AES-GCM 최적화
&
부채널 공격 및 방어

https://www.youtube.com/watch?v=LEOcrLrK1PM&feature=youtu.be

김경호

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

AES–GCM Multiplication 구현

- Look Up Table(LUT)

- Block-Comb
  - Secure GCM Implementation on AVR
  - SCA-Resistant GCM Implementation on 8-bit AVR Microcontrollers

Secure GCM Implementation on AVR

Algorithm 5 Masked Block Comb on 32-bit

Require: 32-bit wise operands A and B
Ensure: Result $C \leftarrow A \cdot B$

```
for i from 7 by 1 to 0 do
    for j from 3 by 1 to 0 do
        BIT ← A[j]&(1 ≪ i)
        {MASK, T0} ← 0 − BIT
        for k from 3 by 1 to 0 do
            C[k + j] ← C[k + j] ⊕ (B[k]&MASK)
        end for
    end for
    C ← C ≪ 1
end for
return C
```

- Block-Comb를 이용한 곱셈 연산
- If 문에 의한 Timing Attack을 방어하기 위해 MASK 값을 이용하여 방어
- MASK를 사용할 때 Zero 값을 이용하기 때문에 CPA에 취약

SCA-Resistant GCM Implementation

1: $R_{25} \leftarrow 0x07$ // Set displacement value for dummy ADD instruction for ILA
2: **for** $l = 0$ to 15 **do**
3: $R_l \leftarrow 0$
4: **end for**
5: **for** $l = 0$ to 3 **do**
6: $R_{16+l} \leftarrow A[l]$
7: $R_{21+l} \leftarrow B[l]$
8: **end for**
9: $R_{20} \leftarrow 0$
10: // Processing from 0-th bit to 6-th bit
11: **for** $l = 0$ to 6 **do**
12: **for** $m = 0$ to 3 **do**
13: **if** the l-th bit of R_{21+m}==1 **then**
14: $R_0 \leftarrow R_0 + R_{25}$ // Dummy ADD instruction for ILA
15: **for** $k = 0$ to 4 **do**
16: $R_{8+m+k} \leftarrow R_{8+m+k} \oplus R_{16+k}$
17: **end for**
18: **else**
19: // Dummy XOR with the garbage registers
20: **for** $k = 0$ to 4 **do**
21: $R_{m+k} \leftarrow R_{m+k} \oplus R_{16+k}$
22: **end for**
23: **end if**
24: **end for**
25: $(R_{20}, \ldots, R_{16}) \leftarrow (R_{20}, \ldots, R_{16}) \ll 1$
26: **end for**
27: // Processing the final 7-th bit
28: **for** $m = 0$ to 3 **do**
29: **if** the 7-th bit of R_{21+m}==1 **then**
30: $R_0 \leftarrow R_0 + R_{25}$ // Dummy ADD instruction for ILA
31: **for** $k = 0$ to 4 **do**
32: $R_{8+m+k} \leftarrow R_{8+m+k} \oplus R_{16+k}$
33: **end for**
34: **else**
35: // Dummy XOR with the garbage registers
36: **for** $k = 0$ to 4 **do**
37: $R_{m+k} \leftarrow R_{m+k} \oplus R_{16+k}$
38: **end for**
39: **end if**
40: **end for**
41: (Return $C = (R_{15}, \ldots, R_8)$)

- Secure GCM Implementation on AVR에서 구현한 곱셈기의 CPA 취약점을 보완한 곱셈기 구현
- Garbage 레지스터를 이용한 CPA 공격 방어
- If else에 따라 Real Register와 Garbage Register에 동일 연산

SCA-Resistant GCM Implementation

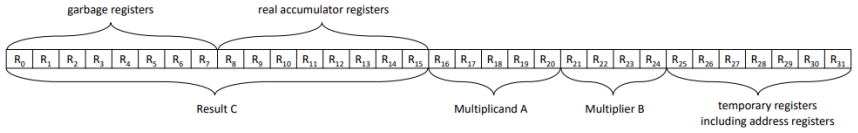


Figure 3: Register assignment in the proposed Block-Comb (BC) method.

```
; l = 1, m = 0
l_1_m_0:
   SBRS    R21, 1
   RJMP    l_1_m0_Dummy

   EOR     R8,  R16
   EOR     R9,  R17
   EOR     R10, R18
   EOR     R11, R19
   EOR     R12, R20
   RJMP    l_1_m_1

l_1_m0_Dummy:
   EOR     R0,  R16
   EOR     R1,  R17
   EOR     R2,  R18
   EOR     R3,  R19
   EOR     R4,  R20
   RJMP    l_1_m_1

; l = 1, m = 1
l_1_m_1:
   ...
   ...
```

