

# 스마트디바이스 부채널 분석 경진대회

## (정답 발표)

2017년 02월 17일 (금)

SICADA



## ▣ 스마트디바이스 부채널 분석 경진대회 문제

ARIA			
No		분석 장비	ATmega-128
01		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

LEA			
No		분석 장비	ChipWhisperer-Lite
04		분석 대상	Normal LEA
		구현 비트	16비트

SEED			
No		분석 장비	ChipWhisperer-Lite
05		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

AES			
No		분석 장비	ChipWhisperer-Lite
07		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

ARIA			
No		분석 장비	ATmega-128
01		분석 대상	Normal ARIA
		구현 비트	08비트
		분석 장비	ARM 902T
02		분석 대상	Normal ARIA
		구현 비트	08비트
		분석 장비	ATmega-128
03		분석 대상	First-Order Masked ARIA
		구현 비트	08비트
		분석 장비	ATmega-128

LEA			
No		분석 장비	ChipWhisperer-Lite
04		분석 대상	Normal LEA
		구현 비트	16비트
		분석 장비	ChipWhisperer-Lite

SEED			
No		분석 장비	ChipWhisperer-Lite
05		분석 대상	Normal SEED
		구현 비트	08비트
		분석 장비	ChipWhisperer-Lite
06		분석 대상	First-Order Masked SEED
		구현 비트	08비트
		분석 장비	ChipWhisperer-Lite

AES			
No		분석 장비	ChipWhisperer-Lite
07		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
		분석 장비	ATmega-328P
08		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
		분석 장비	ATmega-328P
09		분석 대상	First-Order Masked AES
		구현 비트	08비트
		분석 장비	ATmega-328P

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

연번	알고리즘	대응기법	분석장비	구현비트	파형수	포인트수	용량	제출문제	제공소스
01	ARIA	Normal	ATmega-128	08	10,000	10,002	381M	1. CPA	ARIA 표준문서
02	ARIA	Normal	ARM 920T	08	5,000	130,000	2.6G	1. SPA 2. CPA	
03	ARIA	First-Order Masking	ATmega-128	08	2,000	125,002	1G	1. CPA	
04	LEA	Normal	ChipWhisperer	16	5,000	6,000	114M	1. CPA	LEA 표준문서 공개용 LEA 소스코드
05	SEED	Normal	ChipWhisperer	08	10,000	3792	145M	1. SPA 2. CPA [Xor된 키] 3. CPA [라운드 키]	
06	SEED	First-Order Masking	ChipWhisperer	08	10,000	24,000	916M	1. CPA [Xor된 키] 2. CPA [라운드 키]	SEED 표준문서 공개용 SEED 소스코드
07	AES	Eight-Shuffling	ChipWhisperer	08	30,000	3,492	401M	1. CPA	
08	AES	Normal with random clock cycle	ATmega-328P	08	200	70,000	53.4M	1. CPA	AES 표준문서
09	AES	First-Order Masking	ATmega-328P	08	2,000	90,000	686M	1. CPA	

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 01

제공되는 정보는 8비트 기반 소프트웨어로 구현된 **ARIA-128 암호** 알고리즘이 Atmega128에서 동작 할 때 소비되는 전력을 **1라운드 부분**을 타겟으로 수집한 결과이다.

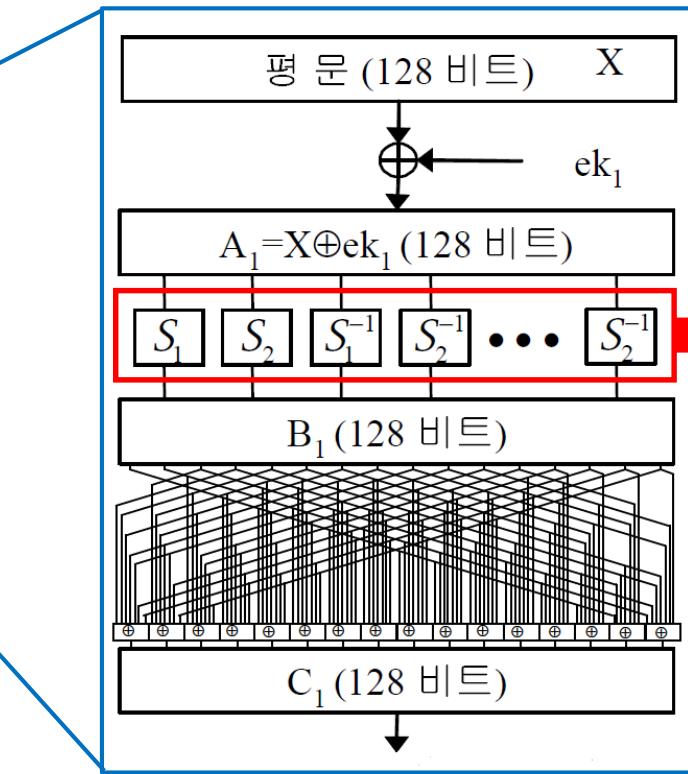
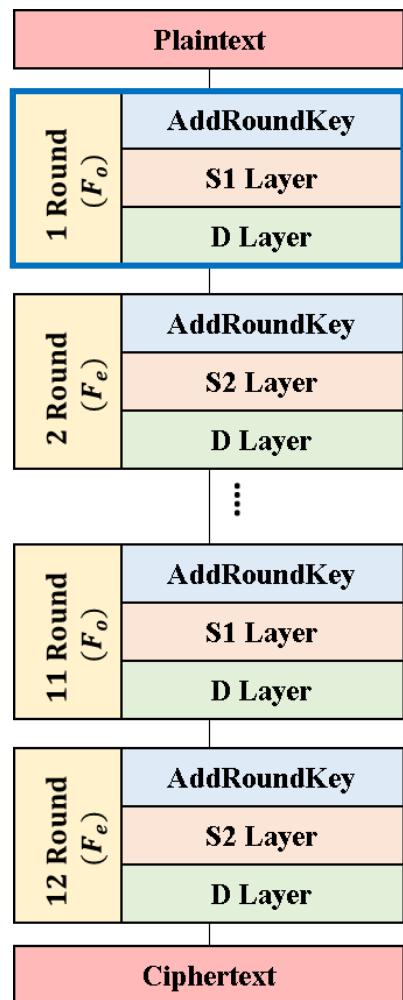
binary 파일 01\_Normal\_ARIA\_10000tr\_10002pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 ARIA 알고리즘이 10,000번 시행된 소비 전력이 포함되어 있는 파일이다.

01\_Normal\_ARIA\_10000tr\_10002pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 소비 전력 및 평문 정보를 이용하여 ARIA 암호 알고리즘 동작 시 사용된 1라운드 키를 찾으시오.

## 01\_Normal ARIA / ATmega128 / 8 bit / 분석 논리

### Normal ARIA



공격 위치

- S-box Output
- Non-linear
- 8 bit Guessing

중간 값

t : S-box 위치  
i : i 번째 파형

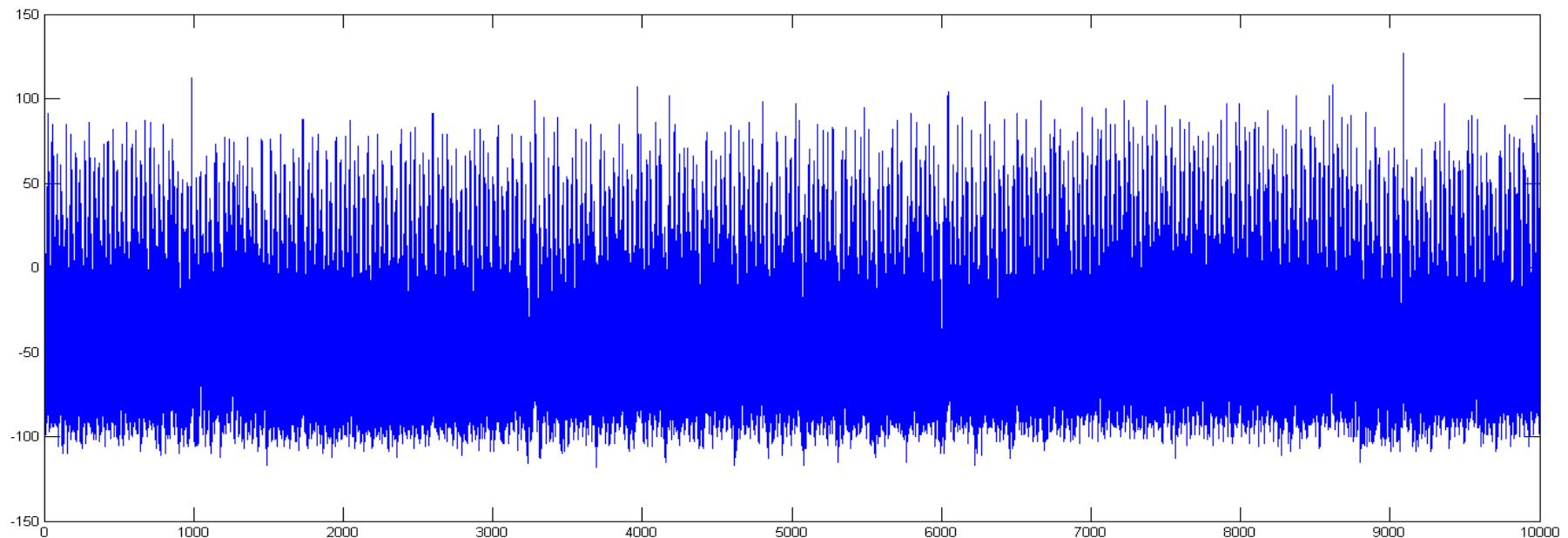
// 01\_Normal\_ARIA 중간값

key = \_ARIA\_SBOX\_[t%4][plaintext[i][t] ^ (guess\_key)];

## ▣ 01\_Normal ARIA / ATmega128 / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

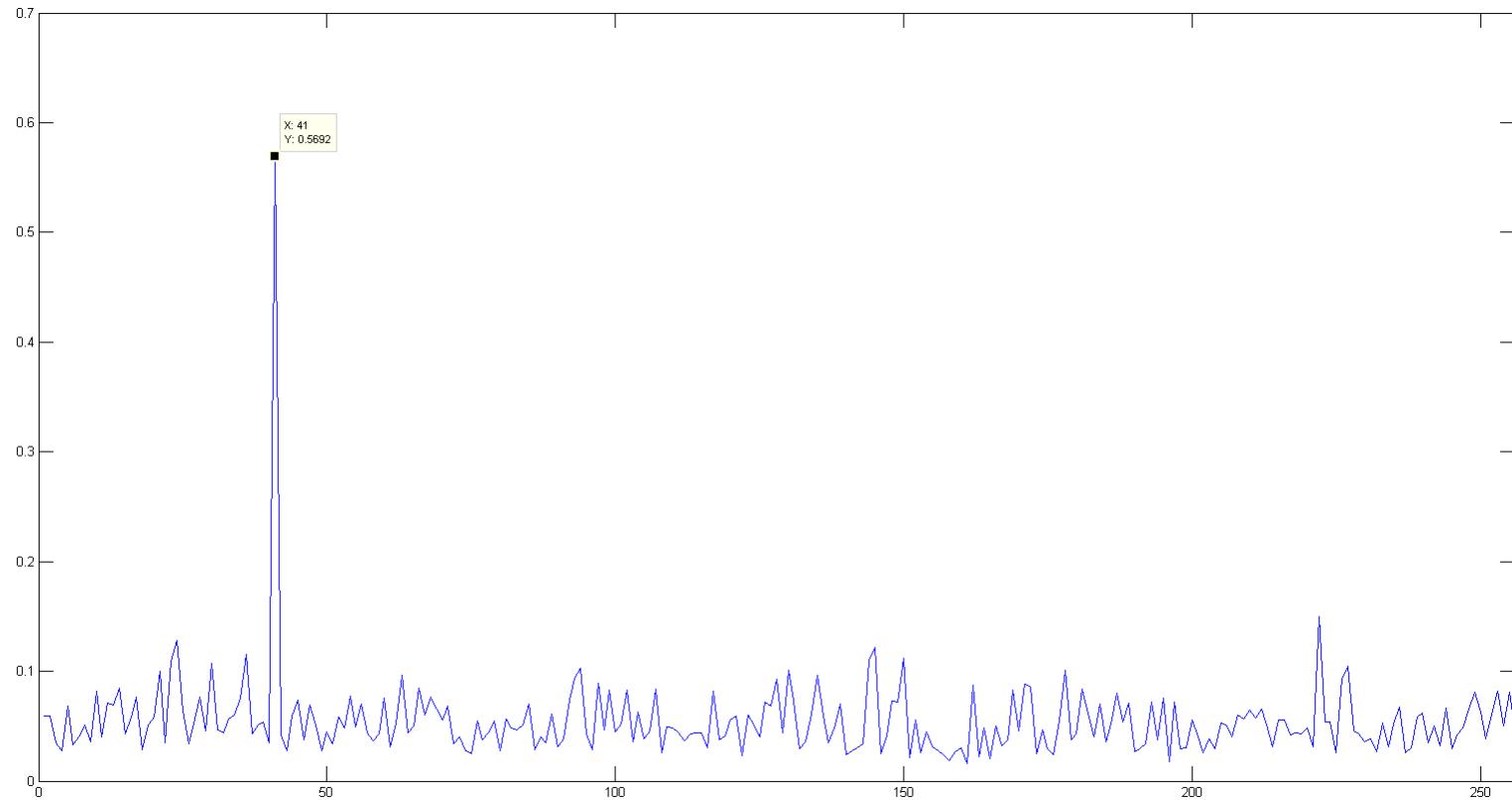
파형 정보	
파형 수	10,000
포인트 수	10,002



- ❖ 분석 위치 (S-Box)를 구분하기 어려움
- ❖ 따라서 전 구간 분석 수행

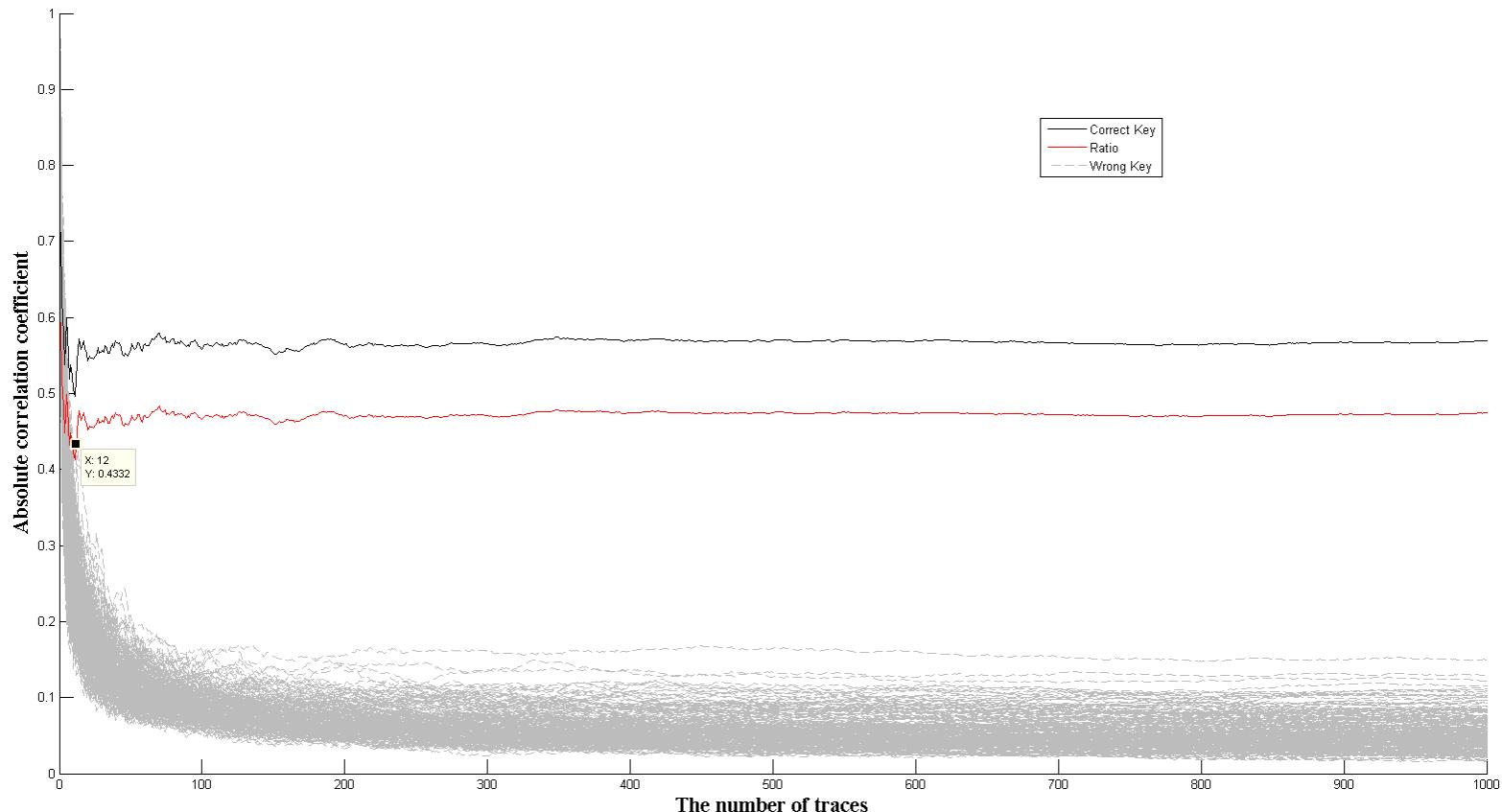
## ▣ 01\_Normal ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 01\_Normal ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte 패형 증가에 따른 분석 결과



## ▣ 01\_Normal ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) First Order CPA 결과

✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	28	43	07	1A	0F	10	89	A5	9E	54	9E	C1	00	A2	19	A1

✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	3.79	1.94	2.10	2.09	3.30	3.73	3.54	3.94	3.53	3.72	3.69	2.45	3.99	3.48	3.75	4.63

✓ 최소 분석 파형 수

( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
파형 수	12	106	124	64	26	22	25	13	6	14	11	27	11	12	19	10

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	<b>ARM 902T</b>
		분석 대상	<b>Normal ARIA</b>
		구현 비트	<b>08비트</b>
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 02

제공되는 정보는 32비트 기반 소프트웨어로 구현된 **ARIA-128 암호 알고리즘**이 ARM920T 보드에서 동작 할 때 소비되는 전력을 **라운드 부분**을 타겟으로 수집한 결과이다.

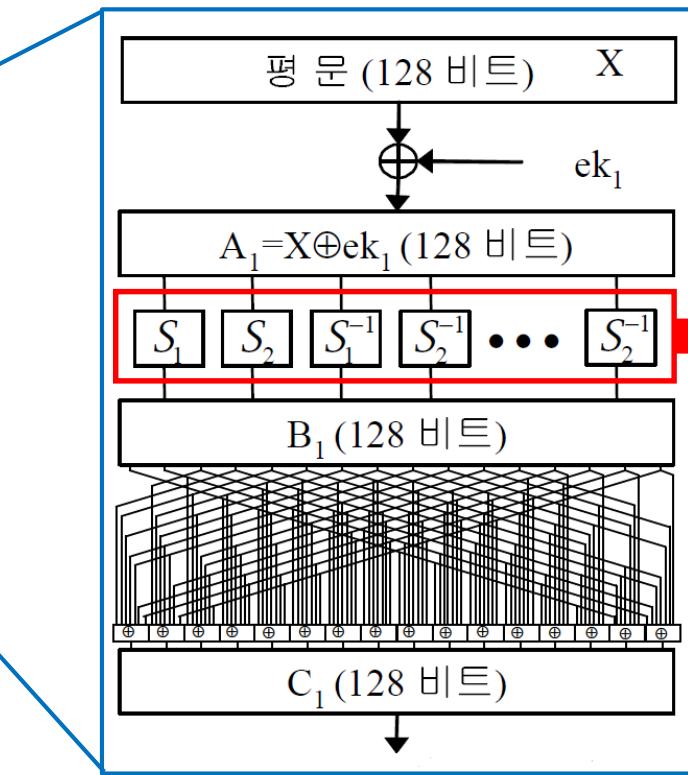
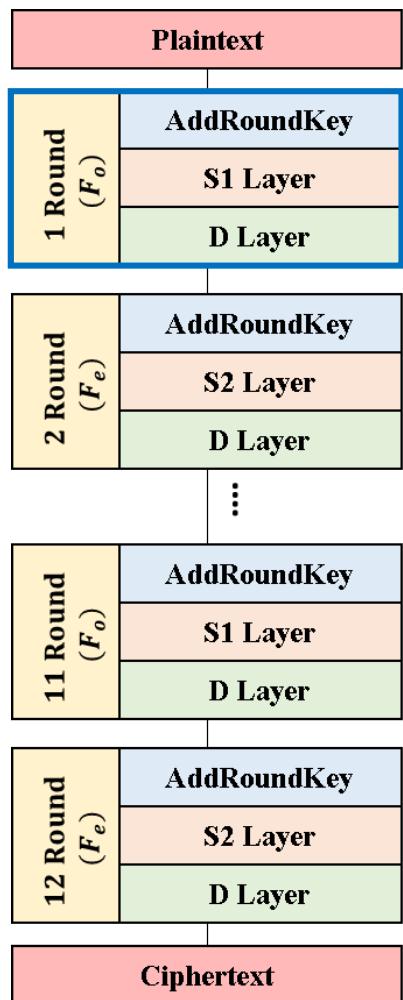
binary 파일 02\_Normal\_ARIA\_5000tr\_130000pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 ARIA-128 알고리즘에서 라운드 부분이 5,000번 시행된 소비 전력이 포함되어 있는 파일이다. ([참고 1]은 키 스케줄링이 포함된 전체 소비 전력 파형이다.)

02\_Normal\_ARIA\_5000tr\_130000pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 주어진 소비 전력에 대하여 동봉된의 ARIA 알고리즘 명세서([01\_ARIA\_표준문서.pdf])을 활용하여 각각의 라운드 위치를 추측하여 나타내시오.
- 2) 소비 전력 및 평문 정보를 이용하여 ARIA-128 암호 알고리즘 동작 시 사용된 1라운드 키를 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)

## 02\_Normal ARIA / ARM902T / 8 bit / 분석 논리

### Normal ARIA



공격 위치

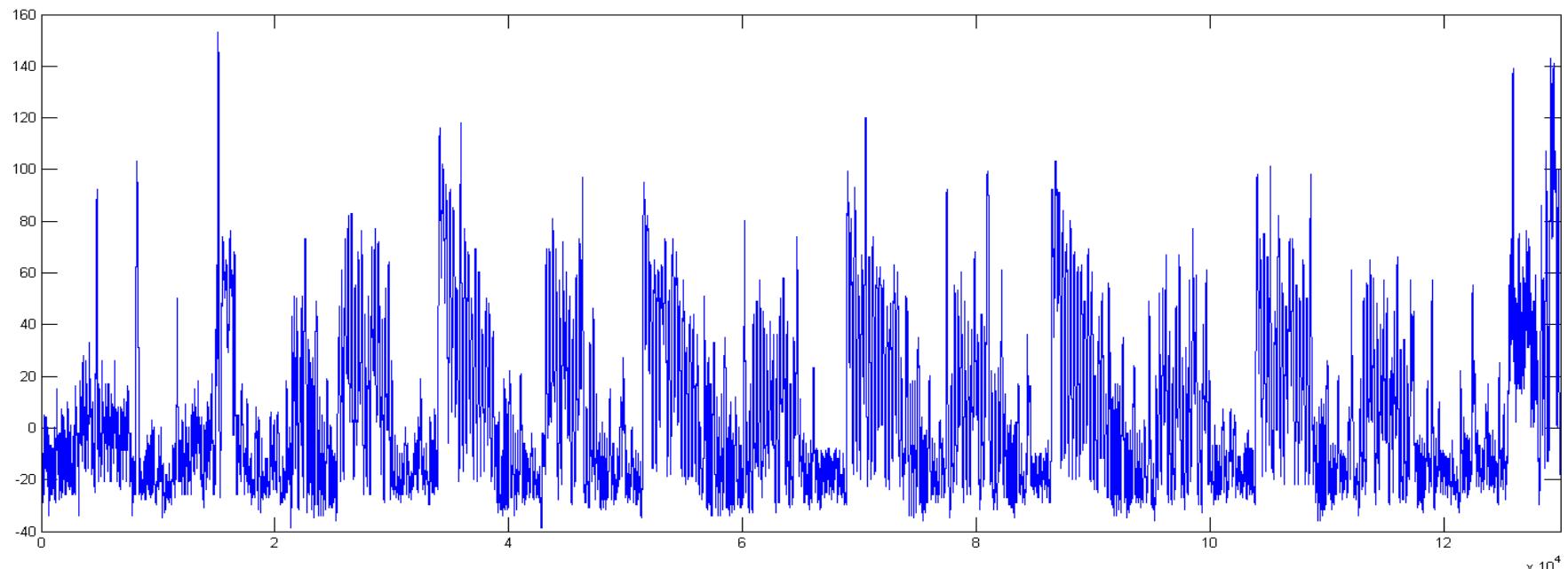
- S-box Output
- Non-linear
- 8 bit Guessing



## ■ 02\_Normal ARIA / ARM902T / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

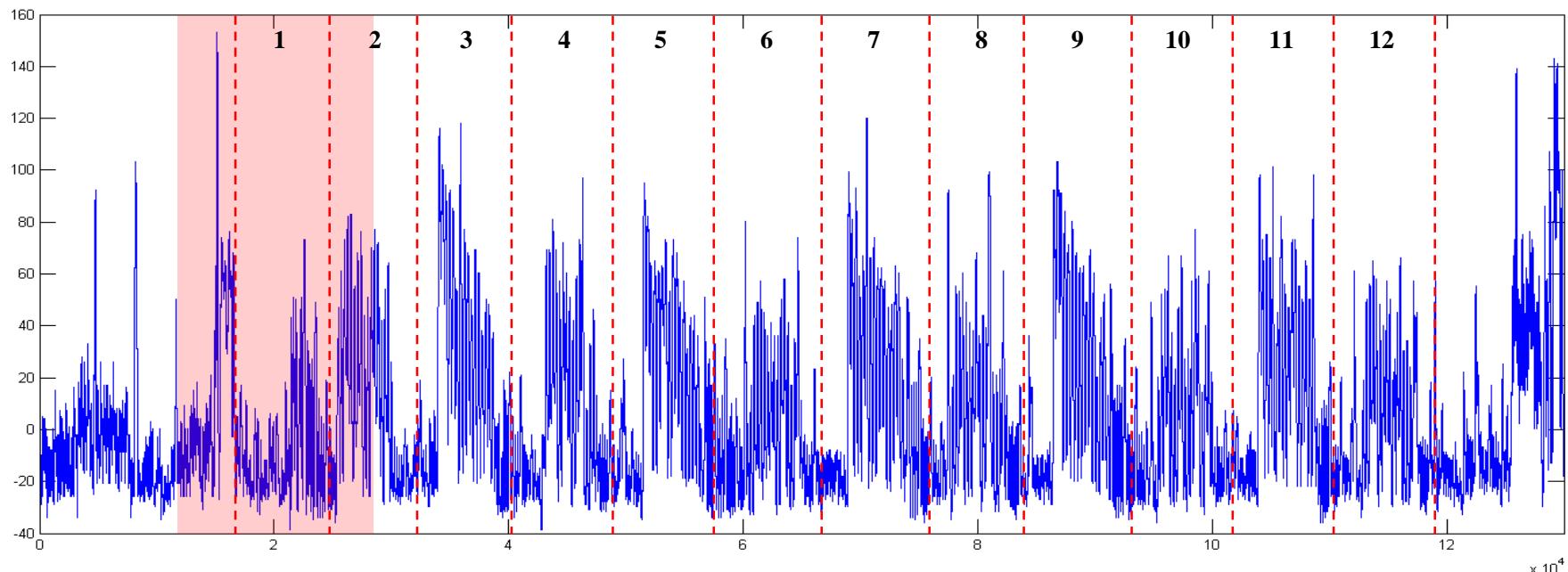
파형 정보	
파형 수	5,000
포인트 수	130,000



## ▣ 02\_Normal ARIA / ARM902T / 8 bit / 분석 결과

### ❖ (1) 각 라운드 위치 추측

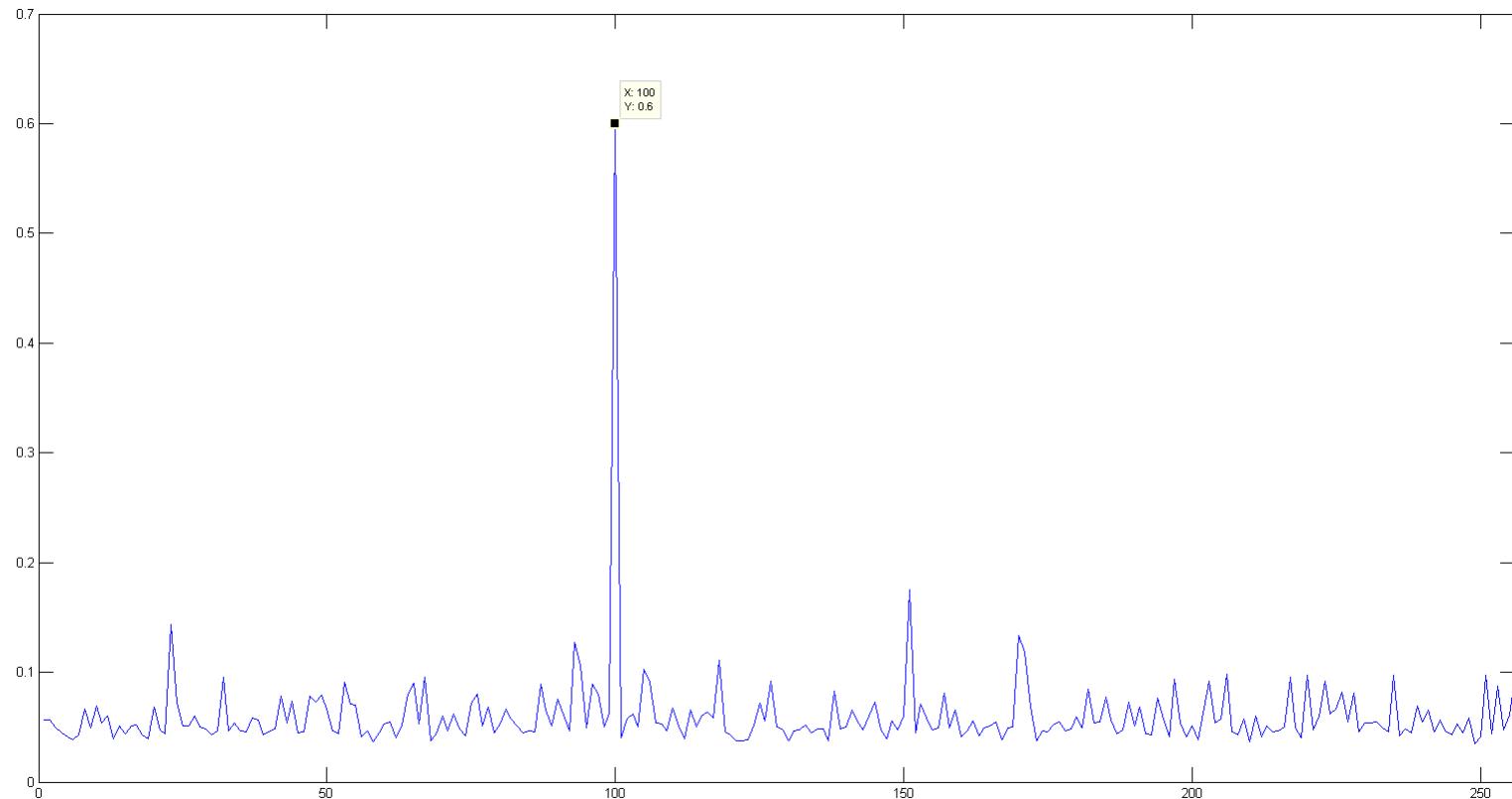
파형 정보	
파형 수	5,000
포인트 수	130,000



- ❖ ARIA 12 라운드 구분 가능하다고 추측 가능
- ❖ 또한 전체 포인트 수가 많음 (130000pts) → 전 구간 분석은 비효율적
- ❖ 따라서 1 Round 예상 구간만 분석 수행 (대략 15000 – 30000pts)

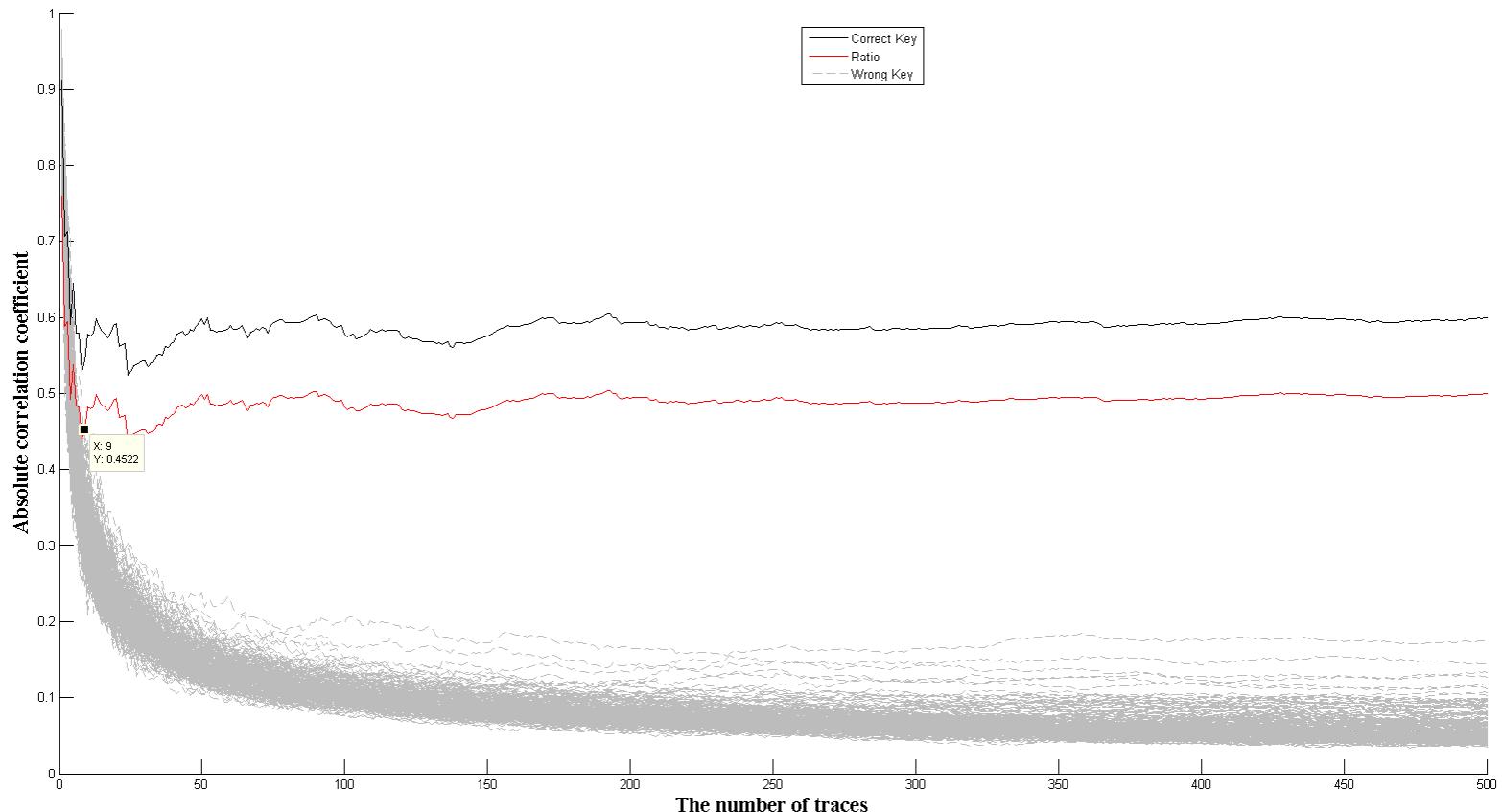
## ▣ 02\_Normal ARIA / ARM902T / 8 bit / 분석 결과

### ❖ (2) S-Box 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 02\_Normal ARIA / ARM902T / 8 bit / 분석 결과

### ❖ (2) S-Box 1<sup>st</sup> Byte 패형 증가에 따른 분석 결과



## ▣ 02\_Normal ARIA / ARM902T / 8 bit / 분석 결과

### ❖ (2) First Order CPA 결과

✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	63	CD	F6	C4	8E	A6	AB	28	84	19	46	87	62	F0	AE	E4

✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	3.42	3.20	2.93	2.65	3.35	2.33	2.20	2.25	2.42	2.35	3.39	4.35	3.87	3.98	4.52	1.94

✓ 최소 분석 파형 수

( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
파형 수	9	68	46	148	39	150	79	138	116	97	53	8	20	6	10	58

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	<b>ATmega-128</b>
		분석 대상	<b>First-Order Masked ARIA</b>
		구현 비트	<b>08비트</b>

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 03

제공되는 정보는 1차 전력 분석 대응 기법으로 **마스킹이 적용된 8비트 기반 소프트웨어로 구현된 ARIA-128 암호 알고리즘이 ATmega128에서 동작 할 때 소비되는 전력을 1라운드 부분을 타겟으로 수집한 결과이다.**

binary 파일 03\_First-Order\_Masked\_ARIA\_2000tr\_125002pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 ARIA알고리즘이 2,000번 시행된 소비 전력이 포함되어 있는 파일이다.

03\_First-Order\_Masked\_ARIA\_2000tr\_125002pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

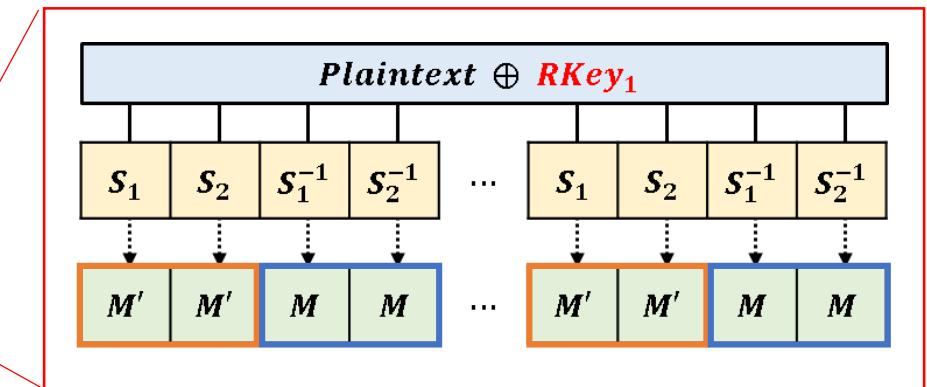
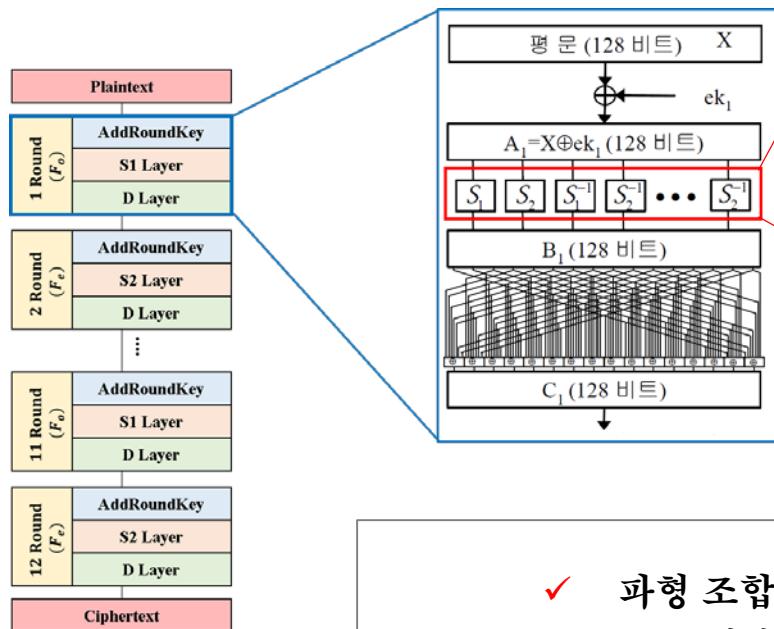
1) 소비 전력 및 평문 정보를 이용하여 ARIA 암호 알고리즘 동작 시 사용된 암호화키의 1라운드키를 최소의 소비 전력 정보를 이용하여 찾으시오.

