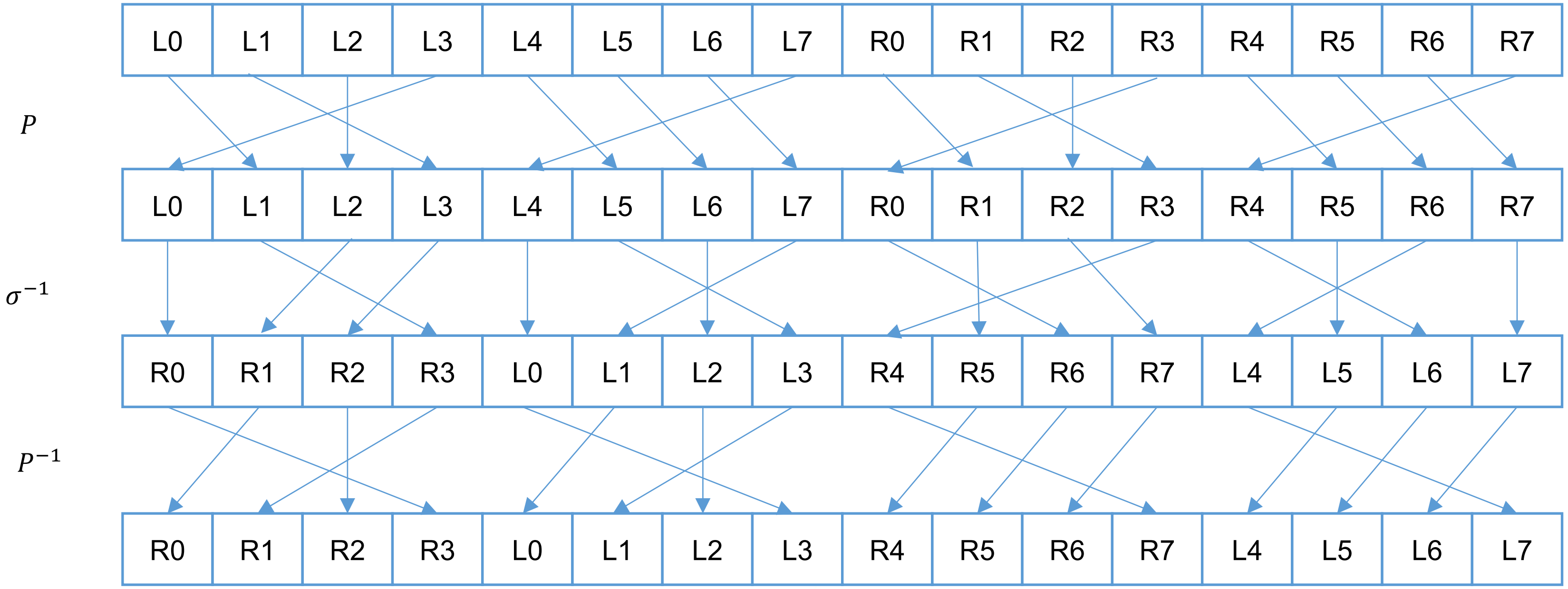
```
.macro mix_even
    add.4s      v5, v5, v7                   //v5 : cvl0~3 / v6: cvl4~7
    add.4s      v6, v6, v8                   //add_blk(cv_l, cv_r);

    shl.4s      v20, v5, #29                 //even_rot_alpha(29, 3)
    sri.4s      v20, v5, #3
    mov.4s      v5, v20

    shl.4s      v21, v6, #29
    sri.4s      v21, v6, #3                  //    rotate_blk(cv_l, rot_alpha);
    mov.4s      v6, v21

    eor         v5.16b, v5.16b, v18.16b
    eor         v6.16b, v6.16b, v19.16b      //    xor_with_const(cv_l, const_v);

    add.4s      v7, v7, v5                   //v7 : cvr0~3 / v8: cvr4~7
    add.4s      v8, v8, v6                   //    add_blk(cv_r, cv_l);

    shl.4s      v20, v7, #1                  //even_rot_beta (1, 31)
    sri.4s      v20, v7, #31
    mov.4s      v7, v20

    shl.4s      v21, v8, #1
    sri.4s      v21, v8, #31                 //    rotate_blk(cv_r, rot_beta);
    mov.4s      v8, v21

    add.4s      v5, v5, v7                   //v5 : cvl0~3 / v6: cvl4~7
    add.4s      v6, v6, v8                   //add_blk(cv_l, cv_r);

    new_rotate_msg_gamma        //    new_rotate_msg_gamma(cv_r);

    movi.8h     v14, 0x00
    mov.8h      v26, v14
    mov.8h      v27, v14

.endm
```




```
.macro new_rotate_msg_gamma
    mov         v26.s[0], v7.s[3]
    mov         v26.s[1], v8.s[3]

    mov         v27.s[0], v7.s[0]
    mov         v27.s[1], v8.s[1]

    shl.4s      v20, v26, #8
    sri.4s      v20, v26, #24
    mov.4s      v26, v20

    shl.4s      v21, v27, #24
    sri.4s      v21, v27, #8
    mov.4s      v27, v21

    mov         v7.s[3], v26.s[0]
    mov         v8.s[3], v26.s[1]

    mov         v7.s[0], v27.s[0]
    mov         v8.s[1], v27.s[1]

    mov         v26.s[0], v7.s[2]
    mov         v26.s[1], v8.s[2]
    rev32       v26.8h, v26.8h

    mov         v7.s[2],v26.s[0]
    mov         v8.s[2],v26.s[1]
.endm
```

