부채널 분석 대응을 위한 Masking AES 최적화 구현

IT 융합공학과
김경호
https://youtu.be/VSP2EMvWLJU

HSU 한성대학교 HANSUNG UNIVERSITY
CryptoCraft LAB

Contents

1 Side Channel Attack

2 1st-Order Masked AES

3 Implement Optimization

4 Results

CryptoCraft LAB

1 Side Channel Attack

- **부채널 공격**

  암호화 알고리즘의 취약점을 찾거나 Brute Force Attack 을 이용하여 암호화 알고리즘을 공격하는게 아니라 암호화 과정에서 발생하는 부가적인 정보를 이용하여 공격하는 방법

- **부채널 공격 종류**
  - 전력 소모량 분석
  - 오류 주입
  - 시차 분석
  - 전자기파 분석

CryptoCraft LAB

1 Side Channel Attack

- **전력 소모량 분석**

  암호화 알고리즘을 사용하는 하드웨어 모듈에서 암호화 과정을 진행할 때 전력량을 측정하여 키 값을 분석하는 공격 방법

- **전력 소모량 분석 종류**
  - DPA (Differential Power Analysis)
  - CPA (Correlation Power Analysis)

Results Table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	00 0.6502	01 0.6685	02 0.6533	03 0.6974	04 0.5344	05 0.7788	06 0.7953	07 0.6533	08 0.6065	09 0.6710	0A 0.6844	0B 0.7352	0C 0.5600	0D 0.6268	0E 0.7319	0F 0.7598
1	01 0.5566	00 0.4924	1D 0.4636	02 0.4917	CF 0.4882	E2 0.4482	AA 0.4700	06 0.5545	D9 0.4474	85 0.4628	6C 0.4511	29 0.4689	2F 0.4376	82 0.4685	54 0.4876	0C 0.4619
2	C9 0.4667	7E 0.4467	E3 0.4540	AC 0.4514	2B 0.4750	01 0.4457	08 0.4627	84 0.4681	E7 0.4466	2A 0.4497	65 0.4455	EC 0.4465	8D 0.4361	18 0.4580	0F 0.4840	BC 0.4614
3	A3 0.4411	E1 0.4409	E7 0.4429	58 0.4461	26 0.4667	C8 0.4435	71 0.4573	E9 0.4529	23 0.4397	74 0.4437	DC 0.4361	10 0.4448	37 0.4346	B9 0.4538	8A 0.4575	B8 0.4588
4	0B 0.4411	44 0.4390	A8 0.4314	43 0.4434	A2 0.4634	FE 0.4427	A9 0.4449	6F 0.4484	8A 0.4388	F9 0.4395	AB 0.4302	0A 0.4402	87 0.4328	09 0.4520	A6 0.4491	7C 0.4458
5	36 0.4377	48 0.4237	32 0.4289	F7 0.4385	BD 0.4625	99 0.4399	A8 0.4370	87 0.4461	2D 0.4319	DA 0.4386	17 0.4257	78 0.4283	23 0.4288	A5 0.4502	2B 0.4490	D1 0.4444
6	E0 0.4371	2F 0.4236	F7 0.4287	4E 0.4378	D2 0.4546	31 0.4299	94 0.4339	97 0.4448	F0 0.4278	64 0.4350	41 0.4255	00 0.4277	42 0.4283	0C 0.4489	0C 0.4457	CF 0.4419
7	F6 0.4226	2B 0.4205	0E 0.4278	7C 0.4372	EE 0.4468	52 0.4268	D7 0.4320	85 0.4380	1C 0.4223	4E 0.4332	5F 0.4236	4B 0.4254	92 0.4277	96 0.4486	A3 0.4337	5A 0.4341
8	B1 0.4210	84 0.4166	40 0.4265	B4 0.4274	37 0.4444	0B 0.4216	D4 0.4315	05 0.4214	FA 0.4204	A1 0.4318	3E 0.4202	8B 0.4209	CC 0.4235	BB 0.4457	3F 0.4304	55 0.4334
9	10 0.4208	F8 0.4159	F3 0.4244	DC 0.4256	AF 0.4370	F0 0.4212	52 0.4260	C7 0.4194	B4 0.4163	E6 0.4290	8B 0.4182	3F 0.4167	EA 0.4231	7C 0.4377	85 0.4271	DC 0.4331
10	63 0.4160	5F 0.4138	70 0.4241	3C 0.4236	95 0.4364	DF 0.4183	B5 0.4251	9F 0.4181	DE 0.4136	76 0.4277	A9 0.4180	D0 0.4143	D9 0.4219	CF 0.4315	FA 0.4227	1D 0.4298
11	BA 0.4137	5D 0.4125	CA 0.4154	9A 0.4223	77 0.4319	03 0.4179	C0 0.4207	8E 0.4166	D4 0.4089	1E 0.4270	05 0.4136	6A 0.4143	0D 0.4192	9B 0.4301	6A 0.4218	56 0.4273
12	B3 0.4134	2C 0.4103	0A 0.4123	1B 0.4181	9E 0.4266	7D 0.4167	69 0.4202	E5 0.4148	F9 0.4080	B5 0.4264	C6 0.4077	E5 0.4137	FD 0.4182	32 0.4260	BB 0.4211	AA 0.4260