[표 1] 각 Sbox에 대한 peak의 파형에서 위치

Sbox	0	1	2	3	4	5	6	7
위치 ( $\times 10^4$ )	6.343	6.361	6.379	6.396	6.411	6.429	6.447	6.464
Sbox	8	9	10	11	12	13	14	15
위치 ( $\times 10^4$ )	6.479	6.497	6.515	6.532	6.547	6.565	6.583	6.609

## ■ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 논리

### ❖ Normal ARIA



- 출력 마스킹이 동일한 두 바이트 선택  
(01&02) / (02&03) / ... / (15&16)

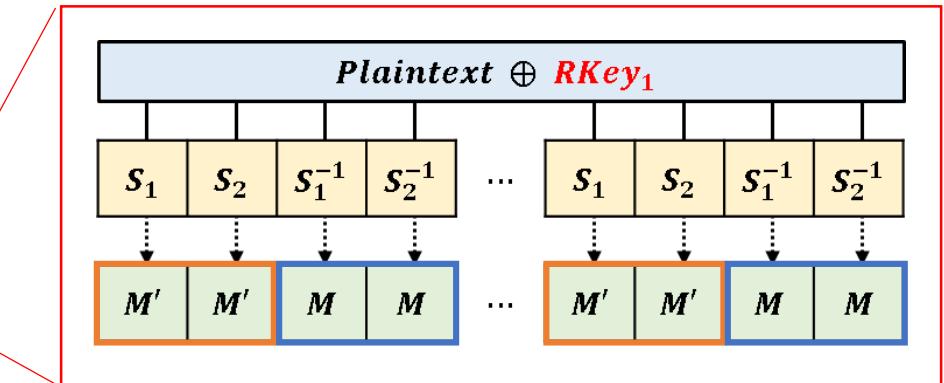
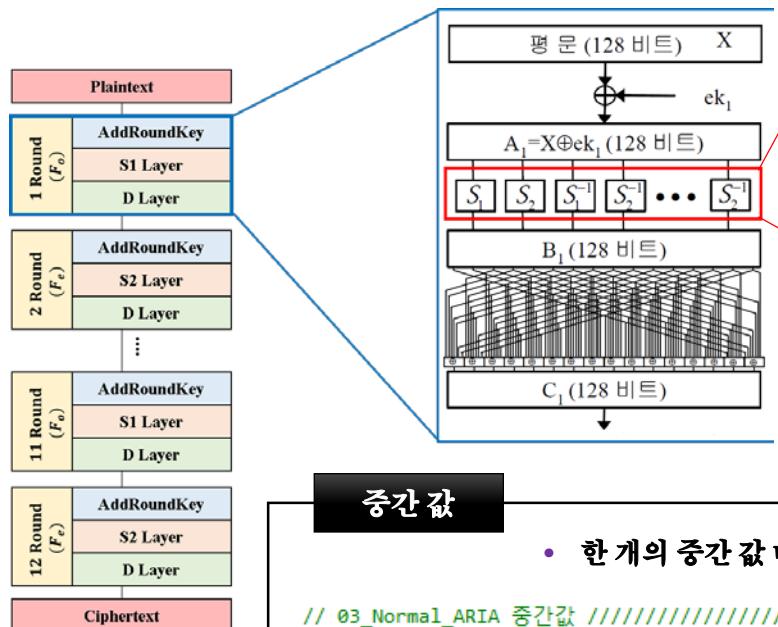
### ✓ 파형 조합 방법 (Preprocessing)

$C_i : i$  번째 S-Box 연산으로 예상되는 전력 값

1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
3. 수정된 차분 조합 :  $| C_i - C_j |$

## ■ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 논리

#### ❖ Normal ARIA



- 출력 마스킹이 동일한 두 바이트 선택  
(01&02) / (02&03) / ... / (15&16)

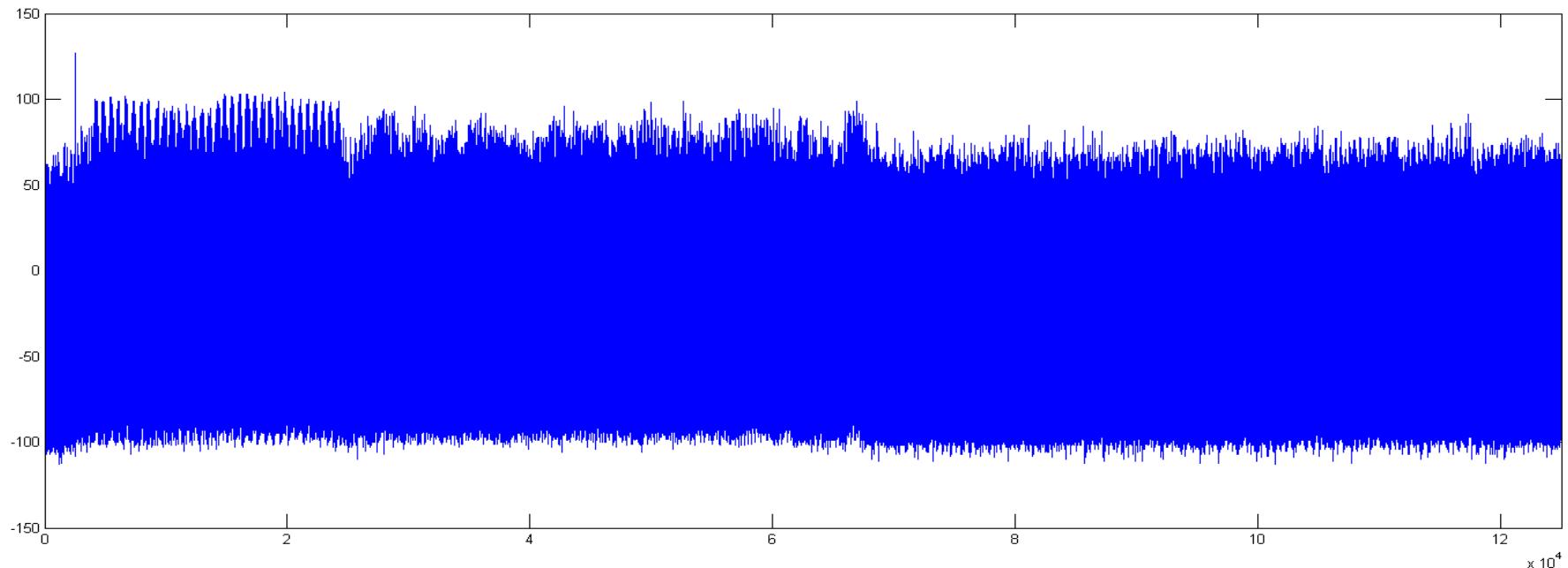
$t : S\text{-box 위치}$   
 $i : i$  번째 파형

- 한 개의 중간 값마다 독립적으로 수행 (총 8번)

## ■ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ 전력 파형 SPA (1<sup>st</sup> trace)

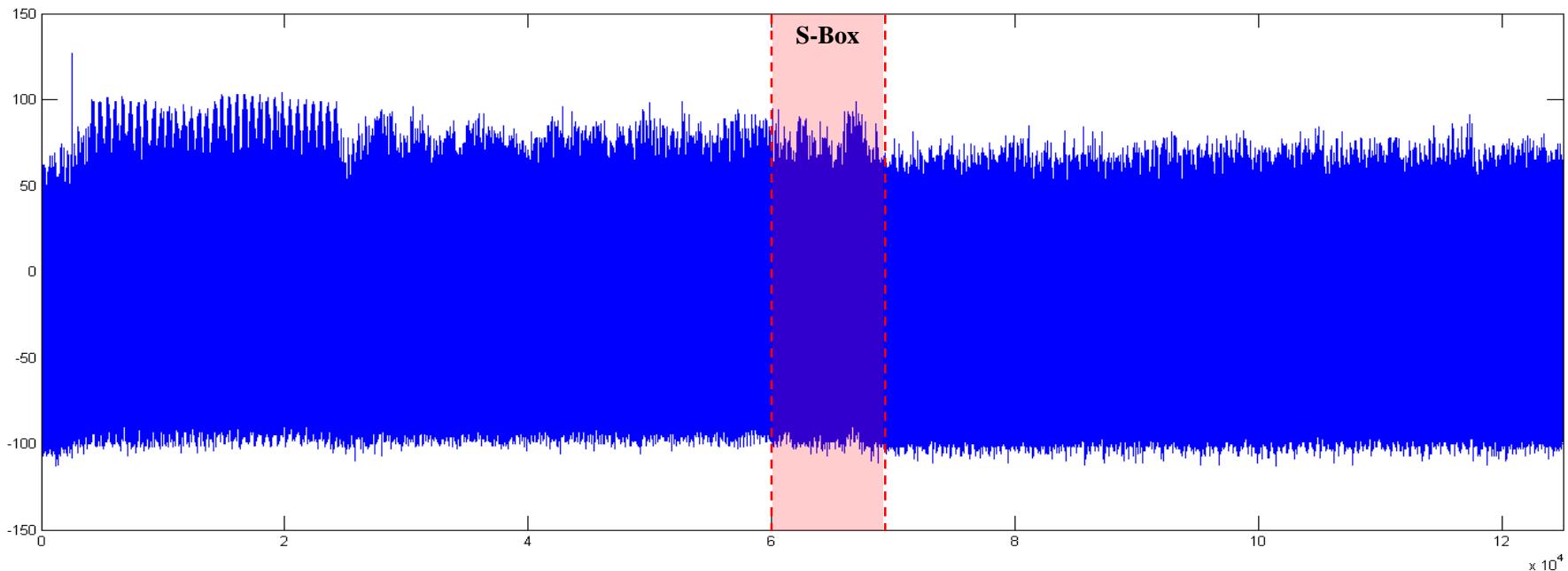
파형 정보	
파형 수	2,000
포인트 수	125,002



## ■ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) S-Box 연산 위치

파형 정보	
파형 수	2,000
포인트 수	125,002

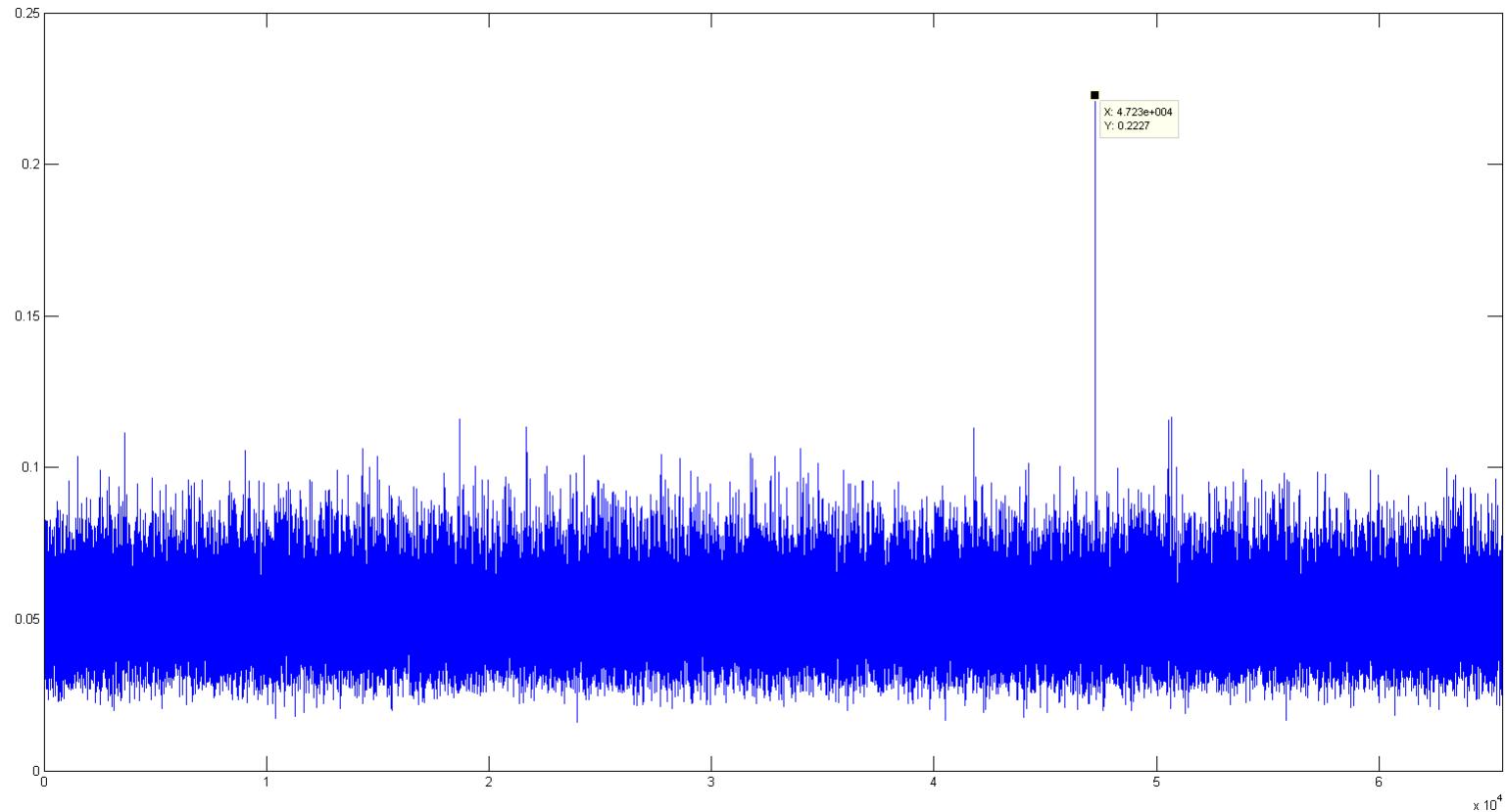


### ❖ SPA 를 통한 S-Box 구분 어려움 → 문제에서 S-Box 포인트 제공

S-box	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
위치	63430	63610	63790	63960	64110	64290	64470	64640	64790	64970	65150	65320	65470	65650	65830	66090

## ▣ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) S-Box ( $1^{\text{st}}$ Byte $\oplus$ $2^{\text{nd}}$ Byte) Maxpeak 분석 결과



## ■ 03\_Masked ARIA / ATmega128 / 8 bit / 분석 결과

### ❖ (1) Second Order CPA 결과

✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	B8	7F	05	12	CD	52	DD	CB	EE	38	6F	90	EE	CD	38	A9

✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	1.91	1.70	2.16	1.22	2.06	1.49	2.08	2.03								

### ❖ 마스킹이 동일한 두 바이트씩 조합하여 분석 진행

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 04

제공되는 정보는 16비트 기반 소프트웨어로 구현된 **LEA-128 암호 알고리즘**이 ChipWhisperer-Lite에서 동작 할 때 소비되는 전력을 **1라운드 부분**을 타겟으로 수집한 결과이다.

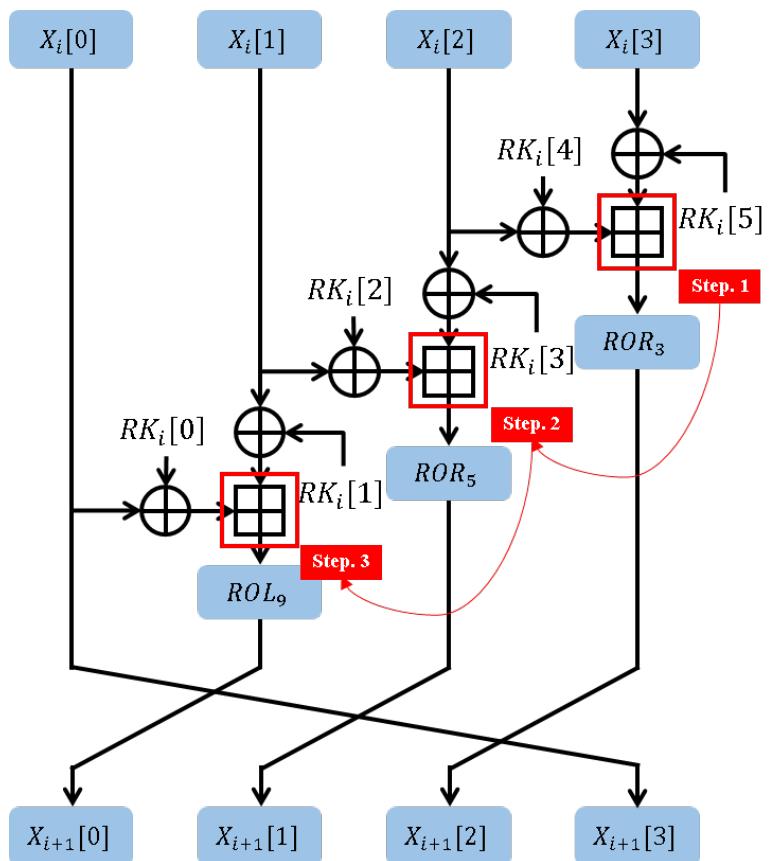
Binary 파일 04\_Normal\_LEA\_5000tr\_6000pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 LEA-128 알고리즘이 5,000번 시행된 소비 전력이 포함되어 있는 파일이다.

04\_Normal\_LEA\_5000tr\_6000pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 소비 전력 및 평문 정보를 이용하여 LEA-128 암호 알고리즘 동작 시 사용된 1라운드키( $RK^0$ )를 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)

## Normal LEA / ChipWhisperer-Lite / 16 bit

### ❖ Normal LEA



- $ROL_i(x)$  : the i-bit left rotation on a 32-bit value x
- $ROR_i(x)$  : the i-bit right rotation on a 32-bit value x
- 24 라운드 :  $0 \leq i \leq 23$ ,  $RK_i[1] = RK_i[3] = RK_i[5]$

### Step. 1

- Nibble 단위 공격 (공격 시간 고려)
- 하위 Nibble 부터 공격 (Carry 값 고려)
- 하위 Nibble에서 발생한 Ghost Key Filtering 필요

### Step. 2

- $RK_i[3]$  후보군 존재 시, Ghost Key Filtering 필요
- Byte 단위 공격 가능
- 하위 Byte에서 발생한 Ghost Key Filtering 필요

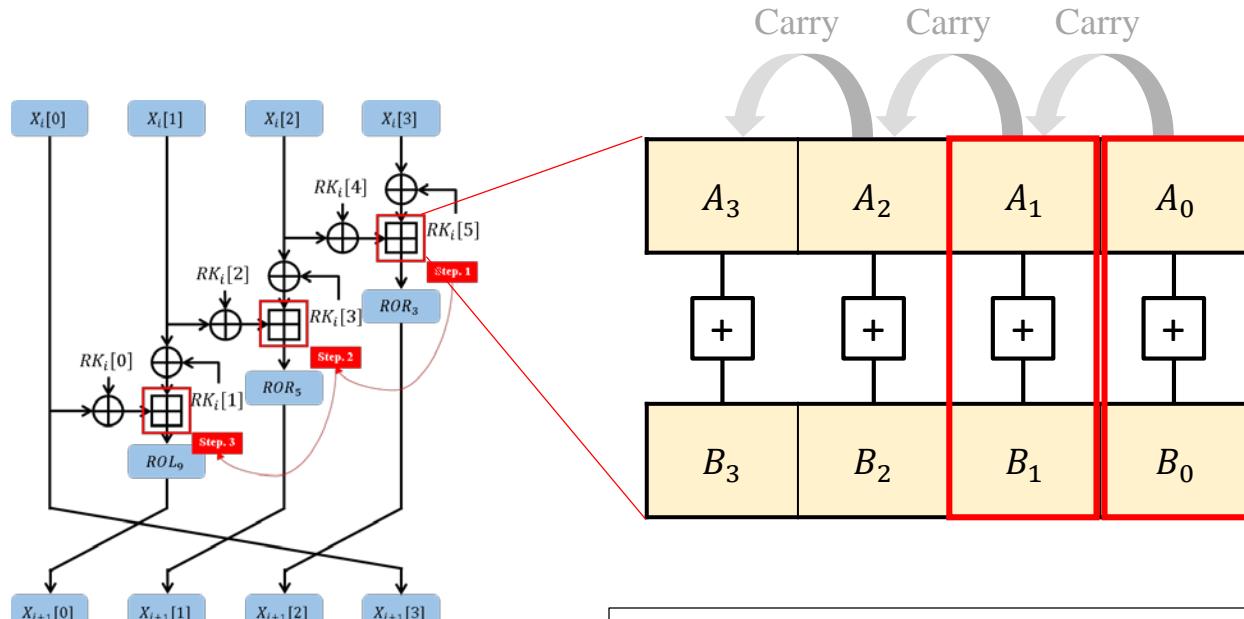
### Step. 3

- $RK_i[1]$  후보군 존재 시, Ghost Key Filtering 필요
- Byte 단위 공격 가능
- 하위 Byte에서 발생한 Ghost Key Filtering 필요

\* 공격 위치에 대한 순서는 바뀔 수 있음

## Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

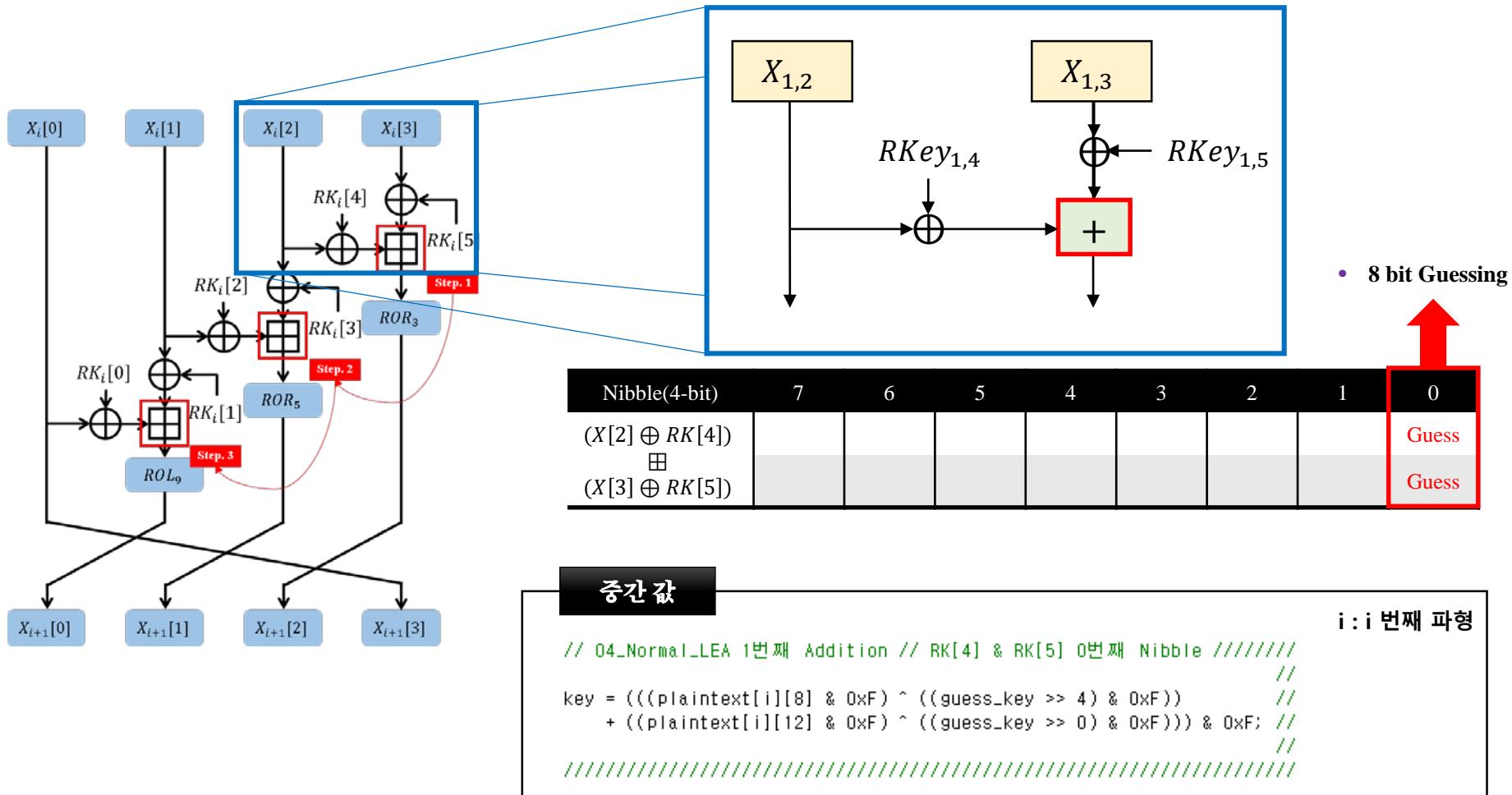
### ❖ Addition Carry



- ❖ 32-bit 덧셈 연산을 8-bit 씩 수행할 경우 상위 8-bit에 대해 Carry가 발생함
- ❖ 따라서 최하위  $A_0 + B_0$  부터 연산한 후 Carry를 고려해서  
 $Carry(A_0 + B_0) + A_1 + B_1$  연산 수행

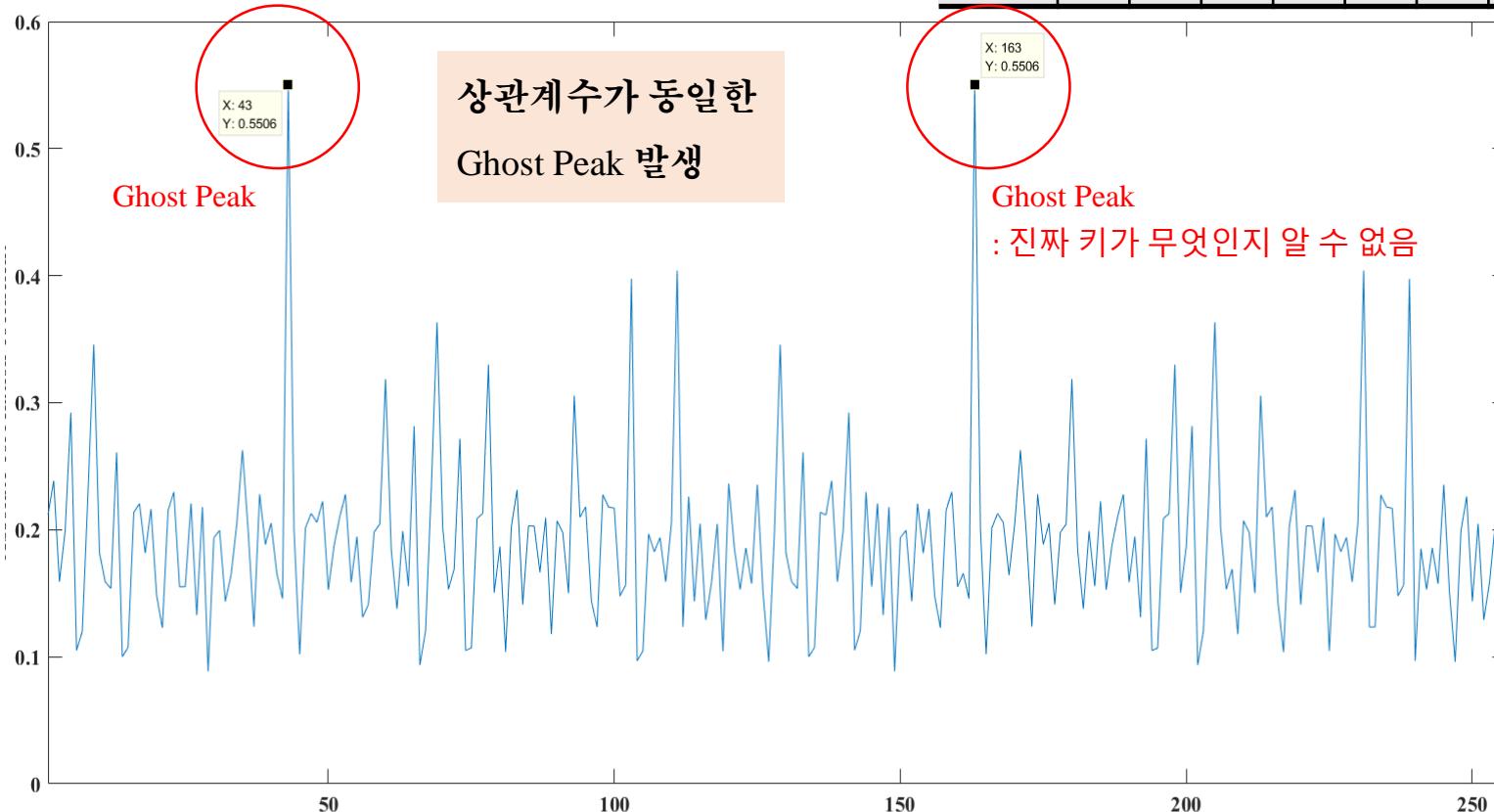
## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- ❖ [Step. 1] 1번째 Addition 연산 / 0번째 Nibble → 7번째 Nibble 덧셈 순으로 분석



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

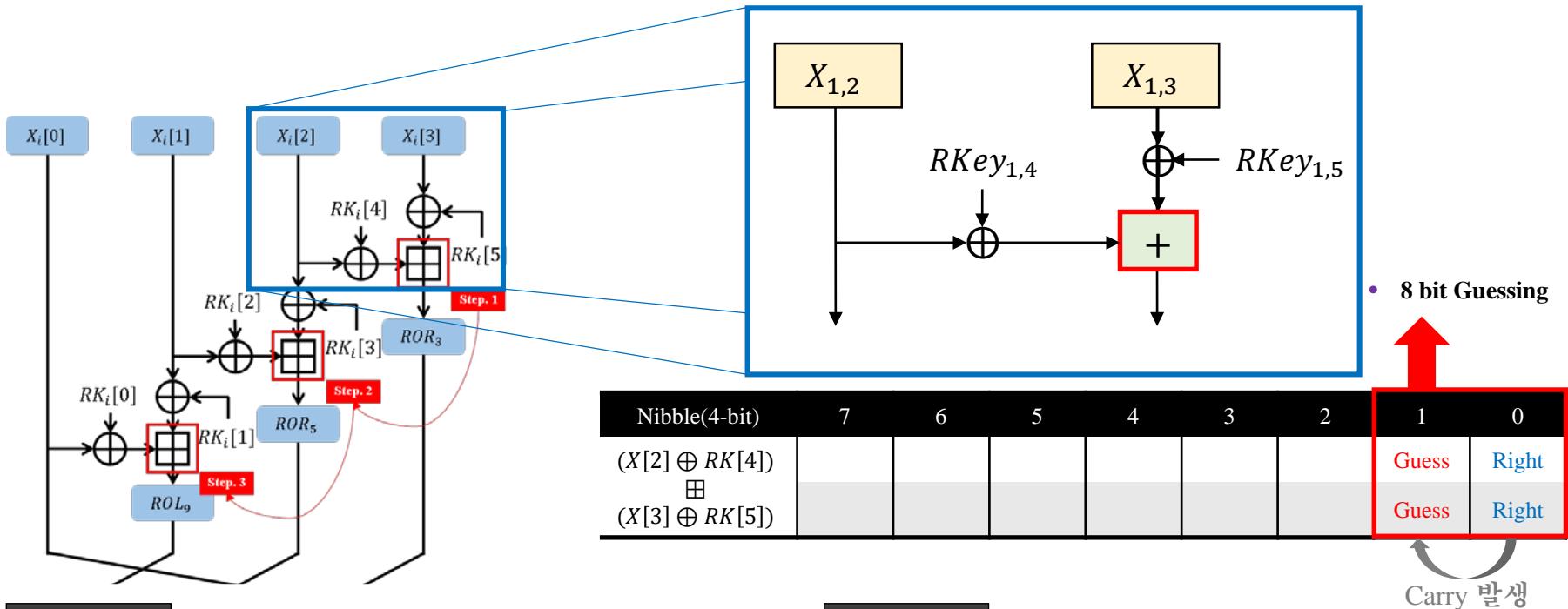
### ❖ [Step. 1] 0번째 Nibble / Maxpeak 분석 결과



### ❖ Ghost Peak를 필터링 하기 위해 분석된 두 개의 키를 각각 다음 Nibble 추측에 사용

▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- ❖ [Step. 1] 1번째 Addition 연산 / 0번째 Nibble → 7번째 Nibble 덧셈 순으로 분석

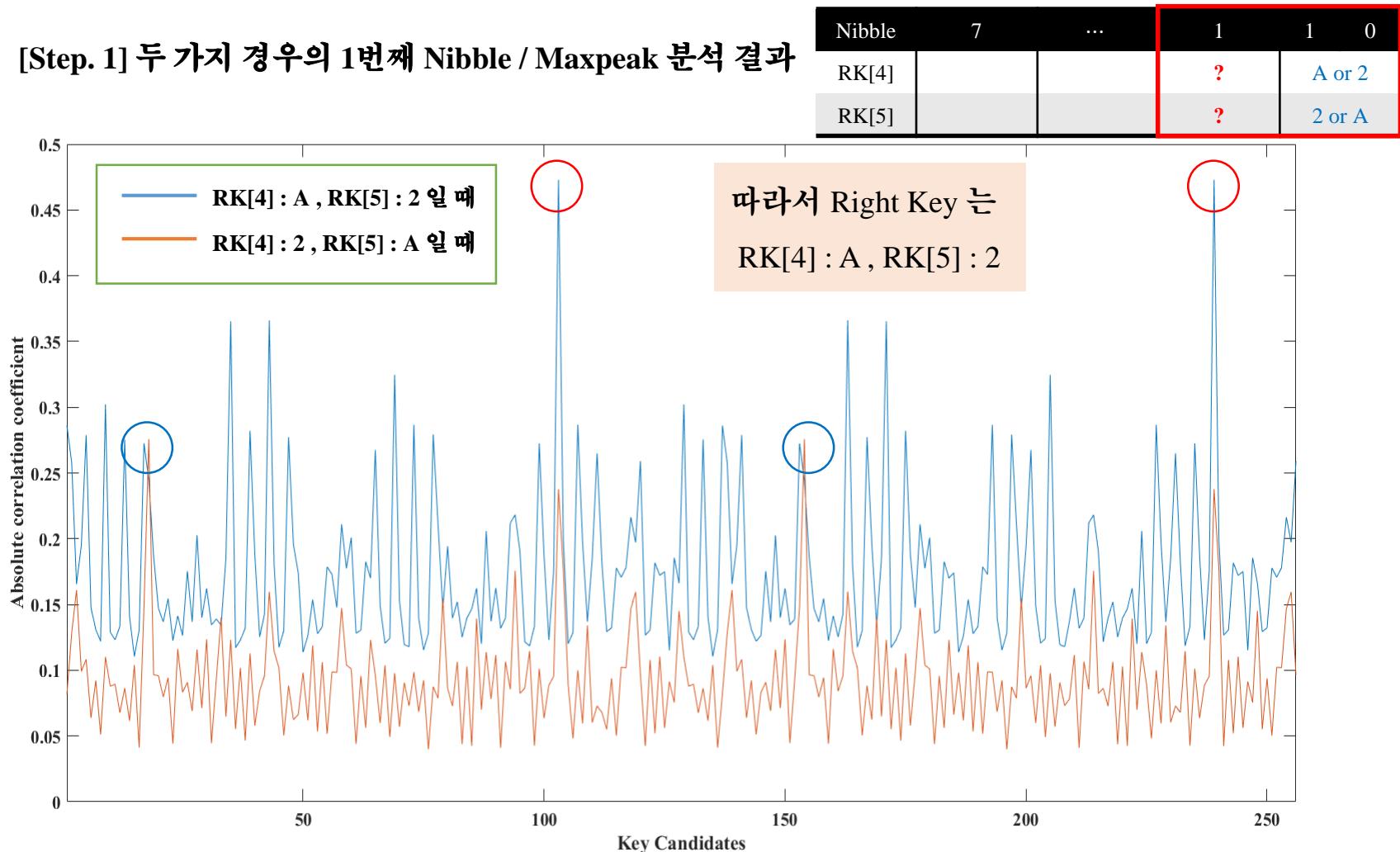


```
// 04_Normal_LEA 1번째 Addition // RK[4] & RK[5] 1번째 Nibble //////////////////////  
key = (((plaintext[i][8] >> 4) ^ ((guess_key >> 4) & 0xF))  
      + ((plaintext[i][12] >> 4) ^ ((guess_key >> 0) & 0xF));  
key <= 4;  
key += ((plaintext[i][8] & 0xF) ^ 0xA) + ((plaintext[i][12] & 0xF) ^ 0x2);
```

```
// 04_Normal_Lea 1번째 Addition // RK[4] & RK[5] 1번째 Nibble //////////////////////  
키 후보2 //  
key = ((plaintext[i][8] >> 4) ^ ((guess_key >> 4) & 0xF)) //  
+ ((plaintext[i][12] >> 4) ^ ((guess_key >> 0) & 0xF)); //  
key <= 4; //  
key += ((plaintext[i][8] & 0xF) ^ 0x2) + ((plaintext[i][12] & 0xF) ^ 0xA); //
```

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

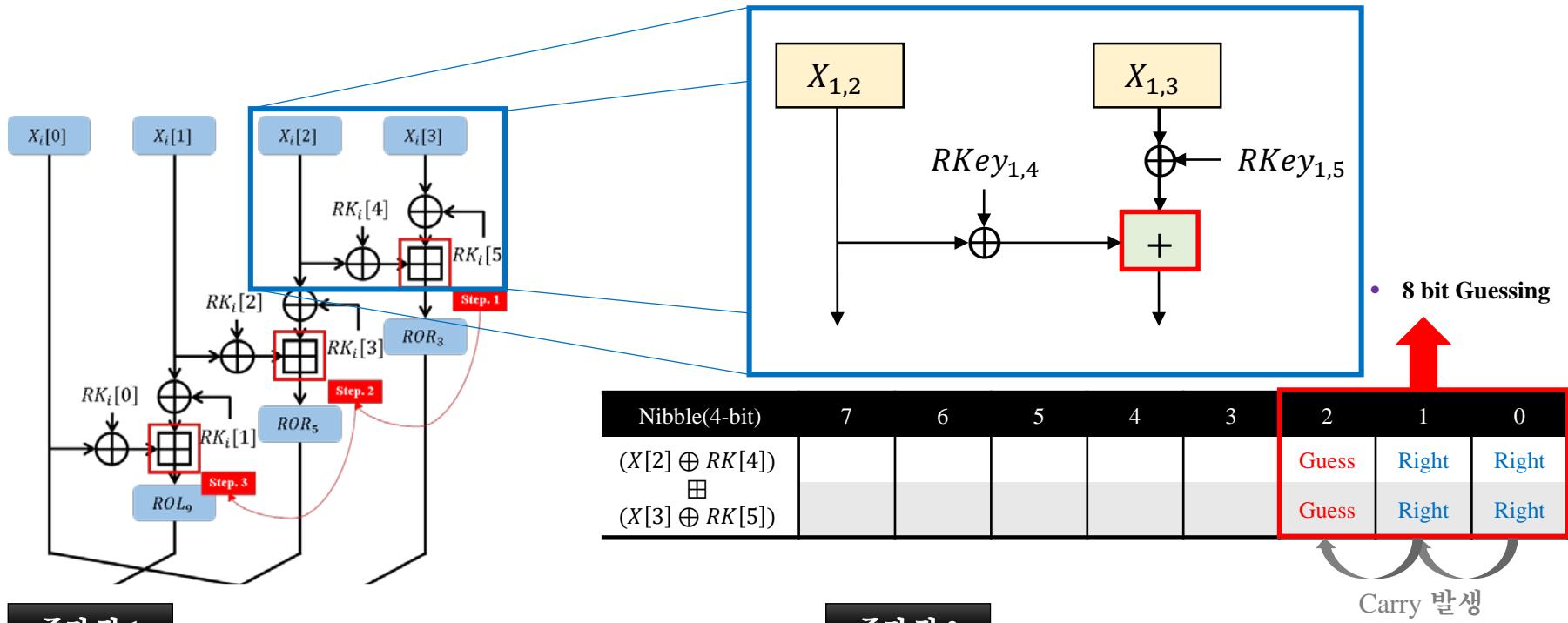
- ❖ [Step. 1] 두 가지 경우의 1번째 Nibble / Maxpeak 분석 결과



- ❖ 이전에 분석된 Nibble을 사용하여 다음 Nibble 분석 할 때 더 높은 상관계수가 나오는 키가 Right Key

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- [Step. 1] 1번째 Addition 연산 / 0번째 Nibble → 7번째 Nibble 덧셈 순으로 분석



중간 값 1

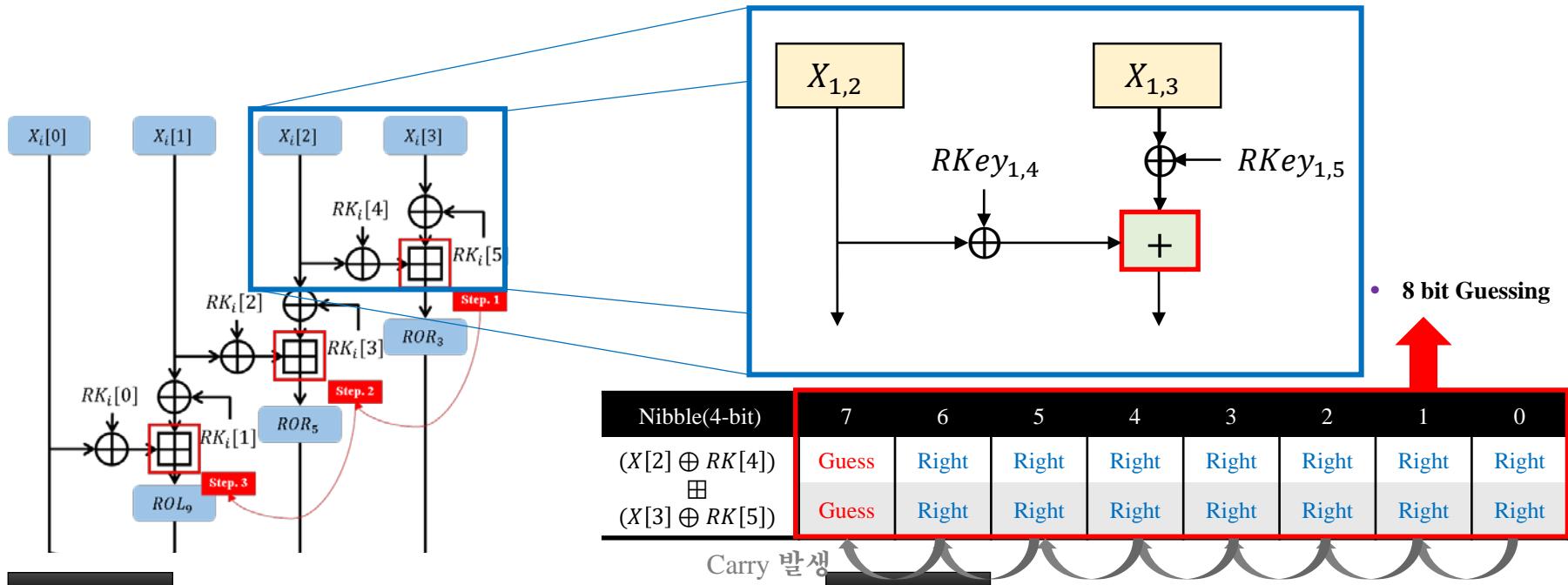
```
// 04_Normal_LEA 1번째 Addition // RK[4] & RK[5] 2번째 Nibble ///////
key = (((plaintext[i][9] & 0xF) ^ ((guess_key >> 4) & 0xF)) 키 후보 1
    + ((plaintext[i][13] & 0xF) ^ ((guess_key >> 0) & 0xF)) & 0xF; //
key <= 8;
key += ((plaintext[i][8] ^ 0xEA) + (plaintext[i][12] ^ 0xE2)); //
///////////////////////////////////////////////////////////////////
```

중간 값 2

```
// 04_Normal_LEA 1번째 Addition // RK[4] & RK[5] 2번째 Nibble ///////
key = (((plaintext[i][9] & 0xF) ^ ((guess_key >> 4) & 0xF)) 키 후보 2
    + ((plaintext[i][13] & 0xF) ^ ((guess_key >> 0) & 0xF)) & 0xF; //
key <= 8;
key += ((plaintext[i][8] ^ 0x6A) + (plaintext[i][12] ^ 0x62)); //
///////////////////////////////////////////////////////////////////
```

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- [Step. 1] 1번째 Addition 연산 / 0번째 Nibble → 7번째 Nibble 덧셈 순으로 분석