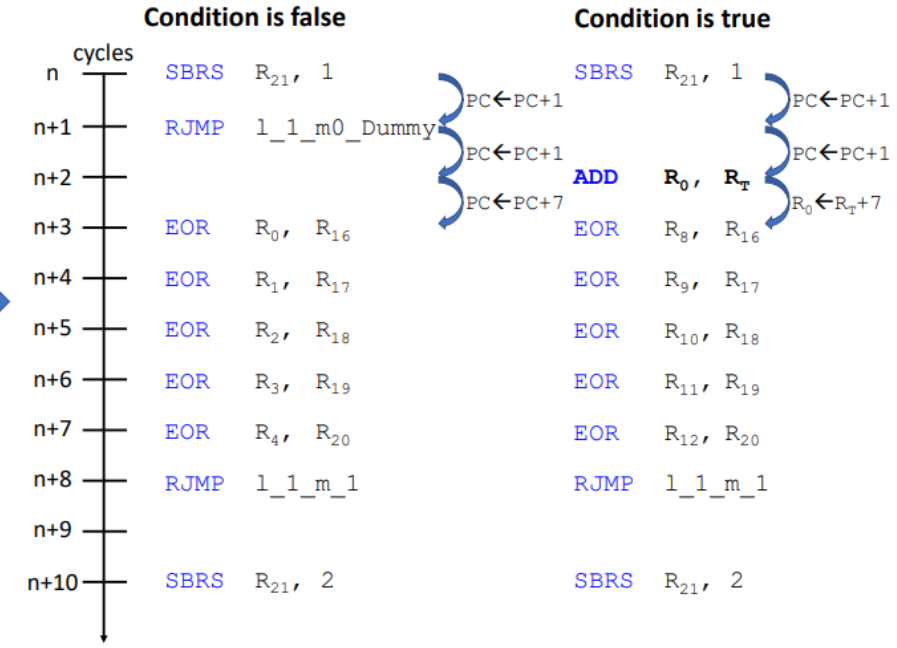
cycles	Condition is false	Condition is true
n	SBRS R_{21}, 1	SBRS R_{21}, 1
n+1	RJMP l_1_m0_Dummy	
n+2		EOR R_8, R_{16}
n+3	EOR R_0, R_{16}	EOR R_9, R_{17}
n+4	EOR R_1, R_{17}	EOR R_{10}, R_{18}
n+5	EOR R_2, R_{18}	EOR R_{11}, R_{19}
n+6	EOR R_3, R_{19}	EOR R_{12}, R_{20}
n+7	EOR R_4, R_{20}	RJMP l_1_m_1
n+8	RJMP l_1_m_1	
n+9		SBRS R_{21}, 2
n+10	SBRS R_{21}, 2	

```
; l = 1, m = 0
l_1_m_0:
   SBRS    R21, 1
   RJMP    l_1_m0_Dummy

   ADD     R0,  RT
   EOR     R8,  R16
   EOR     R9,  R17
   EOR     R10, R18
   EOR     R11, R19
   EOR     R12, R20
   RJMP    l_1_m_1

l_1_m0_Dummy:
   EOR     R0,  R16
   EOR     R1,  R17
   EOR     R2,  R18
   EOR     R3,  R19
   EOR     R4,  R20
   RJMP    l_1_m_1

; l = 1, m = 1
l_1_m_1:
   ...
   ...
```



This Work

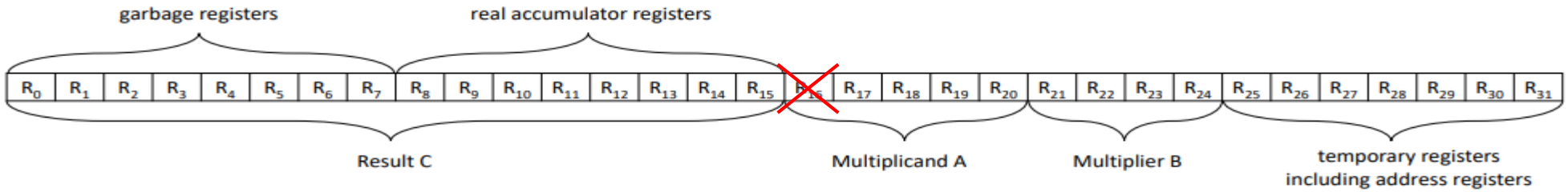


Figure 3: Register assignment in the proposed Block-Comb (BC) method.