CryptoCraft LAB

2 1^{st}-Order Masked AES

- **Masking 이란?**

  공격자의 전력 분석 공격을 막기 위해 암호화 과정에서 필요 없는 연산과정을 추가하여 제대로 된 전력 소모량 모델을 찾지 못하게 하는 대응 방법

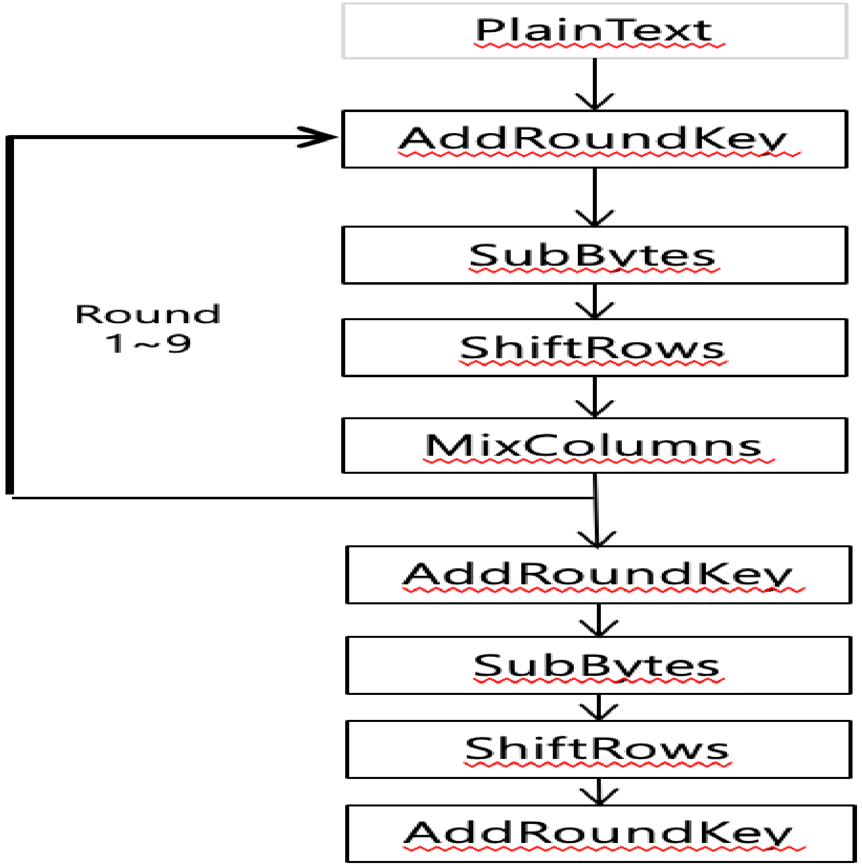
- **Herbst, C., Oswald, E., & Mangard, S. (2006)**

  An AES Smart Card Implementation Resistant to Power Analysis Attacks.