중간 값 1

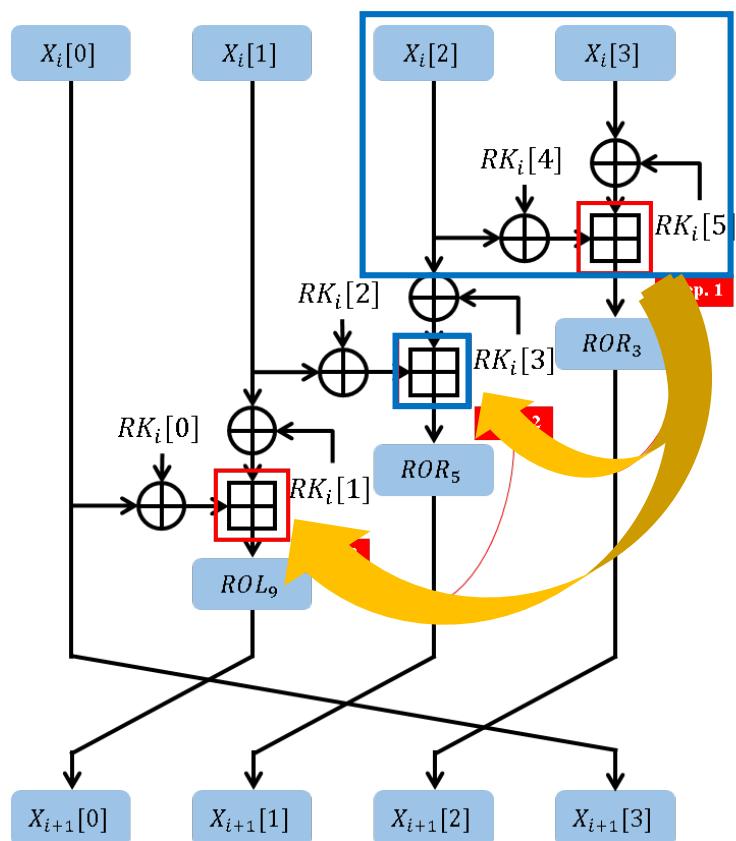
```
// 04_Normal_LEA 1번째 Addition // RK[4] & RK[5] 7번째 Nibble // 키 후보/1 //
key = (((plaintext[i][11] >> 4) ^ ((guess_key >> 4) & 0xF))
    + ((plaintext[i][15] >> 4) ^ ((guess_key >> 0) & 0xF))) & 0xF;
key <= 4;
key += ((plaintext[i][11] & 0xF) ^ 0x8) + ((plaintext[i][15] & 0xF) ^ 0xC);
key <= 8;
key += ((plaintext[i][10] ^ 0xE9) + (plaintext[i][14] ^ 0xFA));
key <= 8;
key += ((plaintext[i][9] ^ 0x3F) + (plaintext[i][13] ^ 0x52));
key <= 8;
key += ((plaintext[i][8] ^ 0xEA) + (plaintext[i][12] ^ 0xE2));
//
```

중간 값 2

```
// 04_Normal_LEA 1번째 Addition // RK[4] & RK[5] 7번째 Nibble // 키 후보/2 //
key = (((plaintext[i][11] >> 4) ^ ((guess_key >> 4) & 0xF))
    + ((plaintext[i][15] >> 4) ^ ((guess_key >> 0) & 0xF))) & 0xF;
key <= 4;
key += ((plaintext[i][11] & 0xF) ^ 0x0) + ((plaintext[i][15] & 0xF) ^ 0x4);
key <= 8;
key += ((plaintext[i][10] ^ 0xE9) + (plaintext[i][14] ^ 0xFA));
key <= 8;
key += ((plaintext[i][9] ^ 0x3F) + (plaintext[i][13] ^ 0x52));
key <= 8;
key += ((plaintext[i][8] ^ 0xEA) + (plaintext[i][12] ^ 0xE2));
//
```

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

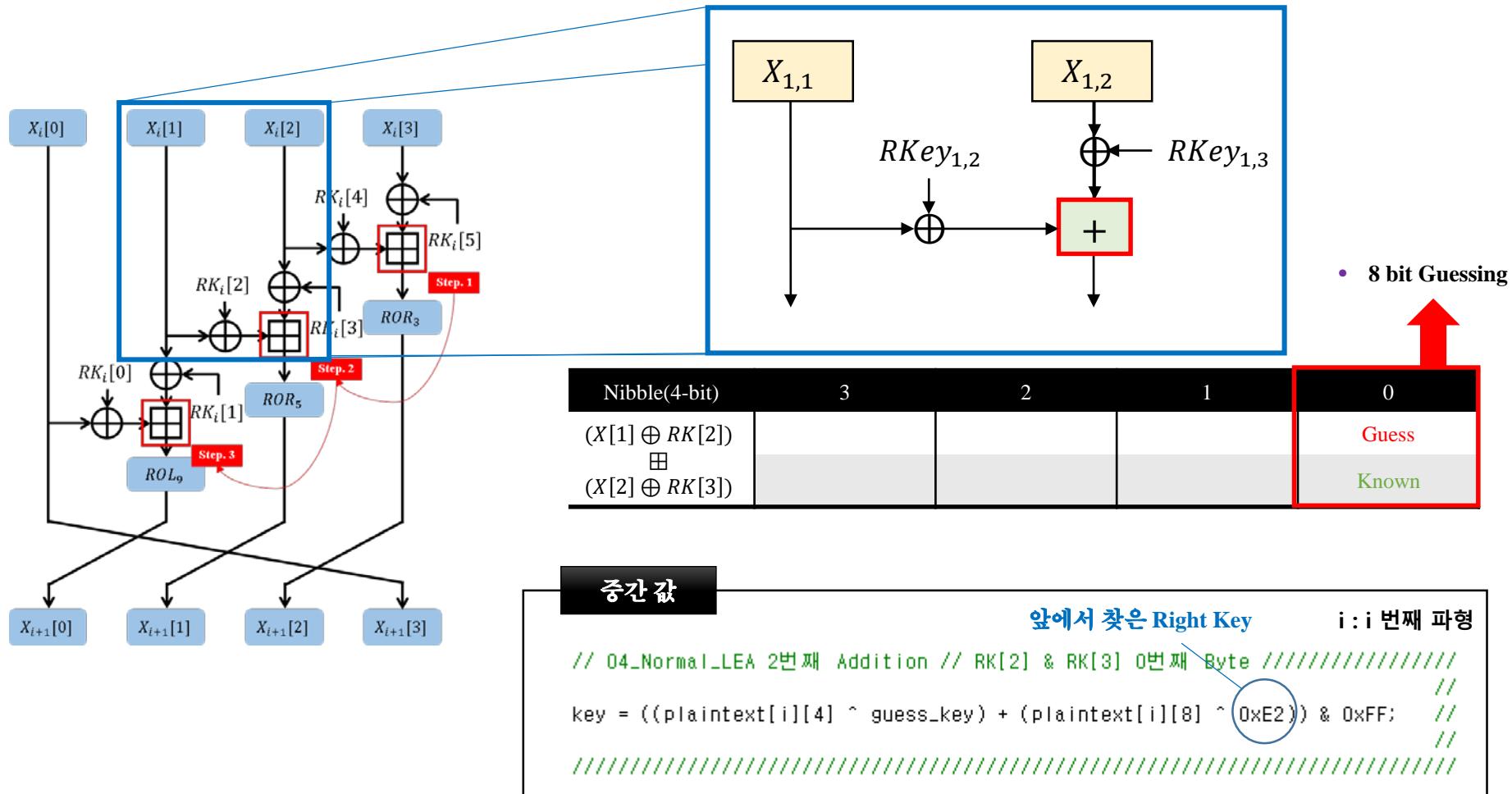
- ❖ [Step. 1] → [Step. 2] / [Step. 1] → [Step. 3] 연결



- ❖ LEA는  $RK[5] = RK[3] = RK[1]$  이므로  
[Step. 1]에서 나온 최종 라운드 키  $RK[5]$  후보 2개를  
[Step. 2]에서 사용하여 실제 라운드 키 찾음
- ❖ [Step. 1]을 통해  $RK[5] = RK[3]$  값을 알기 때문에  
[Step. 2]부터는  $RK[2]$ 에 대한  $2^8$  Guessing
- ❖ 또한  $RK[1] = RK[5]$  이므로  
[Step. 3] 또한  $RK[0]$ 에 대한  $2^8$  Guessing

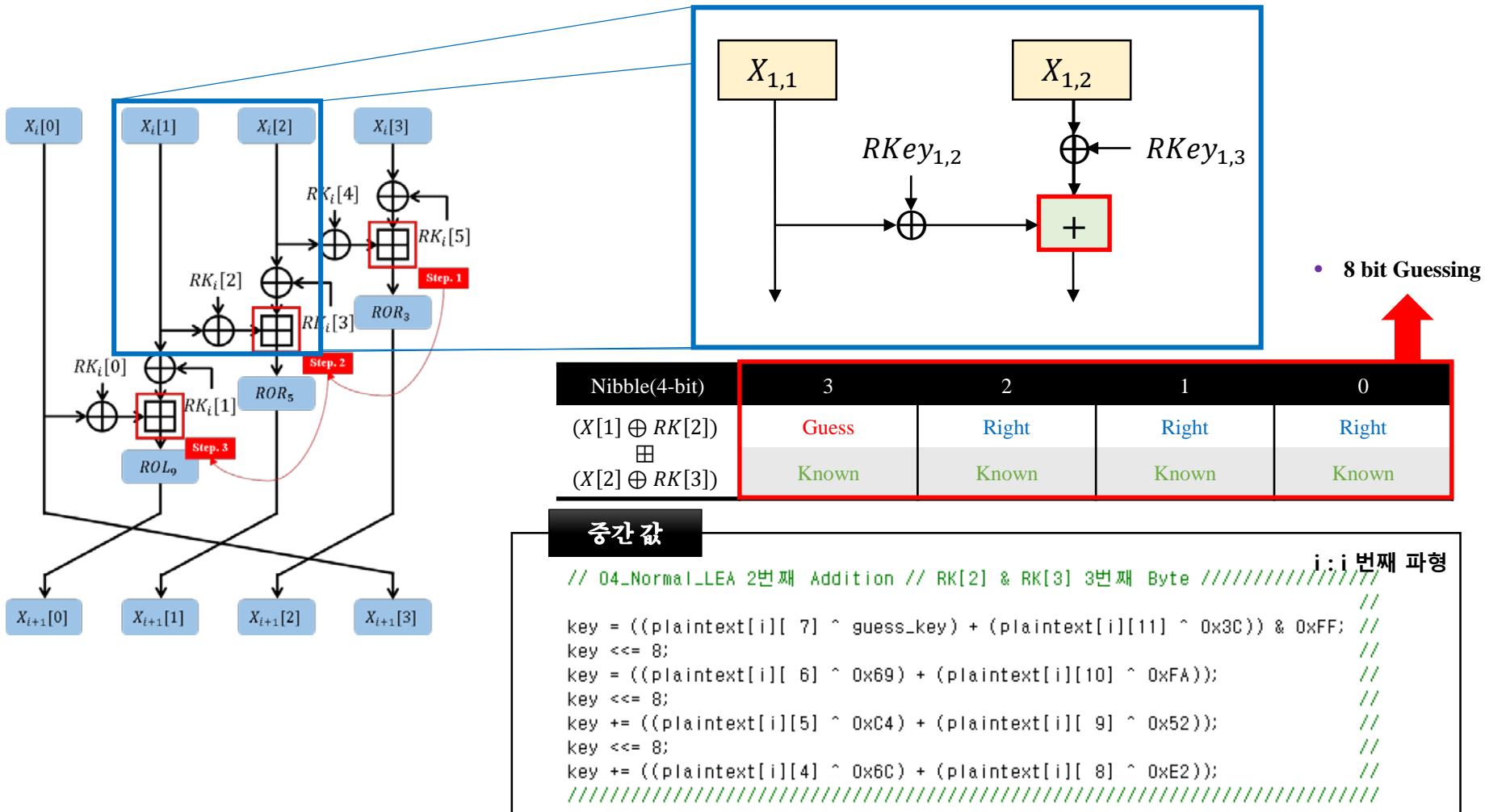
## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- [Step. 2] 2번째 Addition 연산 / 0 Byte → 3 번째 Byte 덧셈 순으로 분석



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

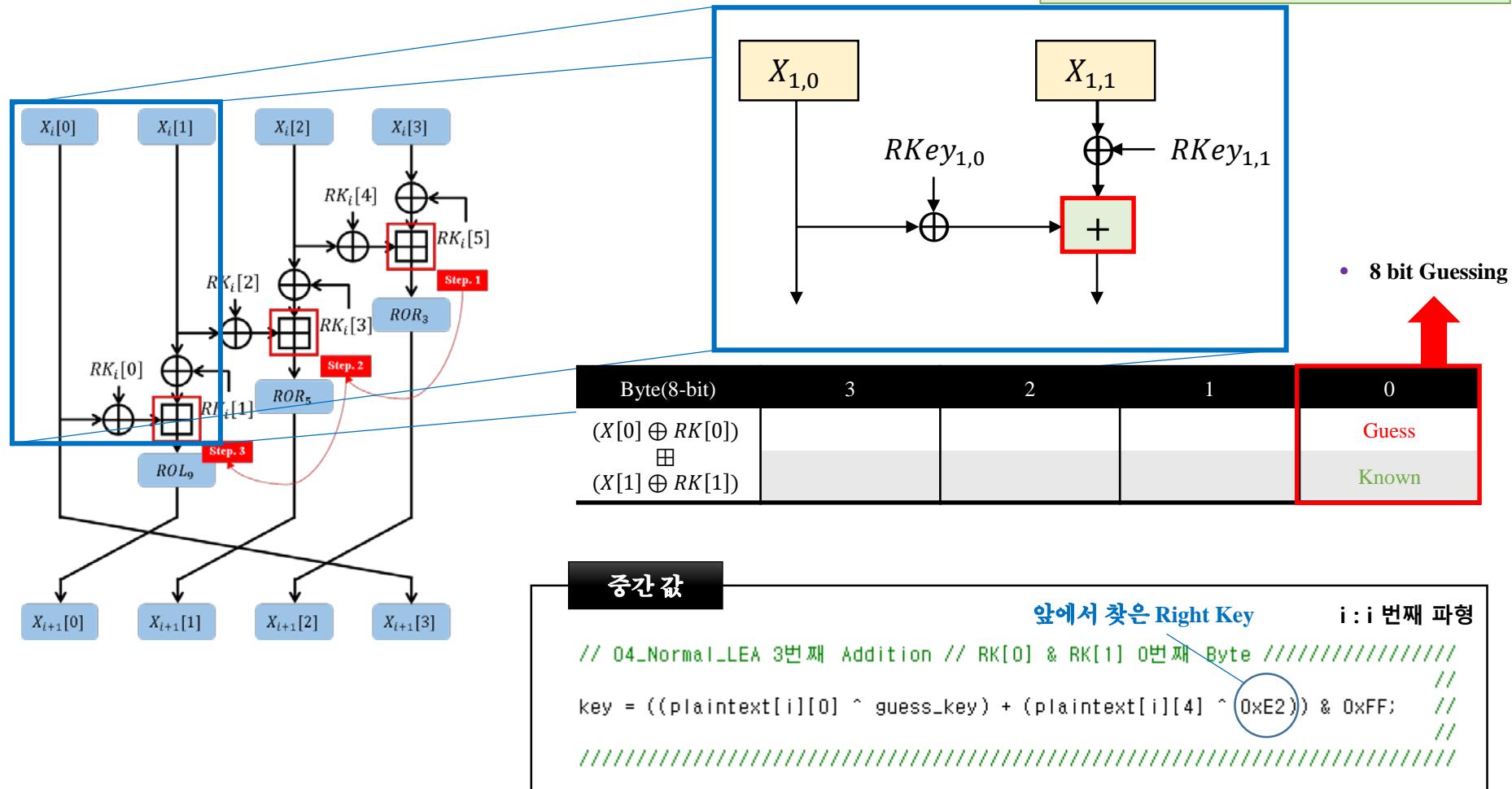
- ❖ [Step. 2] 2번째 Addition 연산 / 0 Byte → 3 번째 Byte 덧셈 순으로 분석



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- [Step. 3] 3번째 Addition 연산 / 0 Byte → 3 번째 Byte 덧셈 순으로 분석

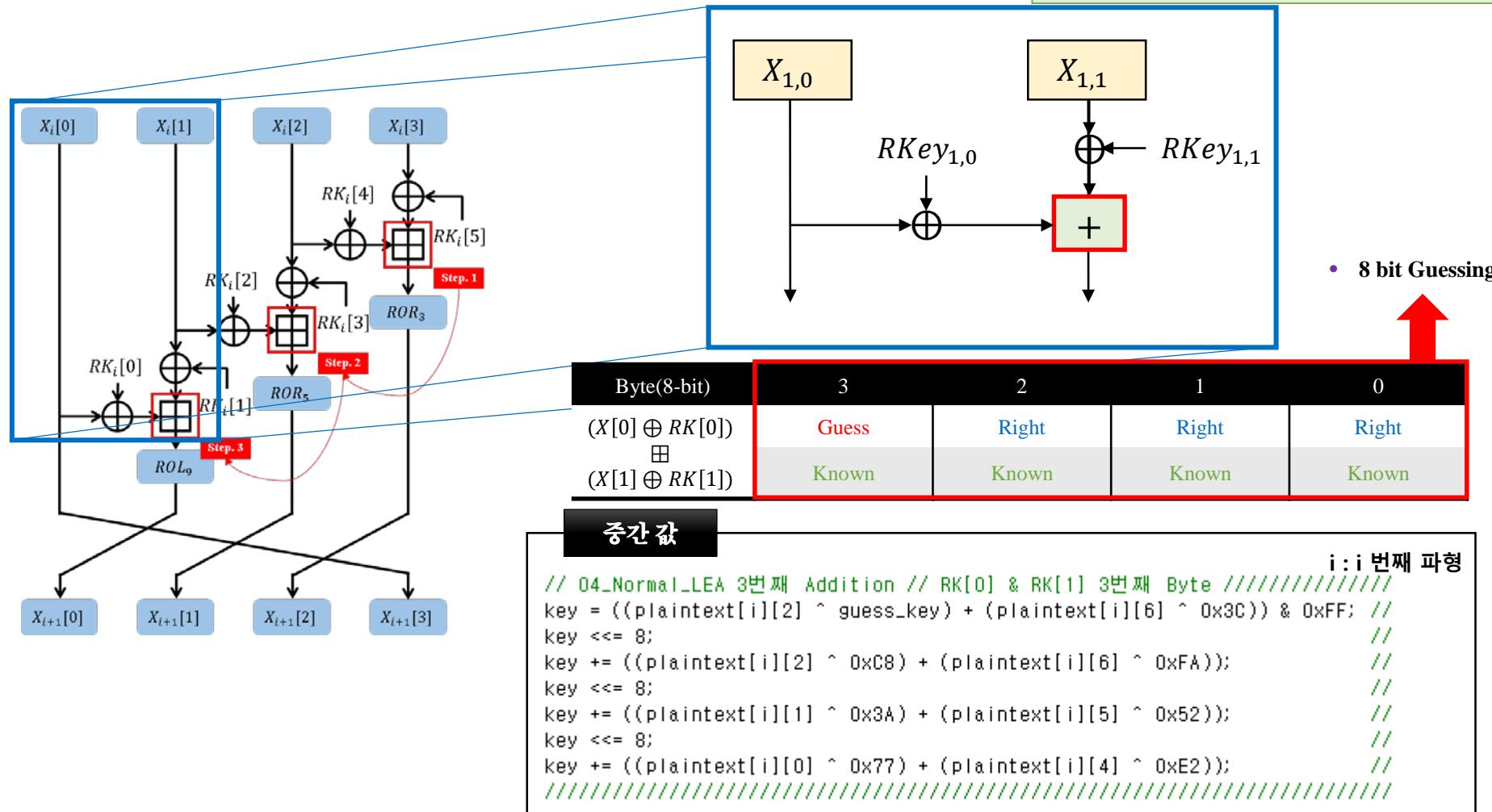
[Step. 2] 와 동일한 방법으로 진행



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- [Step. 3] 3번째 Addition 연산 / 0 Byte → 3 번째 Byte 덧셈 순으로 분석

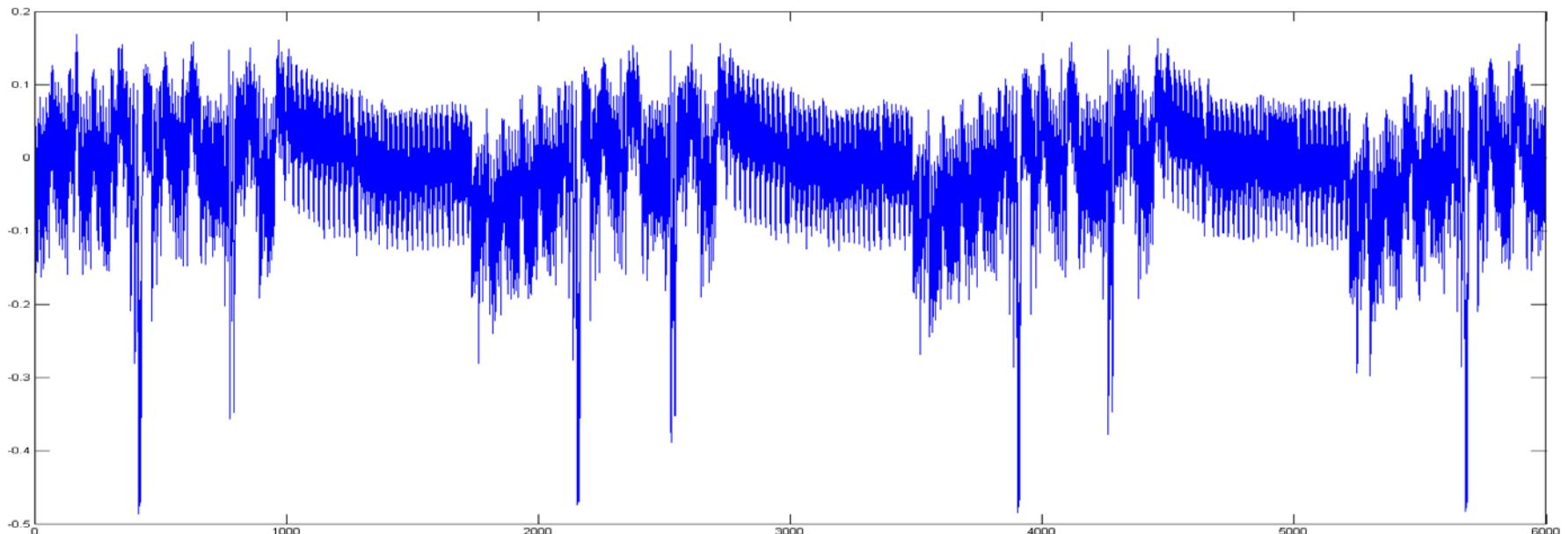
[Step. 2] 와 동일한 방법으로 진행



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

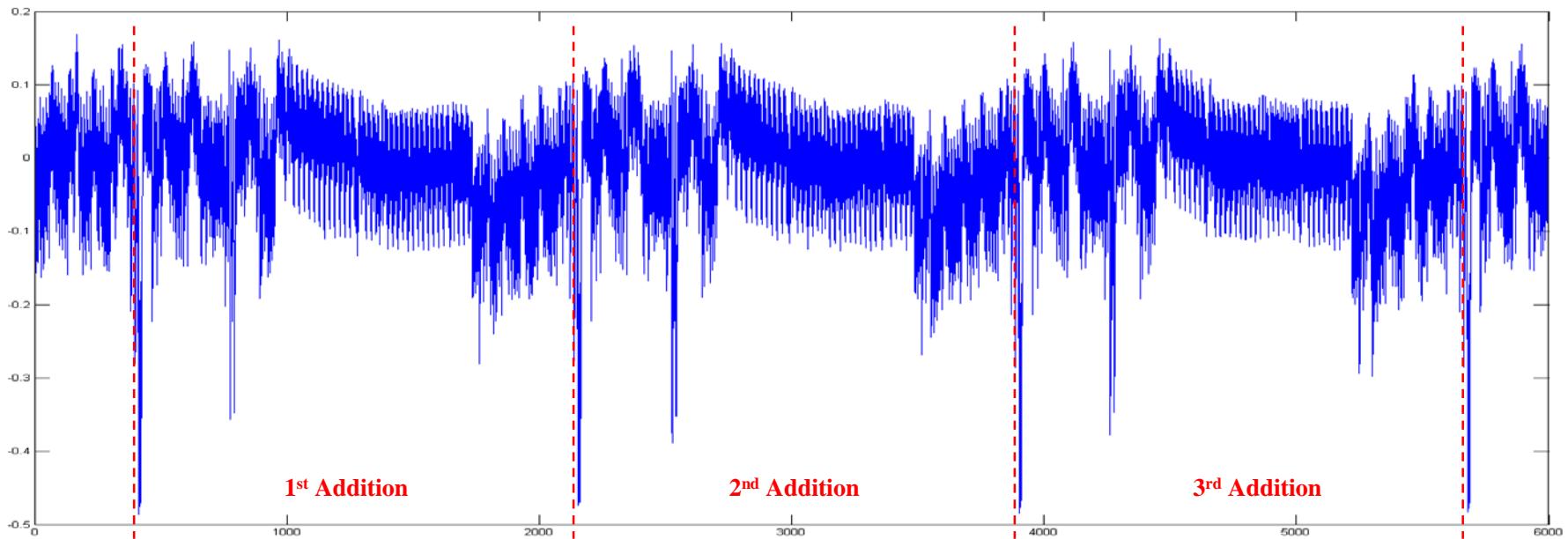
파형 정보	
파형 수	5,000
포인트 수	6,000



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	5,000
포인트 수	6,000

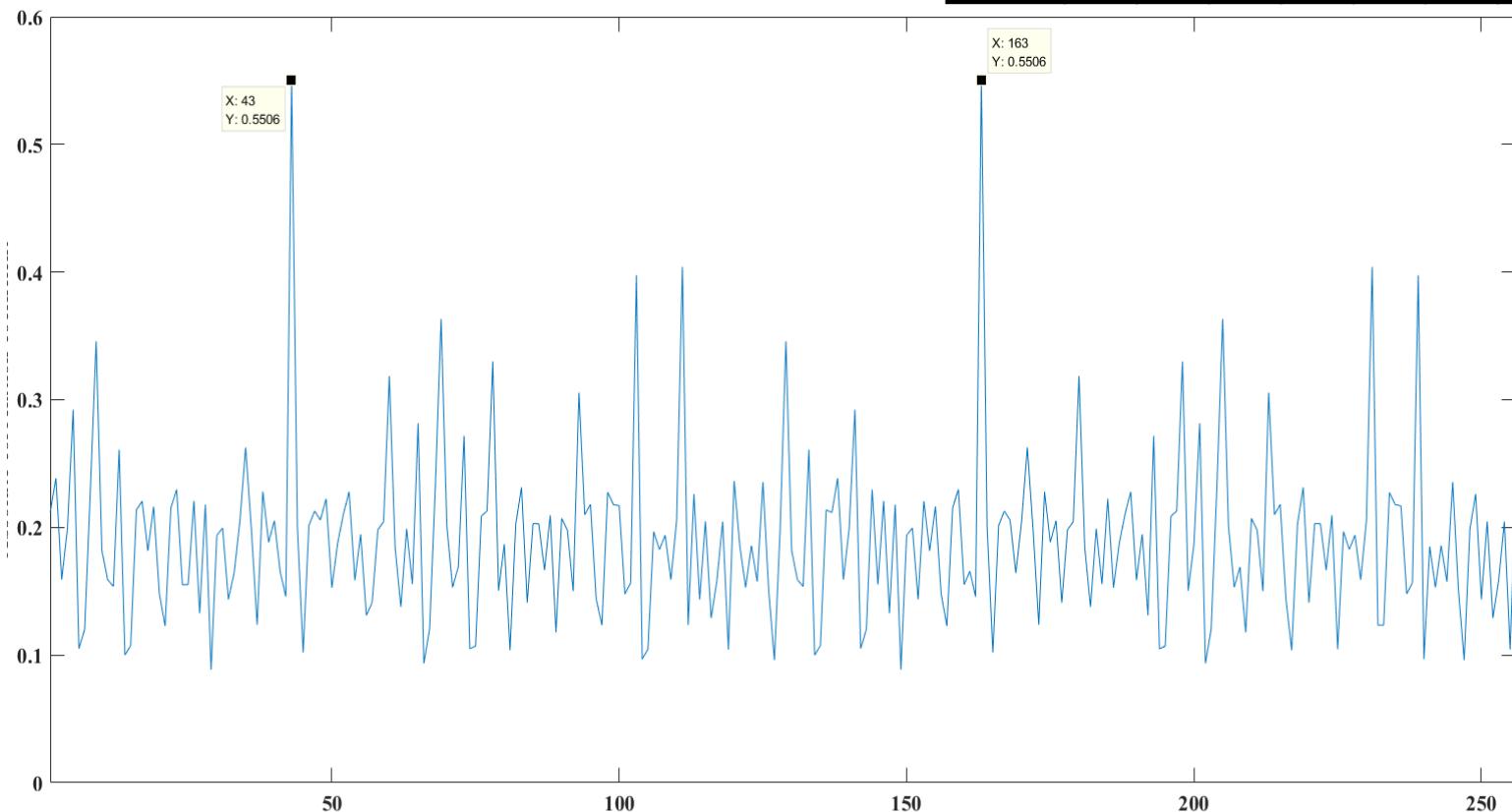


- ❖ 파형 SPA와 문제에서 제공된 알고리즘으로 보아 동일한 3개의 부분을 1라운드 덧셈 연산으로 추측

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

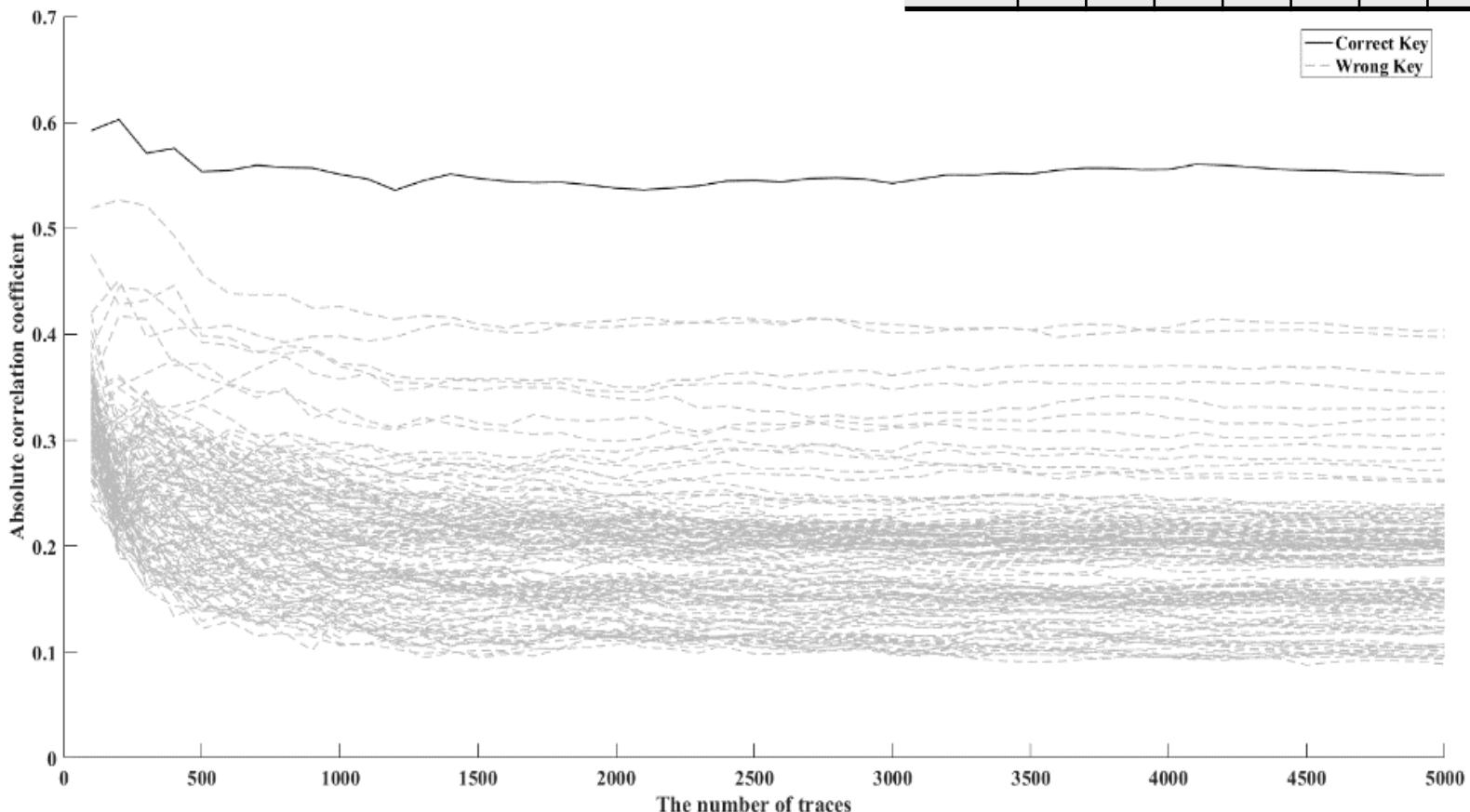
### ❖ [Step. 1] 0번째 Nibble / Maxpeak 분석 결과

Nibble	7	6	5	4	3	2	1	0
RK[4]	C	8	E	9	3	F	E	A
RK[5]	3	C	F	A	5	2	E	2



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

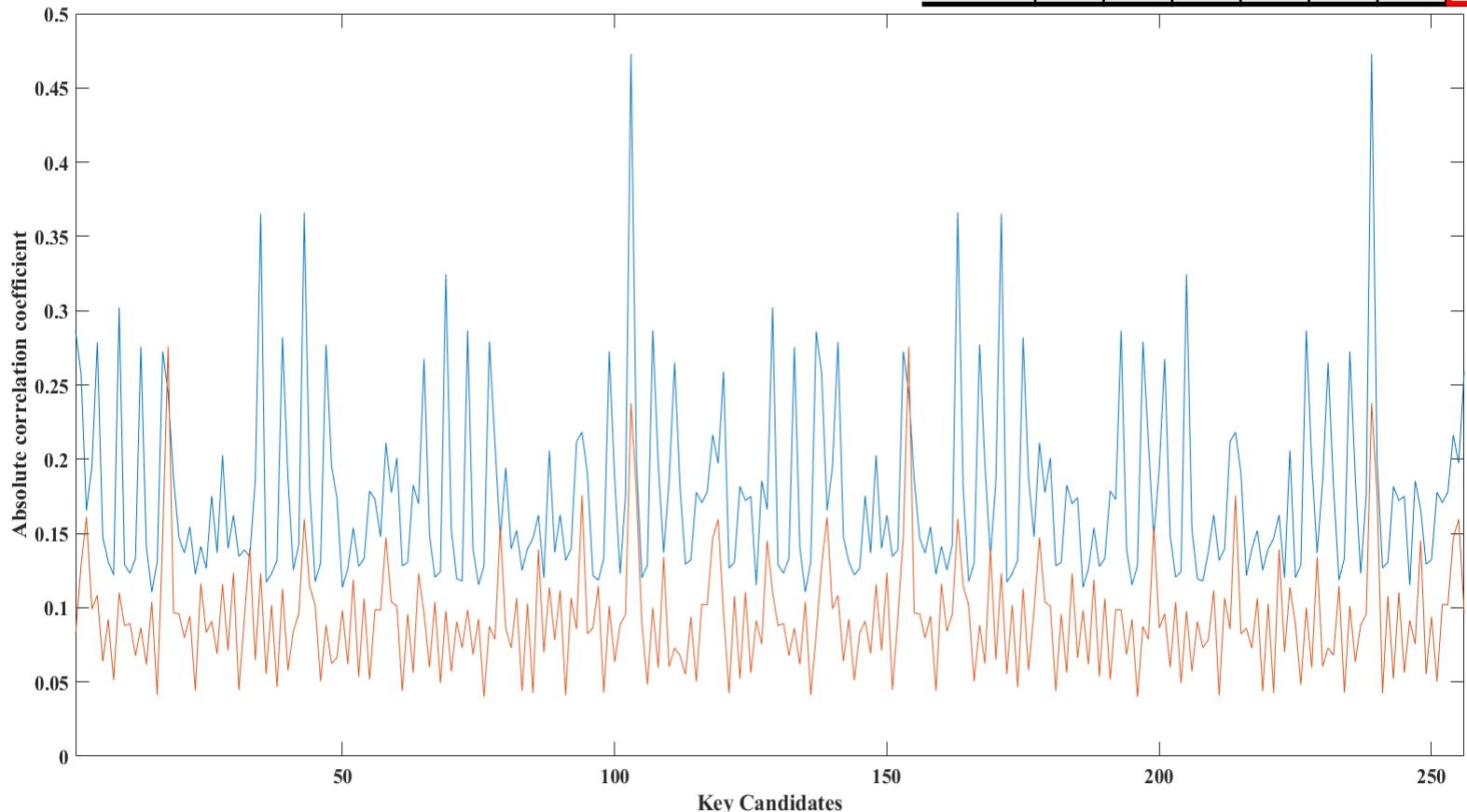
- ❖ [Step. 1] 0번째 Nibble / 파형 증가에 따른 분석 결과



Nibble	7	6	5	4	3	2	1	0
RK[4]	C	8	E	9	3	F	E	A
RK[5]	3	C	F	A	5	2	E	2

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

### ❖ [Step. 1] 1번째 Nibble / Maxpeak 분석 결과

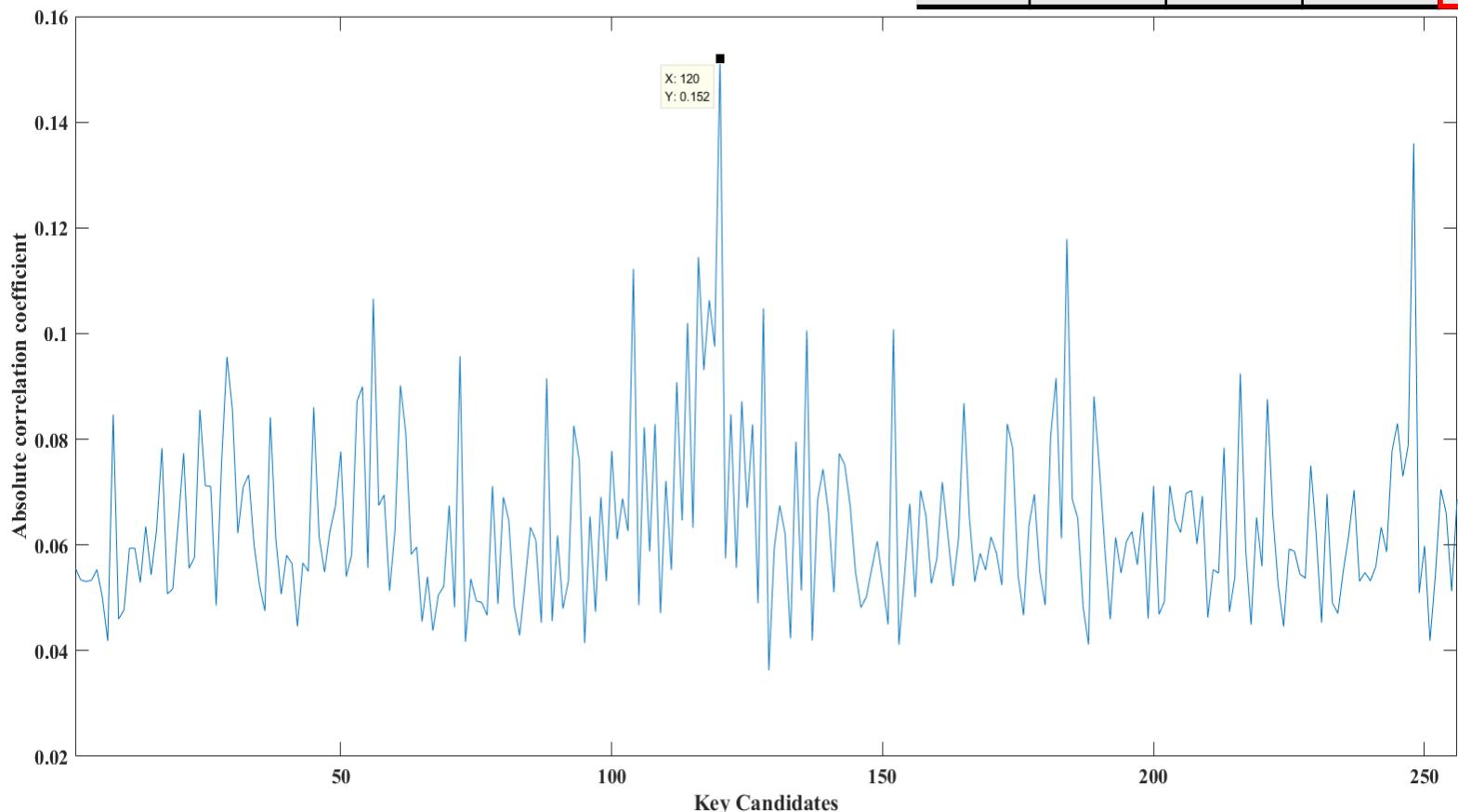


Nibble	7	6	5	4	3	2	1	0
RK[4]	C	8	E	9	3	F	E	A
RK[5]	3	C	F	A	5	2	E	2

## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

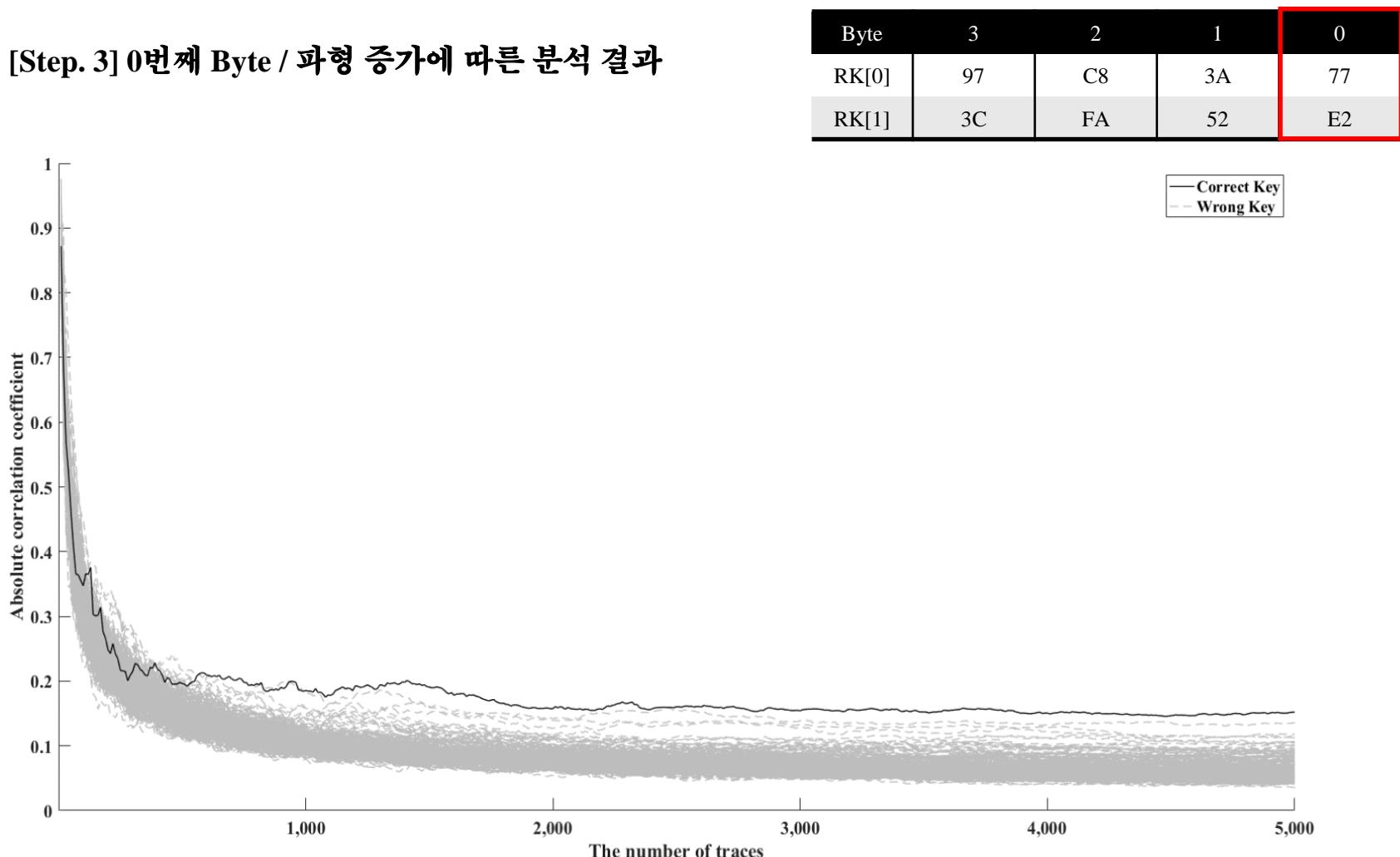
### ❖ [Step. 3] 0번째 Byte / Maxpeak 분석 결과

Byte	3	2	1	0
RK[0]	97	C8	3A	77
RK[1]	3C	FA	52	E2



## ▣ Normal LEA / ChipWhisperer-Lite / 16 bit / 분석 결과

- ❖ [Step. 3] 0번째 Byte / 파형 증가에 따른 분석 결과



## ▣ 02\_Normal ARIA / ARM902T / 8 bit / 분석 결과

### ❖ (2) First Order CPA 결과

- Ratio 1.0 기준
- RK[1] = RK[3] = RK[5]

Byte	3	2	1	0
RK[0]	97	C8	3A	77
RK[1]	3C	FA	52	E2

최소분석파형개수	600	310	2100	1000
Byte	3	2	1	0

RK[2]	5D	69	C4	6C
RK[3]	3C	FA	52	E2
최소분석파형 수	500	230	110	540
Byte	3	2	1	0

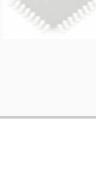
Nibble	07	06	05	04	03	02	01	00
RK[4]	C	8	E	9	3	F	E	A
RK[5]	3	C	F	A	5	2	E	2
최소분석파형 수	60	80	240	370	30	40	100	60

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 05

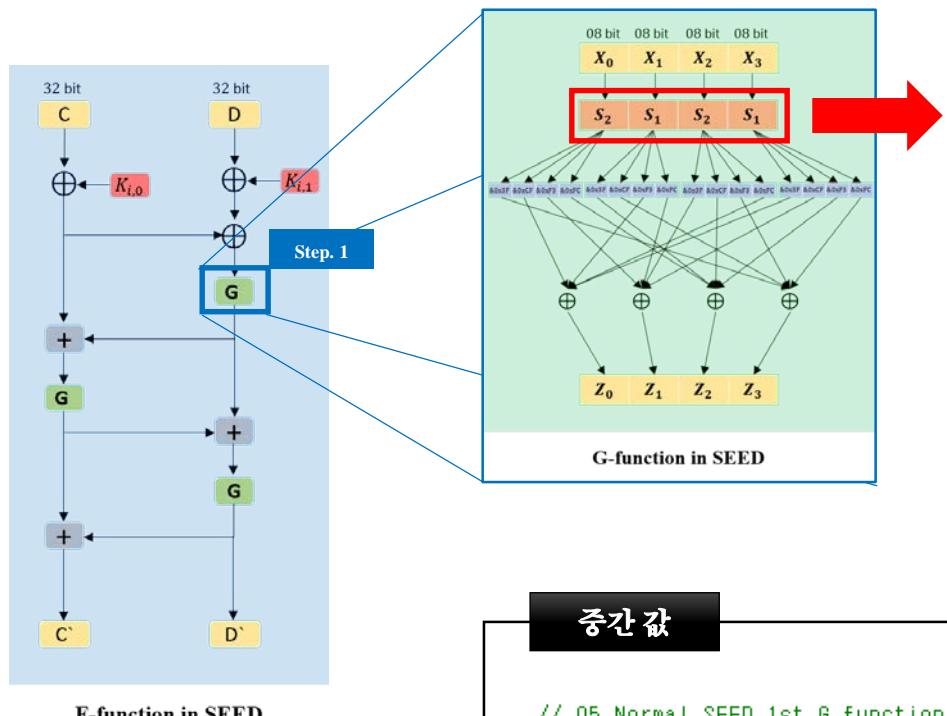
제공되는 정보는 8비트 기반 소프트웨어로 구현된 **SEED-128 암호** 알고리즘이 ChipWhisperer-Lite에서 동작 할 때 소비되는 전력을 **1라운드 부분**을 타겟으로 수집한 결과이다.

binary 파일 05\_Normal\_SEED\_10000tr\_3792pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 SEED 알고리즘이 10,000번 시행된 소비 전력이 포함되어 있는 파일이다.