Algorithm 5 Masked Block Comb on 32-bit

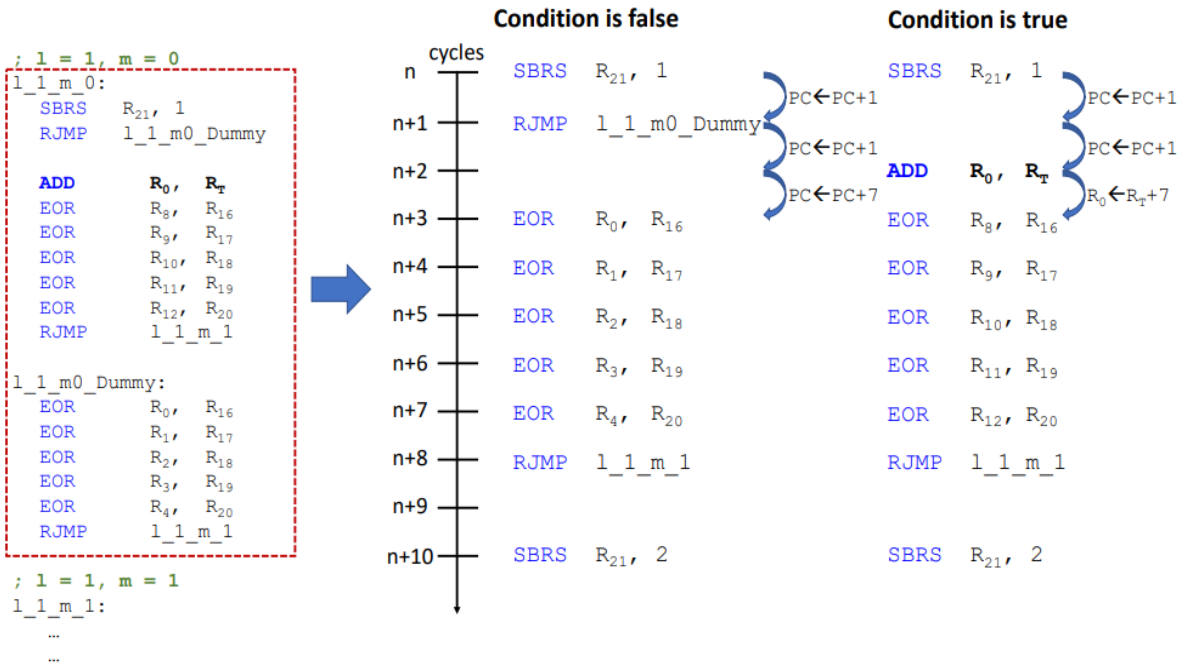
Require: 32-bit wise operands A and B
Ensure: Result $C \leftarrow A \cdot B$

1: **for** i from 7 by 1 to 0 **do**
2: **for** j from 3 by 1 to 0 **do**
3: $BIT \leftarrow A[j]\&(1 \ll i)$
4: $\{MASK, T0\} \leftarrow 0 - BIT$
5: **for** k from 3 by 1 to 0 **do**
6: $C[k+j] \leftarrow C[k+j] \oplus (B[k]\&MASK)$
7: **end for**
8: **end for**
9: $C \leftarrow C \ll 1$
10: **end for**
11: **return** C