CryptoCraft LAB

2 1st-Order Masked AES

- **일반적인 AES**



CryptoCraft LAB

2 1^{st}-Order Masked AES

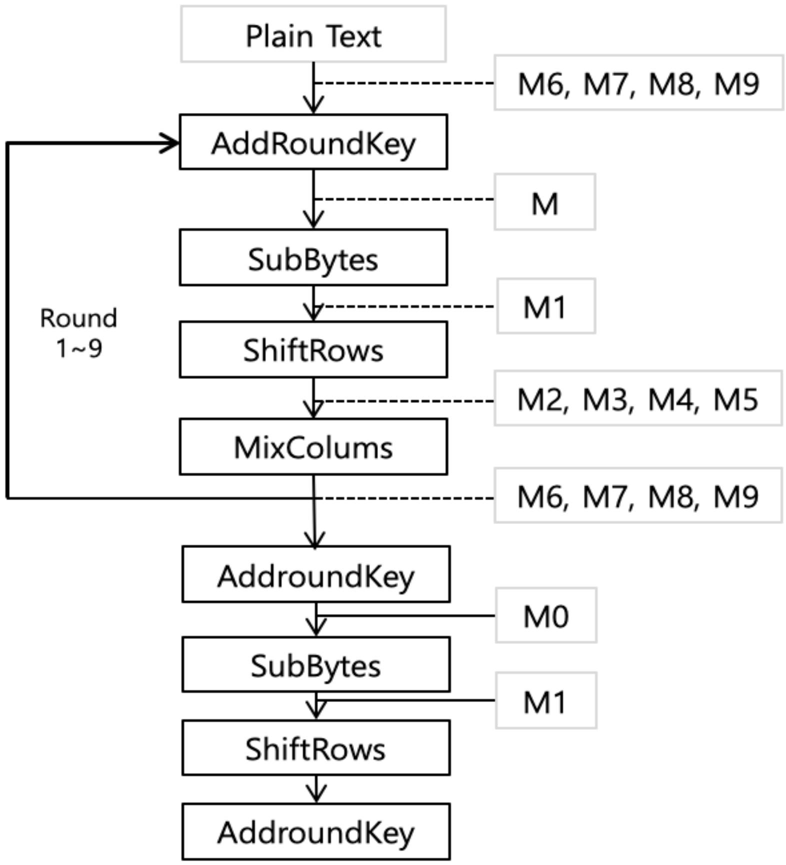
- **MakingMask**

```
MakingMask      mask[0] = rand(); \
                mask[1] = rand(); \
                mask[2] = rand(); \
                mask[3] = rand(); \
                mask[4] = rand(); \
                mask[5] = rand(); \
            \
                mask[6] = (mask[2] << 1) ^ ((mask[2] >> 7) * 0x1b) ^ (mask[3] << 1) ^ ((mask[3] >> 7) * 0x1b) ^ mask[3] ^ mask[4] ^ mask[5]; \
                mask[7] = (mask[3] << 1) ^ ((mask[3] >> 7) * 0x1b) ^ (mask[4] << 1) ^ ((mask[4] >> 7) * 0x1b) ^ mask[2] ^ mask[4] ^ mask[5];  \
                mask[8] = (mask[4] << 1) ^ ((mask[4] >> 7) * 0x1b) ^ (mask[5] << 1) ^ ((mask[5] >> 7) * 0x1b) ^ mask[2] ^ mask[3] ^ mask[5];  \
                mask[9] = (mask[2] << 1) ^ ((mask[2] >> 7) * 0x1b) ^ (mask[5] << 1) ^ ((mask[5] >> 7) * 0x1b) ^ mask[2] ^ mask[3] ^ mask[4]; \
            \
                for(int i = 0; i < 256; i++) \
                    MaskingSBOX[i ^ mask[0]] = SBOX[i] ^ mask[1]; \
            \
                mask[2] ^= mask[1]; \
                mask[3] ^= mask[1]; \
                mask[4] ^= mask[1]; \
                mask[5] ^= mask[1];
```