05\_Normal\_SEED\_10000tr\_3792pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 소비 전력 및 평문 정보를 이용하여 SEED 암호 알고리즘 동작 시 사용된 1라운드키 상위 32 비트와 하위 32 비트의 exclusive OR 값( $RK - K1^0 \oplus RK - K0^0$ )을 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)
- 2) 소비 전력 및 평문 정보를 이용하여 SEED 암호 알고리즘 동작 시 사용된 암호화키의 1라운드키( $RK^0$ )를 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)
- 3) 주어진 소비 전력에 대하여 아래의 pseudocode를 활용하여 SEED 암호 알고리즘의 연산위치를 최대한 자세히 추측하여 나타내시오.

## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리



### ❖ Step. 1 1 라운드 XOR Key 분석

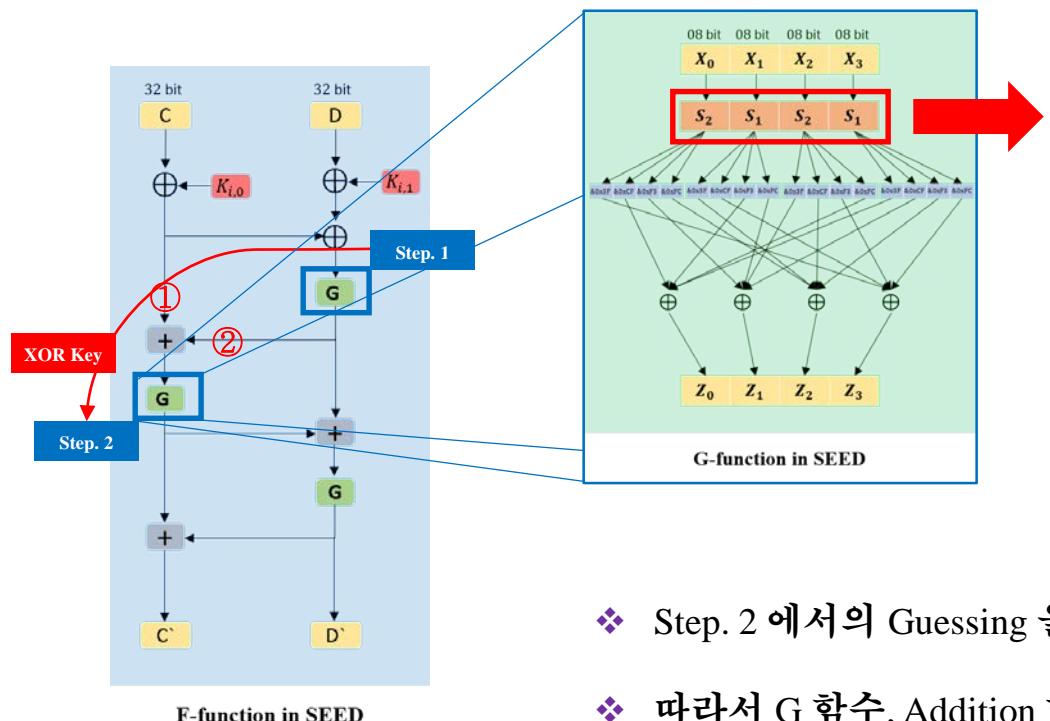
- 1라운드 1번째 G함수에서 4비트 XOR Key ( $K_{0,0} \oplus K_{1,0}$ )를 획득
- S-Box 순서 상관 없이 Byte 단위 공격 가능

중간 값

```
// 05_Normal_SEED 1st G function 중간값 /////////////////////////////////
// key = _SEED_SBOX_[t + 1] % 2[ plaintext[i][t] ^ plaintext[i][t + 4] ^ guess_key ];
// /////////////////////////////////
```

t : S-box 위치  
i : i 번째 파형

## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리



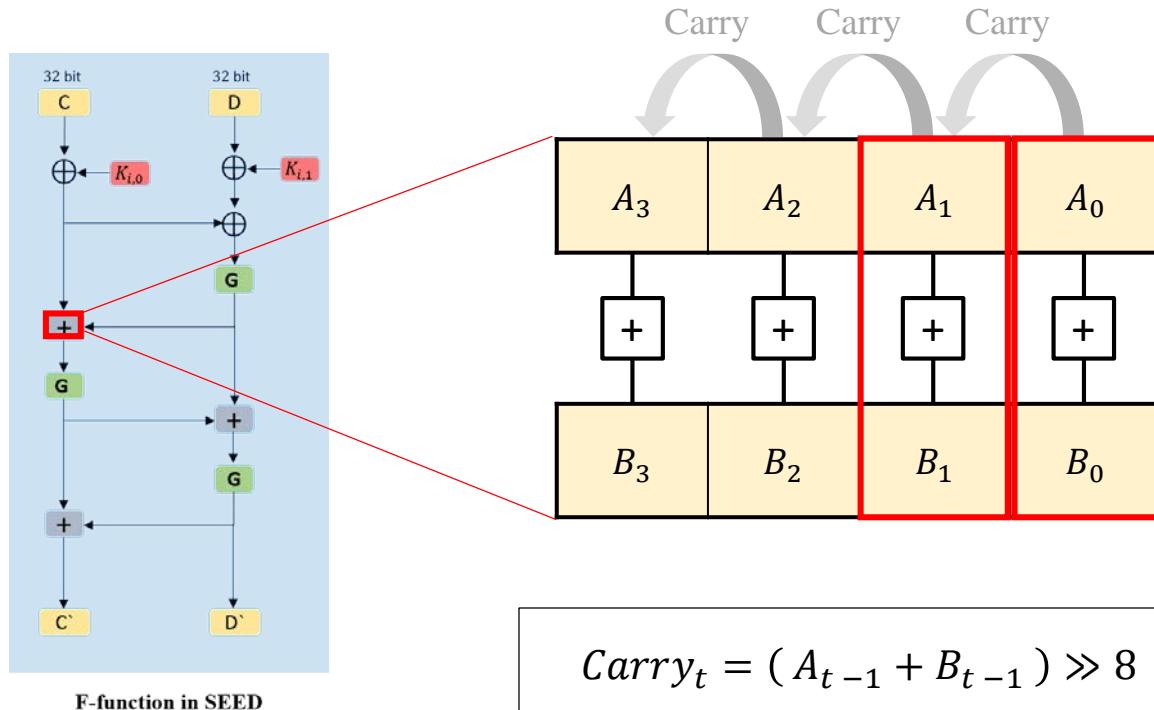
### ❖ Step. 2 1 라운드 Left RoundKey 분석

- 1라운드 1번 째 G함수에서 4바이트 Left RoundKey ( $K_{0,0}$ )를 획득
- 최하위 Byte ( $S_1(X_3)$ ) 부터 Byte 단위 공격 (Carry 고려)

- ❖ Step. 2에서의 Guessing을 위해서는 ①, ② 입력 값 필요
- ❖ 따라서 G 함수, Addition 함수 구현 필요
- ❖ 또한 Addition 연산에서의 Byte 단위의 Carry 고려

## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

### ❖ Addition Carry



- ❖ 32-bit 덧셈 연산을 8-bit 씩 수행할 경우 상위 8-bit에 대해 Carry가 발생함
- ❖ 따라서 최하위  $A_0 + B_0$  부터 연산한 후 Carry를 고려해서  
 $Carry(A_0 + B_0) + A_1 + B_1$  연산 수행

## □ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

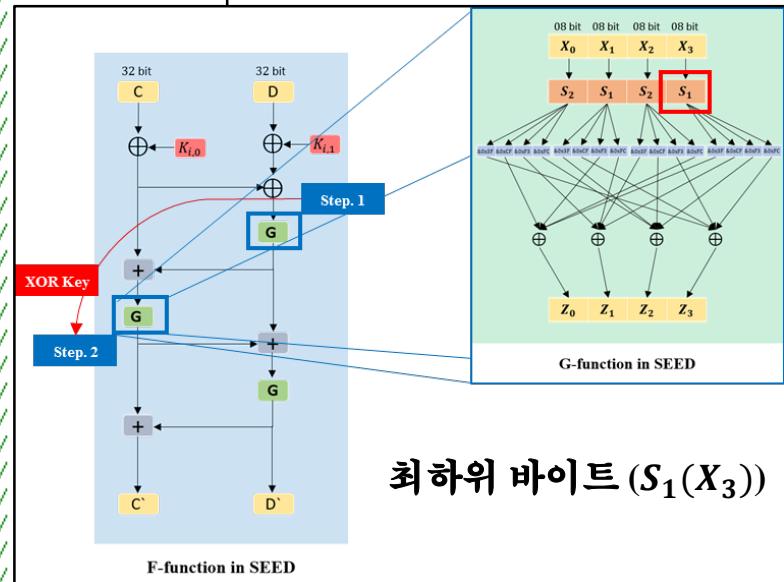
중간 값

```
// 05_Normal_SEED 2nd G function 중간값 //////////////////////////////////////////////////////////////////
// 1 Round XOR Key (4 Byte) -> XOR_KEY[0] || XOR_KEY[1] || XOR_KEY[2] || XOR_KEY[3] //
XOR_KEY[0] = 0x0b;
XOR_KEY[1] = 0x9e;
XOR_KEY[2] = 0xd9;
XOR_KEY[3] = 0x46;

// Plaintext(L) ^ Plaintext(R) ^ Roundkey(L) ^ Roundkey(R)
IN_R[0] = plaintext[i][8] ^ plaintext[i][12] ^ XOR_KEY[0];
IN_R[1] = plaintext[i][9] ^ plaintext[i][13] ^ XOR_KEY[1];
IN_R[2] = plaintext[i][10] ^ plaintext[i][14] ^ XOR_KEY[2];
IN_R[3] = plaintext[i][11] ^ plaintext[i][15] ^ XOR_KEY[3];

// 1 Round G Function Output
SEED_GFunction(IN_R)           • 1 Round XOR Key
                                • XOR 연산
                                • 1st G 함수 연산
                                • 1 Round Left RoundKey Guessing
                                • Left Round Key XOR 연산
                                • 1st Addition 함수 연산 (Carry)
                                • 2nd S-Box 최하위 Byte 중간 값
```

t : S-box 위치  
i : i 번째 파형



## □ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

### 중간 값

```
// 05_Normal_SEED 2nd G function 중간값 //////////////////////////////////////////////////////////////////
// 1 Round XOR Key (4 Byte) -> XOR_KEY[0] || XOR_KEY[1] || XOR_KEY[2] || XOR_KEY[3] //
XOR_KEY[0] = 0x0b;
XOR_KEY[1] = 0x9e;
XOR_KEY[2] = 0xd9;
XOR_KEY[3] = 0x46;

// Plaintext(L) ^ Plaintext(R) ^ Roundkey(L) ^ Roundkey(R)
IN_R[0] = plaintext[i][ 8 ] ^ plaintext[i][12] ^ XOR_KEY[0];
IN_R[1] = plaintext[i][ 9 ] ^ plaintext[i][13] ^ XOR_KEY[1];
IN_R[2] = plaintext[i][10] ^ plaintext[i][14] ^ XOR_KEY[2];
IN_R[3] = plaintext[i][11] ^ plaintext[i][15] ^ XOR_KEY[3];

// 1 Round G Function Output
SEED_GFunction(IN_R)

// Key Guessing
RK_L[0] = guess_key & 0xFF;
RK_L[1] = 0xec;
RK_L[2] = 0xee;
RK_L[3] = 0xf9;

// Plaintext ^ Roundkey(L)
IN_L[0] = plaintext[i][ 8 ] ^ RK_L[0];
IN_L[1] = plaintext[i][ 9 ] ^ RK_L[1];
IN_L[2] = plaintext[i][10] ^ RK_L[2];
IN_L[3] = plaintext[i][11] ^ RK_L[3];

// 1 Round 1st Add
SEED_ADD32_8(IN_L, IN_R, ADD_OUT);

// 1 Round 2nd G Function 중간값
key = _SEED_SBOX_[1][ADD_OUT[0]];
//key = _SEED_SBOX_[0][ADD_OUT[1]];
//key = _SEED_SBOX_[1][ADD_OUT[2]];
//key = _SEED_SBOX_[0][ADD_OUT[3]];

////////////////////////////////////////////////////////////////
```



- 1 Round XOR Key



t : S-box 위치  
i : i 번째 파형

- XOR 연산



- 1<sup>st</sup> G 함수 연산



- 1 Round Left RoundKey Guessing
- 앞서 찾은 Left RoundKey 입력 (Carry)



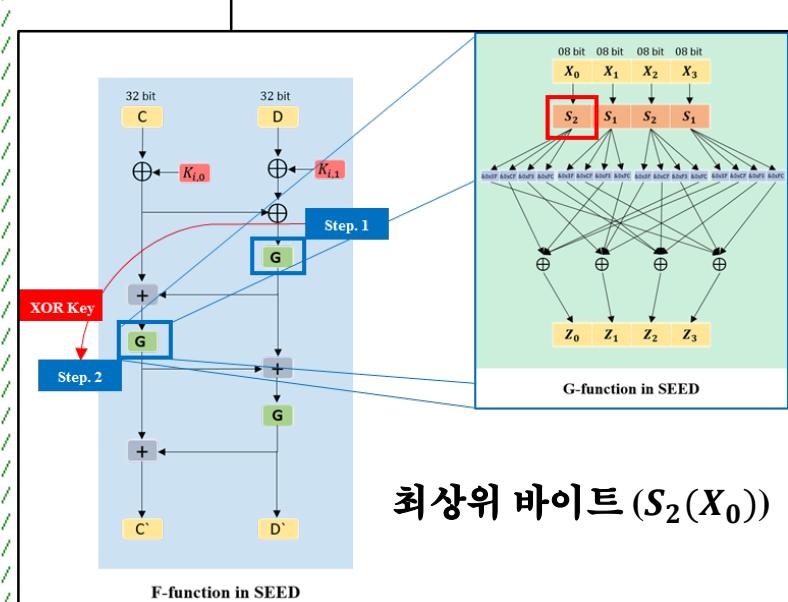
- Left Round Key XOR 연산



- 1<sup>st</sup> Addition 함수 연산 (Carry)



- 2<sup>nd</sup> S-Box 최상위 Byte 중간 값

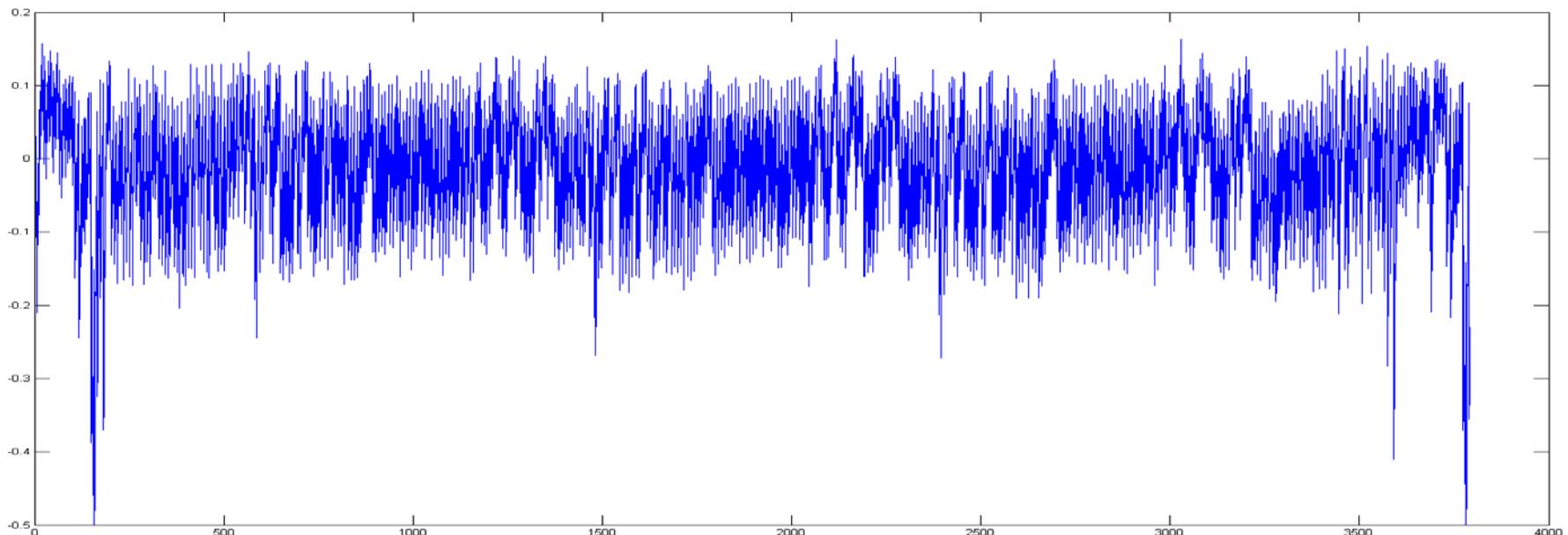


최상위 바이트 ( $S_2(X_0)$ )

## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

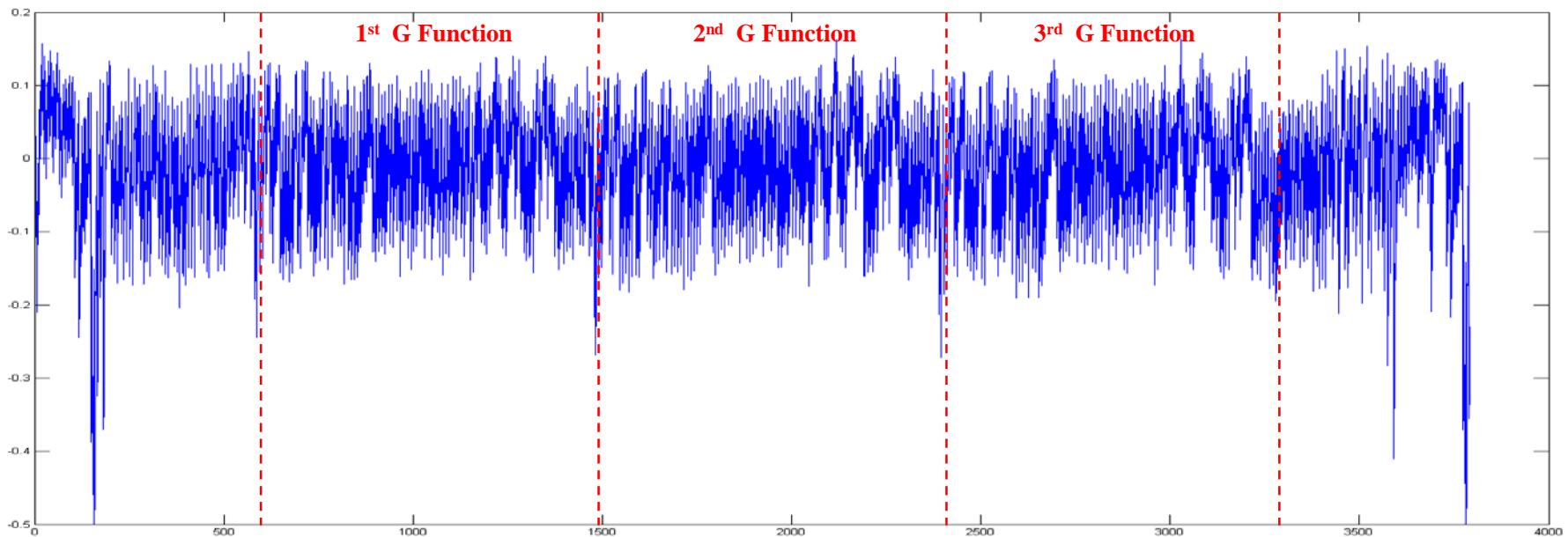
파형 정보	
파형 수	10,000
포인트 수	3,792



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	10,000
포인트 수	3,792

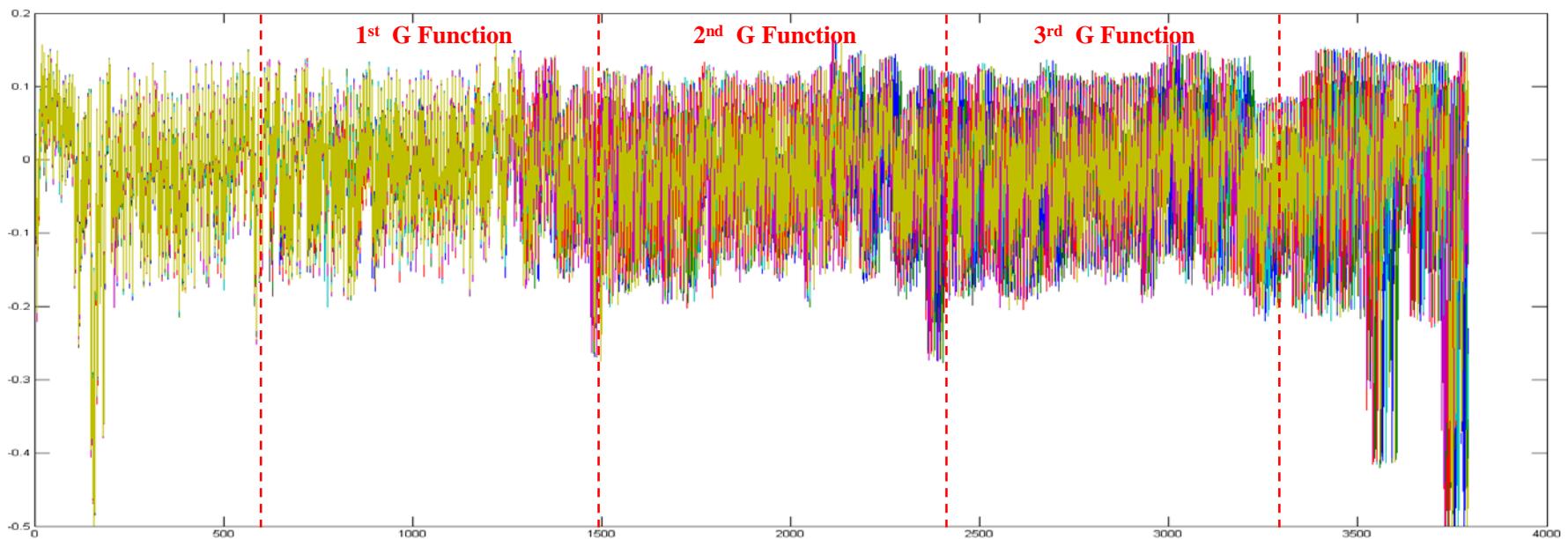


- ❖ SPA 를 통한 G function (3개) 구분가능

## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

- ❖ (1) 전력 파형 SPA (Original / overlap traces)

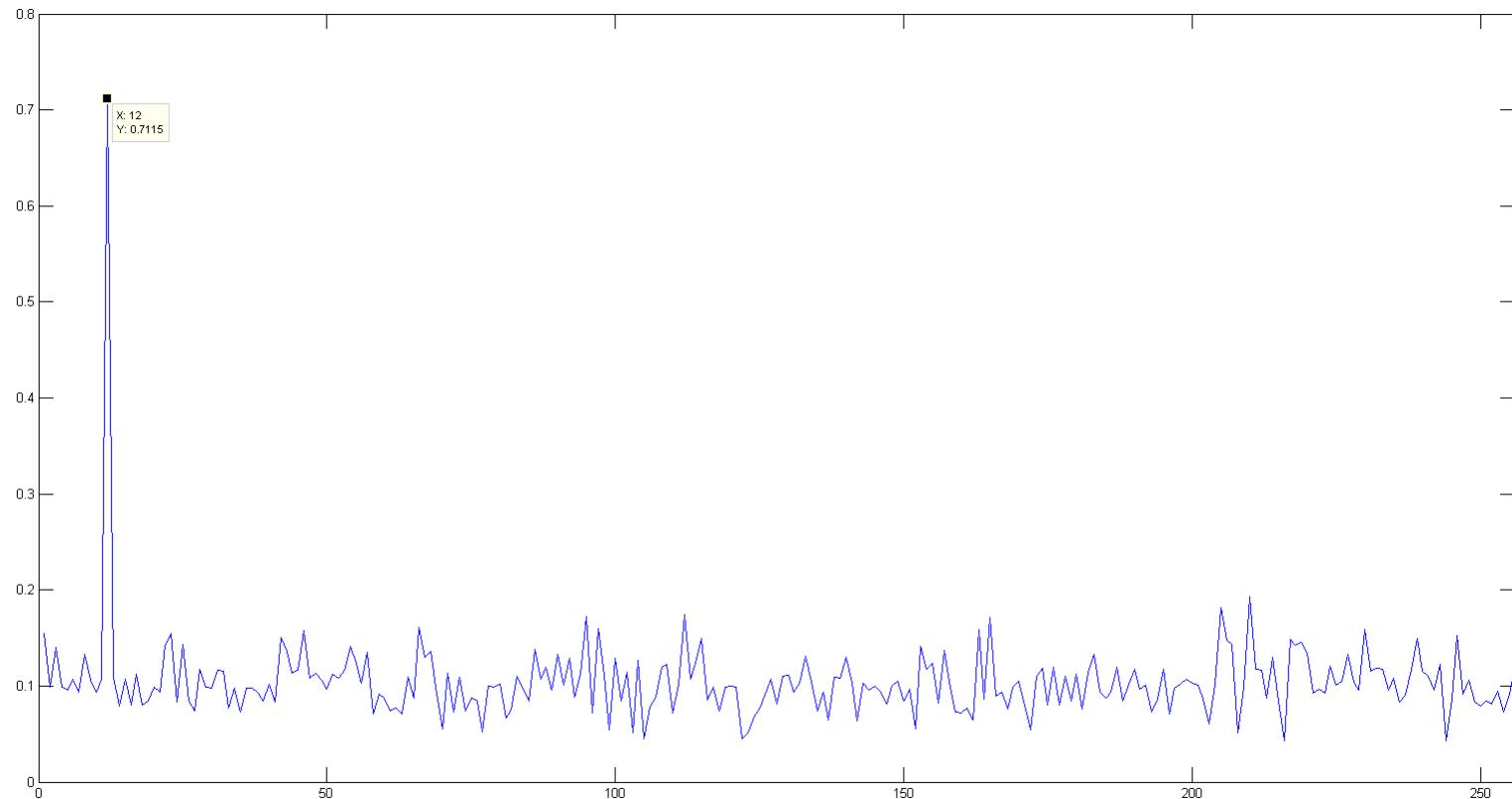
파형 정보	
파형 수	10,000
포인트 수	3,792



- ❖ 앞쪽엔 정렬이 맞지만 두 번째 G 함수부분은 정렬이 맞지 않음
- ❖ 첫 번째 G 함수 → XOR 라운드 키(4-Byte) 분석

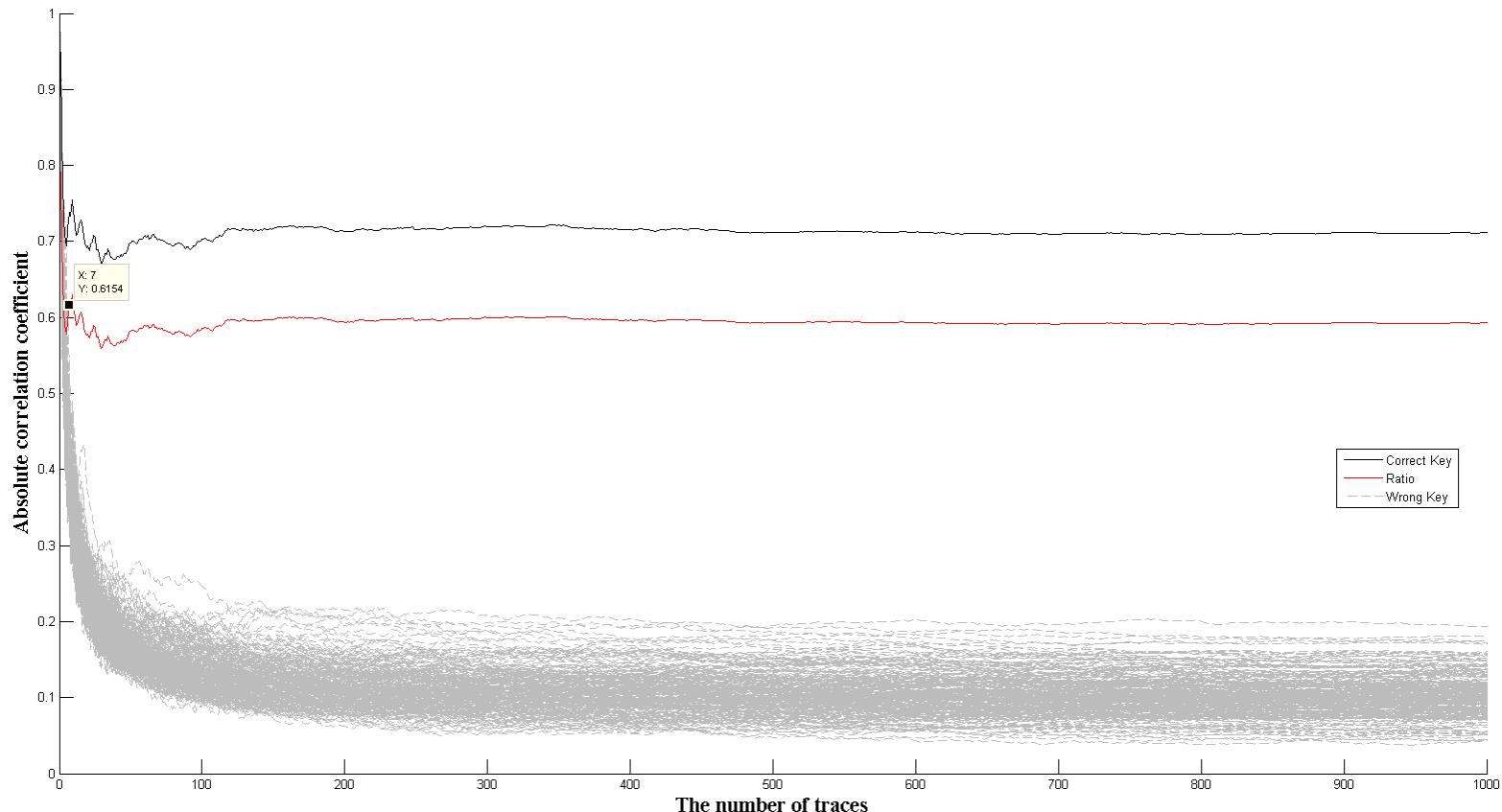
## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (1) 첫 번째 G Function 의 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

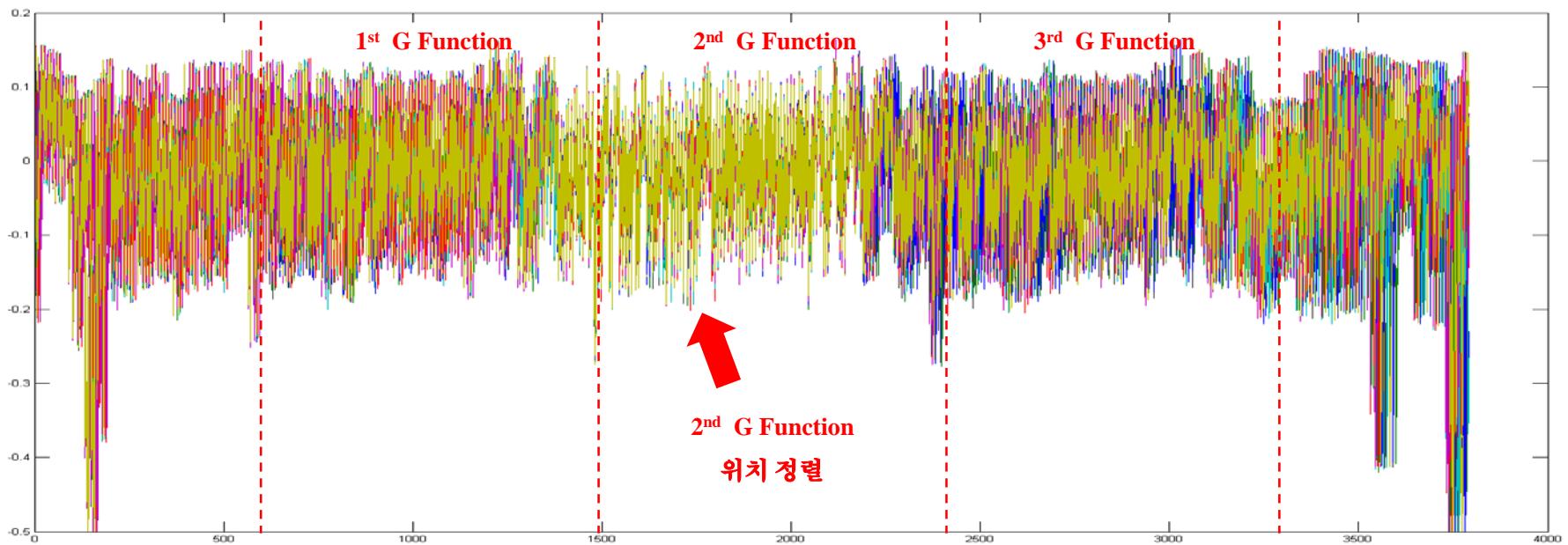
- ❖ (1) 첫 번째 G Function 의 1<sup>st</sup> Byte 파형 증가에 따른 분석 결과



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

- ❖ (2) 전력 파형 SPA (Original / overlap traces)

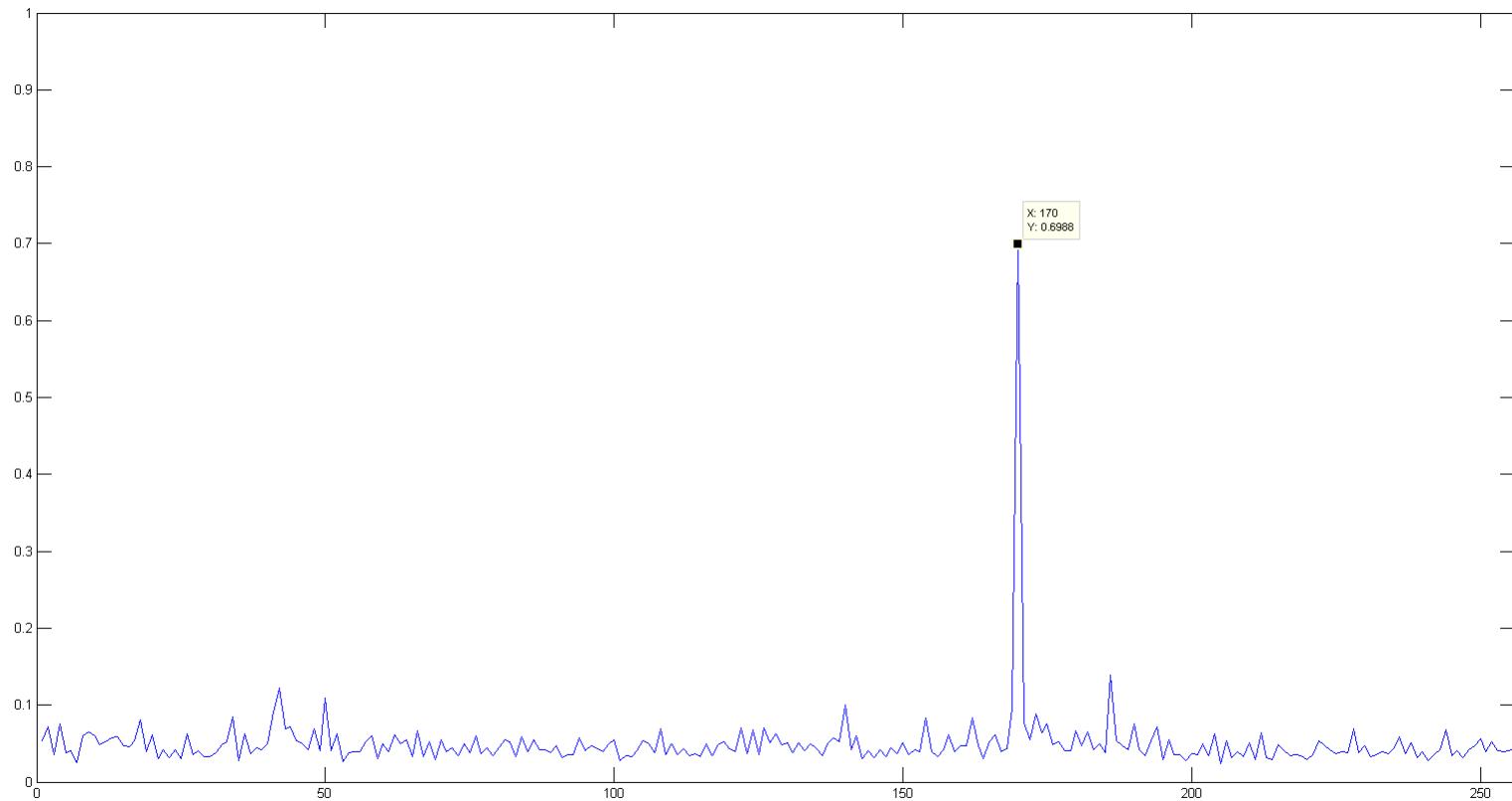
파형 정보	
파형 수	10,000
포인트 수	3,792



- ❖ 두 번째 G 함수부분을 기준으로 정렬 수행
- ❖ 두 번째 G 함수 → 왼쪽 라운드 키(4-Byte) 분석

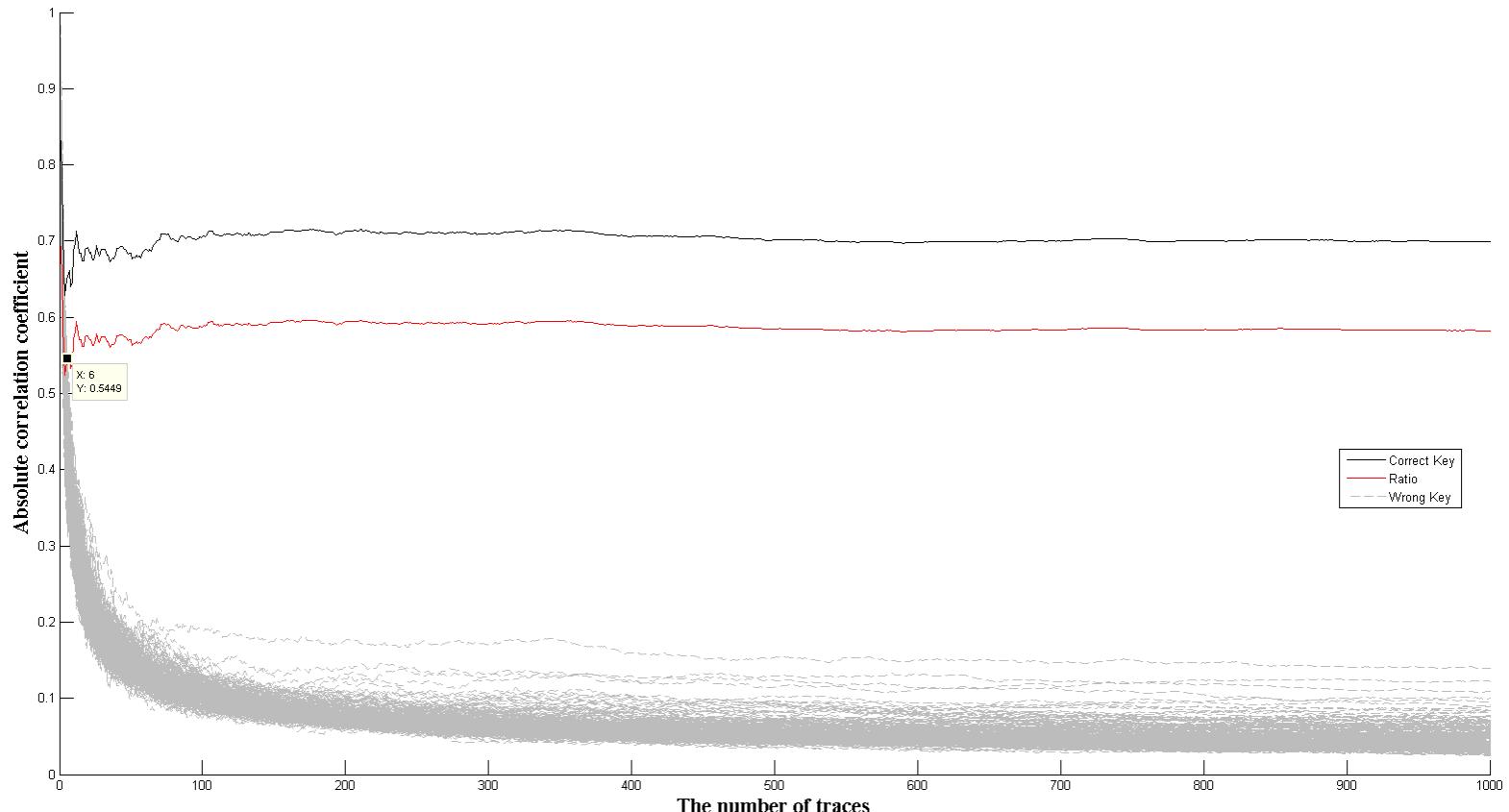
## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (2) 두 번째 G Function 의 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (2) 두 번째 G Function 의 1<sup>st</sup> Byte 파형 증가에 따른 분석 결과