1: $R_{25} \leftarrow 0x07$ // Set displacement value for dummy ADD instruction for ILA
2: **for** $l = 0$ to 15 **do**
3: $R_l \leftarrow 0$
4: **end for**
5: **for** $l = 0$ to 3 **do**
6: $R_{16+l} \leftarrow A[l]$
7: $R_{21+l} \leftarrow B[l]$
8: **end for**
9: $R_{20} \leftarrow 0$
10: // Processing from 0-th bit to 6-th bit
11: **for** $l = 0$ to 6 **do**
12: **for** $m = 0$ to 3 **do**
13: **if** the l-th bit of R_{21+m}==1 **then**
14: $R_0 \leftarrow R_0 + R_{25}$ // Dummy ADD instruction for ILA
15: **for** $k = 0$ to 4 **do**
16: $R_{8+m+k} \leftarrow R_{8+m+k} \oplus R_{16+k}$
17: **end for**
18: **else**
19: // Dummy XOR with the garbage registers
20: **for** $k = 0$ to 4 **do**
21: $R_{m+k} \leftarrow R_{m+k} \oplus R_{16+k}$
22: **end for**
23: **end if**
24: **end for**
25: $(R_{20}, \ldots, R_{16}) \leftarrow (R_{20}, \ldots, R_{16}) \ll 1$
26: **end for**

```
for(int i = 7; i >=0; i--)
{
    for(int j = 3; j >= 0; j--)
    {
        if(a[j] & (1 << i))
        {
            c[j] ^= b[0];
            c[j+1] ^= b[1];
            c[j+2] ^= b[2];
            c[j+3] ^= b[3];
        }
        else
        {
            G[j] ^= b[0];
            G[j+1] ^= b[1];
            G[j+2] ^= b[2];
            G[j+3] ^= b[3];
        }
    }
    c[7] = (c[7] << 1) | (c[6] >> 7);
    c[6] = (c[6] << 1) | (c[5] >> 7);
    c[5] = (c[5] << 1) | (c[4] >> 7);
    c[4] = (c[4] << 1) | (c[3] >> 7);
    c[3] = (c[3] << 1) | (c[2] >> 7);
    c[2] = (c[2] << 1) | (c[1] >> 7);
    c[1] = (c[1] << 1) | (c[0] >> 7);
    c[0] = c[0] << 1;
}
```

This Work




```
1:
   sbrs A3, 7
   rjmp 2f

   add G0, TEMP
   eor C3, B0
   eor C4, B1
   eor C5, B2
   eor C6, B3
   rjmp 1f

2:
   eor G3, B0
   eor G4, B1
   eor G5, B2
   eor G6, B3
   rjmp 1f

1:
   sbrs A2, 7
   rjmp 2f

   add G0, TEMP
   eor C2, B0
   eor C3, B1
   eor C4, B2
   eor C5, B3
   rjmp 1f

2:
   eor G2, B0
   eor G3, B1
   eor G4, B2
   eor G5, B3
   rjmp 1f
```

```
1:
   sbrs A1, 7
   rjmp 2f

   add G0, TEMP
   eor C1, B0
   eor C2, B1
   eor C3, B2
   eor C4, B3
   rjmp 1f

2:
   eor G1, B0
   eor G2, B1
   eor G3, B2
   eor G4, B3
   rjmp 1f

1:
   sbrs A0, 7
   rjmp 2f

   add G0, TEMP
   eor C0, B0
   eor C1, B1
   eor C2, B2
   eor C3, B3
   rjmp 1f

2:
   eor G0, B0
   eor G1, B1
   eor G2, B2
   eor G3, B3
   rjmp 1f
```

```
1:
   LSL C0
   ROL C1
   ROL C2
   ROL C3
   ROL C4
   ROL C5
   ROL C6
   ROL C7
```

This Work (Evaluation)

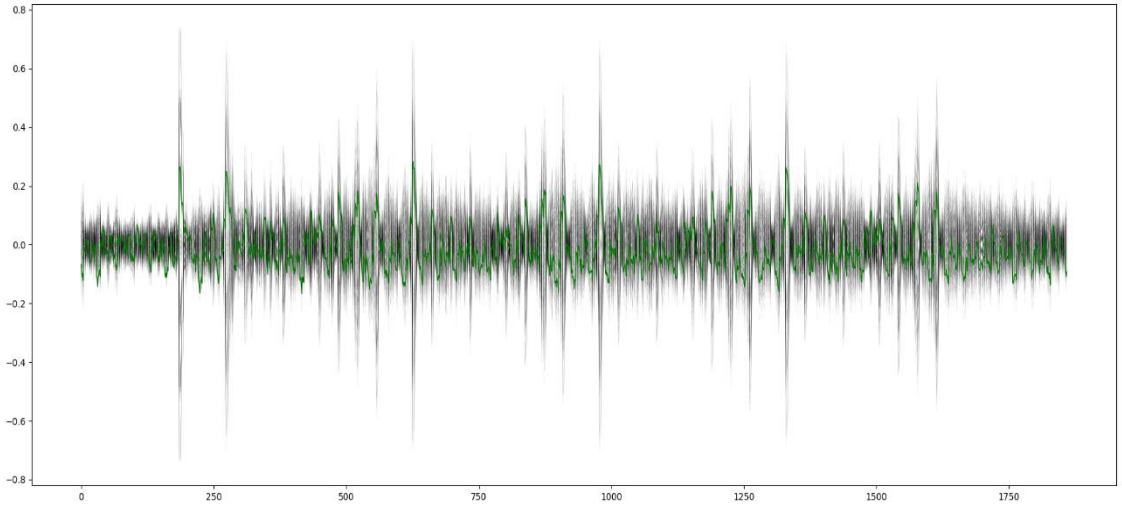
Bit	Method	Timing (*cc*)
32-bit	Proposed Block-Comb	490
64-bit	Proposed Karatsuba Block-Comb (Level 1)	1,330
128-bit	Proposed Karatsuba Block-Comb (Level 2)	5,675