CryptoCraft LAB

2 1^{st}-Order Masked AES

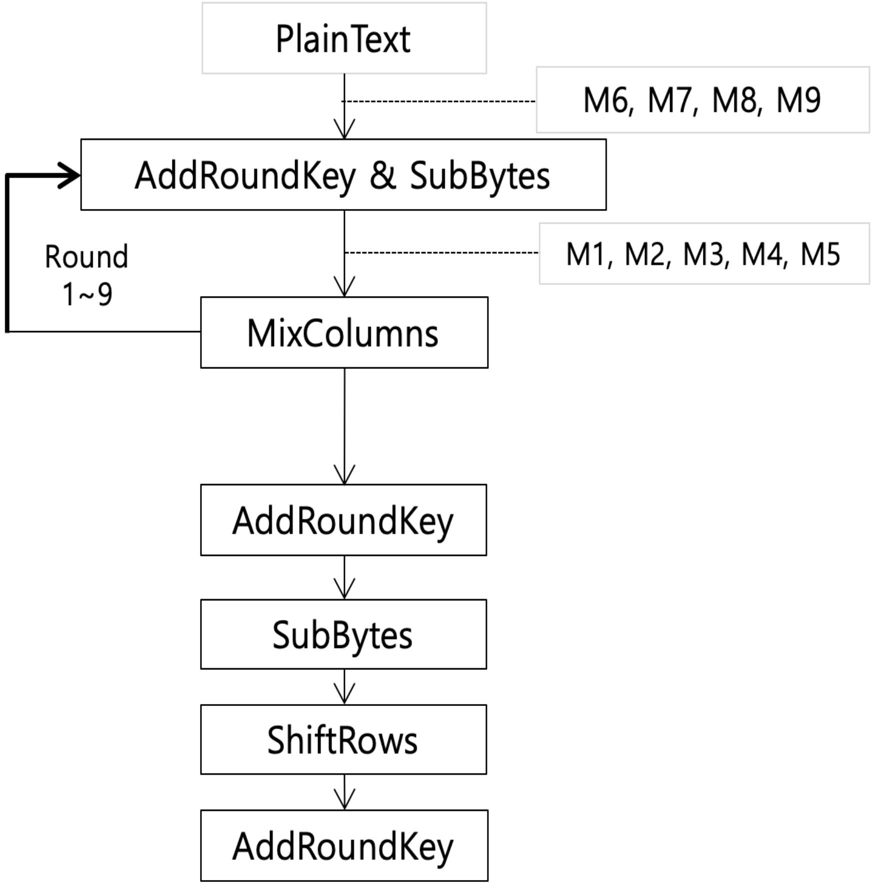
- **Masked AES**




```
int main(int argc, char* argv[])
{
    uint8_t roundKey[4][44] = {0,}; // Round Key
    uint8_t mask[10] = {0,};
    int dx = 0;

    MakingMask;
    KeySchedule;
    MaskingText;

    for(dx = 0; dx < 9; dx++)
    {
        AddRoundKey;
        SubBytes;
        MaskingCT;
        MixColumns;
    }

    AddRoundKey;
    SubBytes;

    dx++;

    ShiftRows;
    AddRoundKey;

    return 0;
}
```

CryptoCraft LAB

3 Implement Optimization



1. AddRoundKey & SubBytes 통합
2. Masking 값을 각 함수에 포함시켜서 연산 시간 단축
3. ShiftRows 연산 생략 (MixColumns)
4. Round 함수 전체를 Inline Assembly로 구현

CryptoCraft LAB

3 Implement Optimization

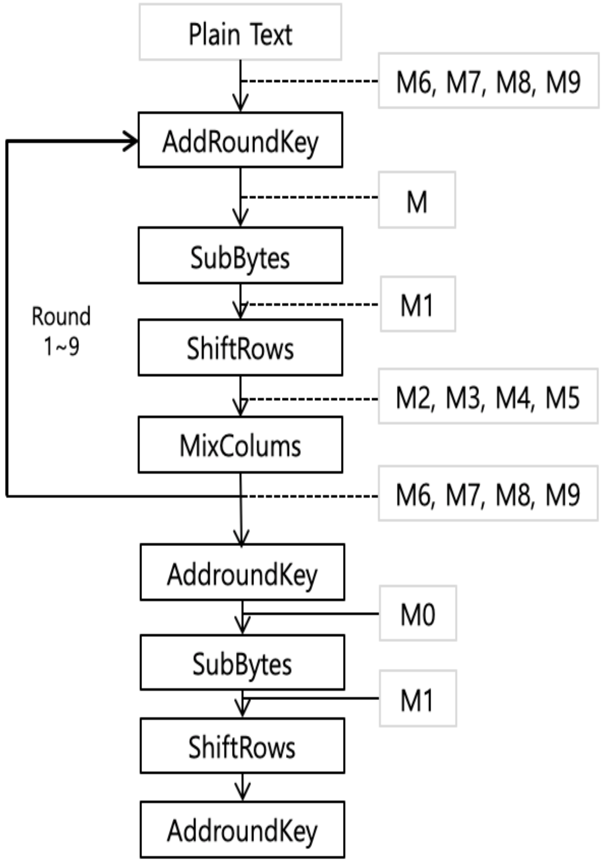
1. AddRoundKey & SubBytes 통합

```
  for(int dx = 0; dx < 9; dx ++)
{
    for(int i = 0; i < 4; i ++){
        for(int j = 0; j < 4; j++){
            plainText[i][j] ^= roundKey[i][j+(dx*4)];
            plainText[i][j] = MaskingSBOX[plainText[i][j]];
        }
    }
    MaskingCT;
    MixColumns;
}
```