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (1), (2) First Order CPA 결과

#### ✓ 1<sup>st</sup> G Function XOR Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Key	0B	9E	9D	46

#### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Ratio	3.68	3.06	4.31	3.73

#### ✓ 최소 분석 파형 수 ( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
파형 수	7	8	6	5

#### ✓ 2<sup>nd</sup> G Function Left Round Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Key	A9	EC	EE	F9

#### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Ratio	5.02	3.94	6.84	5.19

#### ✓ 최소 분석 파형 수 ( 단위 10 )

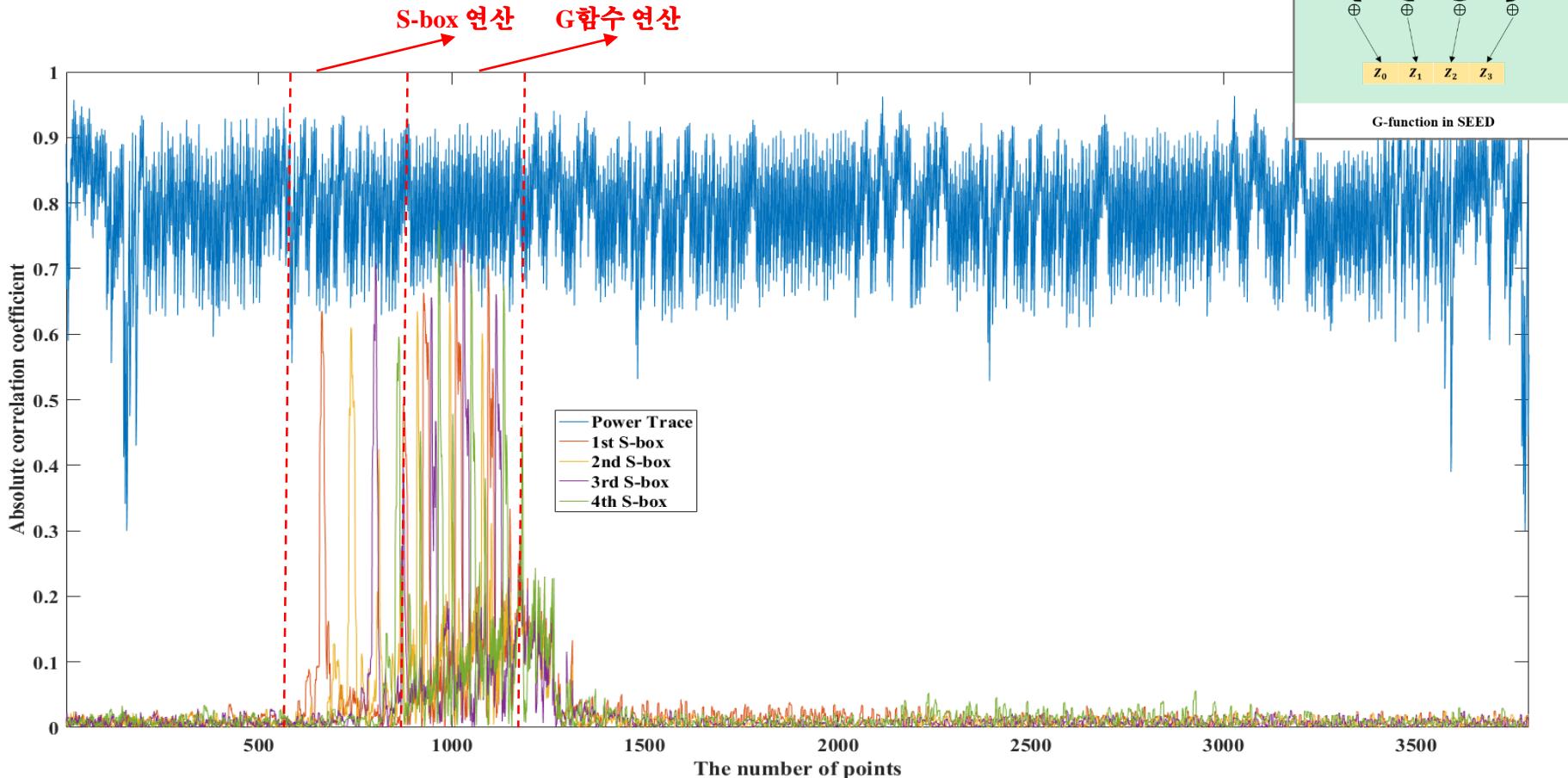
	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
파형 수	6	6	5	6

#### ✓ SEED 1 Round Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>
Key	A9	EC	EE	F9	A2	72	73	BF

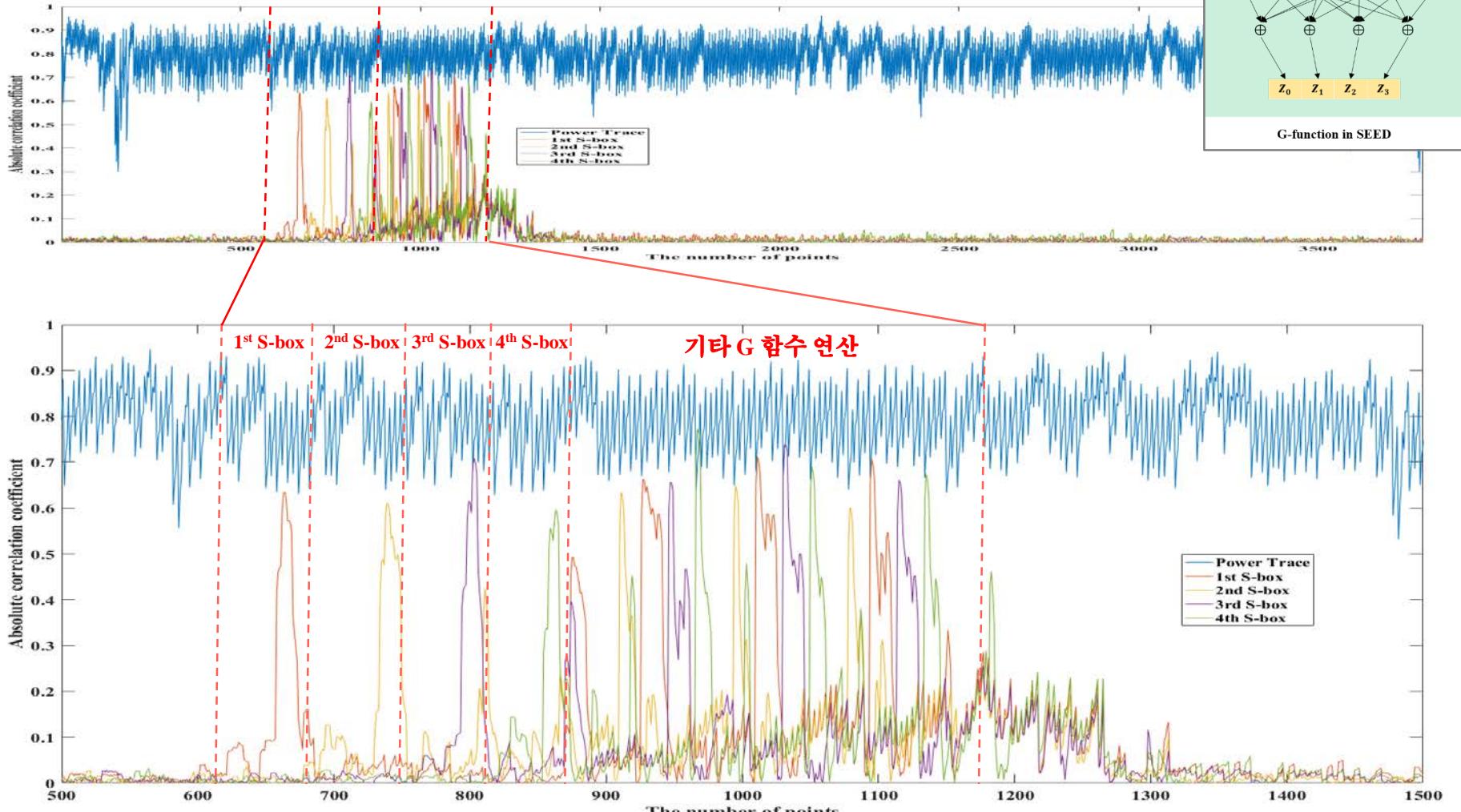
## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (3) SPA 정답 (S-box 연산, 기타 G함수 연산)



## ▣ 05\_Normal SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (3) SPA 정답 (S-box 연산, 기타 G함수 연산)



## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 06

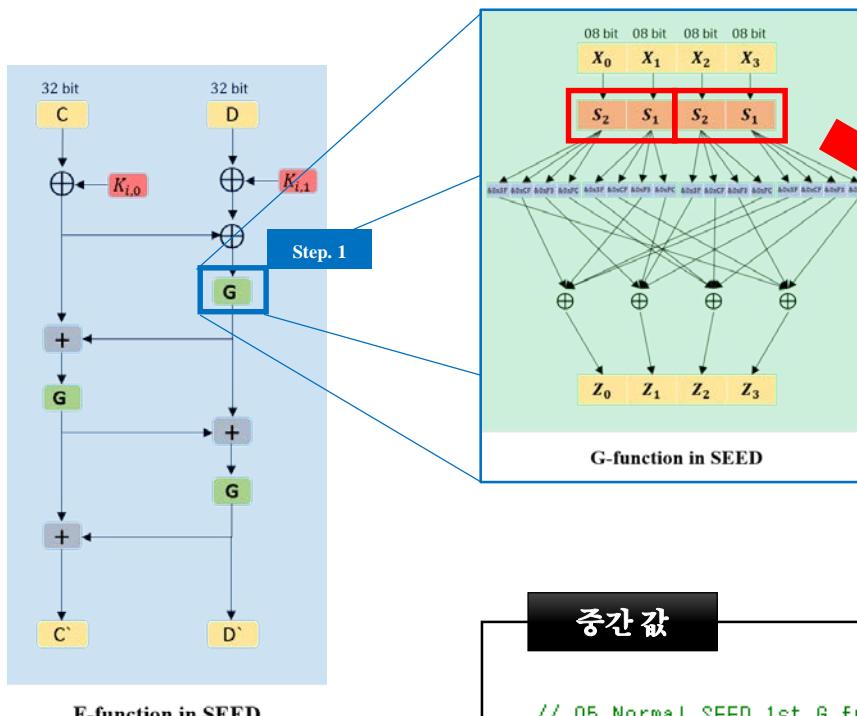
제공되는 정보는 1차 전력 분석 대응 기법으로 **마스킹이 적용된 8비트 기반 소프트웨어 SEED 암호 알고리즘이 ChipWhisperer-Lite에서 동작 할 때 소비되는 전력을 1라운드 부분**을 타겟으로 수집한 결과이다.

binary 파일 06\_First-Order\_Masked\_SEED\_10000tr\_24000pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 SEED 알고리즘이 10,000번 시행된 소비 전력이 포함되어 있는 파일이다.

06\_First-Order\_Masked\_SEED\_10000tr\_24000pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 소비 전력 및 평문 정보를 이용하여 SEED 암호 알고리즘 동작 시 사용된 1라운드키 상위 32 비트와 하위 32 비트의 exclusive OR 값( $RK - K1^0 \oplus RK - K0^0$ )을 소비 전력 정보를 이용하여 찾으시오.
- 2) 소비 전력 및 평문 정보를 이용하여 SEED 암호 알고리즘 동작 시 사용된 암호화키의 1라운드키( $RK^0$ )를 소비 전력 정보를 이용하여 찾으시오.

## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리



- ✓ **파형 조합 방법 (Preprocessing)**  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
  1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$

### ❖ Step. 1 1 라운드 XOR Key 분석

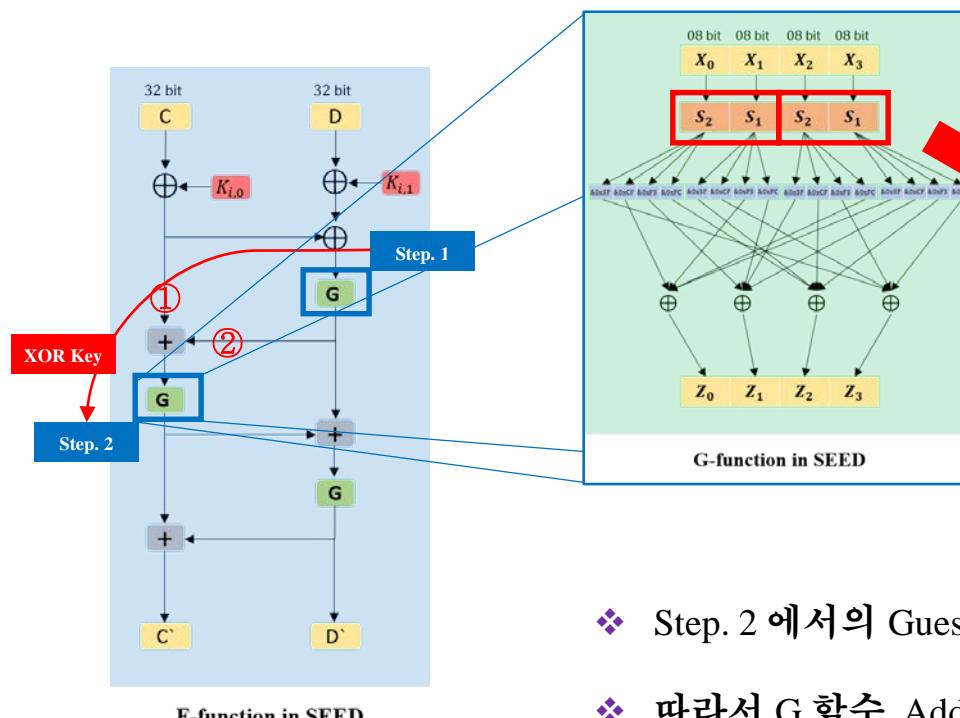
- 1라운드 1번째 G함수에서  
4바이트 XOR Key ( $K_{0,0} \oplus K_{1,0}$ )를 획득
- S-Box 순서 상관 없이 2개 조합으로 공격 가능 (총 6가지)

중간 값

t : S-box 위치  
i : i 번째 파형

```
// 05_Normal_SEED 1st G function 중간값 /////////////////////////////////
key = _SEED_SBOX_[(t + 1) % 2][ plaintext[i][t] ^ plaintext[i][t + 4] ^ guess_key ];//
///////////////////////////////////////////////////////////////////
```

## 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리



- ✓ **파형 조합 방법 (Preprocessing)**  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
  1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$

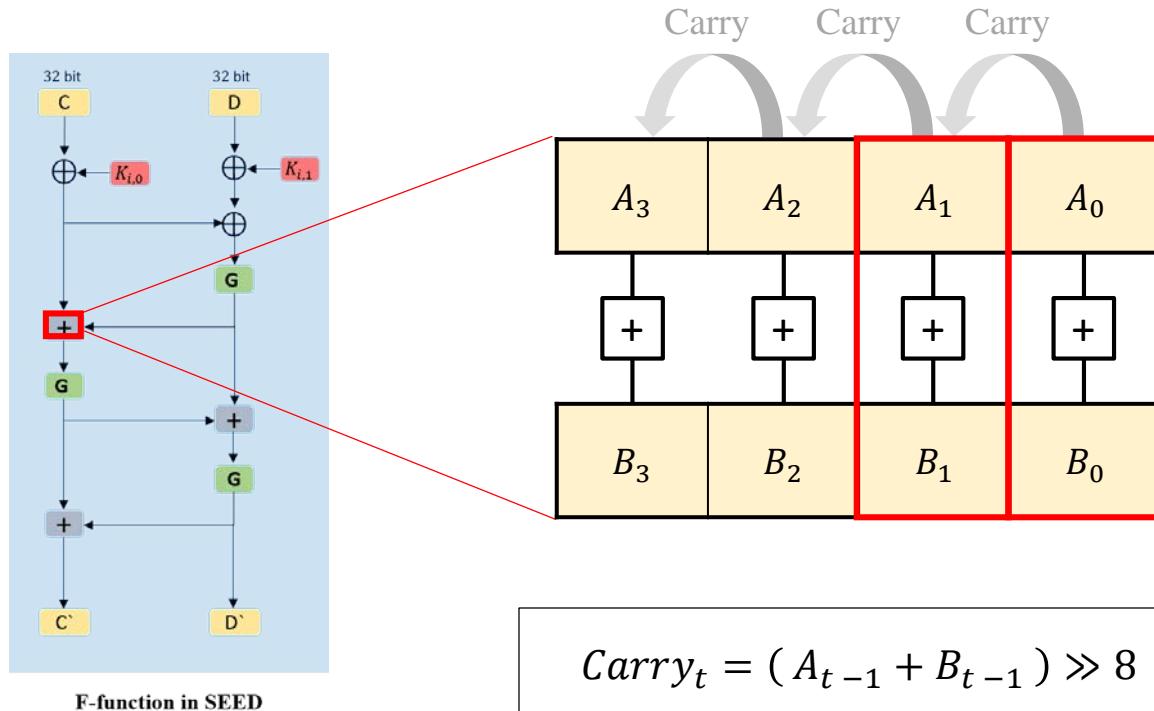
### ❖ Step. 2 1 라운드 Left RoundKey 분석

- 1라운드 1번째 G함수에서 4바이트 Left RoundKey ( $K_{0,0}$ )를 획득
- 최하위 Byte 단위 부터 2개 조합으로 공격 가능 (Carry)

- ❖ Step. 2에서의 Guessing을 위해서는 ①, ② 입력 값 필요
- ❖ 따라서 G 함수, Addition 함수 구현 필요
- ❖ 또한 Addition 연산에서의 Byte 단위의 Carry 고려

## □ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

### ❖ Addition Carry



- ❖ 32-bit 덧셈 연산을 8-bit 씩 수행할 경우 상위 8-bit에 대해 Carry가 발생함
- ❖ 따라서 최하위  $A_0 + B_0$  부터 연산한 후 Carry를 고려해서  
 $Carry(A_0 + B_0) + A_1 + B_1$  연산 수행

## □ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

### 중간 값

```
// 06_Masked_SEED 2nd G function 중간값 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// 1 Round XOR Key (4 Byte) -> XOR_KEY[0] || XOR_KEY[1] || XOR_KEY[2] || XOR_KEY[3]
XOR_KEY[0] = 0xf3;
XOR_KEY[1] = 0x14;
XOR_KEY[2] = 0x80;
XOR_KEY[3] = 0x4e;
```

- 1 Round XOR Key

```
// Plaintext(L) ^ Plaintext(R) ^ Roundkey(L) ^ Roundkey(R)
IN_R[0] = plaintext[i][8] ^ plaintext[i][12] ^ XOR_KEY[0];
IN_R[1] = plaintext[i][9] ^ plaintext[i][13] ^ XOR_KEY[1];
IN_R[2] = plaintext[i][10] ^ plaintext[i][14] ^ XOR_KEY[2];
IN_R[3] = plaintext[i][11] ^ plaintext[i][15] ^ XOR_KEY[3];
```

```
// 1 Round G Function Output
SEED_GFunction(IN_R)
```

- 1<sup>st</sup> G 함수 연산

```
// Key Guessing
//RK_L[0] = ((guess_key >> 8) & 0xFF);
//RK_L[1] = ((guess_key >> 0) & 0xFF);
RK_L[2] = ((guess_key >> 8) & 0xFF);
RK_L[3] = ((guess_key >> 0) & 0xFF);
```

- 1 Round Left RoundKey  
조합 Guessing

```
// Plaintext ^ Roundkey(L)
IN_L[0] = plaintext[i][8] ^ RK_L[0];
IN_L[1] = plaintext[i][9] ^ RK_L[1];
IN_L[2] = plaintext[i][10] ^ RK_L[2];
IN_L[3] = plaintext[i][11] ^ RK_L[3];
```

- Left Round Key XOR 연산

```
// 1 Round 1st Add
SEED_ADD32_8(IN_L, IN_R, ADD_OUT);
```

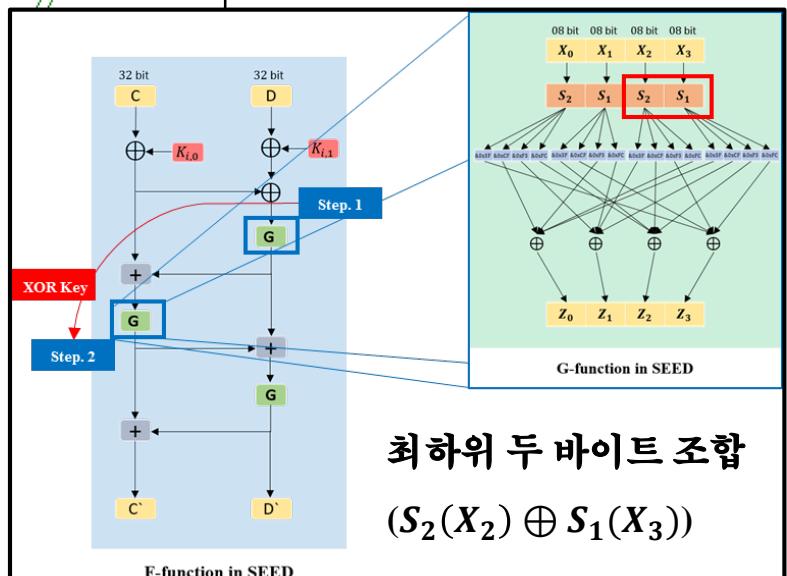
- 1<sup>st</sup> Addition 함수 연산

```
// 1 Round 2nd G Function 중간값
//key = _SEED_SBOX_[1][ADD_OUT[0]] ^ _SEED_SBOX_[0][ADD_OUT[1]];
key = _SEED_SBOX_[1][ADD_OUT[2]] ^ _SEED_SBOX_[0][ADD_OUT[3]];
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

t : S-box 위치  
i : i 번째 파형

//  
//  
//  
//

- ✓ **파형 조합 방법 (Preprocessing)**  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$



## □ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

### 중간 값

```
// 06_Masked_SEED 2nd G function 중간값 //////////////////////////////////////////////////////////////////
// 1 Round XOR Key (4 Byte) -> XOR_KEY[0] || XOR_KEY[1] || XOR_KEY[2] || XOR_KEY[3]
XOR_KEY[0] = 0xf3;
XOR_KEY[1] = 0x14;
XOR_KEY[2] = 0x80;
XOR_KEY[3] = 0x4e;
```

- 1 Round XOR Key

```
// Plaintext(L) ^ Plaintext(R) ^ Roundkey(L) ^ Roundkey(R)
IN_R[0] = plaintext[i][8] ^ plaintext[i][12] ^ XOR_KEY[0];
IN_R[1] = plaintext[i][9] ^ plaintext[i][13] ^ XOR_KEY[1];
IN_R[2] = plaintext[i][10] ^ plaintext[i][14] ^ XOR_KEY[2];
IN_R[3] = plaintext[i][11] ^ plaintext[i][15] ^ XOR_KEY[3];
```

```
// 1 Round G Function Output
SEED_GFuction(IN_R)
```

- 1<sup>st</sup> G 함수 연산

```
// Key Guessing
RK_L[0] = ((guess_key >> 8) & 0xFF);
RK_L[1] = ((guess_key >> 0) & 0xFF);
RK_L[2] = 0x98;
RK_L[3] = 0xa8;
```

- 1 Round Left RoundKey Guessing
- 앞서 찾은 Left RoundKey

2 Byte 입력 (Carry)

```
// Plaintext ^ Roundkey(L)
IN_L[0] = plaintext[i][8] ^ RK_L[0];
IN_L[1] = plaintext[i][9] ^ RK_L[1];
IN_L[2] = plaintext[i][10] ^ RK_L[2];
IN_L[3] = plaintext[i][11] ^ RK_L[3];
```

- Left Round Key XOR 연산

```
// 1 Round 1st Add
SEED_ADD32_8(IN_L, IN_R, ADD_OUT);
```

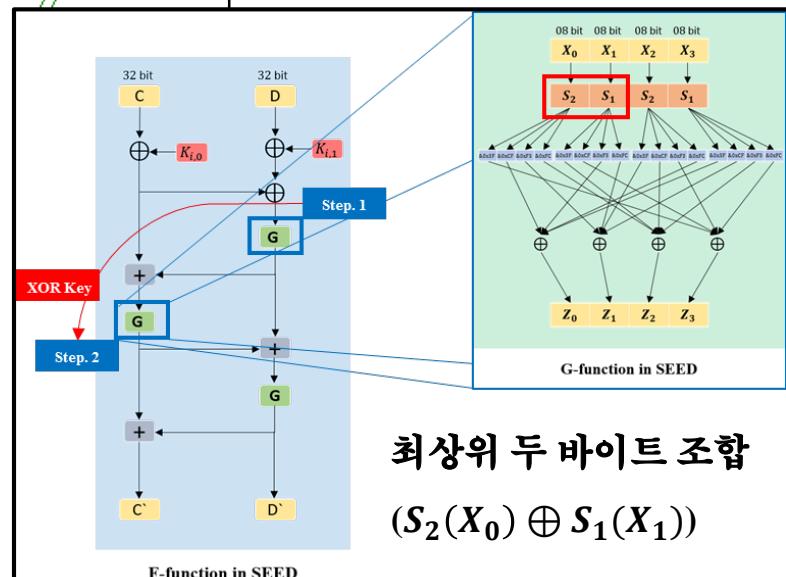
- 1<sup>st</sup> Addition 함수 연산 (Carry)

```
// 1 Round 2nd G Function 중간값
```

```
key = _SEED_SBOX_[1][ADD_OUT[0]] ^ _SEED_SBOX_[0][ADD_OUT[1]];
//key = _SEED_SBOX_[1][ADD_OUT[2]] ^ _SEED_SBOX_[0][ADD_OUT[3]]; //
```



- ✓ **파형 조합 방법 (Preprocessing)**  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$

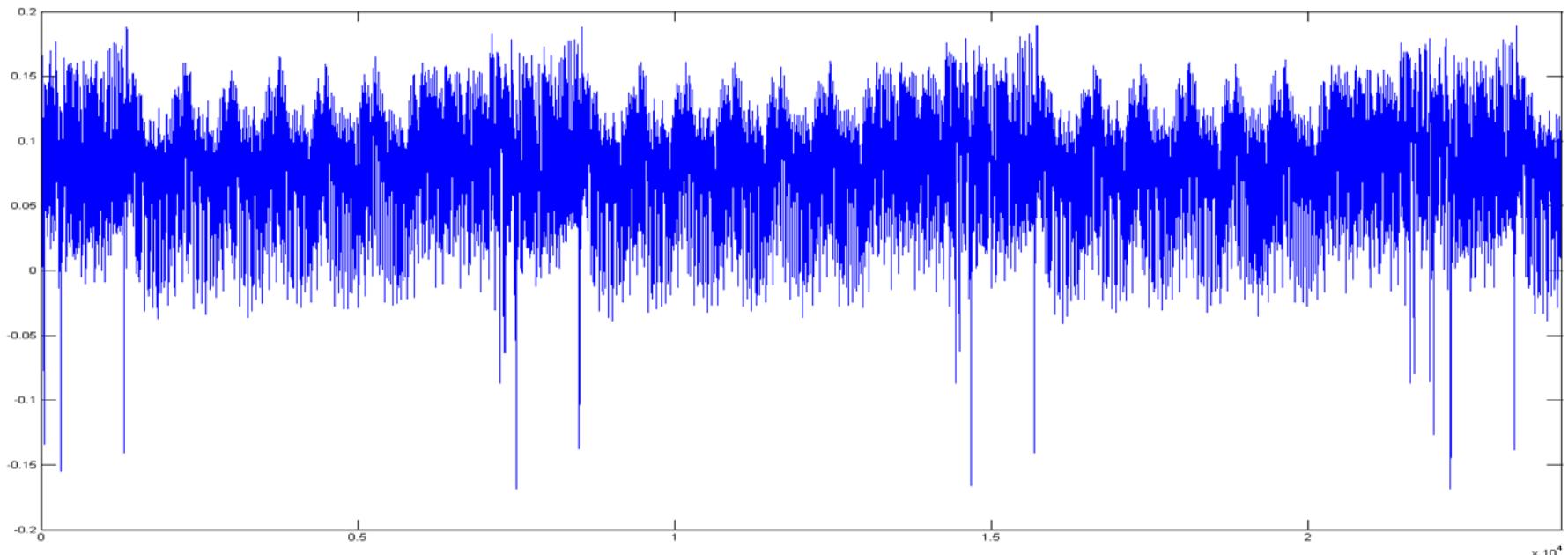


```
////////////////////////////////////////////////////////////////
```

## ▣ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

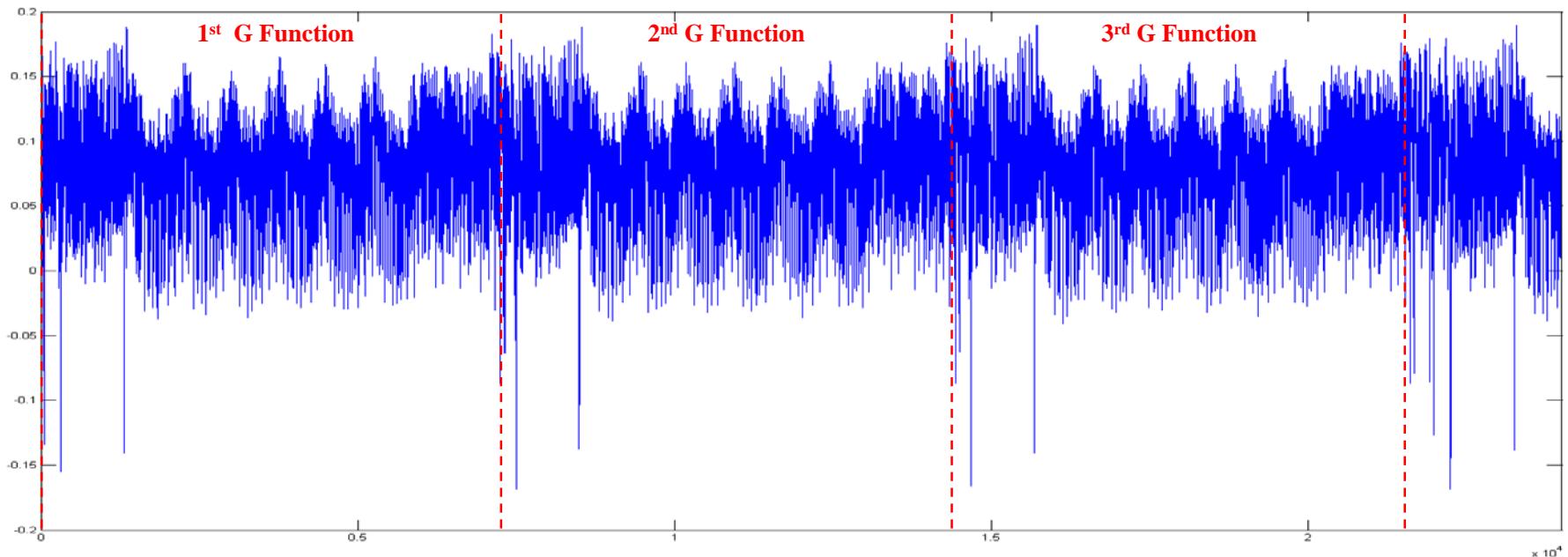
파형 정보	
파형 수	10,000
포인트 수	24,000



## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	10,000
포인트 수	24,000

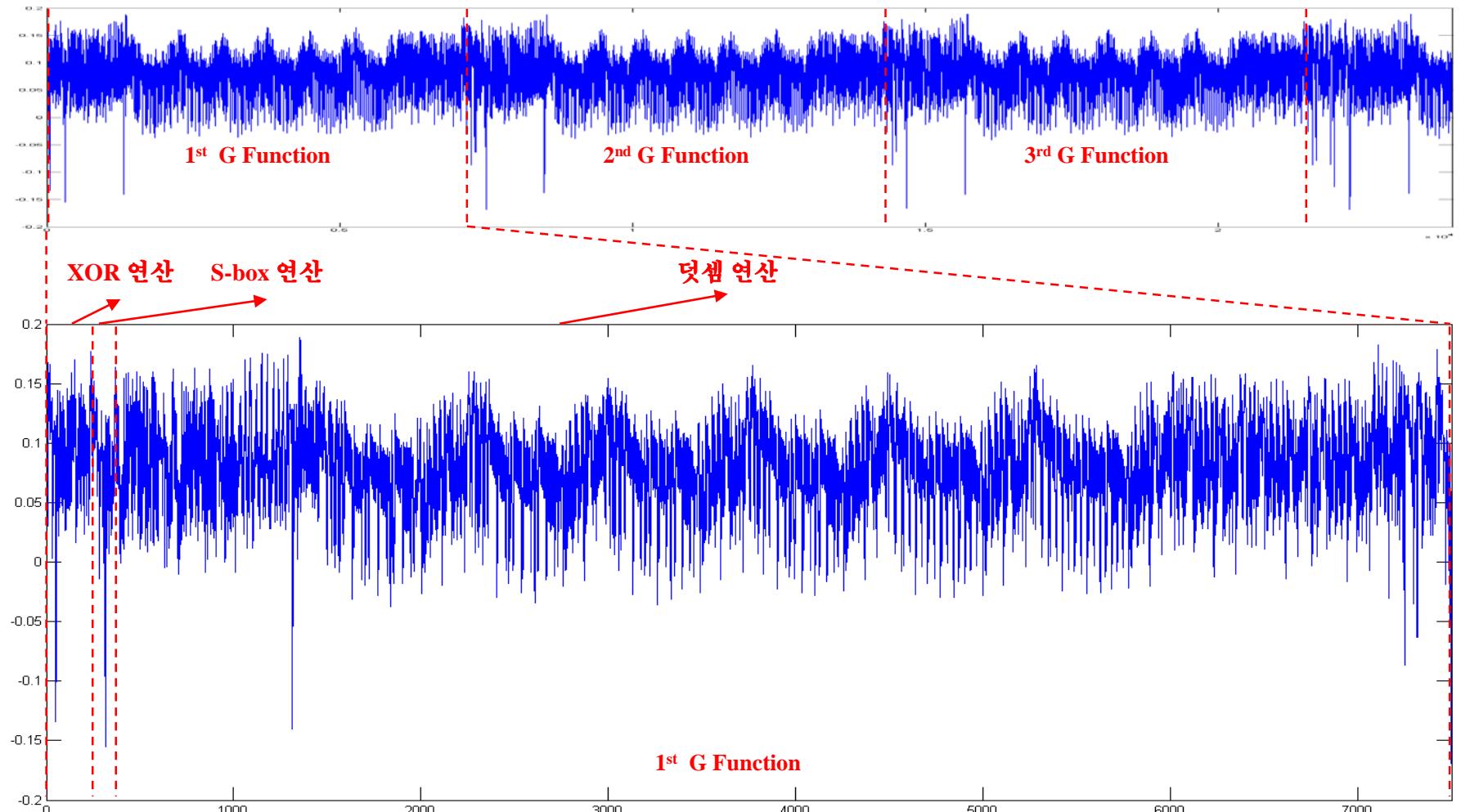


- ❖ SPA를 통해 3개의 G 함수 구분 가능

## 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ◆ 전력 파형 SPA (1<sup>st</sup> G 함수 확대)

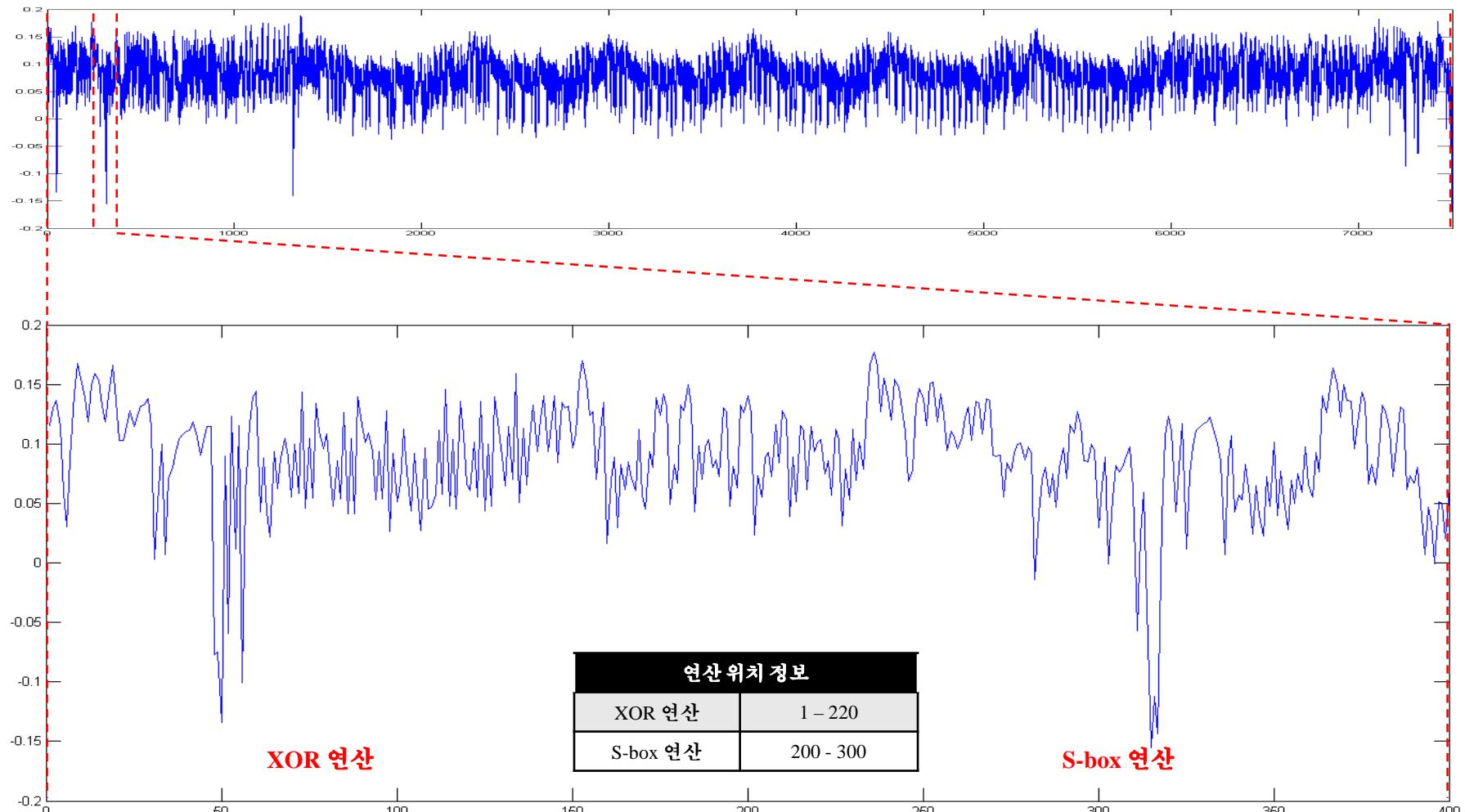
파형 정보	
파형 수	10,000
포인트 수	24,000



## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

❖ 전력 파형 SPA (XOR 연산 & S-Box 연산 확대)

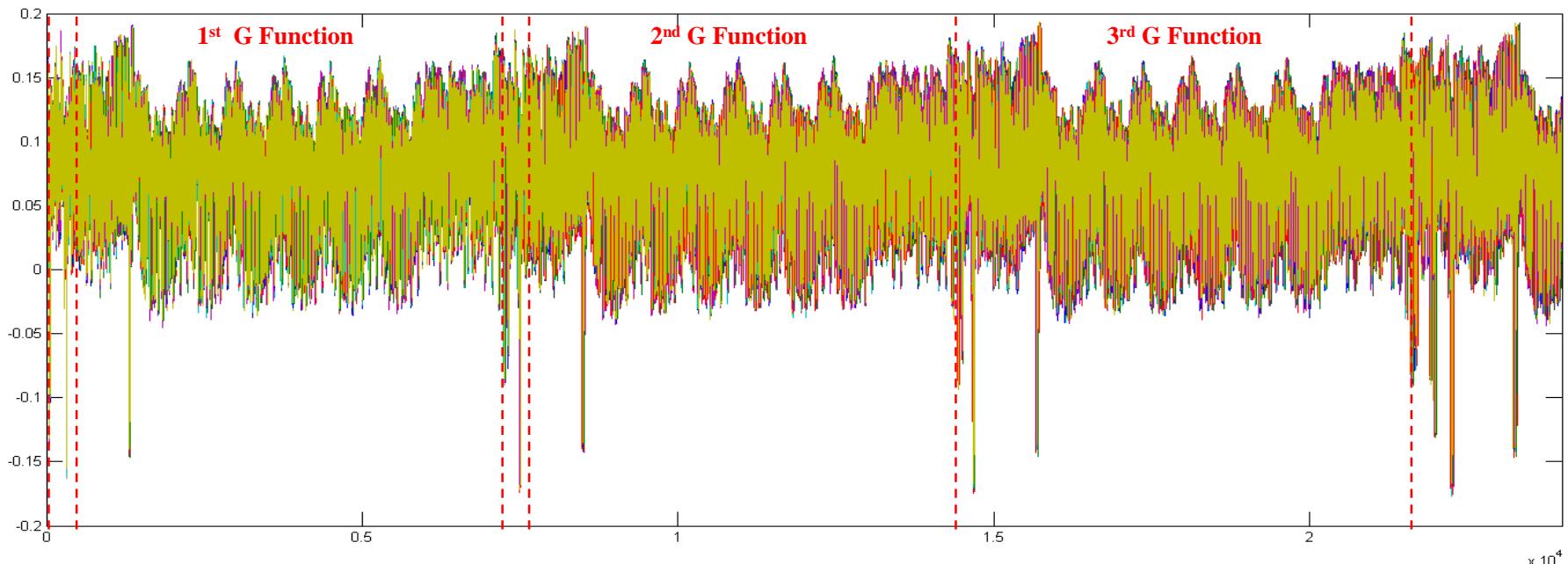
파형 정보	
파형 수	10,000
포인트 수	24,000



## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

- ◆ 전력 파형 SPA (Original / Overlab)

파형 정보	
파형 수	10,000
포인트 수	24,000

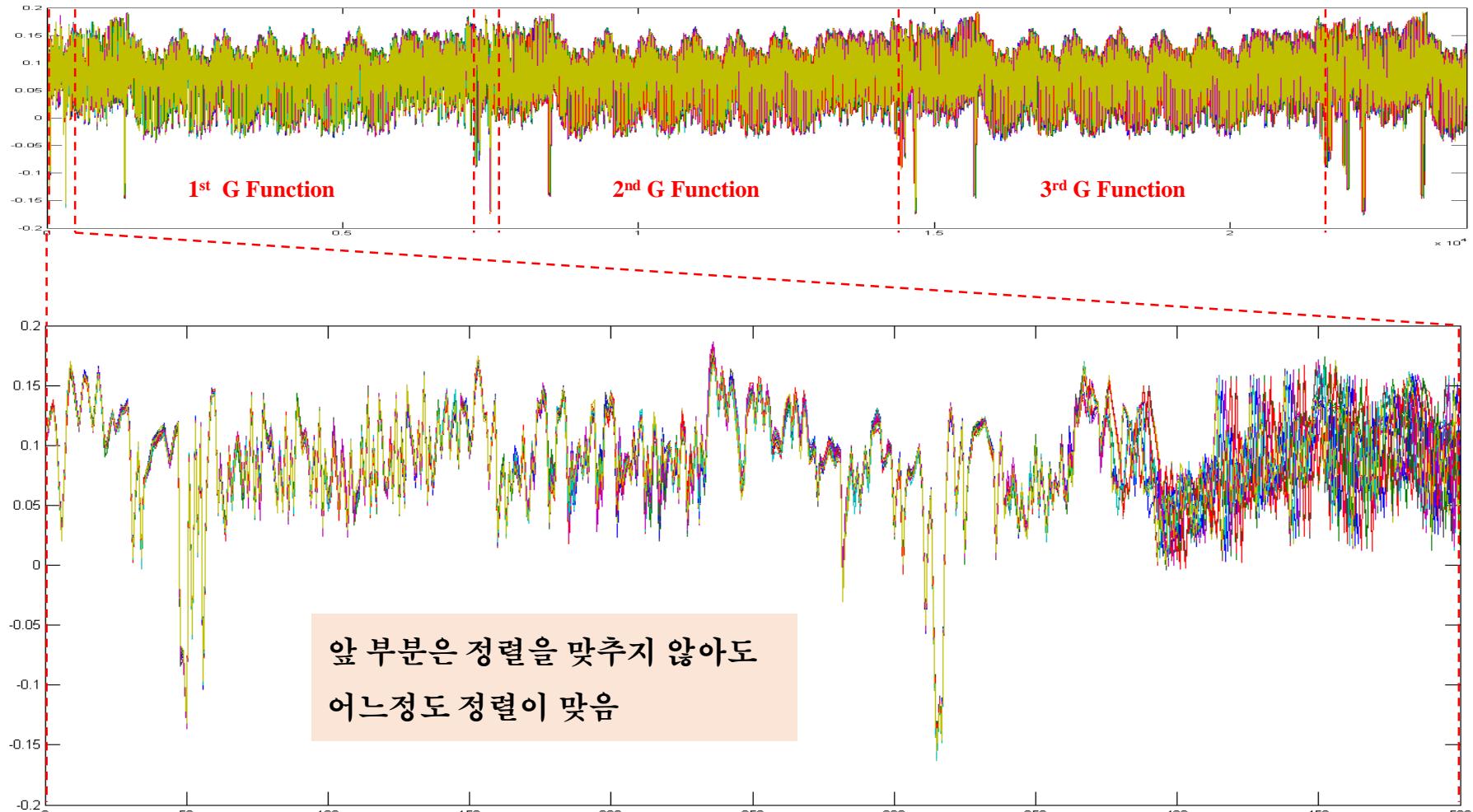


- ◆ 파형을 중첩해보면 뒤로 갈수록 파형이 흔들리는 것을 볼 수 있음 (정렬 필요)

## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

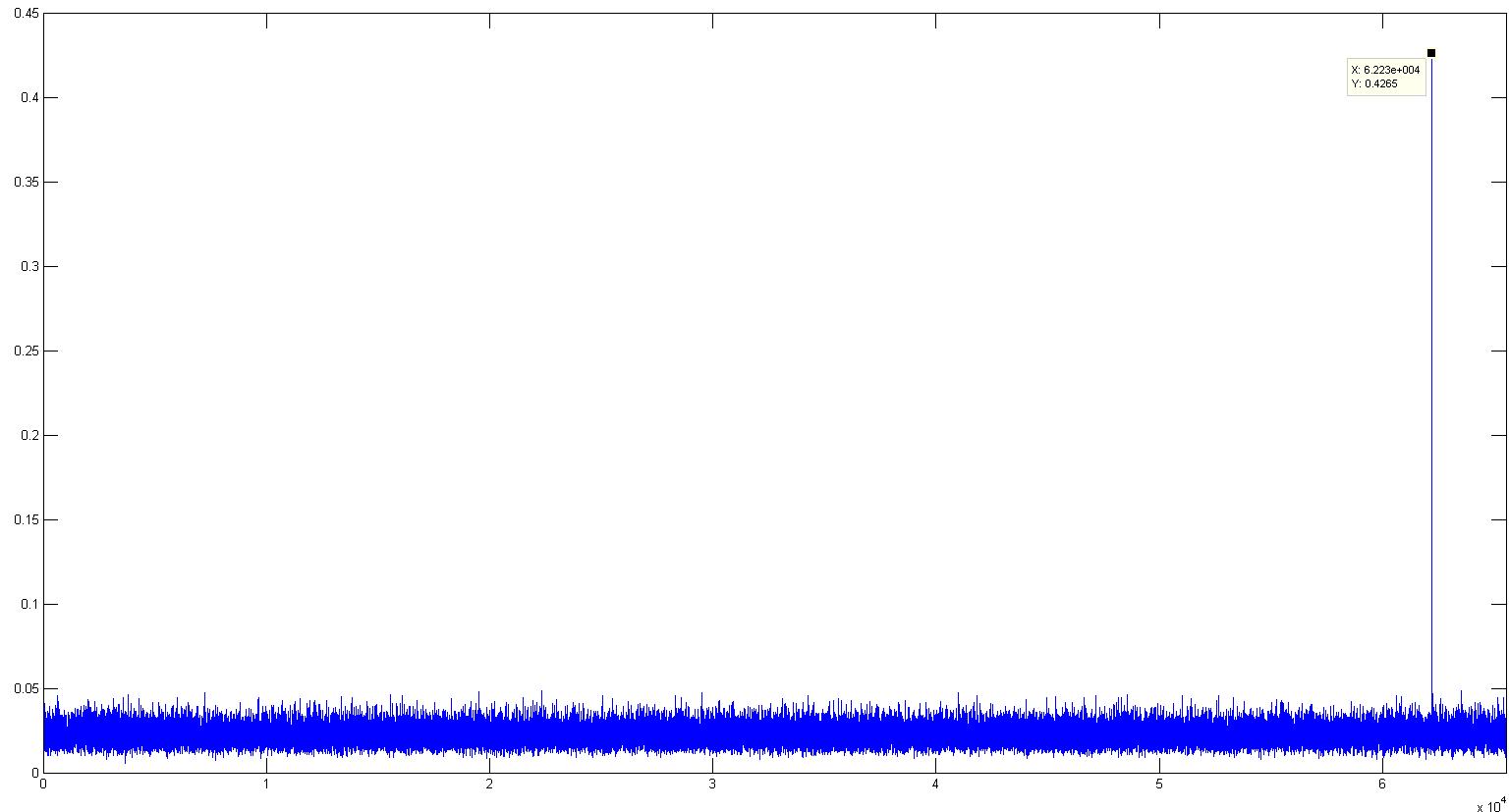
### ❖ (1) 첫 번째 G 함수 확대 파형

파형 정보	
파형 수	10,000
포인트 수	24,000



## ▣ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

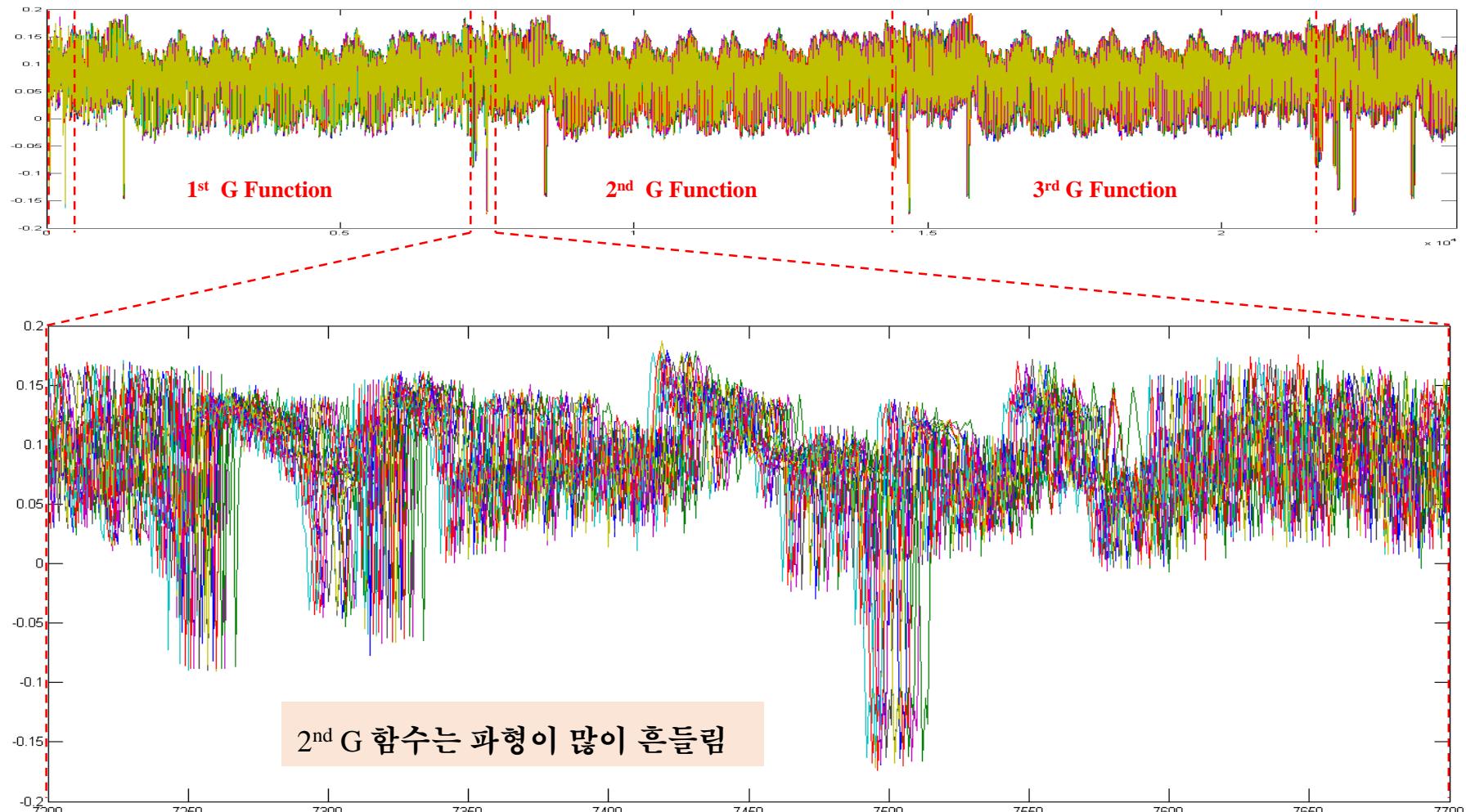
### ❖ (1) 첫 번째 G Function 의 (1<sup>st</sup> Byte $\oplus$ 2<sup>nd</sup> Byte) Maxpeak 분석 결과



## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (2) 두 번째 G 함수 확대 파형

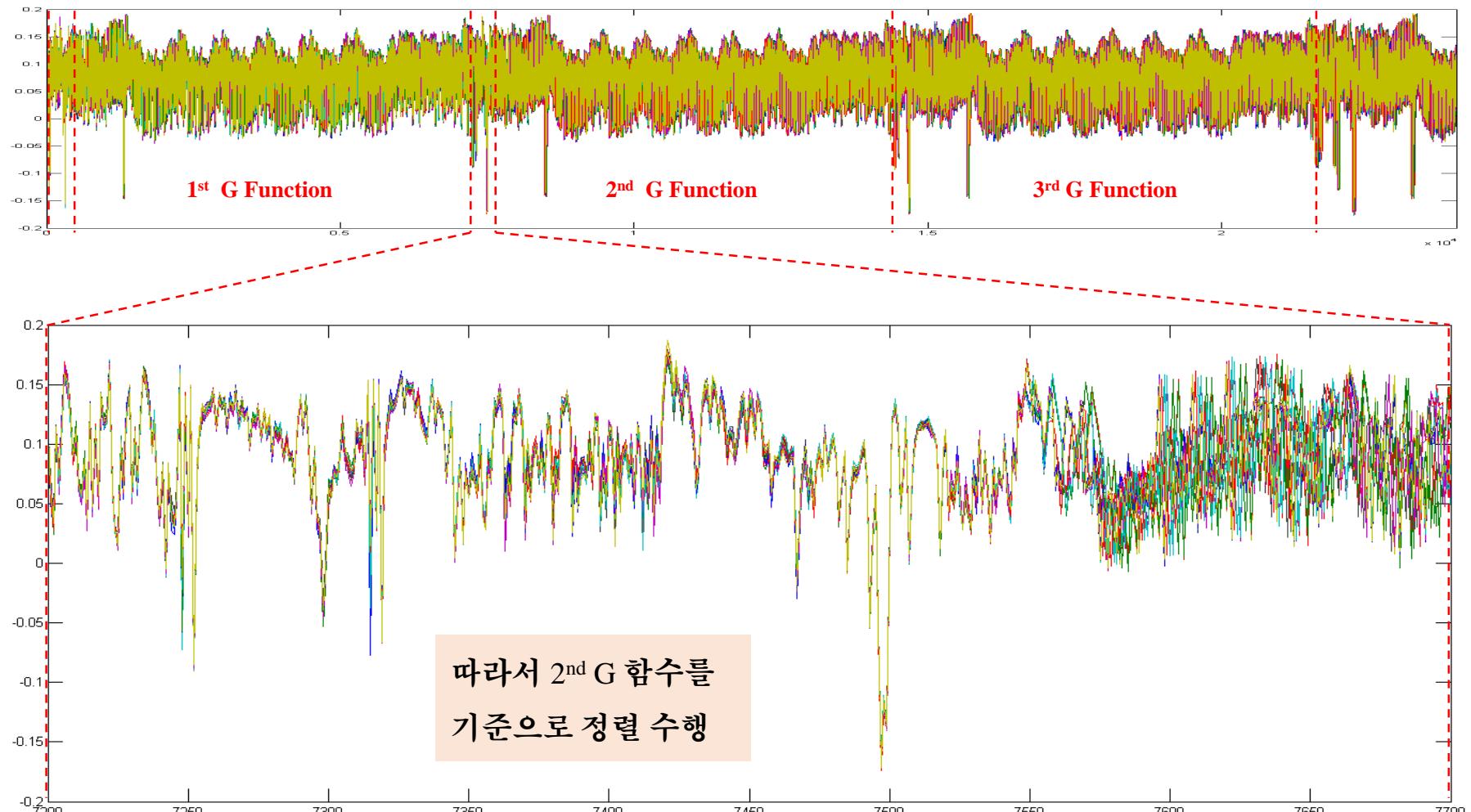
파형 정보	
파형 수	10,000
포인트 수	24,000



## 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

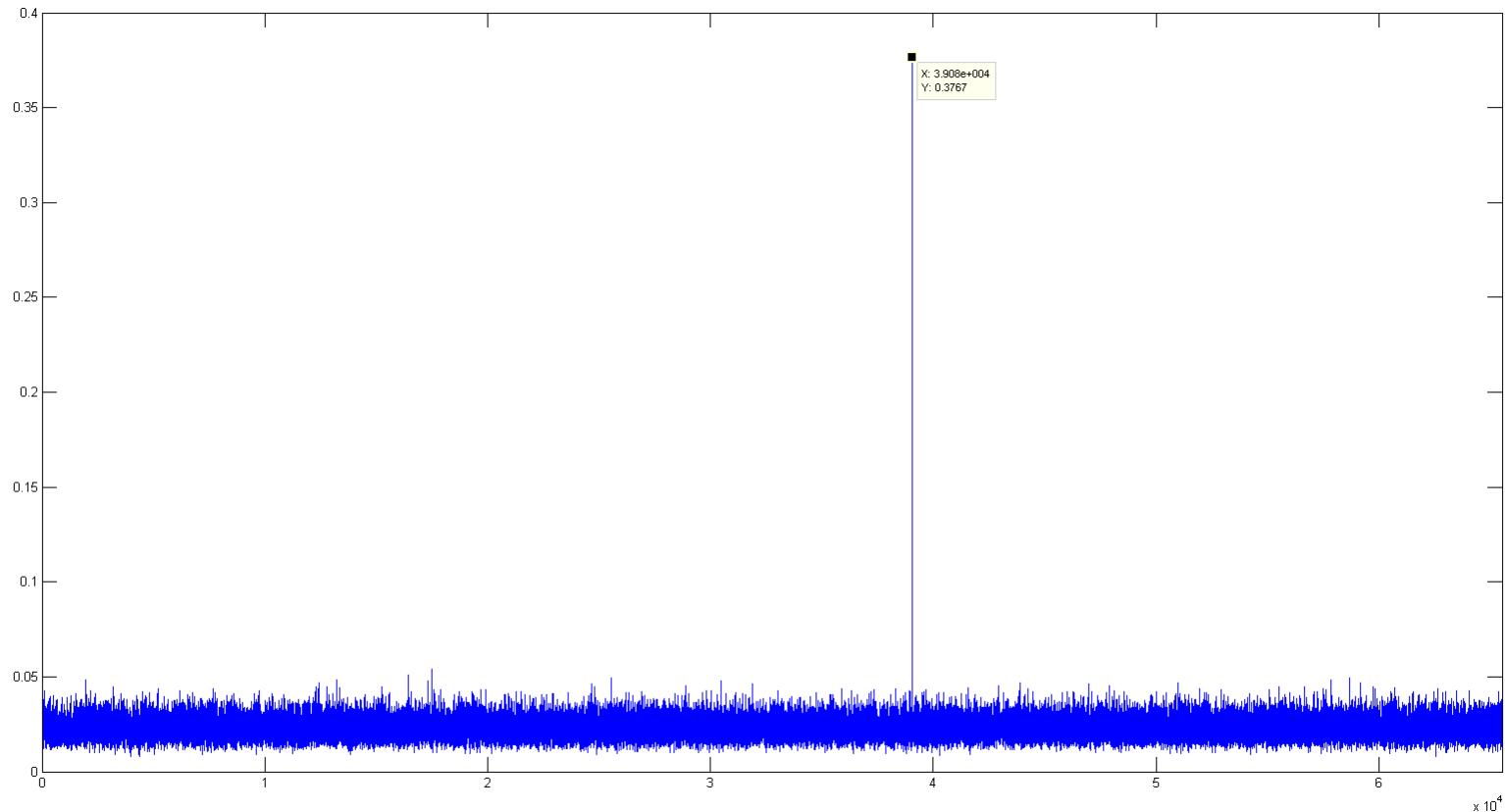
### ❖ (2) 두 번째 G 함수 확대 정렬 과정

파형 정보	
파형 수	10,000
포인트 수	24,000



## ▣ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (2) 두 번째 G Function 의 (3<sup>rd</sup> Byte $\oplus$ 4<sup>th</sup> Byte) Maxpeak 분석 결과



## ■ 06\_Masked SEED / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (1), (2) First Order CPA 결과

#### ✓ 1<sup>st</sup> G Function XOR Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Key	F3	14	80	4E

#### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Ratio	3.68	3.06	4.31	3.73

#### ✓ 분석 위치

분석 위치	
1 & 2번째	(237, 238)
3 & 4번째	(260, 261)

#### ✓ 2<sup>nd</sup> G Function Left Round Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Key	9E	82	98	A8

#### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Ratio	5.02	3.94	6.84	5.19

#### ✓ 분석 위치

분석 위치	
1 & 2번째	(7435, 7436)
3 & 4번째	(7445, 7446)

#### ✓ 1 Round Key

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>
Key	9E	82	98	A8	6D	96	18	E6

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 07

제공되는 정보는 Eight-Shuffling AES 암호 알고리즘이 ChipWhisperer-Lite에서 동작 할 때 소비되는 전력을 각 AES의 대략 2~3라운드까지 수집한 결과이다. Eight-Shuffling AES는 8가지의 서로 다른 AES-128 구현을 이용하여 셔플링 기법을 제공하는 알고리즘으로 자세한 내용은 참고 논문 [02]에서 확인할 수 있다.

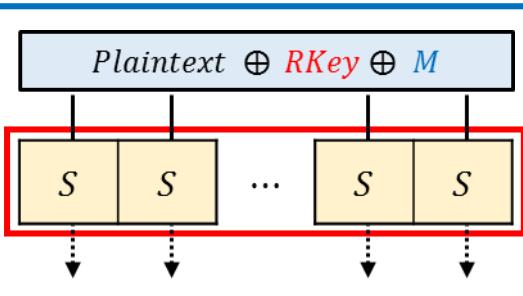
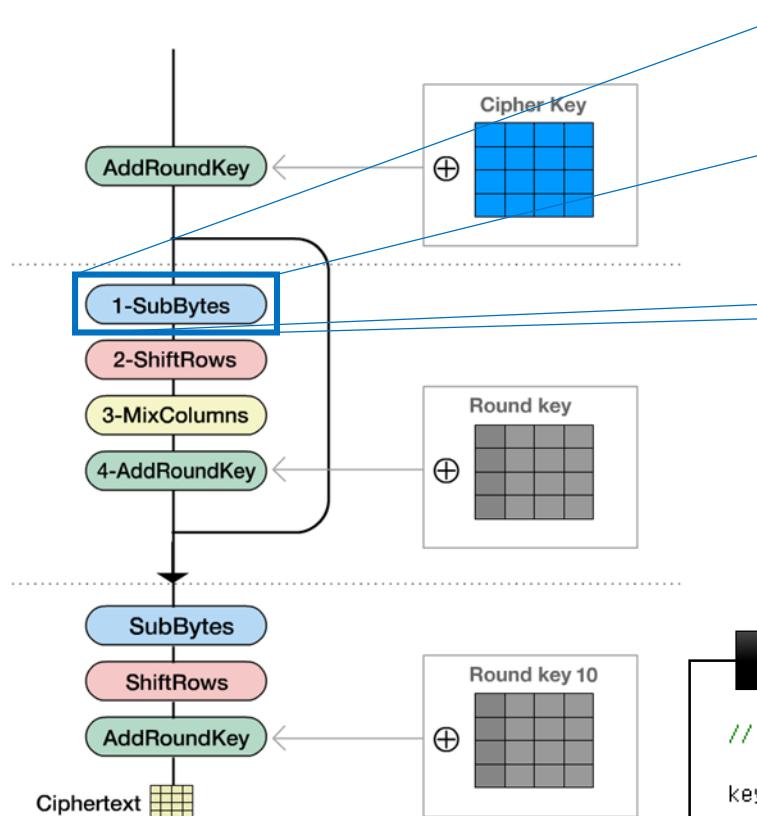
binary 파일 07\_Eight-Shuffling\_AES\_30000tr\_3492pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 AES 알고리즘이 30,000번 시행된 소비 전력이 포함되어 있는 파일이다.

07\_Eight-Shuffling\_AES\_30000tr\_3492pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

1) 소비 전력 및 평문 정보를 이용하여 AES-128 암호 알고리즘 동작 시 사용된 암호화키의 1라운드키 ( $RK^0$ )를 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)

▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 논리

## ❖ Normal AES

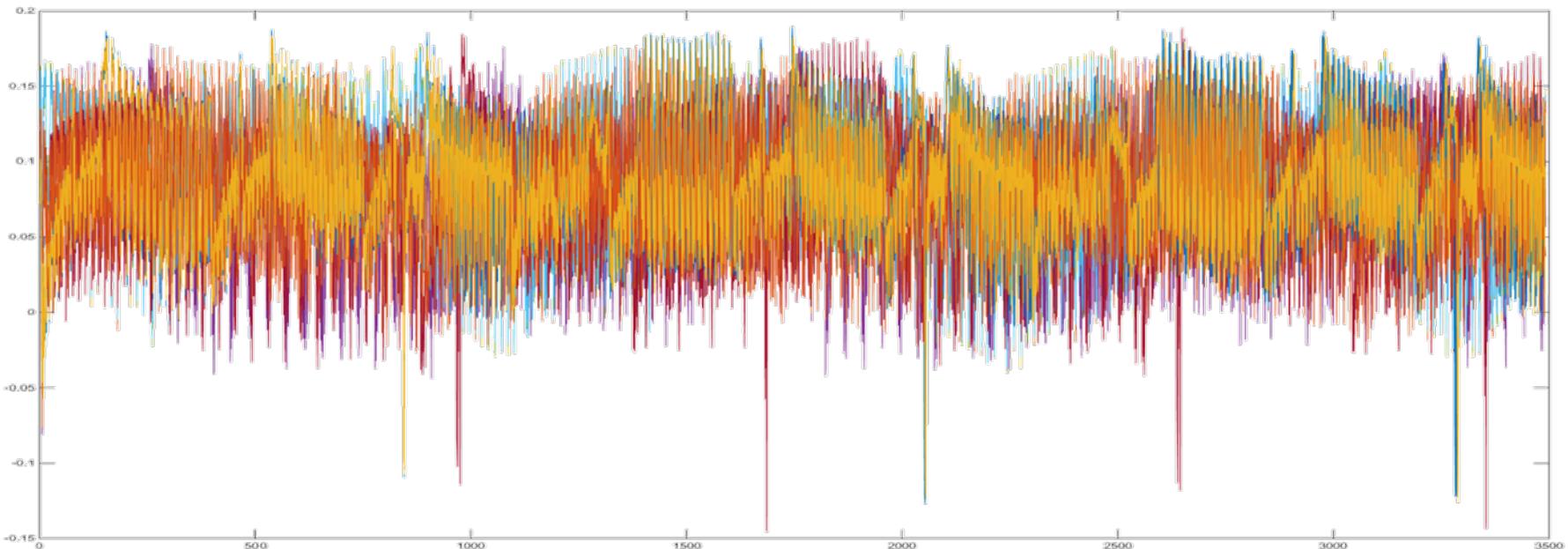


### ❖ S-box 출력 지점 분석

- Non-linear
  - 8 bit Guessing

## ▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 논리

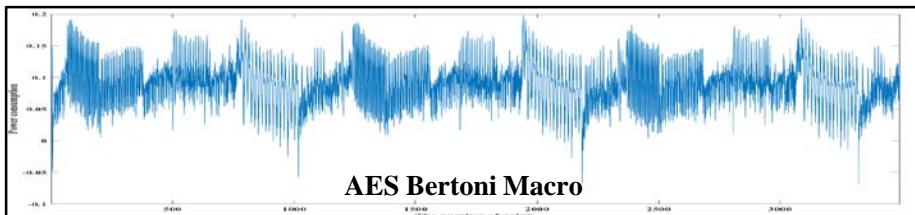
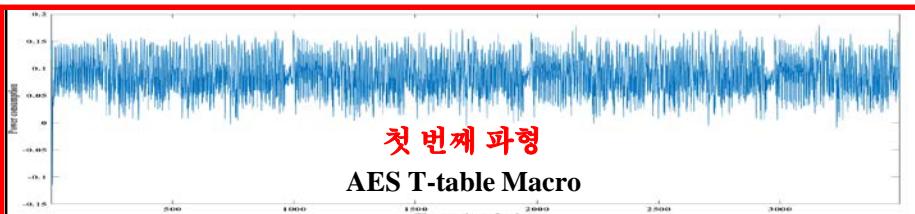
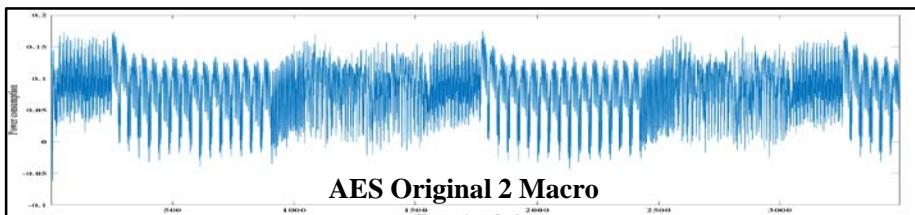
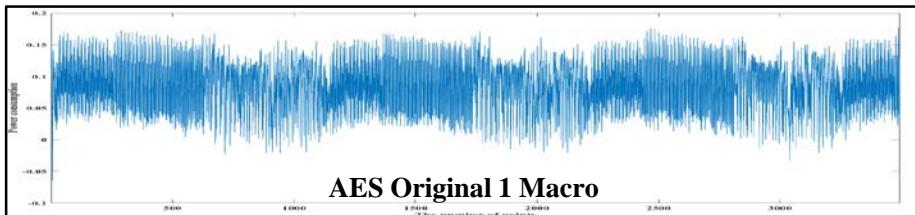
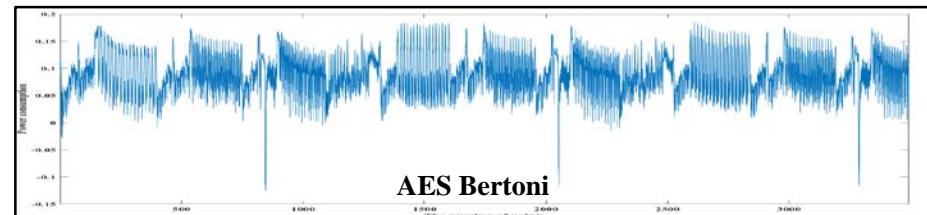
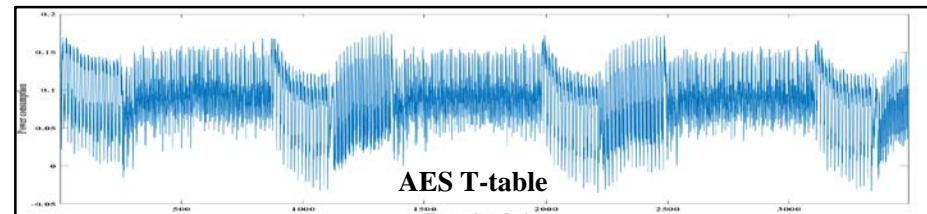
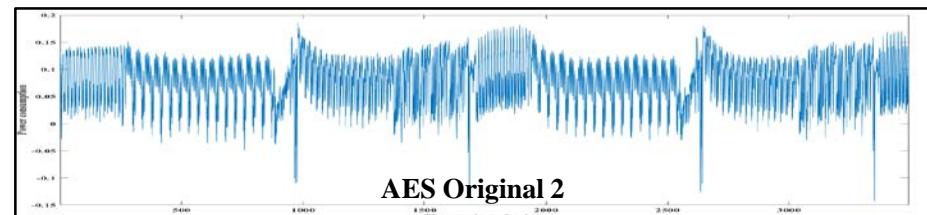
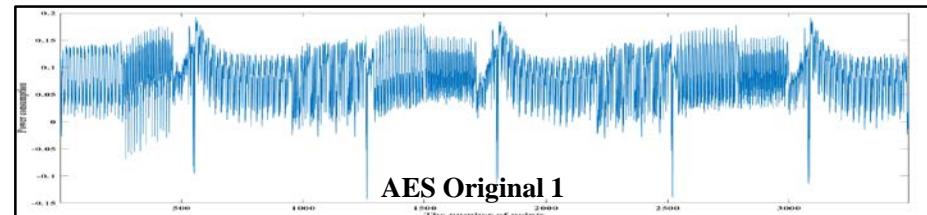
- ❖ 전력 파형 SPA (Original / Overlab)



- ❖ 8 가지 종류의 AES 가 랜덤하게 수행
- ❖ 일반적인 정렬 불가능

## ▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 논리

### ❖ (1) 전력 파형 필터링 (8 가지)

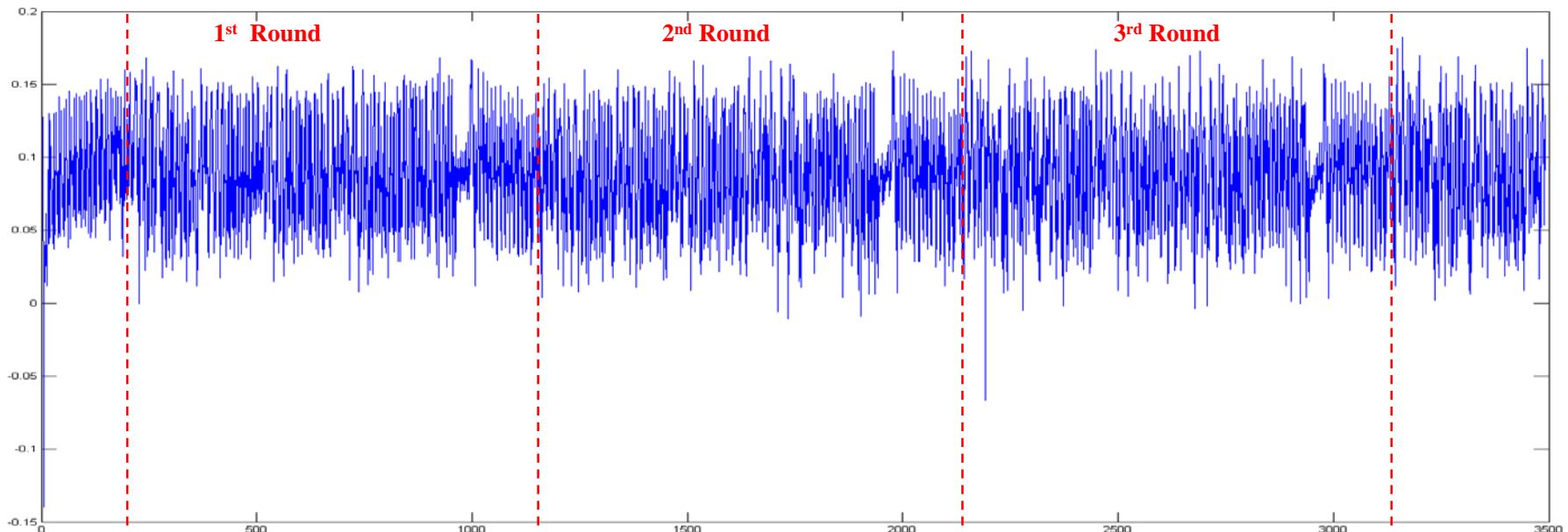


### ❖ 8 가지 AES 중 특정 파형에 대하여 파형 필터링을 수행 (상관계수 기반)

## ■ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석

- ❖ (1) 필터링 된 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	30,000
포인트 수	3,492

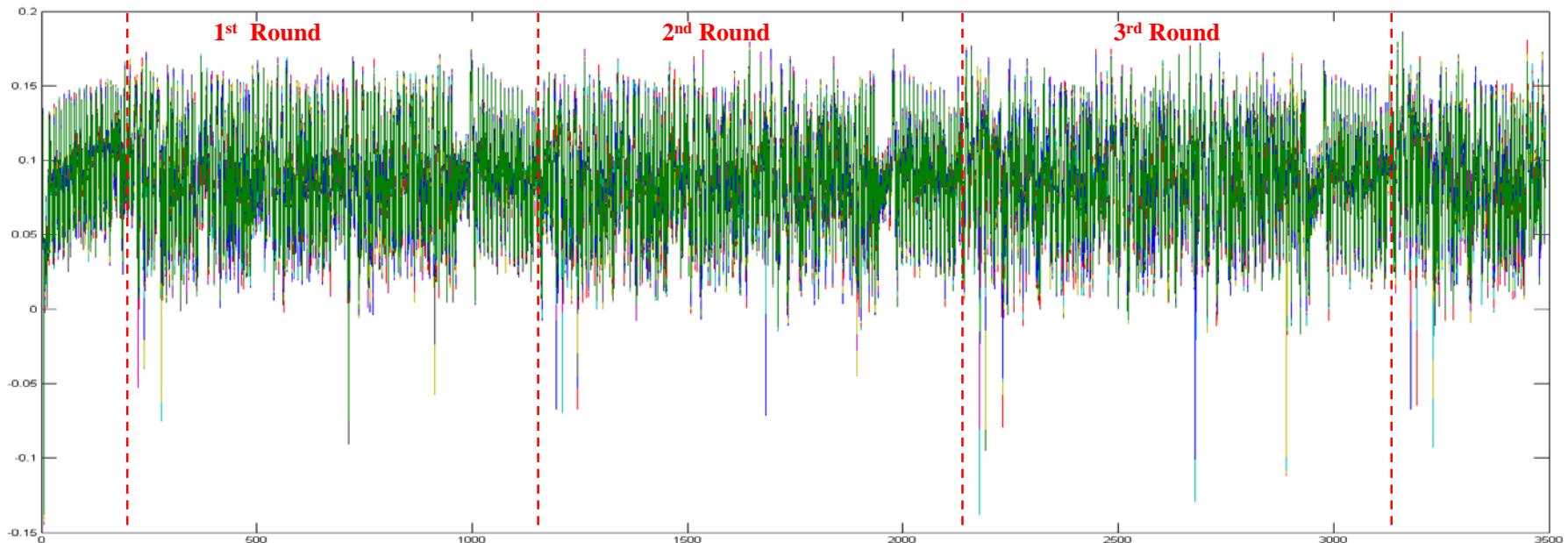


- ❖ 주어진 파형에서 첫 번째 파형을 기준으로 필터링 진행 (다른 파형을 기준으로도 필터링 가능)
- ❖ 전체적인 파형을 SPA 하면 동일한 패턴 3개로 구분 되는 것을 볼 수 있음
- ❖ 하지만 세부적인 AES 함수를 구분하기는 어려움

## ■ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석

- ❖ (1) 필터링 된 파형 SPA (1<sup>st</sup> trace)

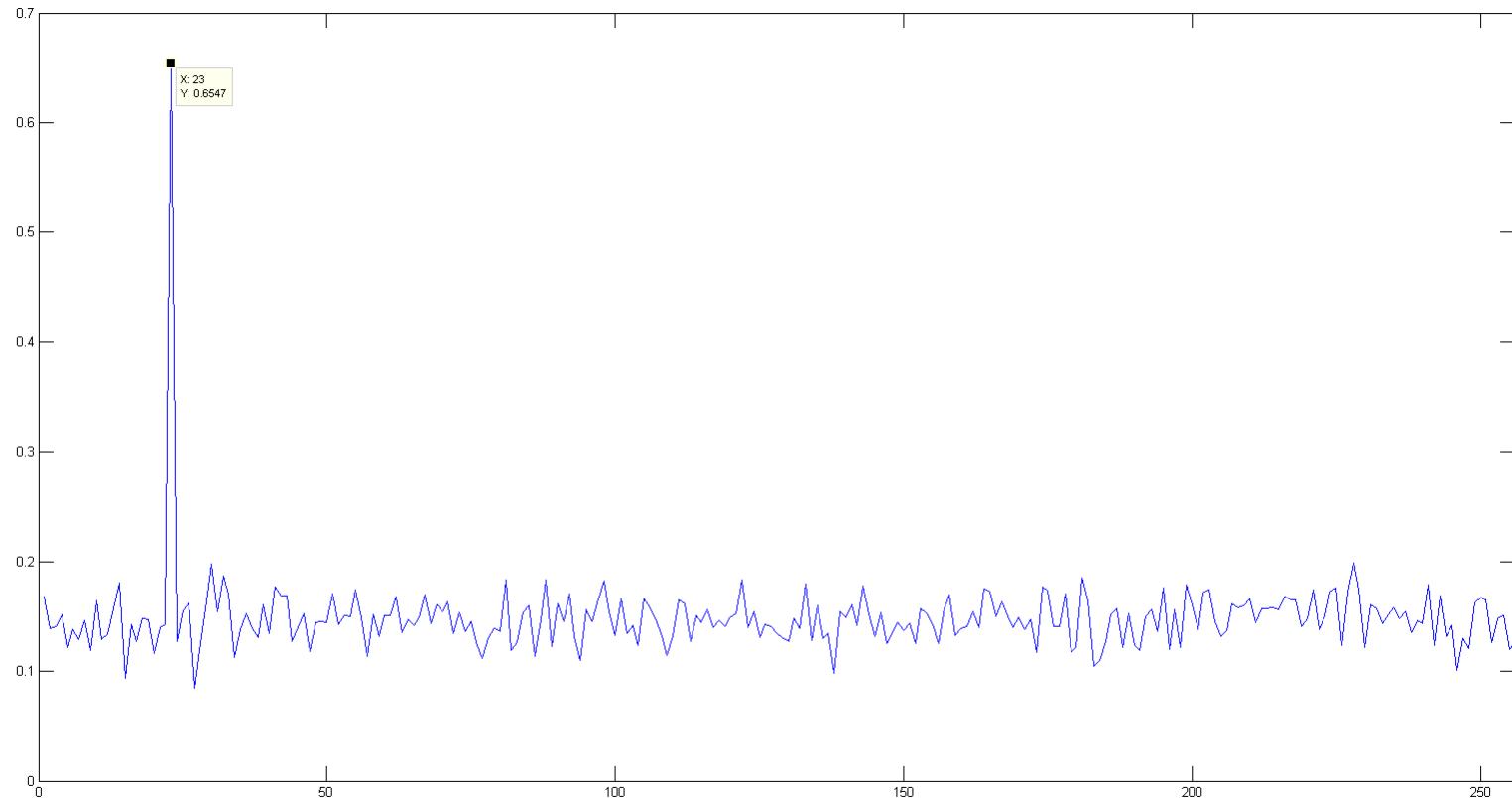
파형 정보	
파형 수	30,000
포인트 수	3,492



- ❖ 필터링 후 파형을 중첩하여 보면 한 종류의 AES 파형으로 필터링 된 것을 볼 수 있음
- ❖ 다른 종류의 AES 파형으로 필터링 해도 분석 가능

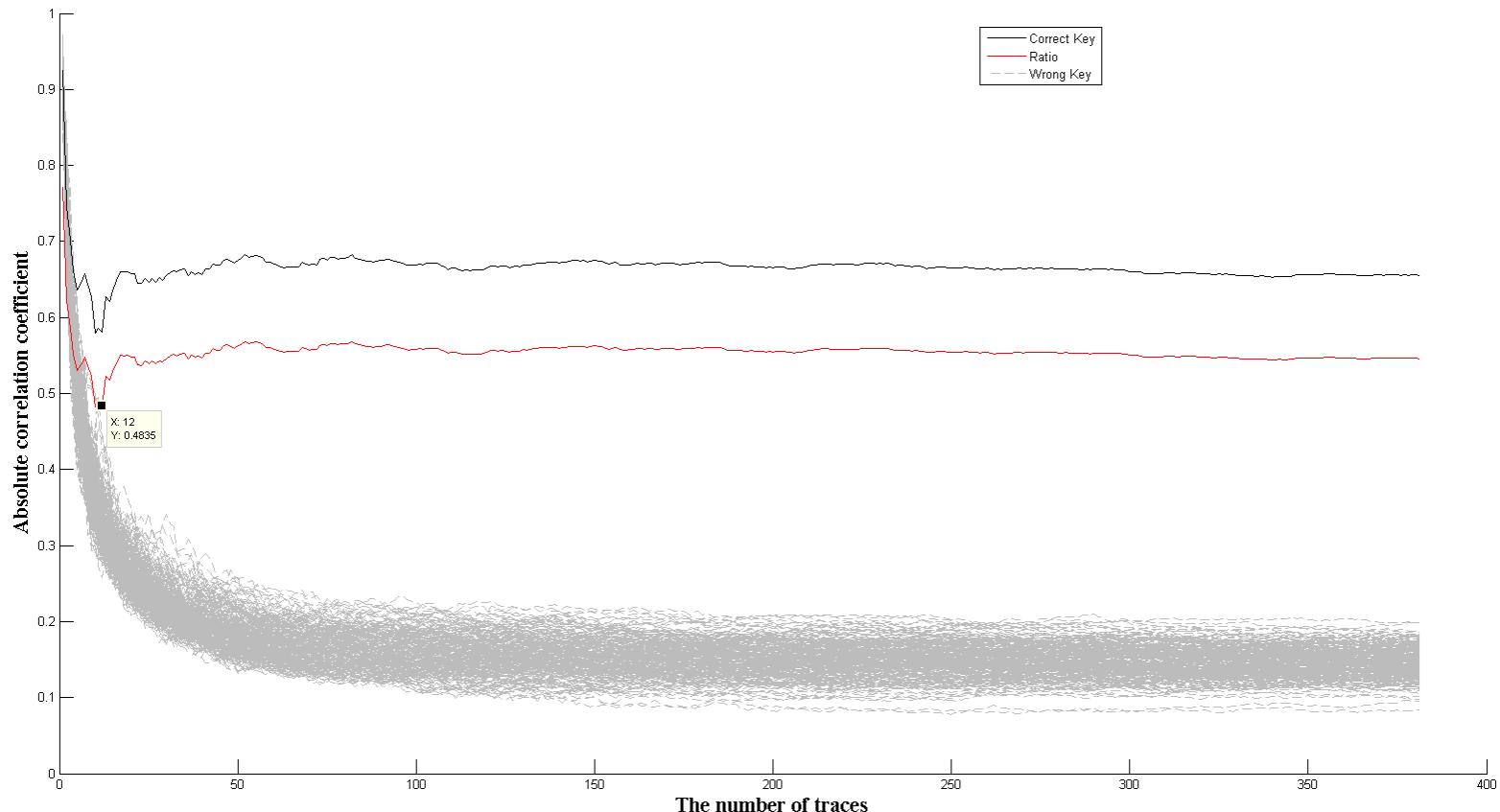
## ▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte 패형 증가에 따른 분석 결과



## ▣ 07\_Eight Shuffling AES / ChipWhisperer-Lite / 8 bit / 분석 결과

### ✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	16	8F	2B	4F	ED	AA	87	78	B2	CE	14	27	28	8C	67	C8

### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	3.29	2.90	3.72	3.51	3.06	3.35	3.00	2.29	2.50	3.24	3.49	3.73	2.47	3.73	2.54	3.65

### ✓ 최소 분석 파형 수

( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
파형 수	12	9	6	10	15	7	7	10	11	12	7	5	11	12	9	6

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 08

제공되는 정보는 하드웨어 대응기법인 **랜덤클록**이 적용된 8비트 기반 소프트웨어 AES-128 암호 알고리즘이 32비트 스마트카드에서 동작 할 때 소비되는 전력을 수집한 결과이다. 동작하는 8비트 기반 AES-128 암호 알고리즘은 아래 Algorithm 8과 같다.

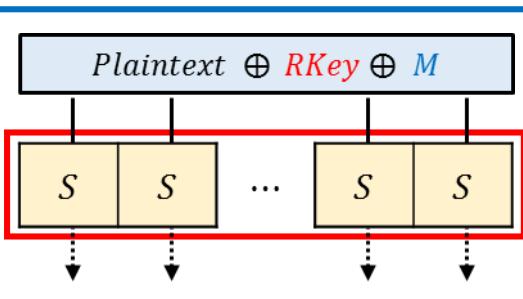
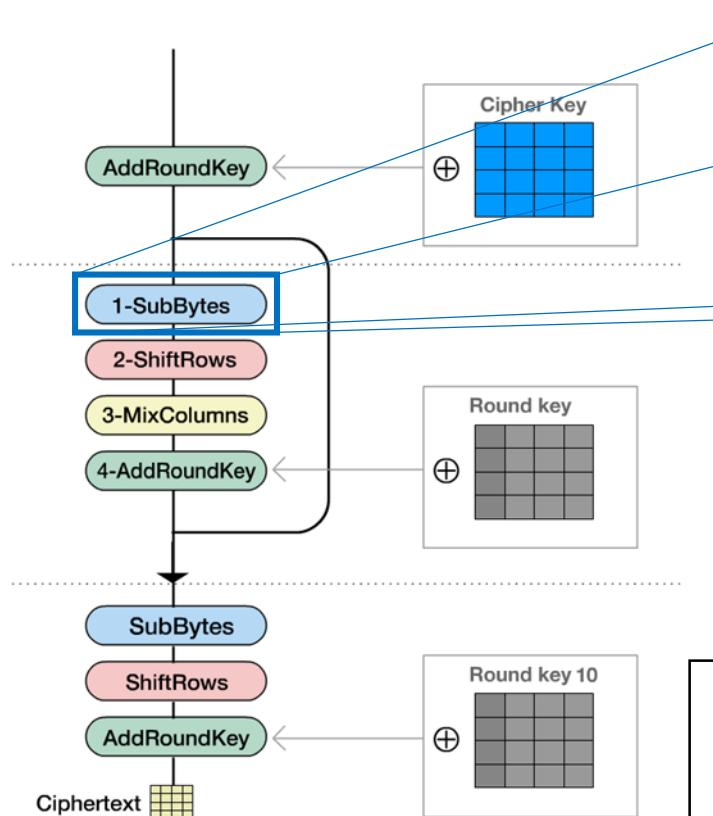
binary 파일 08\_Normal\_AES\_200tr\_70000pt.trace는 서로 다른 평문 및 동일한 암호화키에 대하여 AES-128 알고리즘이 200번 시행된 소비 전력이 포함되어 있는 파일이다.

08\_Normal\_AES\_200tr\_70000pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

- 1) 소비 전력 및 평문 정보를 이용하여 AES-128 암호 알고리즘 동작 시 사용된 암호화키(KEY)를 최소의 소비 전력 정보를 이용하여 찾으시오.(단, 최소 분석 파형 수는 10개 단위 측정)

## □ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 논리

### ❖ Normal AES



### ❖ S-box 출력 지점 분석

- Non-linear / 8 bit Guessing
- Random Clock 적용되었기 때문에 전처리 (Ex. 정렬) 필요

**중간 값**

```

t : S-box 위치
i : i 번째 파형

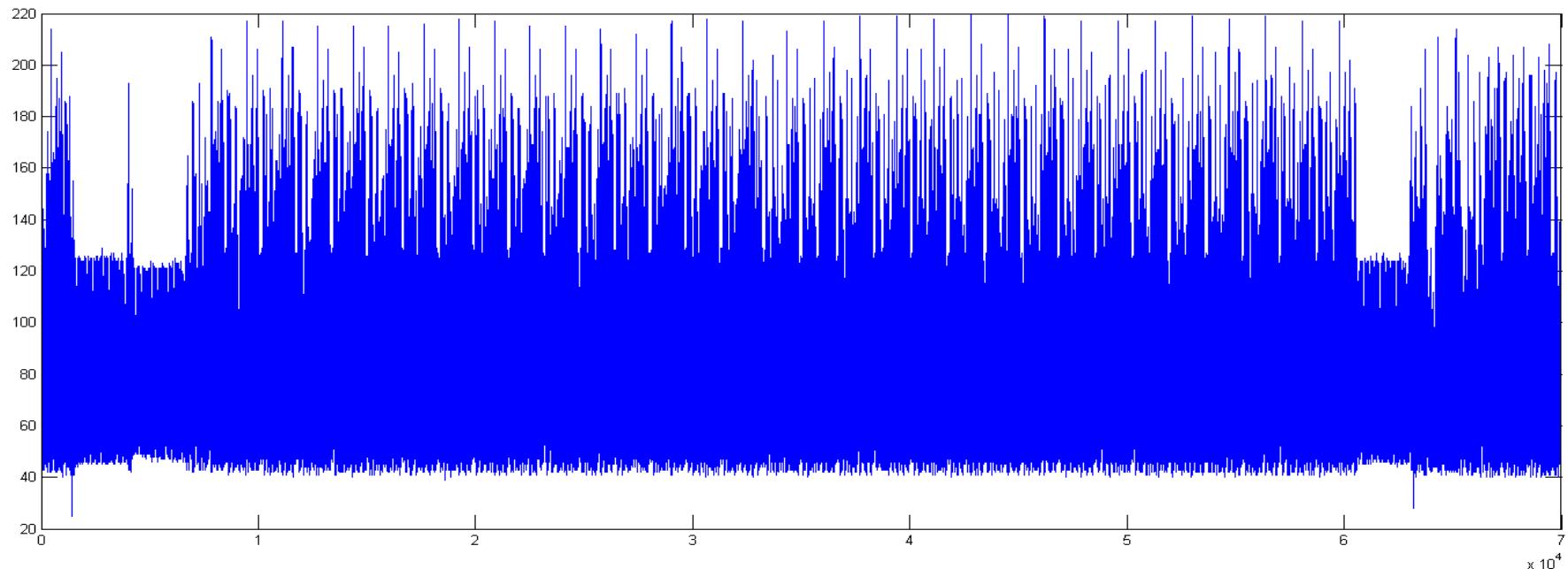
// 08_Normal_AES 중간값 /////////////////////////////////
key = _AES_SBOX_[plaintext[i][t] ^ (guess_key)];
// // //
///////////////////////////////

```

## ■ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

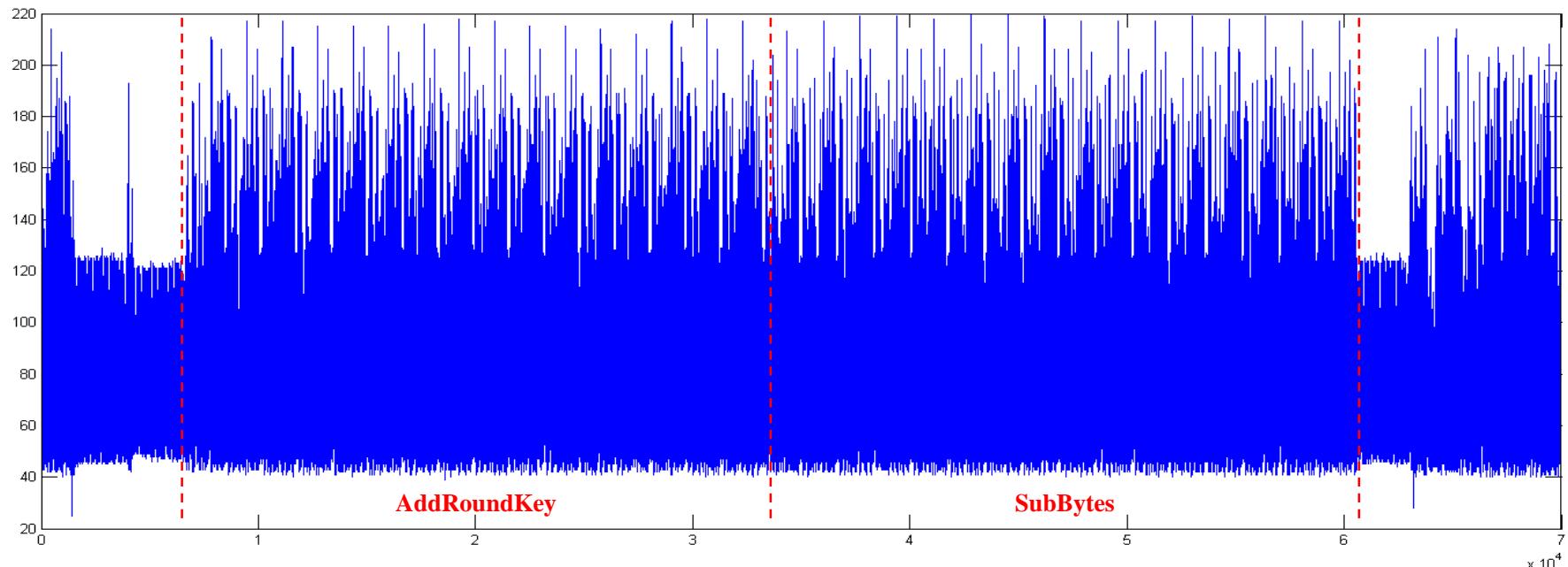
파형 정보	
파형 수	200
포인트 수	70,000



## ▣ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	200
포인트 수	70,000

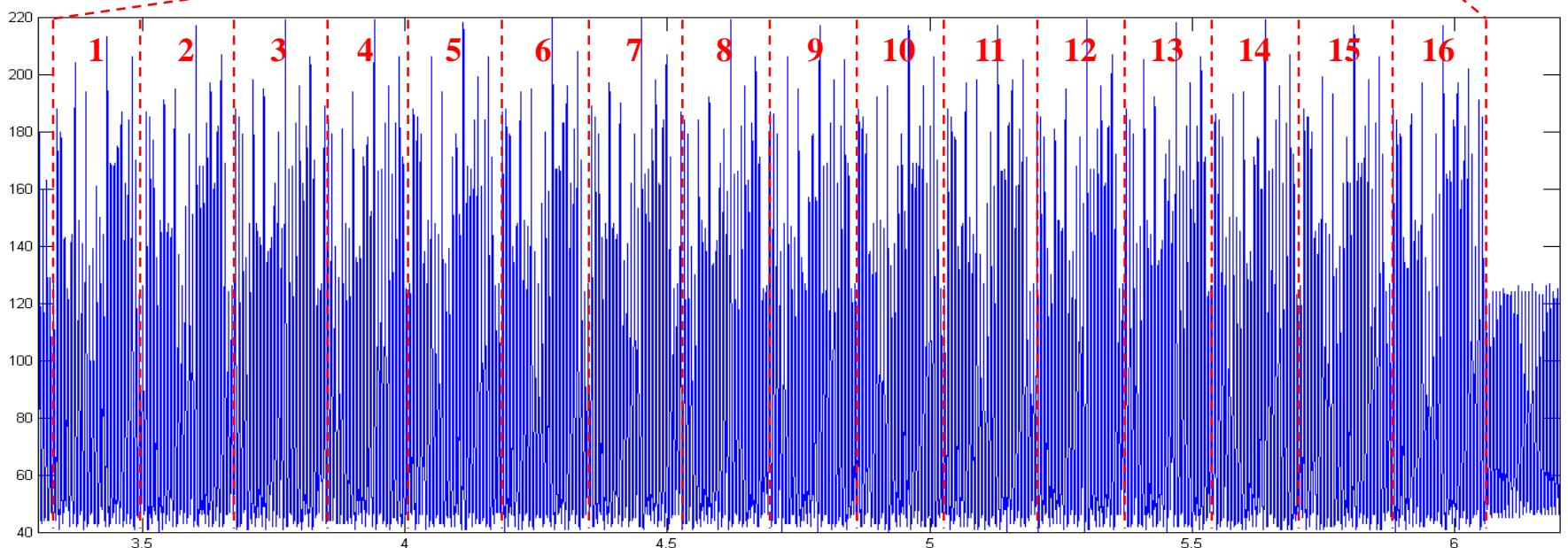
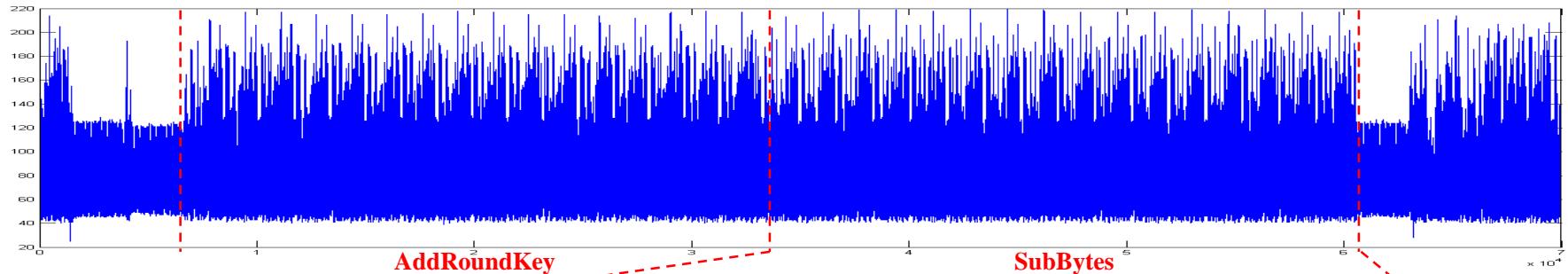


- ❖ 파형을 문제에서 제공하는 알고리즘과 비교하여 AddRoundKey & SubBytes 구분 가능
- ❖ AddRoundKey : 16개 & SubBytes : 16개 구분 가능

## ▣ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

❖ 전력 파형 SPA (1<sup>st</sup> trace)

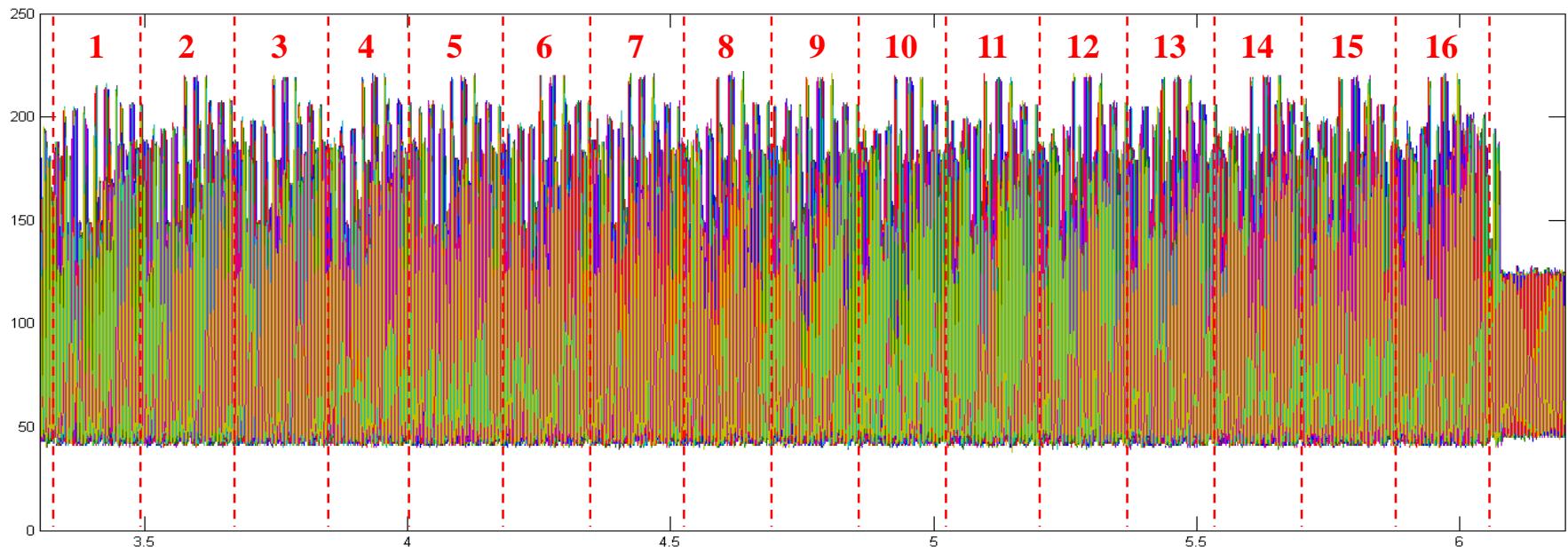
파형 정보	
파형 수	200
포인트 수	70,000



## ■ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

- ❖ 전력 파형 SPA (S-Box 확대 / Overlap traces)

파형 정보	
파형 수	200
포인트 수	70,000

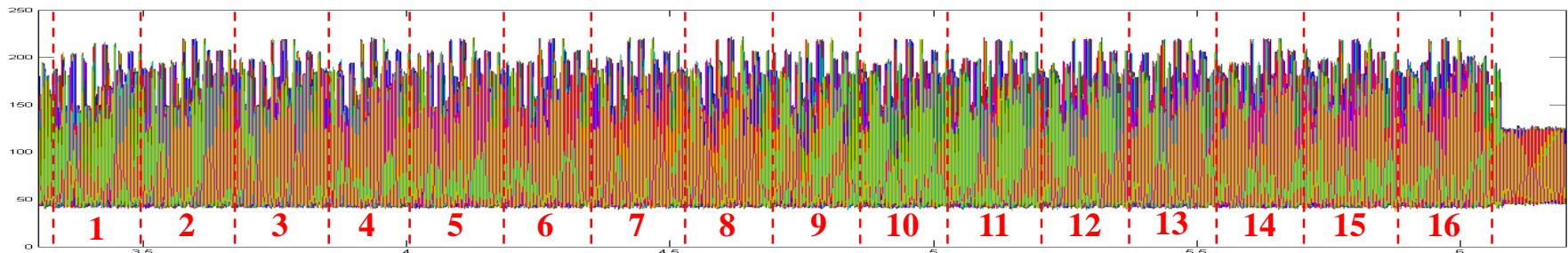


- ❖ 전체적으로 S-Box 예상 부분이 흔들려서 분석이 불가능
- ❖ 따라서 부분적으로 정렬을 맞춰 분석 진행
- ❖ 분석 효율을 위해 압축 & Moving Average 등 전처리 가능

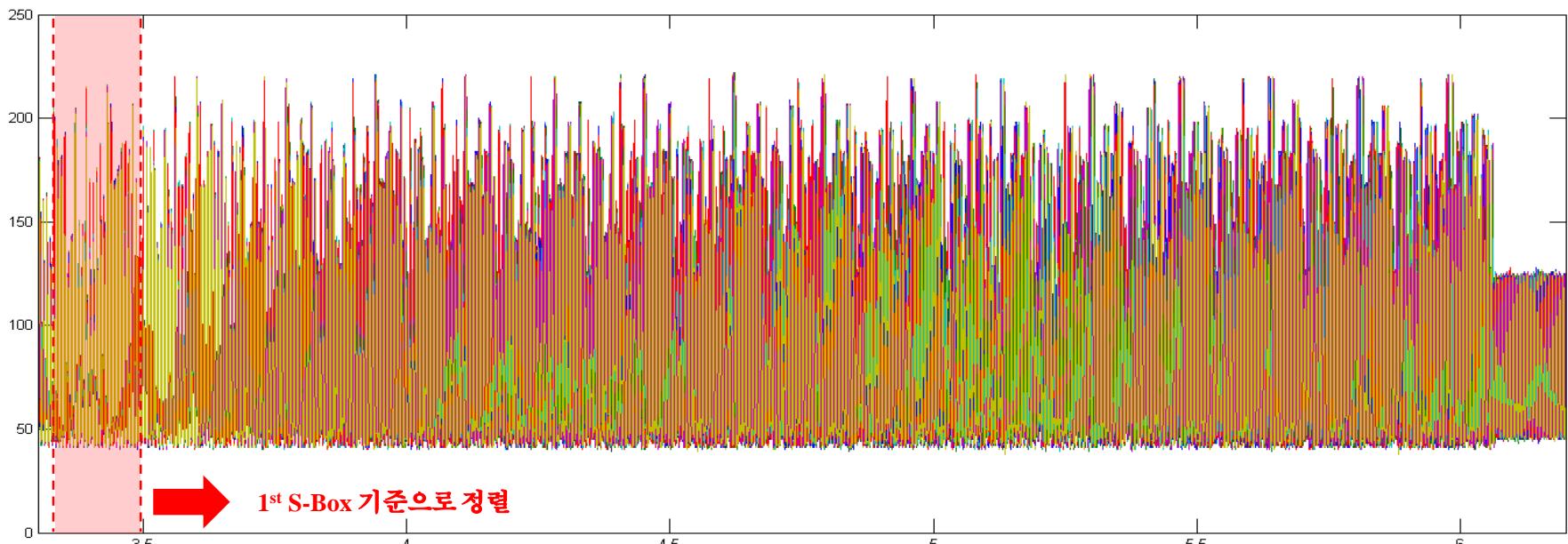
## 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

- ◆ 전력 파형 SPA (S-Box 확대 / Overlap traces)

파형 정보	
파형 수	200
포인트 수	70,000



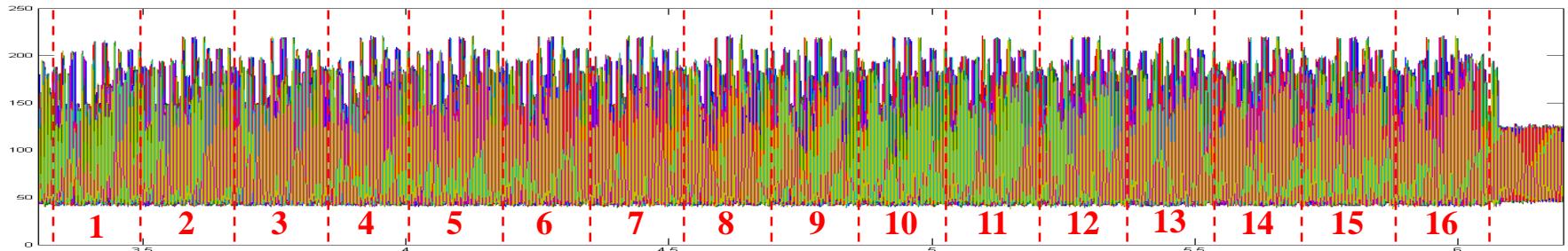
- ◆ 전력 파형 SPA (Aligned 1st S-Box / Overlap traces)



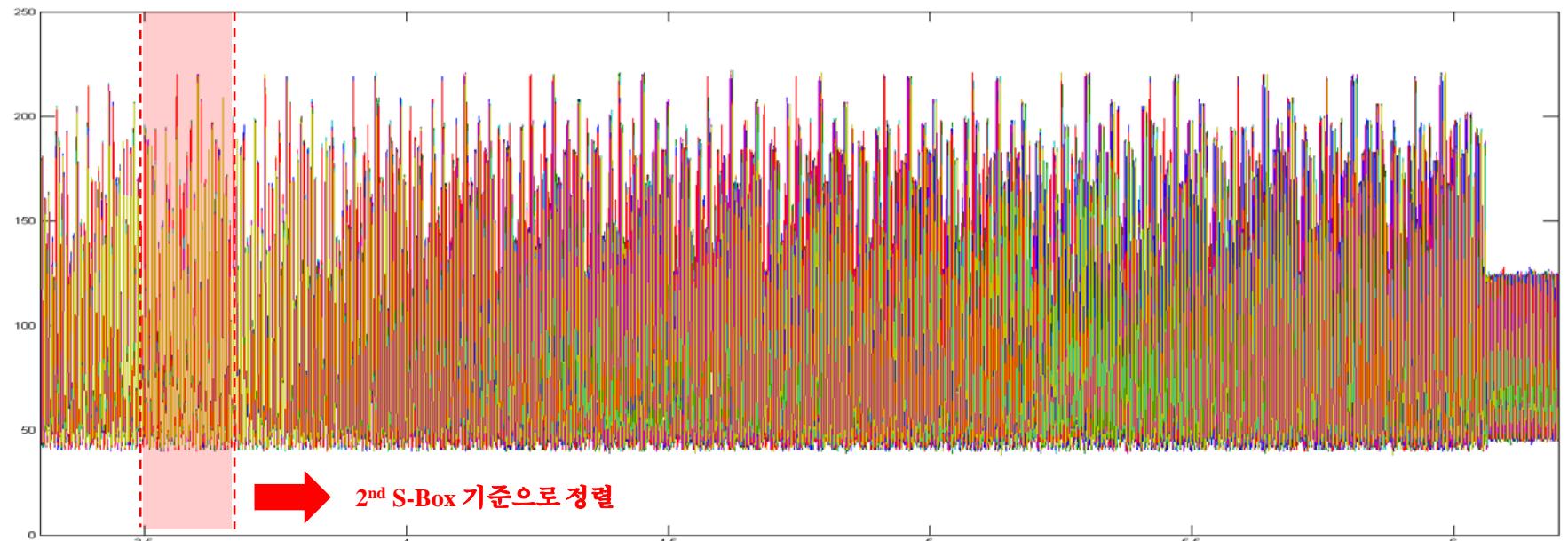
## 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

- ◆ 전력 파형 SPA (S-Box 확대 / Overlap traces)

파형 정보	
파형 수	200
포인트 수	70,000

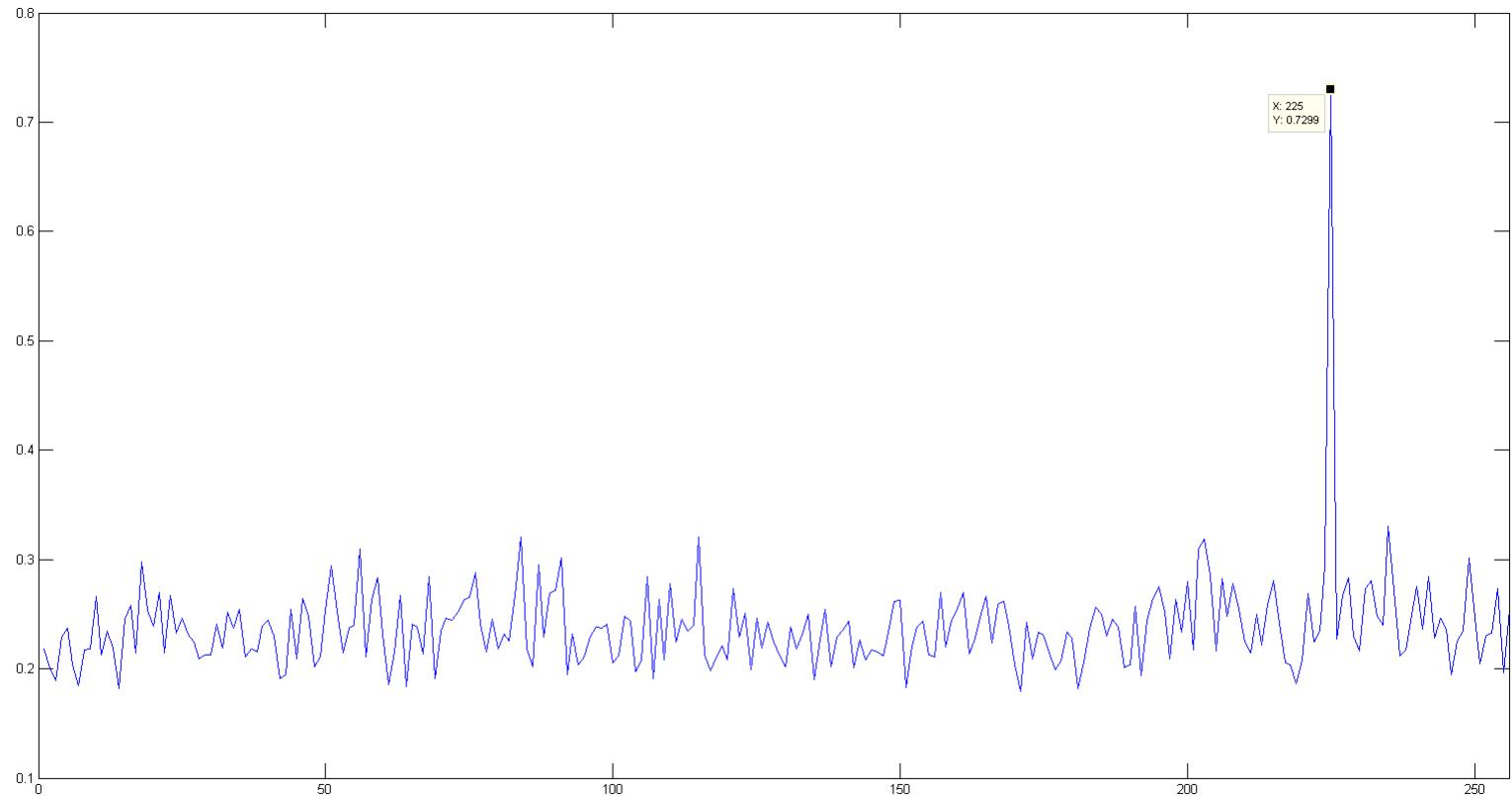


- ◆ 전력 파형 SPA (Aligned 2nd S-Box / Overlap traces)



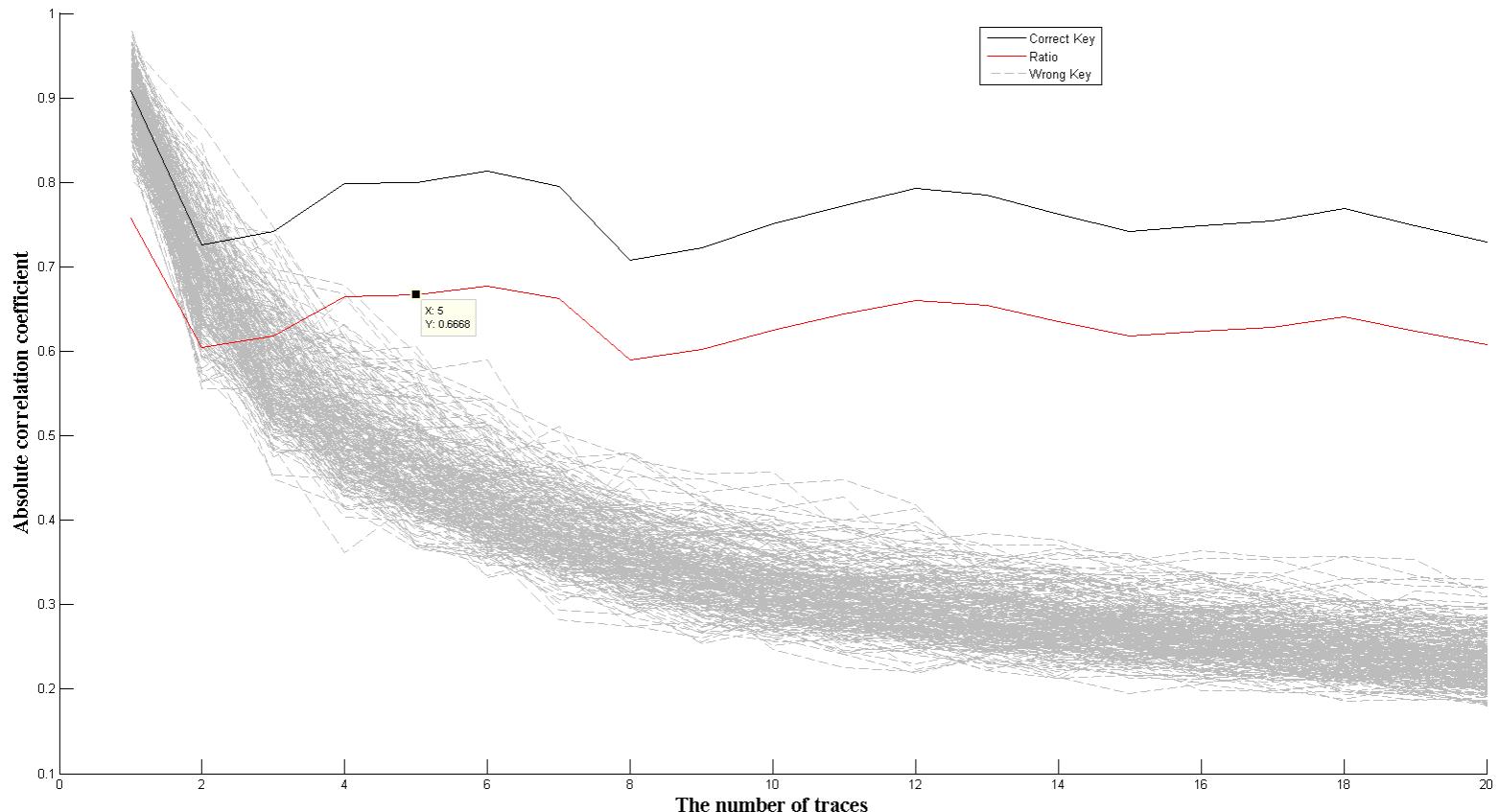
## ▣ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte Maxpeak 분석 결과



## ▣ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

### ❖ (1) S-Box 1<sup>st</sup> Byte 패형 증가에 따른 분석 결과



## ▣ 08\_Random Clock AES / ATmega-328P / 8 bit / 분석 결과

### ✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79

### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	2.20	1.68	1.67	1.61	1.52	1.42	1.51	1.62	1.67	1.65	1.33	1.76	1.65	1.49	1.23	1.28

### ✓ 최소 분석 파형 수

( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
파형 수	5	11	5	12	12	14	13	13	10	10	15	9	10	7	19	16

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

No	ARIA		
01		분석 장비	ATmega-128
		분석 대상	Normal ARIA
		구현 비트	08비트
02		분석 장비	ARM 902T
		분석 대상	Normal ARIA
		구현 비트	08비트
03		분석 장비	ATmega-128
		분석 대상	First-Order Masked ARIA
		구현 비트	08비트

No	LEA		
04		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal LEA
		구현 비트	16비트

No	SEED		
05		분석 장비	ChipWhisperer-Lite
		분석 대상	Normal SEED
		구현 비트	08비트
06		분석 장비	ChipWhisperer-Lite
		분석 대상	First-Order Masked SEED
		구현 비트	08비트

No	AES		
07		분석 장비	ChipWhisperer-Lite
		분석 대상	Eight-Shuffling AES
		구현 비트	08비트
08		분석 장비	ATmega-328P
		분석 대상	Normal AES with random clock cycle
		구현 비트	08비트
09		분석 장비	ATmega-328P
		분석 대상	First-Order Masked AES
		구현 비트	08비트

## ▣ 스마트디바이스 부채널 분석 경진대회 문제

### 문제 09

제공되는 정보는 **마스킹이 적용된** 8비트 기반 소프트웨어 AES-128 암호 알고리즘이 32비트 스마트카드에서 동작 할 때 소비되는 전력을 아래의 알고리즘을 타겟으로 수집한 결과이다.

binary 파일 09\_First-Order\_Masked\_AES\_2000tr\_90000pt.trace는 서로 다른 평문 및 동일한 암호화 키에 대하여 AES-128 알고리즘이 2000번 시행된 소비 전력이 포함되어 있는 파일이다.

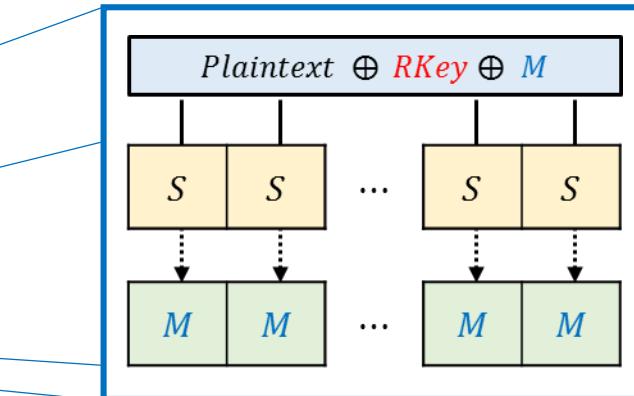
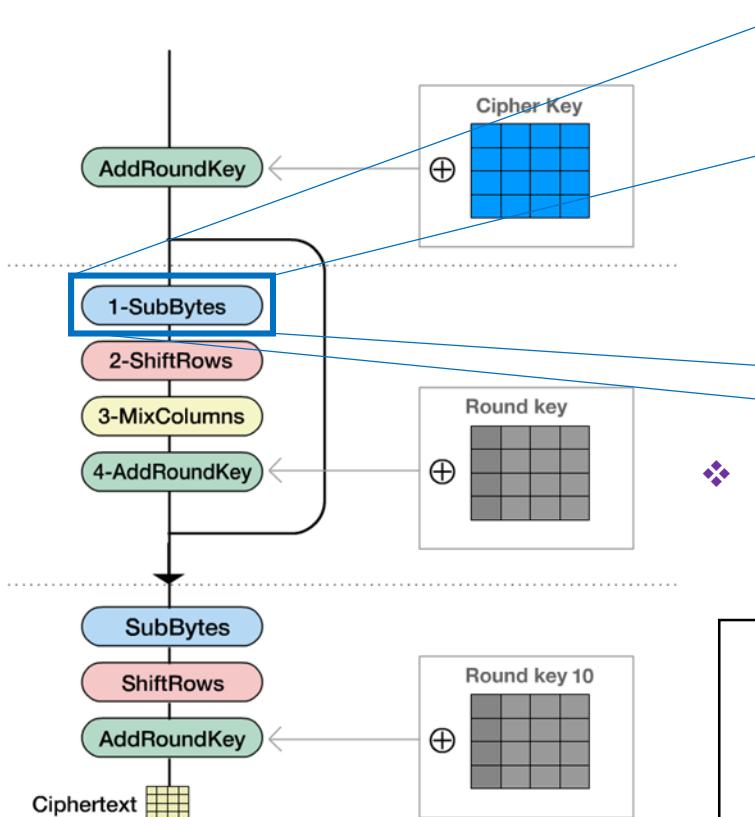
09\_First-Order\_Masked\_AES\_2000tr\_90000pt\_plain.txt는 각 알고리즘 시행에 사용된 평문 정보이다.

1) 소비 전력 및 평문 정보를 이용하여 AES-128 암호 알고리즘 동작 시 사용된 암호화키(KEY)를 전력 정보를 이용하여 찾으시오.

## 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

### ❖ Masked AES

- ✓ 과정 조합 방법 (Preprocessing)  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
  1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$



❖ AddRoundKey 와 MS-Box 마스킹이 동일하게 적용  
따라서 아래 두 가지 공격 방법이 모두 가능

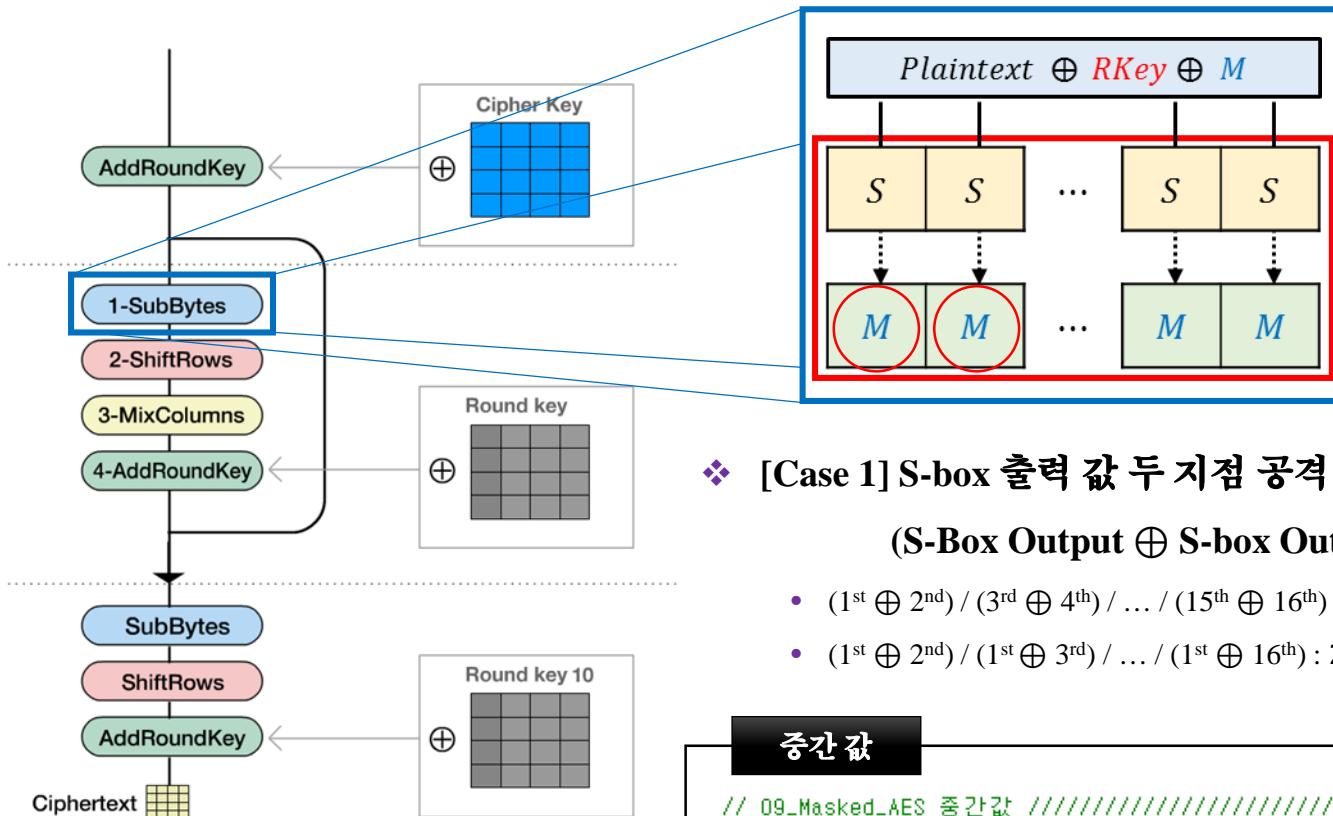
[Case 1] S-box 출력 값 두 지점 공격  
(S-Box Output  $\oplus$  S-Box Output)

[Case 2] Hamming Distance 논리 및 S-box 입·출력 공격  
(S-Box Input  $\oplus$  S-Box Output)

## 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

### ❖ Masked AES

- ✓ **파형 조합 방법 (Preprocessing)**  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
  1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$



### ❖ [Case 1] S-box 출력 값 두 지점 공격 (S-Box Output $\oplus$ S-box Output)

- $(1^{\text{st}} \oplus 2^{\text{nd}}) / (3^{\text{rd}} \oplus 4^{\text{th}}) / \dots / (15^{\text{th}} \oplus 16^{\text{th}})$  :  $2^{16}$  guessing  $\times$  8 또는
- $(1^{\text{st}} \oplus 2^{\text{nd}}) / (1^{\text{st}} \oplus 3^{\text{rd}}) / \dots / (1^{\text{st}} \oplus 16^{\text{th}})$  :  $2^{16} + 2^8$  guessing  $\times$  14

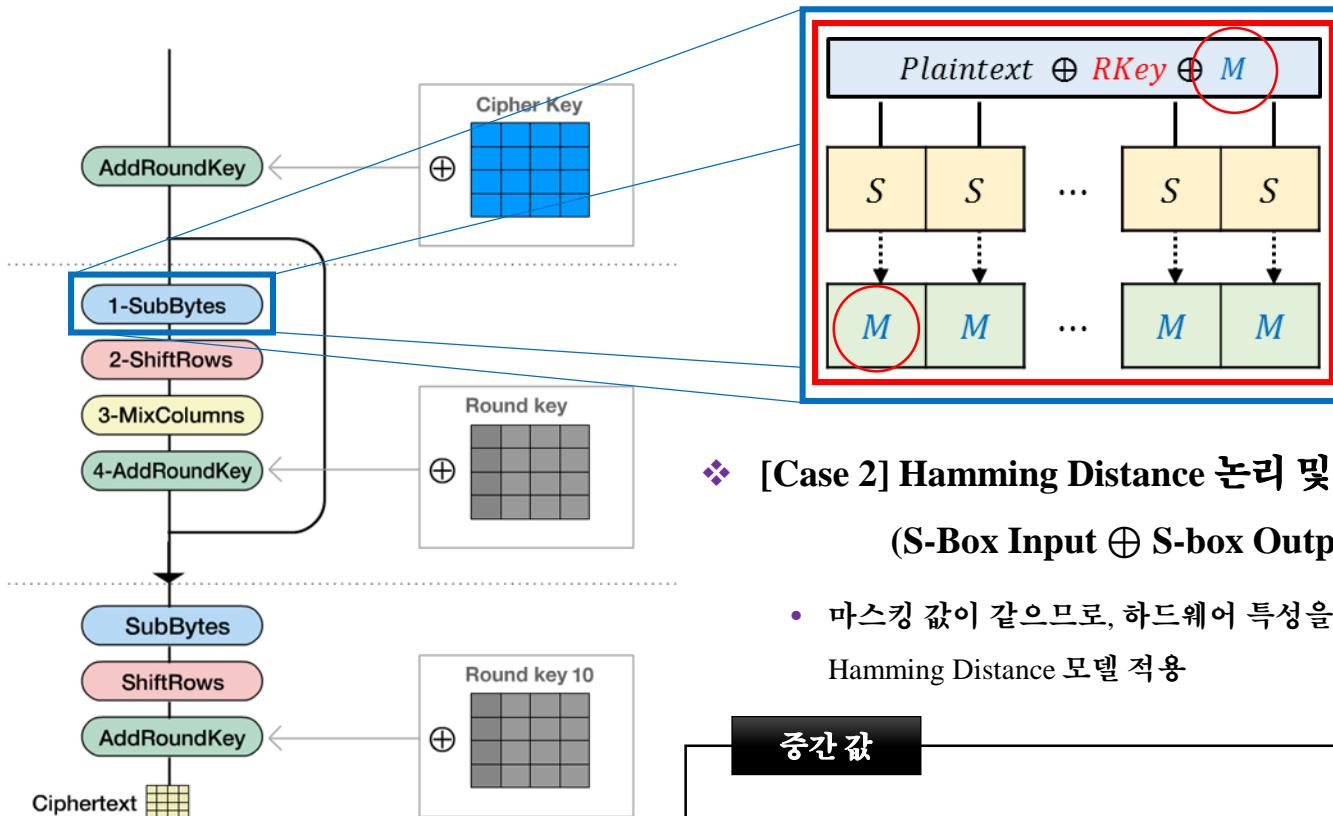
**중간 값**

```
// 09_Masked_AES 중간값 //////////////////////////////////////////////////////////////////
key = _AES_SBOX_[plaintxt[i][t] ^ (guess_key >> 8) // t : S-box 위치
              ^ _AES_SBOX_[plaintxt[i][t+1] ^ (guess_key >> 0)]; // i : i 번째 파형
//////////////////////////////////////////////////////////////////
```

## 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

### ❖ Masked AES

- ✓ 과정 조합 방법 (Preprocessing)  
(모든 방법 다 분석 가능 / 자료는 3번 선택)
  1. 곱셈 조합 :  $| \{C_i - E[C_i]\} \cdot \{C_j - E[C_j]\} |$
  2. 차분 조합 :  $| \{C_i - E[C_i]\} - \{C_j - E[C_j]\} |$
  3. 수정된 차분 조합 :  $| C_i - C_j |$



### ❖ [Case 2] Hamming Distance 논리 및 S-box 입·출력 공격 (S-Box Input $\oplus$ S-box Output)

- 마스킹 값이 같으므로, 하드웨어 특성을 이용하여 Hamming Distance 모델 적용

중간 값

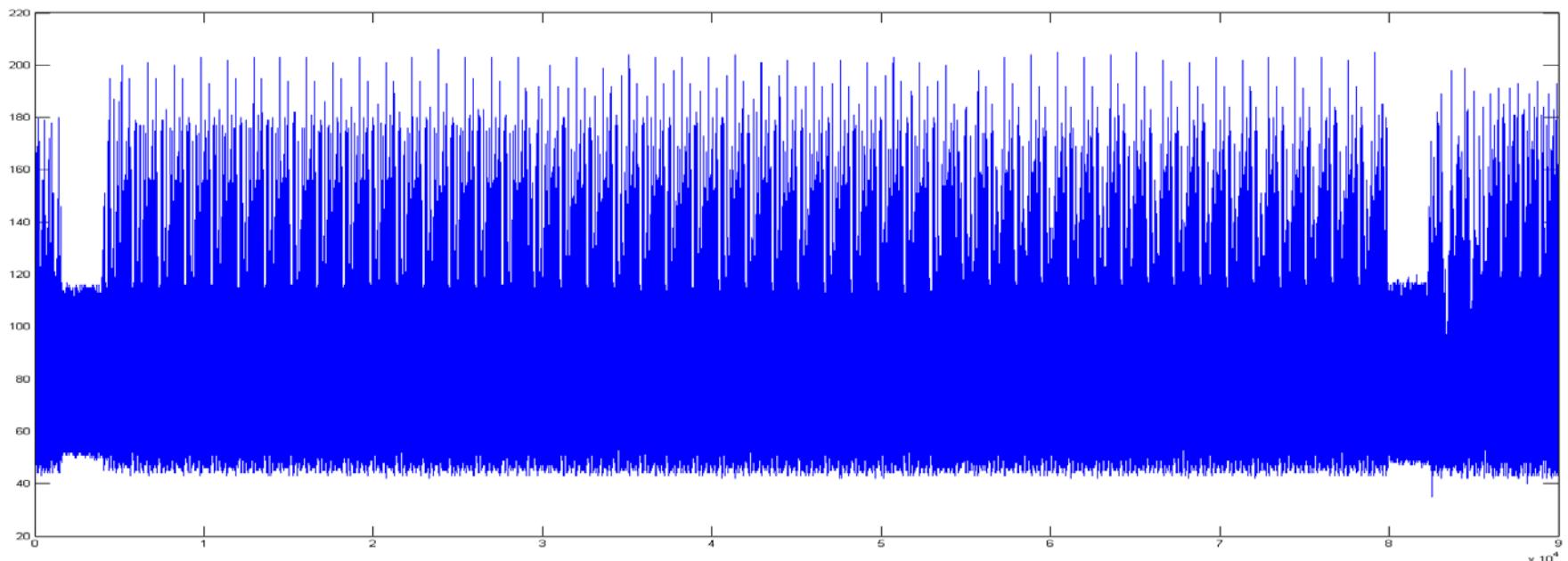
t : S-box 위치  
i : i 번째 파형

```
// 09_Masked_AES 중간값 /////////////////////////////////////////////////////////////////////
key = plaintext[i][t] ^ guess_key ^ _AES_SBOX_[plaintext[i][t] ^ (guess_key)]; /////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
```

## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

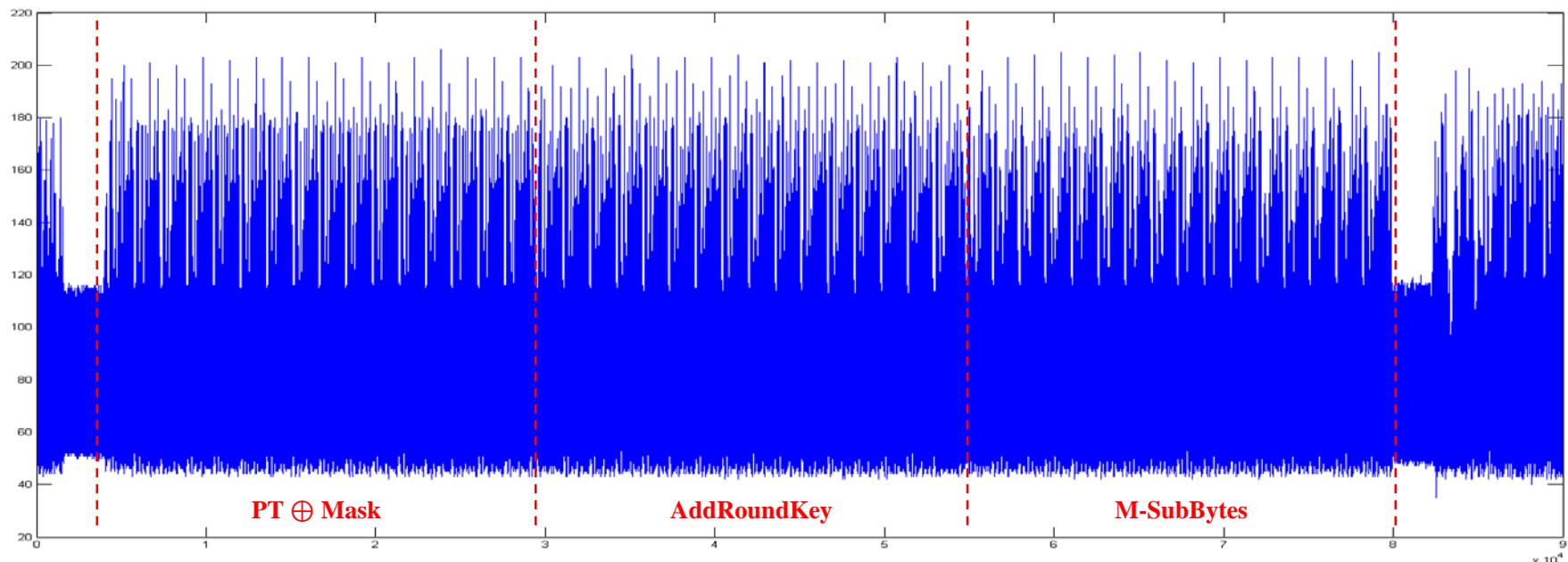
파형 정보	
파형 수	2,000
포인트 수	90,000



## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

- ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	2,000
포인트 수	90,000

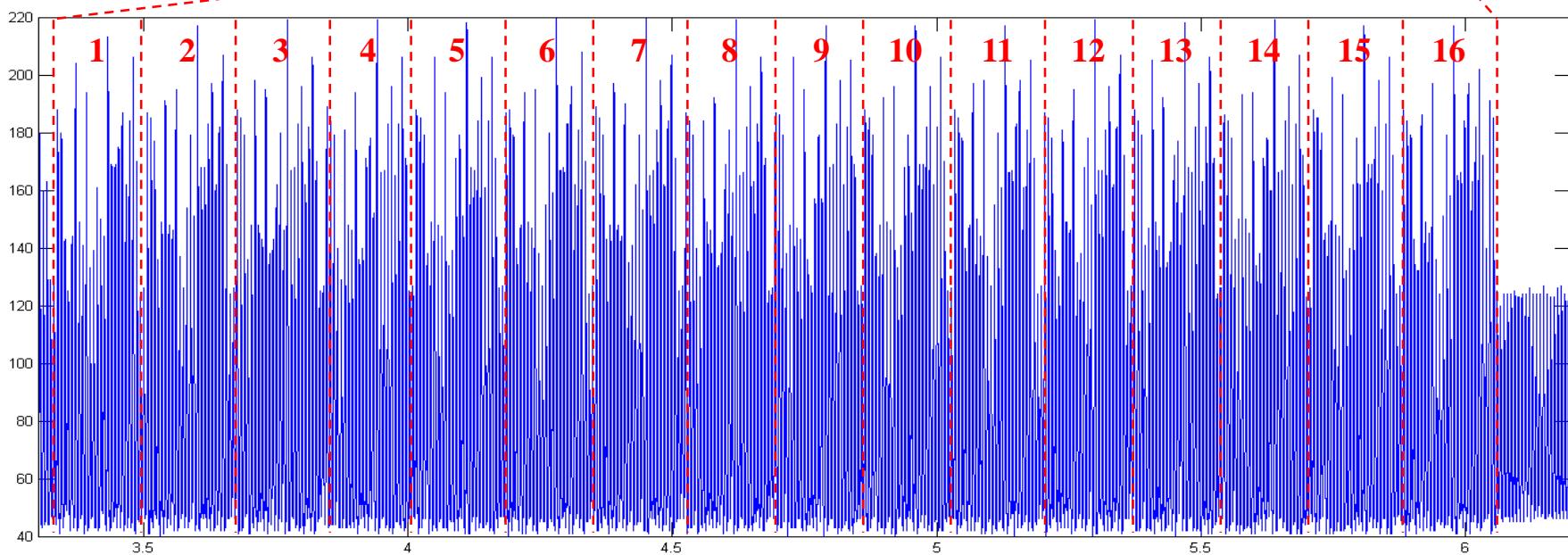
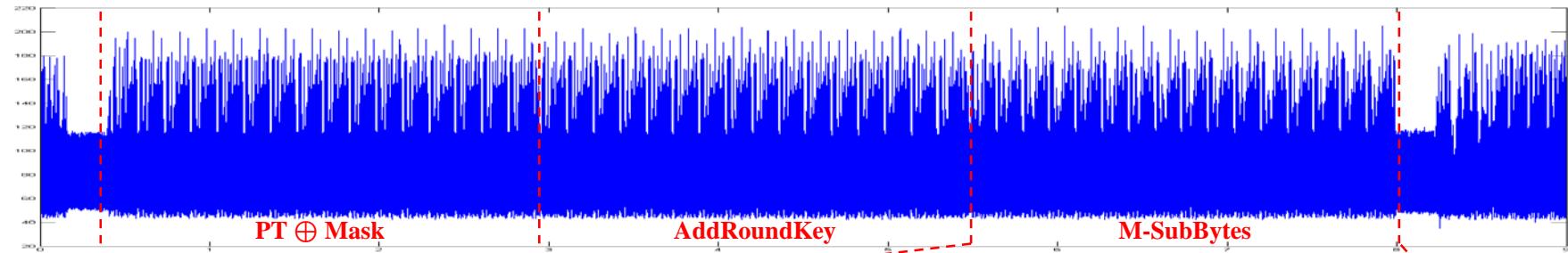


- ❖ SPA와 문제에 주어진 알고리즘 비교를 통해 Masked S-box 부분 파악 가능  
(대략 54000–80000pts)

## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 논리

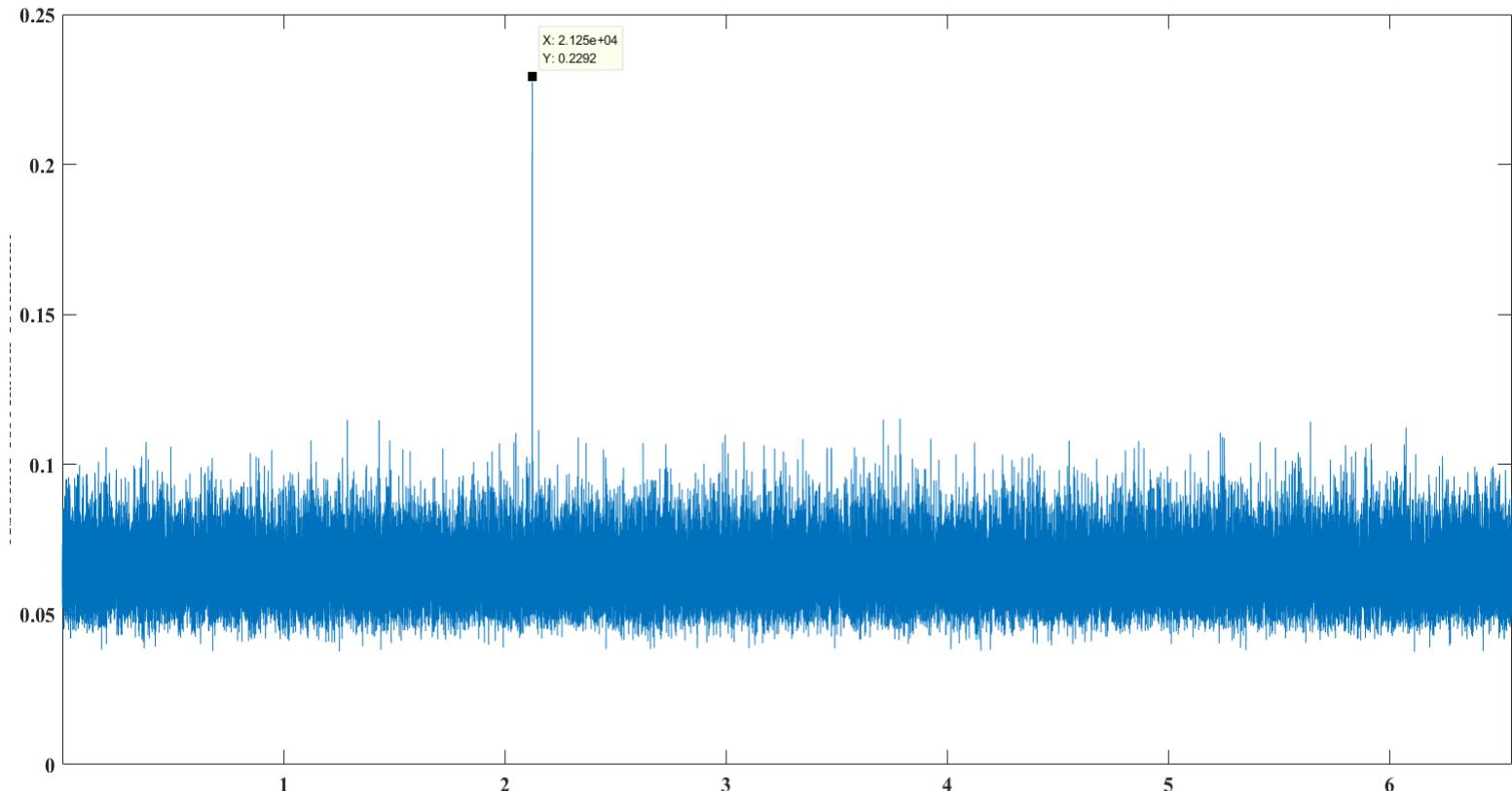
### ❖ 전력 파형 SPA (1<sup>st</sup> trace)

파형 정보	
파형 수	2,000
포인트 수	90,000



## ▣ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

### ❖ [Case 1] S-Box 1<sup>st</sup> & 2<sup>nd</sup> Byte Maxpeak 분석 결과



## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

### ✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	53	01	02	03	04	05	06	07	08	09	0F	0E	0D	0C	0B	0A

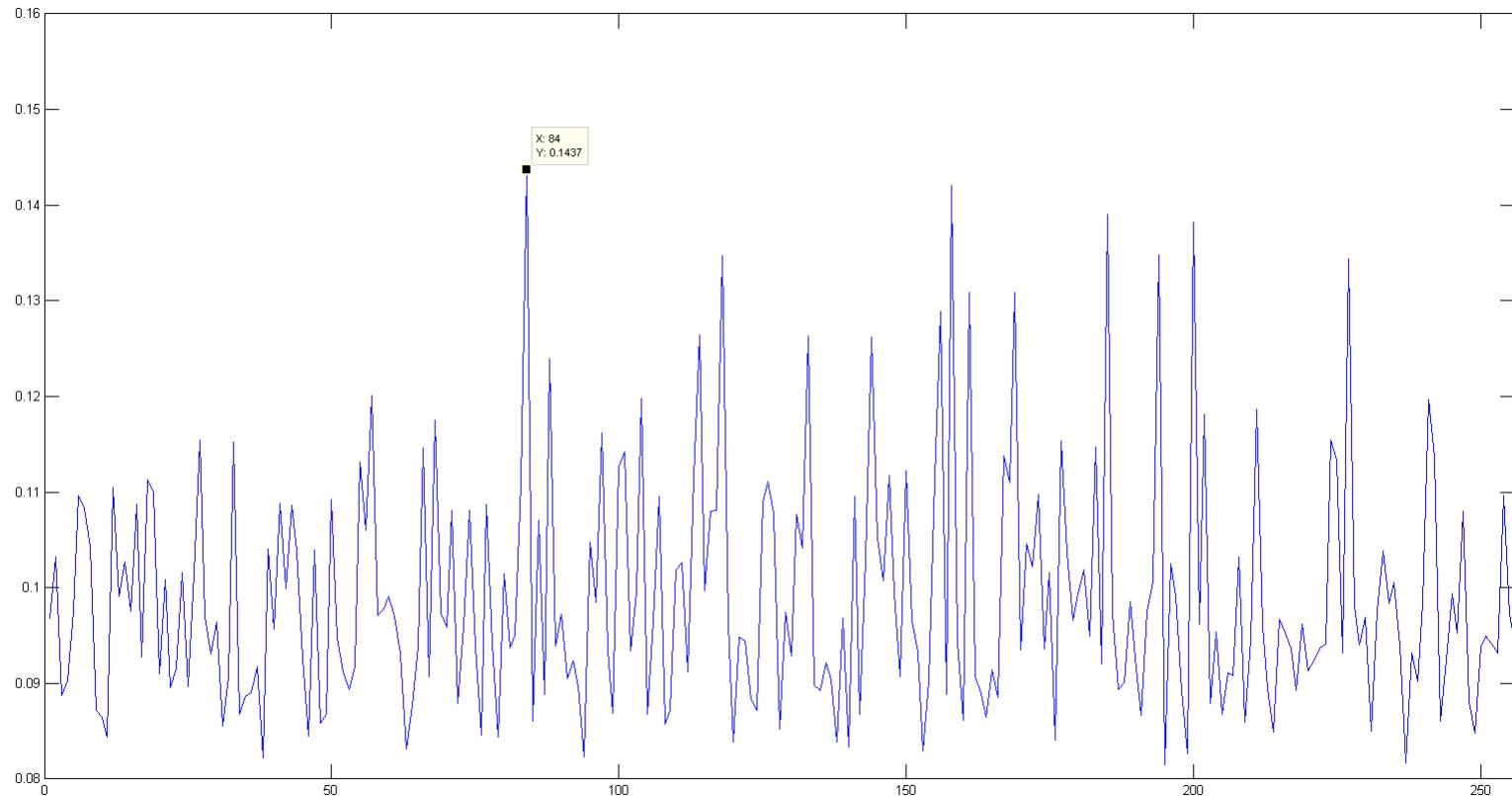
### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	2.29	2.34	2.18	2.73	2.17	2.26	2.01	2.80	2.34	2.37	2.26	2.23	2.13	1.84	1.86	

- ❖ 본 실험에서는 처음 두 바이트의 조합으로 1<sup>st</sup>, 2<sup>nd</sup> Byte 키를 찾고 ( $2^{16}$  Guessing)  
 이후에는 찾은 1<sup>st</sup> Byte 와 나머지 Byte 의 조합 ( $2^8 \times 14$  Guessing) 으로 분석

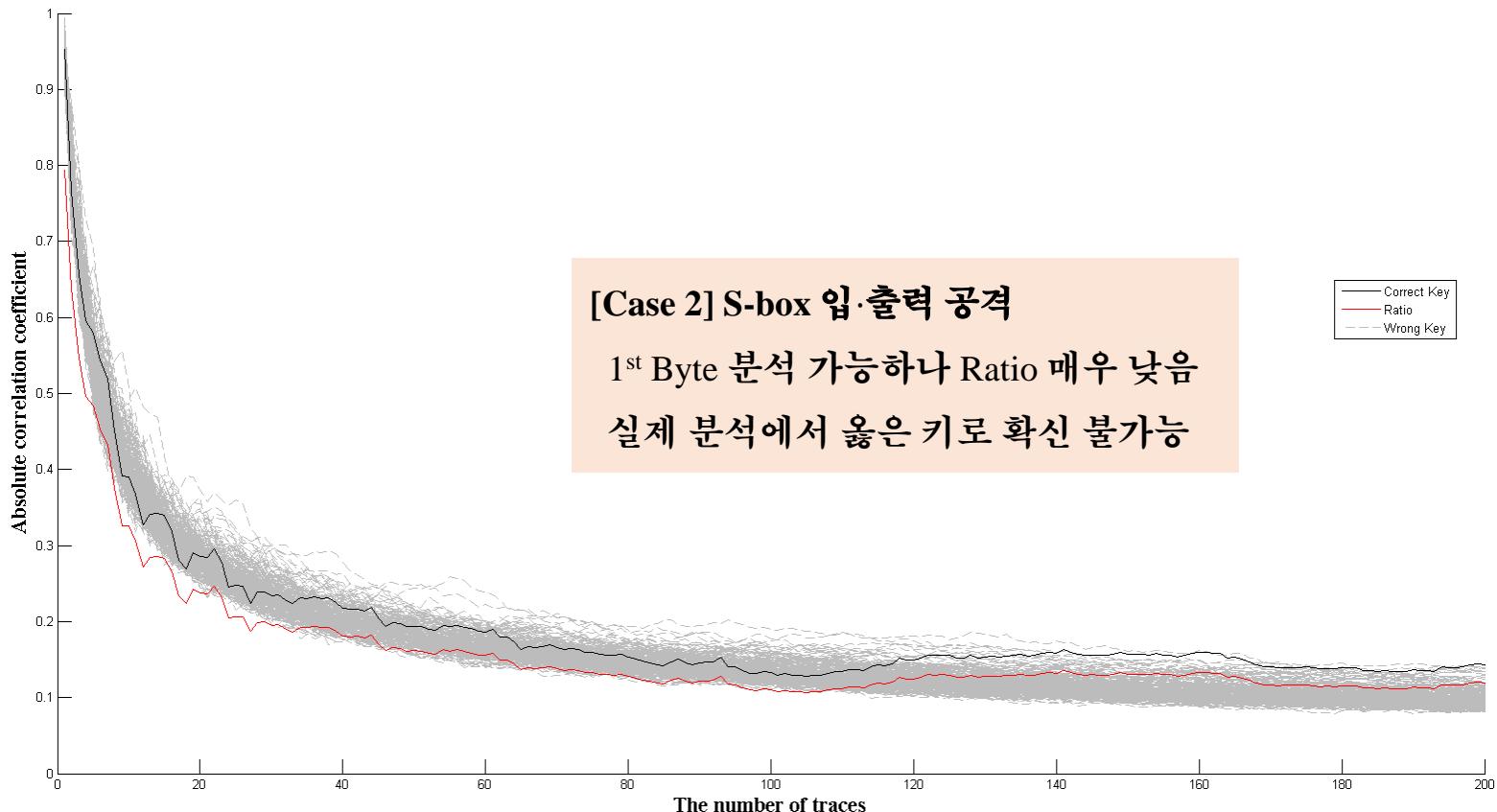
## ▣ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

### ❖ [Case 2] S-Box 1<sup>st</sup> Byte Maxpeak 분석 결과



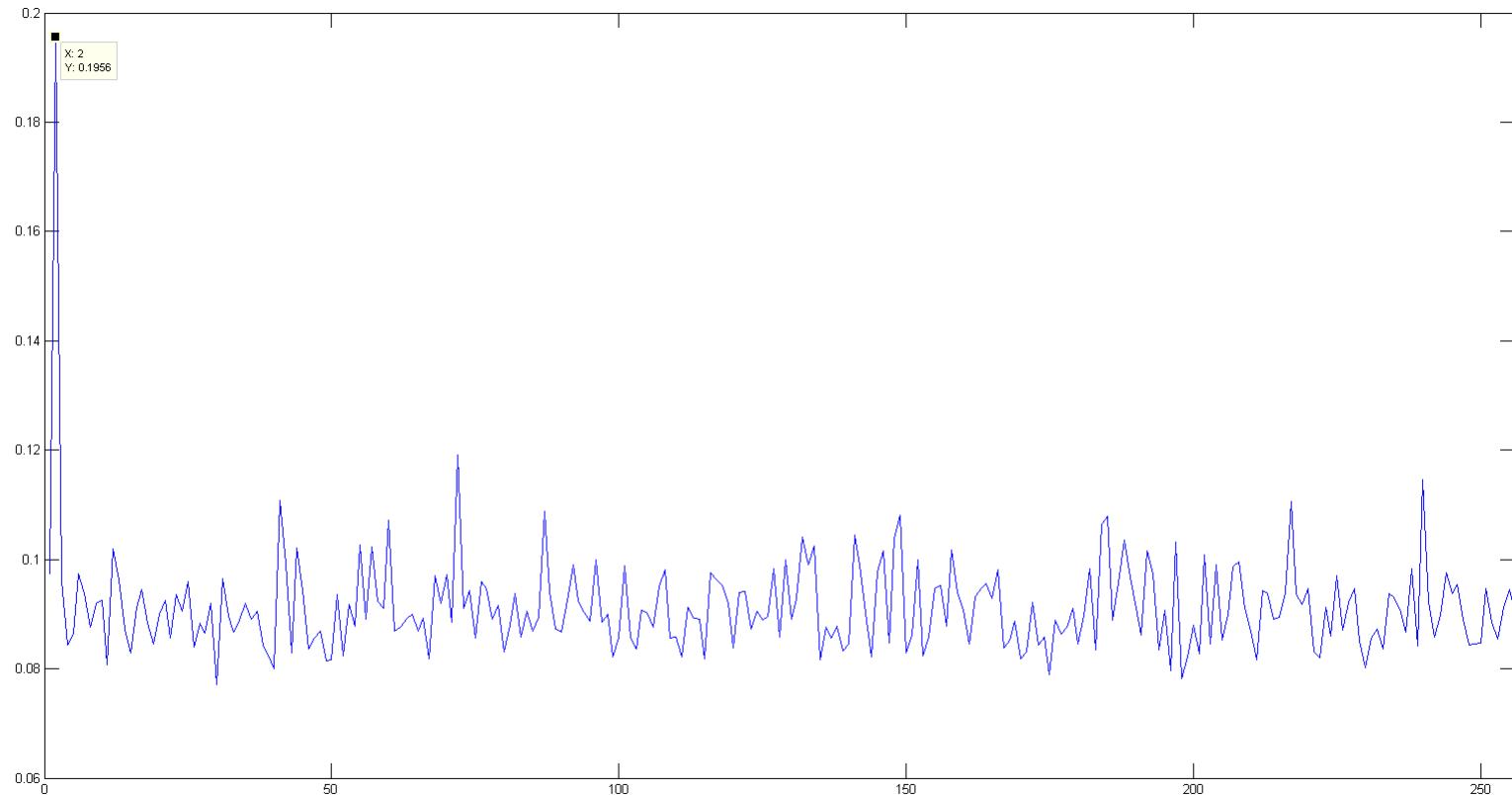
## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

- ❖ [Case 2] S-Box 1<sup>st</sup> Byte 파형 증가에 따른 분석 결과



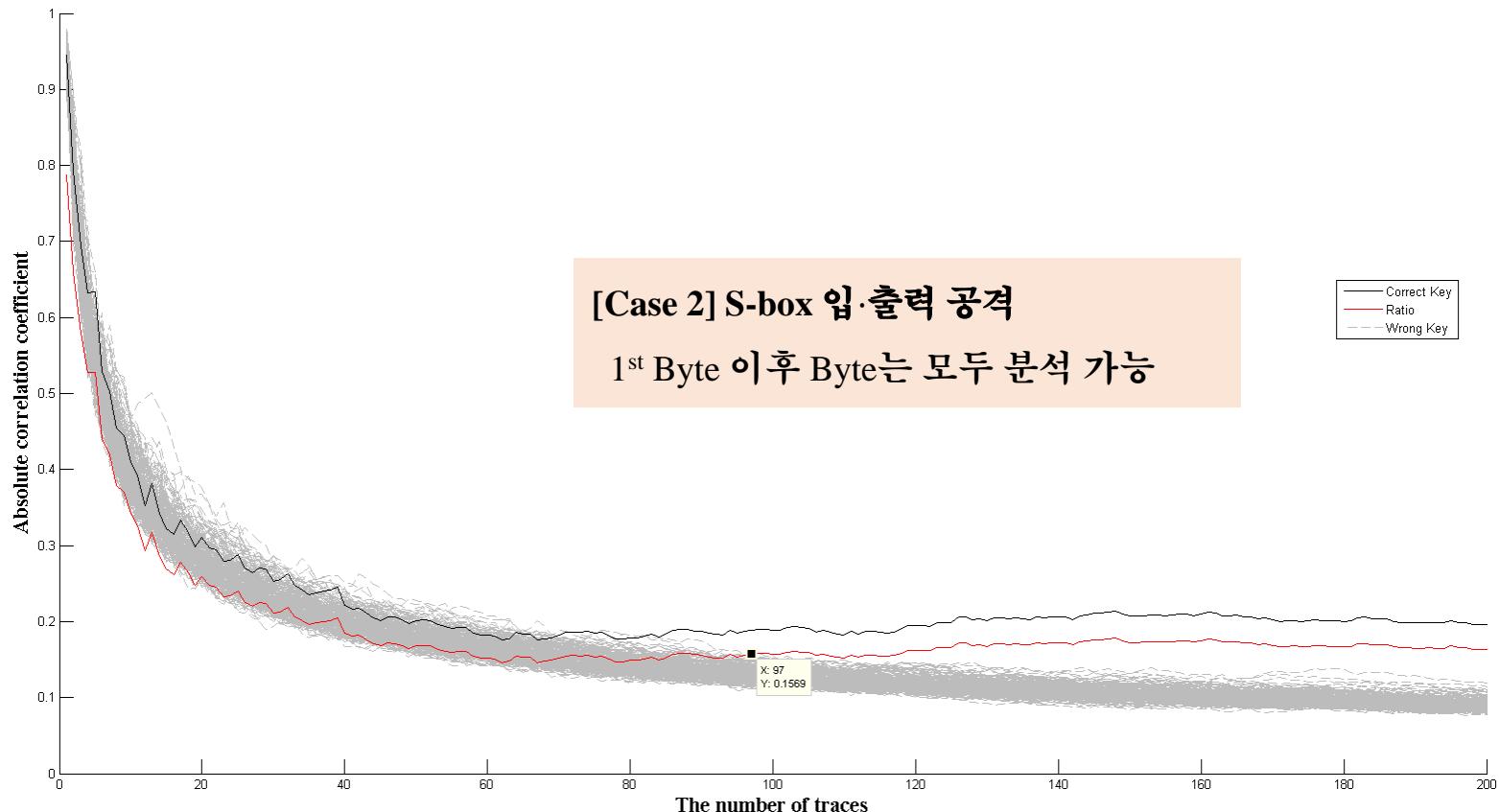
## ▣ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

### ❖ [Case 2] S-Box 2<sup>nd</sup> Byte Maxpeak 분석 결과



## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

- ❖ [Case 2] S-Box 2<sup>nd</sup> Byte 파형 증가에 따른 분석 결과



## ■ 09\_Masked AES / ATmega-328P / 8 bit / 분석 결과

### ✓ 분석된 키

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Key	53	01	02	03	04	05	06	07	08	09	0F	0E	0D	0C	0B	0A

### ✓ Ratio

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
Ratio	1.01	1.79	1.61	1.60	1.59	1.44	1.61	1.60	1.82	1.63	1.71	1.21	1.83	1.46	1.55	1.25

### ✓ 최소 분석 파형 수

( 단위 10 )

	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>	11 <sup>th</sup>	12 <sup>th</sup>	13 <sup>th</sup>	14 <sup>th</sup>	15 <sup>th</sup>	16 <sup>th</sup>
파형 수	115	93	107	60	127	116	50	151	68	120	93	106	83	136	106	107

## ▣ 최종 분석 키

연번	알고리즘	대응기법	제출문제	
01	ARIA	Normal	1. CPA	<b>28 43 07 1A 0F 10 89 A5 9E 54 9E C1 00 A2 19 A1</b>
02	ARIA	Normal	1. SPA 2. CPA	<b>63 CD F6 C4 8E A6 AB 28 84 19 46 87 62 F0 AE E4</b>
03	ARIA	First-Order Masking	1. CPA	<b>B8 7F 05 12 CD 52 DD CB EE 38 6F 90 EE CD 38 A9</b>
04	LEA	Normal	1. CPA	<b>97 C8 3A 77 3C FA 52 E2 5D 69 C4 6C 3C FA 52 E2 C8 E9 3F EA 3C FA 52 E2</b>
05	SEED	Normal	1. SPA 2. CPA [Xor된 키] 3. CPA [라운드 키]	<b>0B 9E 9D 46 A9 EC EE F9 A2 72 73 BF</b>
06	SEED	First-Order Masking	1. CPA [Xor된 키] 2. CPA [라운드 키]	<b>F3 14 80 4E 9E 82 98 A8 6D 96 18 E6</b>
07	AES	Eight-Shuffling	1. CPA	<b>16 8F 2B 4F ED AA 87 78 B2 CE 14 27 28 8C 67 C8</b>
08	AES	Normal with random clock cycle	1. CPA	<b>E0 32 3A 0A 49 06 24 5C C2 D3 AC 62 91 95 E4 79</b>
09	AES	First-Order Masking	1. CPA	<b>53 01 02 03 04 05 06 07 08 09 0F 0E 0D 0C 0B 0A</b>