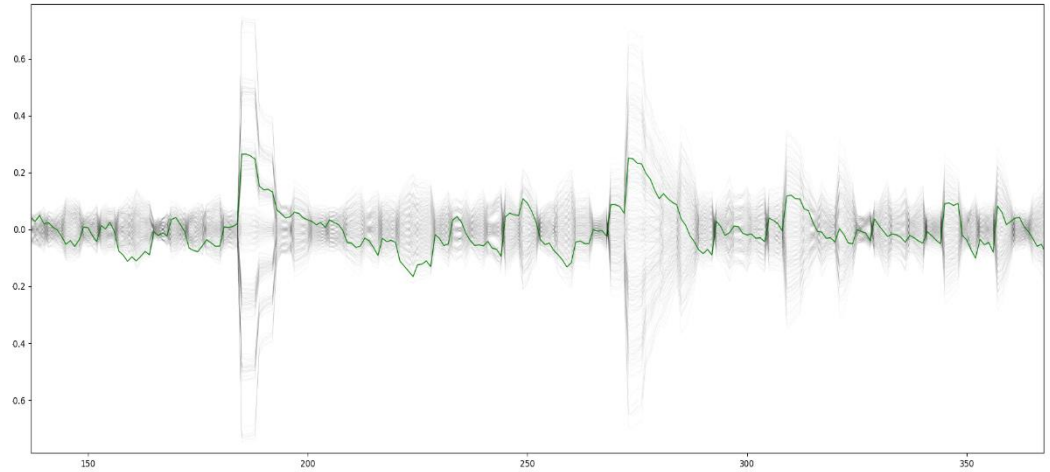
<SCA-Resistant GCM Implementation>

Bit	Method	Clock Cycle 최적화(cc)	Code 최적화 (cc)
32-bit	Proposed Block-Comb	478	-
64-bit	Karatsuba (Level 1)	1318	1383
128-bit	Karatsuba (Level 2)	5750 – a	-

<This Work>

This Work (SCA - CPA)

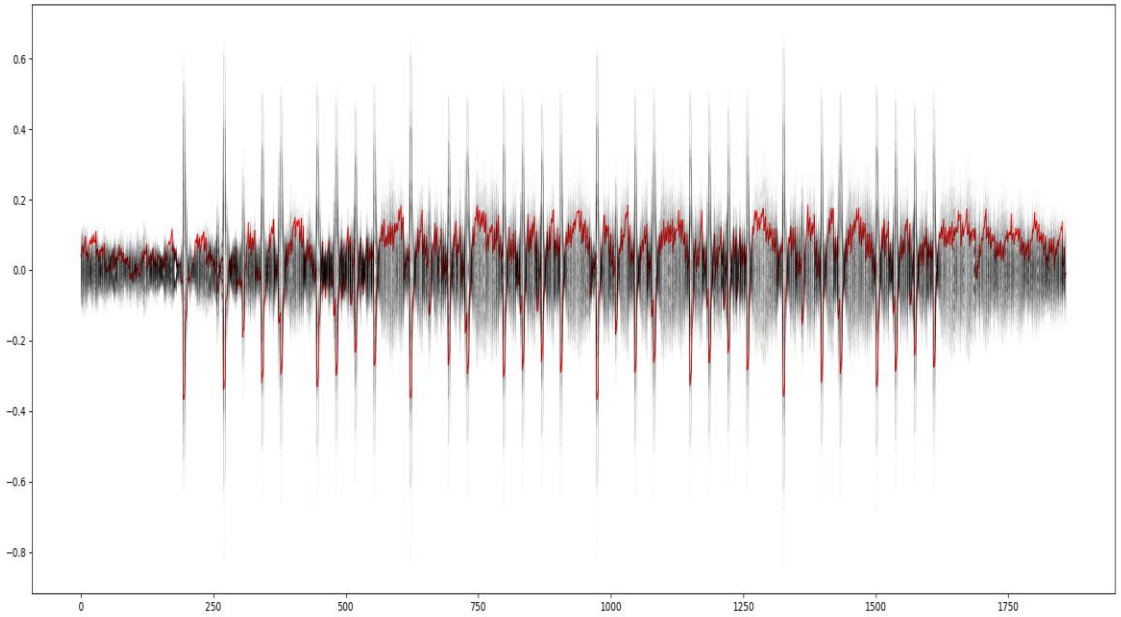


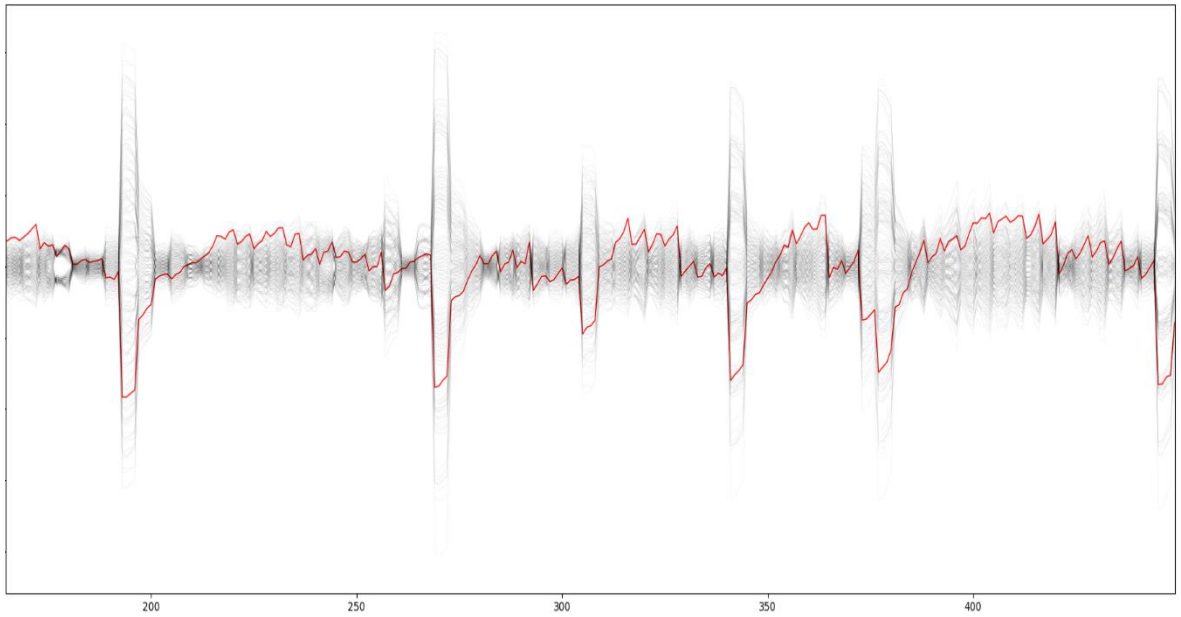


<B[0]>

- 구현 결과에 CPA 공격
- c[j+k] ^= b[k] 를 중간값으로 공격
- 성공적으로 방어된 것을 확인

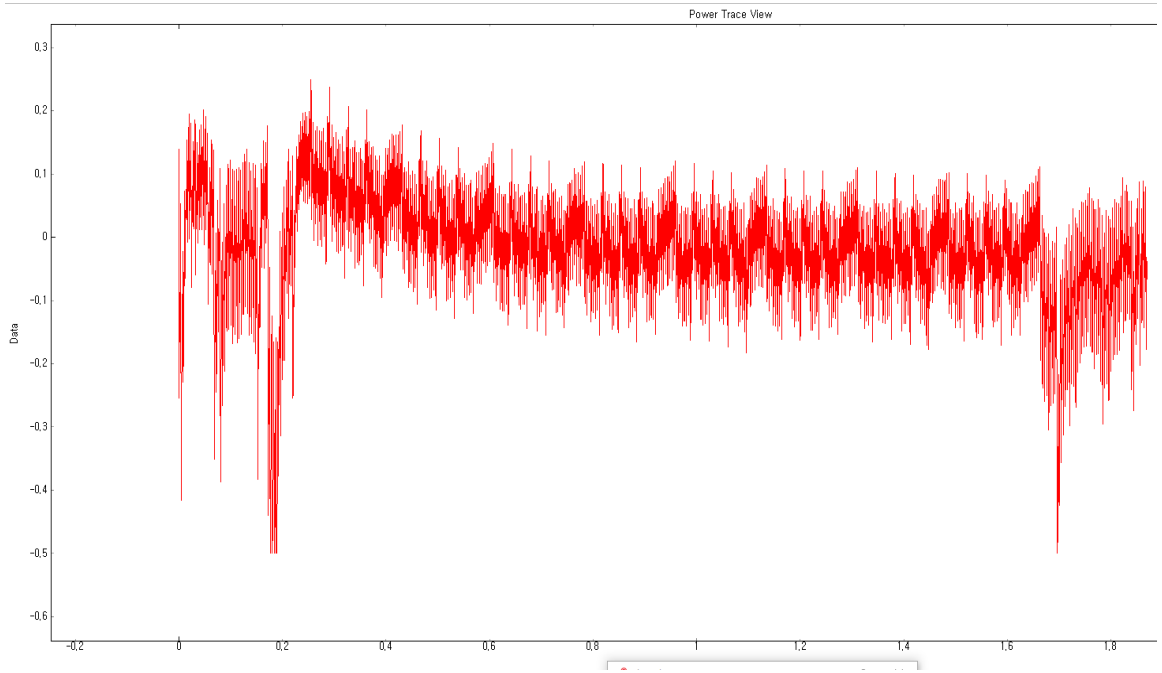
This Work (SCA - CPA)



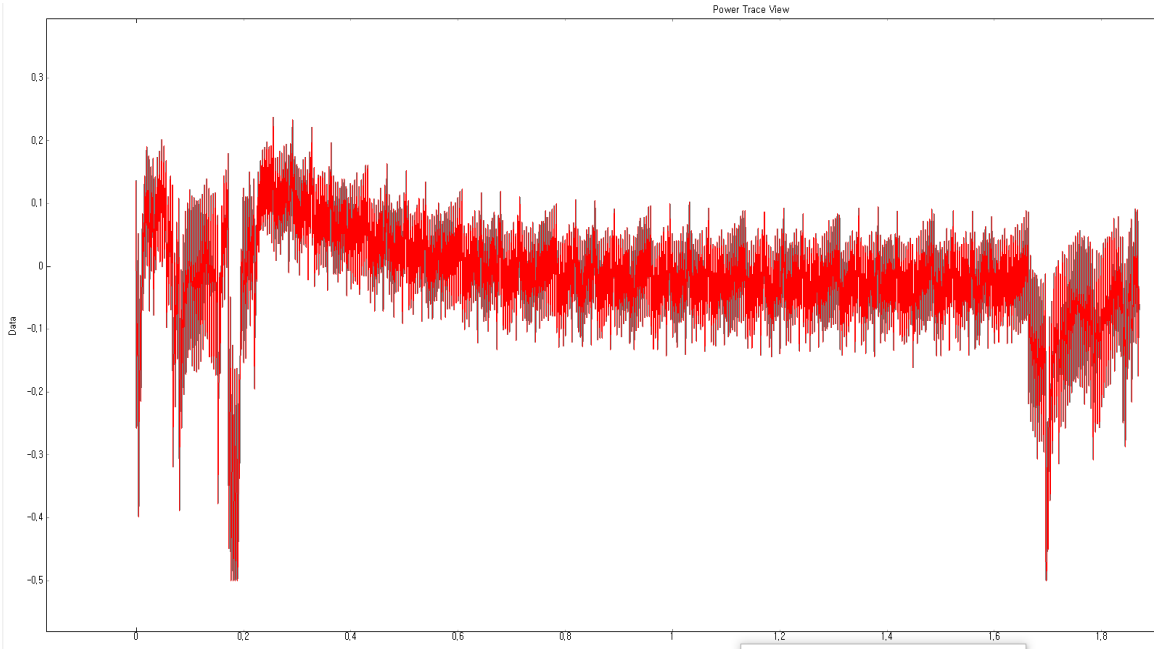


<B[1]>

This Work (SCA - SPA)



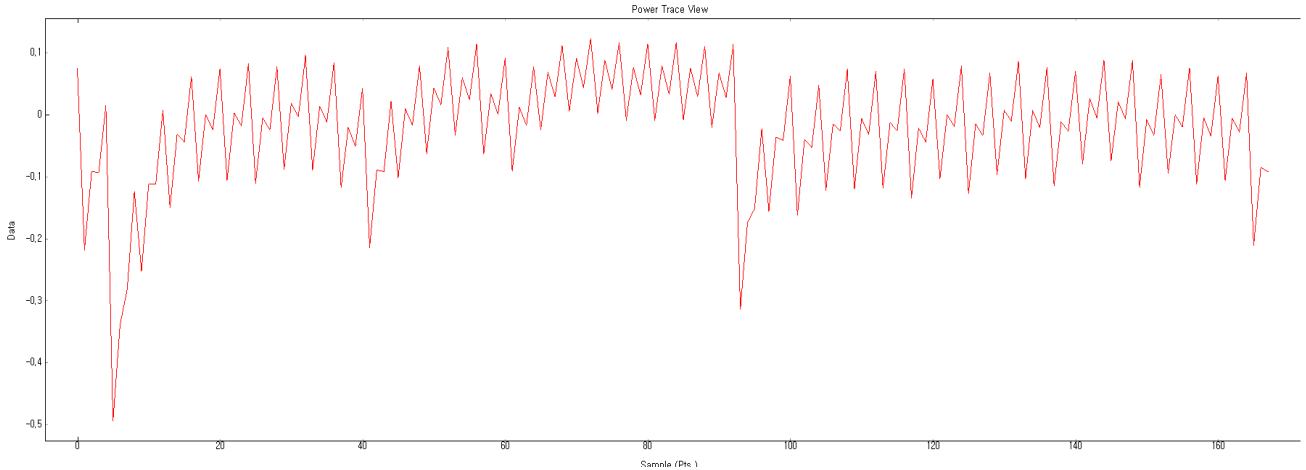
A = 0xffffffff

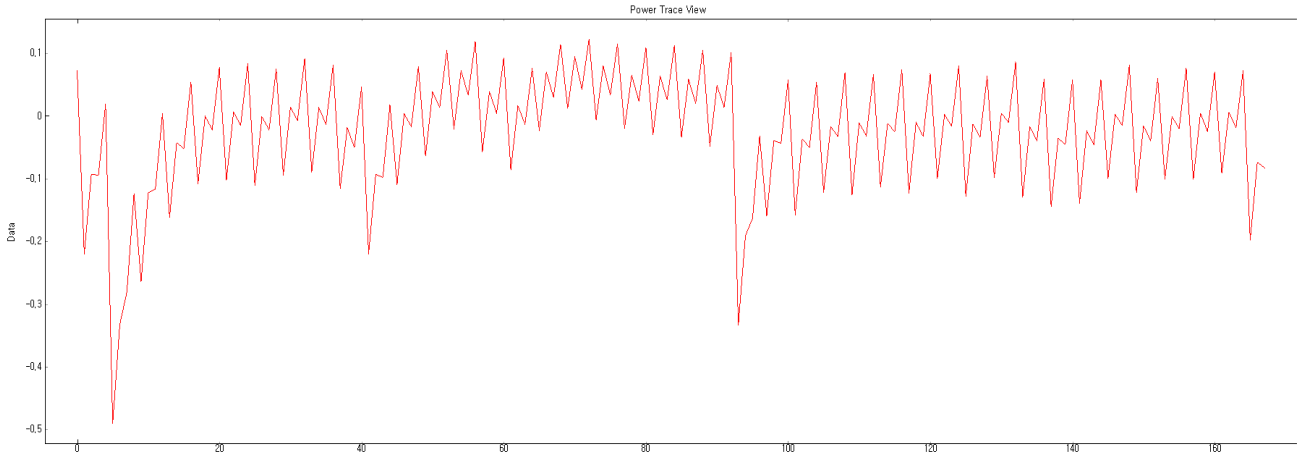


A = 0

- Real Register가 연산되는 0xffffffff 값 Garbage Register가 연산되는 0 값 파형이 거의 비슷한 것을 확인
- Timing Attack 불가능

This Work





Garbage Register 8개 사용

vs

Garbage Register 1개 사용

- 전력 파형에서 큰 변화가 없는 것을 확인
- 7개의 Register 확보 가능
- 구현 후 CPA 공격 확인 필요

Future Work

- 고정 값인 H 값을 사전 연산하여 저장한 후 최적화 작업
- 확보한 7개 레지스터를 이용한 최적화 작업
- 구현 후 CPA 공격 시도 후 안전성 확보

Q & A