CryptoCraft LAB

3 Implement Optimization

2. Masking 값을 각 함수에 포함시켜서 연산 시간 단축



<KeySchedule>

```
for(int i = 0; i < 40; i++)\
{\
      roundKey[0][i] ^= (mask[6] ^ mask[0]);\
      roundKey[1][i] ^= (mask[7] ^ mask[0]);\
      roundKey[2][i] ^= (mask[8] ^ mask[0]);\
      roundKey[3][i] ^= (mask[9] ^ mask[0]);\
}\
for(int i = 40; i < 44; i++)\
{\
      roundKey[0][i] ^= mask[1];\
      roundKey[1][i] ^= mask[1];\
      roundKey[2][i] ^= mask[1];\
      roundKey[3][i] ^= mask[1];\
}
```

```
MakingMask      mask[0] = rand(); \
                mask[1] = rand(); \
                mask[2] = rand(); \
                mask[3] = rand(); \
                mask[4] = rand(); \
                mask[5] = rand(); \
            \
                mask[6] = (mask[2] << 1) ^ ((mask[2] >> 7) * 0x1b) ^ (mask[3] << 1) ^ ((mask[3] >> 7) * 0x1b) ^ mask[3] ^ mask[4] ^ mask[5]; \
                mask[7] = (mask[3] << 1) ^ ((mask[3] >> 7) * 0x1b) ^ (mask[4] << 1) ^ ((mask[4] >> 7) * 0x1b) ^ mask[2] ^ mask[4] ^ mask[5];  \
                mask[8] = (mask[4] << 1) ^ ((mask[4] >> 7) * 0x1b) ^ (mask[5] << 1) ^ ((mask[5] >> 7) * 0x1b) ^ mask[2] ^ mask[3] ^ mask[5];  \
                mask[9] = (mask[2] << 1) ^ ((mask[2] >> 7) * 0x1b) ^ (mask[5] << 1) ^ ((mask[5] >> 7) * 0x1b) ^ mask[2] ^ mask[3] ^ mask[4]; \
            \
                for(int i = 0; i < 256; i++) \
                    MaskingSBOX[i ^ mask[0]] = SBOX[i] ^ mask[1]; \
            \
                mask[2] ^= mask[1]; \
                mask[3] ^= mask[1]; \
                mask[4] ^= mask[1]; \
                mask[5] ^= mask[1];
```

ryptoCraft LAB

3 Implement Optimization

3. ShiftRows 연산 생략 (MixColumns)

```
MixColumns    uint8_t temp[4][4] = {0,};\

        for(int i = 0; i < 4; i ++)\
        {\
        temp[0][i] = (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ (plainText[1][(i+1)%4] << 1) ^ ((plainText[1][(i+1)%4] >> 7) * 0x1b) ^ plainText[1][(i+1)%4] ^ plainText[2][(i+2)%4] ^ plainText[3][(i+3)%4];\
        temp[1][i] = (plainText[0][i]) ^ (plainText[1][(i+1)%4] << 1) ^ ((plainText[1][(i+1)%4] >> 7) * 0x1b) ^ (plainText[2][(i+2)%4] << 1) ^ ((plainText[2][(i+2)%4] >> 7) * 0x1b) ^ plainText[2][(i+2)%4] ^ plainText[3][(i+3)%4];\
        temp[2][i] = plainText[0][i] ^ plainText[1][(i+1)%4] ^ (plainText[2][(i+2)%4] << 1) ^ ((plainText[2][(i+2)%4] >> 7) * 0x1b) ^ (plainText[3][(i+3)%4] << 1) ^ ((plainText[3][(i+3)%4] >> 7) * 0x1b) ^ plainText[3][(i+3)%4];\
        temp[3][i] = (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ plainText[0][i] ^ plainText[1][(i+1)%4] ^ plainText[2][(i+2)%4] ^ (plainText[3][(i+3)%4] << 1) ^ ((plainText[3][(i+3)%4] >> 7) * 0x1b);\
        }\

        for(int i = 0; i < 4; i ++)\
            for(int j = 0; j < 4; j++)\
                plainText[i][j] = temp[i][j];
```