ARM64상에서의 LSH 최적구현

wordperm




```
mov         v20.s[0], v8.s[1]          //v20 : tmp
mov         v8.s[1], v7.s[1]
mov         v7.s[1], v5.s[2]
mov         v5.s[2], v6.s[2]
mov         v6.s[2], v8.s[0]
mov         v8.s[0], v7.s[3]

mov         v7.s[3], v5.s[1]
mov         v5.s[1], v6.s[3]
mov         v6.s[3], v8.s[3]
mov         v8.s[3], v7.s[2]

mov         v7.s[2], v5.s[3]
mov         v5.s[3], v6.s[1]
mov         v6.s[1], v20.s[0]

mov         v20.s[0], v7.s[0]

mov         v7.s[0], v5.s[0]
mov         v5.s[0], v6.s[0]
mov         v6.s[0], v8.s[2]
mov         v8.s[2], v20.s[0]
```

동일하게 18개의 명령어 사용
But, 오른쪽 구현이 더 적은 cpb

```
uzp1.4s     v20, v5, v5
uzp2.4s     v21, v5, v5
rev64.4s    v21, v21
zip1.2d     v16, v20, v21
mov.4s      v23, v16 //r0에 저장할 L0

mov         v20.s[0], v6.s[1]
mov         v6.s[1], v6.s[3]
mov         v6.s[3], v20.s[0] //L4
mov.4s      v5, v6 //L0

mov         v20.s[0], v8.s[0]
mov         v8.s[0], v8.s[2]
mov         v8.s[2], v20.s[0]//L4에서 저장되는 R4
mov.4s      v6, v8

uzp1.4s     v20, v7, v7
uzp2.4s     v21, v7, v7
rev64.4s    v21, v21
zip1.2d     v8, v21, v20

mov.4s      v7, v23
```

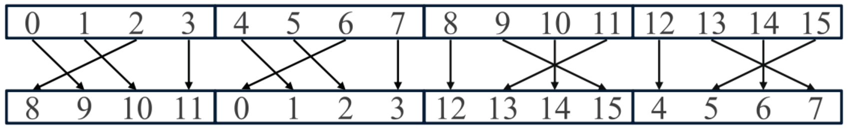
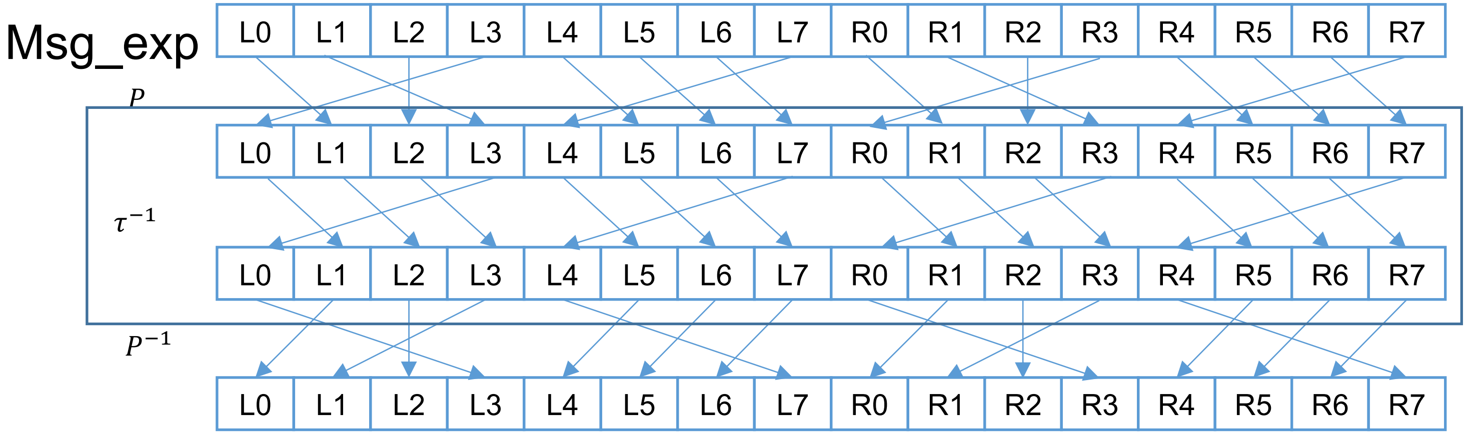


FIGURE 4. **Permutation σ of $Step_j$.**

ARM64상에서의 LSH 최적구현



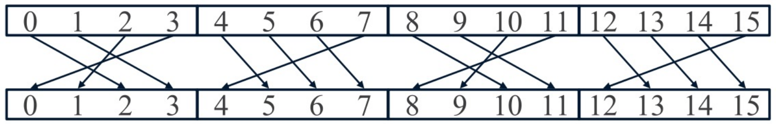
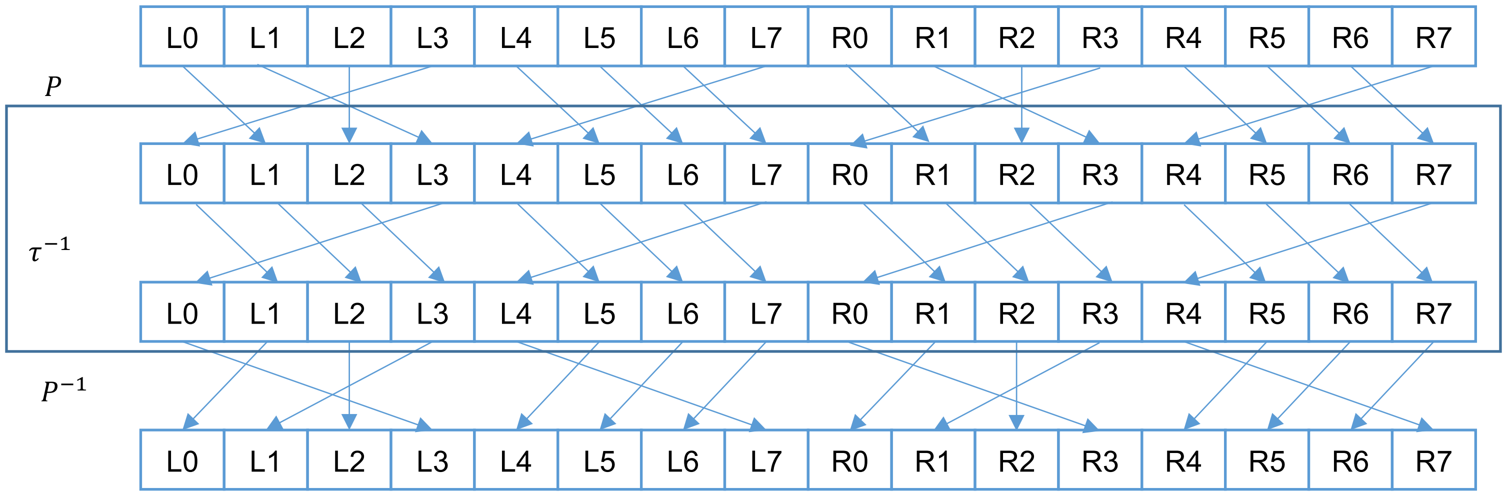


FIGURE 3. Permutation τ of *MsgExp*.

```
 .macro zip_msg_exp1 s0, s1, s2, s3
 //s24, 26, 28, 30
     trn1.4s     v3, \s0, \s1
     trn2.4s     v9, \s0, \s1
     trn1.4s     v10, \s2, \s3
     trn2.4s     v14, \s2, \s3

     uzp1.2d     \s0, v3, v10
     uzp2.2d     \s2, v3, v10
     uzp1.2d     \s1, v9, v14
     uzp2.2d     \s3, v9, v14
 .endm
```


```
.macro uzp_msg_exp1 s0, s1, s2, s3
    zip1.4s     v3, \s0, \s1
    zip2.4s     v9, \s0, \s1
    zip1.4s     v10, \s2, \s3
    zip2.4s     v14, \s2, \s3

    zip1.2d     \s0, v3, v10
    zip2.2d     \s1, v3, v10
    zip1.2d     \s2, v9, v14
    zip2.2d     \s3, v9, v14
.endm
```

ARM64상에서의 LSH 최적구현




```
.macro update_for
    new_msg_exp_even
    msg_add_even
    ld1.4s      {v18,v19},[x3], #32             //load_sc(&const_v, 8);
    update_mix_even
    new_wordperm

    new_msg_exp_odd
    msg_add_odd
    ld1.4s      {v18,v19},[x3], #32             //load_sc(&const_v, 8);
    update_mix_odd
    new_wordperm
.endm
```

$$M_0^{(i)} \leftarrow (M^{(i)}[0], M^{(i)}[1], \cdots, M^{(i)}[15]),$$

$$M_1^{(i)} \leftarrow (M^{(i)}[16], M^{(i)}[17], \cdots, M^{(i)}[31]), \qquad (4.6)$$

$$M_j^{(i)}[l] \leftarrow M_{j-1}^{(i)}[l] \boxplus M_{j-2}^{(i)}[\tau(l)] \qquad (0 \le l \le 15,\ 2 \le j \le N_s).$$

```
.macro new_msg_exp_even
    zip_msg_exp1 v24, v25, v26, v27
    zip_msg_exp2 v28, v29, v30, v31

    mov.4s      v20, v24
    add.4s      v24, v28, v27
    add.4s      v27, v31, v26
    add.4s      v26, v30, v25
    add.4s      v25, v29, v20

    uzp_msg_exp1 v24, v25, v26, v27
    uzp_msg_exp2 v28, v29, v30, v31
.endm
```

성능평가

입력 길이 : 1024-bit

```
gettimeofday(&start, NULL);
    for (int i = 0; i < 10000000; i++) {
        lsh_opt2(data, IV, p_databitlen, g_StepConstants2, hash);

}
    gettimeofday(&end, NULL);
    seconds  = end.tv_sec  - start.tv_sec;
    useconds = end.tv_usec - start.tv_usec;
    mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
    printf("time %ld\n",mtime);
 //millsec
```

Reference C	ARM 상에서의 LSH (3월 세미나 구현물)	현재 구현물
37.460	11.225	9.731

현재 구현물의 경우, Wordperm 부분을 mov 명령어를 사용하여 구현할 경우, **11.279** 로 기존 구현물보다도 좋지 않게 나옴

Q & A