3 Implement Optimization

4. Round 함수 전체를 Inline Assembly로 구현

```
"movw   r30, r24\n\t" // Z = roundKey
"mov    r18, r17\n\t"
"lsr    r18\n\t"
"lsr    r18\n\t" // r18 = i >> 2
"ldi    r21, 0x2c\n\t"
"mul    r21, r18\n\t" // r0 = (i >> 2) * 44

"mov    r21, r16\n\t"
"add    r21, r21\n\t"
"add    r21, r21\n\t" // dx * 4
"mov    r19, r17\n\t"
"andi   r19, 0x03\n\t" // r19 = i & 3
"add    r21, r19\n\t" // (i&3) + (dx*4)
"ld     r9 , X\n\t" // plainText[i][j]
"add    r21, r0 \n\t"
"add    r30, r21\n\t" // Z = roundKey[(i >> 2)][(i&3)+(dx*4)]
"adc    r31, r1 \n\t"
"ld     r14, Z\n\t"
"eor    r9 , r14\n\t" // plainText[i][j] ^ roundKey[(i >> 2)][(i&3)+(dx*4)]
"mov    r30, r9\n\t"
"ldi    r31, 0x00\n\t"
"subi   r30, 0xdc\n\t"
"sbci   r31, 0xfd\n\t" // Z = MaskingSBOX[plainText[i][j]]
"ld     r9 , Z\n\t" // MaskingSBOX[plainText[i][j]]
"movw    r30, r22\n\t" // Z = mask
"add    r30, r18\n\t"
"adc    r31, r1 \n\t" // Z = mask[i>>2]
"ldd    r14, Z+2\n\t" // r14 = mask[2 + (i >>2)]
"eor    r14, r9 \n\t"
"st     X+, r14\n\t"
"inc    r17\n\t"
"cpi    r17, 0x10\n\t"
"brne   .-64\n\t"
```

```
"mov    r21, r17\n\t" // r21 = i
"add    r21, r18\n\t" // i + 1

"andi   r21, 0x03\n\t" // (i+1)&3
"add    r30, r21\n\t" // plainText[][(i+1)&3]
"adc    r31, r1 \n\t"

"ldd    r4 , Z+4\n\t" // plainText[1][(i+1)&3]
"eor    r9 , r4 \n\t" // r9 = (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ plainText[1][(i+1)&3]
"sub    r30, r21\n\t"

"mov    r10, r4 \n\t"
"add    r10, r10\n\t" // r10 = plainText[1][(i+1)&3] << 1
"mov    r12, r4 \n\t"
"add    r12, r12\n\t"
"eor    r12, r12\n\t"
"adc    r12, r12\n\t"
"mul    r12, r19\n\t"
"eor    r10, r0 \n\t" // r10 = (plainText[1][(i+1)&3] << 1) ^ ((plainText[1][(i+1)&3] >> 7) * 0x1b)

"mov    r21, r17\n\t"
"subi   r21, 0xfe\n\t" // i + 2
"andi   r21, 0x03\n\t" // (i+2)&3
"add    r30, r21 \n\t"
"adc    r31, r1  \n\t"

"ldd    r5, Z+8\n\t" // r5 = plainText[2][(i+2)&3]
"eor    r10, r5\n\t" // r10 = (plainText[1][(i+1)&3] << 1) ^ ((plainText[1][(i+1)&3] >> 7) * 0x1b) ^ plainText[2][(i+2)&3]
"sub    r30, r21\n\t"

"mov    r21, r17\n\t"
"subi   r21, 0xfd\n\t"
"andi   r21, 0x03\n\t"
"add    r30, r21\n\t"
"adc    r31, r1 \n\t"

"ldd    r6, Z+12\n\t" // r6 = plainText[3][(i+3)&3]
"eor    r10, r6\n\t" //  r10 = (plainText[1][(i+1)&3] << 1) ^ ((plainText[1][(i+1)&3] >> 7) * 0x1b) ^ plainText[2][(i+2)&3] ^ plainText[3][(i+3)&3]
"sub    r30, r21\n\t"

"mov    r12, r10\n\t"
"eor    r12, r9 \n\t" // temp[0][i]
"st     Y+ , r12\n\t"
////////////////////////// temp[0][i]
```

CryptoCraft LAB

4 Results

- **성능 평가**

1. 명령어 수

	AES	Masked AES	Assembly Masked AES
명령어 수	290	400	191

2. Clock Cycle

	AES	Masked AES	Assembly Masked AES
Clock Cycle	22706	29269	25970

3. CPA Attack

Results Table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	158	147	198	18	248	43	43	133	27	176	120	247	1	27	62	196
0	20 0.1653	6F 0.1533	05 0.1633	3D 0.1837	5C 0.1508	DC 0.1629	AA 0.1588	D5 0.1612	00 0.1547	03 0.1585	82 0.1533	8A 0.1561	0A 0.1557	AE 0.1663	FF 0.1593	47 0.1639
1	4E 0.1563	30 0.1503	90 0.1579	36 0.1800	AD 0.1467	0F 0.1608	4F 0.1550	31 0.1610	B6 0.1541	A3 0.1541	8C 0.1519	3C 0.1525	0C 0.1471	45 0.1492	A7 0.1586	E5 0.1636
2	46 0.1559	A8 0.1495	65 0.1466	2B 0.1699	0F 0.1467	22 0.1566	3C 0.1462	35 0.1557	9F 0.1528	8E 0.1527	A8 0.1508	E6 0.1519	9A 0.1469	29 0.1481	BD 0.1535	29 0.1626
3	E9 0.1498	B1 0.1458	B4 0.1441	6C 0.1690	CE 0.1465	D9 0.1538	C1 0.1452	CA 0.1477	6B 0.1472	AF 0.1501	AC 0.1476	A5 0.1507	C4 0.1469	62 0.1477	BE 0.1534	85 0.1608
4	16 0.1492	39 0.1455	62 0.1408	2F 0.1682	E6 0.1425	C6 0.1510	75 0.1444	39 0.1477	CB 0.1460	FD 0.1428	7E 0.1474	C4 0.1499	71 0.1466	87 0.1435	38 0.1519	DD 0.1579
5	BD 0.1416	7D 0.1445	3D 0.1407	FE 0.1671	D8 0.1412	8C 0.1435	28 0.1438	33 0.1464	8A 0.1443	EE 0.1423	93 0.1454	7A 0.1477	04 0.1422	56 0.1433	07 0.1457	C6 0.1539
6	B1 0.1377	ED 0.1433	A8 0.1404	DA 0.1637	F2 0.1399	42 0.1420	D7 0.1419	A7 0.1463	9E 0.1418	BC 0.1421	F3 0.1426	61 0.1474	3D 0.1416	C1 0.1419	0D 0.1454	68 0.1526
7	6A 0.1375	41 0.1420	48 0.1401	62 0.1633	80 0.1388	E3 0.1398	DF 0.1400	11 0.1454	F4 0.1407	95 0.1419	15 0.1410	50 0.1451	DA 0.1393	06 0.1418	D7 0.1453	2A 0.1503
8	A0 0.1370	E5 0.1420	61 0.1399	CB 0.1628	73 0.1385	E9 0.1397	A5 0.1384	2F 0.1423	E6 0.1399	36 0.1404	4F 0.1407	A0 0.1439	19 0.1389	07 0.1417	52 0.1442	E2 0.1484
9	09	AB	71	66	78	ED	8C	4E	C1	B8	EF	2E	53	BE	00	E1

CryptoCraft LAB

Thank You

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB