Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.2, Apr. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.2.287

더미 기반 부채널 분석 대응기법 신규 취약점*
- Case Study: XMEGA

이 종 혁,[1†] 한 동 국[1,2‡]
[1]국민대학교 금융정보보안학과, [2]국민대학교 정보보안암호수학과

Novel Vulnerability against Dummy Based Side-Channel Countermeasures
- Case Study: XMEGA*

JongHyeok Lee,[1†] Dong-Guk Han[1,2‡]
[1]Dept. of Financial Information Security, Kookmin University,
[2]Dept. of Information Security, Cryptology, and Mathematics, Kookmin University

요 약

부채널 분석에 안전하게 암호 알고리즘을 구현하는 경우, 설계자들은 일반적으로 1차 마스킹 기법과 하이딩 기법을 혼합하여 사용한다. 1차 마스킹 기법과 하이딩 기법을 혼합하여 사용한 구현은 충분한 안전성과 효율성 모두 만족하는 방법이다.

그러나 더미 연산이 실제 연산과 구별될 수 있다면, 공격자는 설계자가 더미 연산을 추가함으로써 의도한 공격 복잡도보다 낮은 공격 복잡도로 비밀 키를 획득할 수 있다. 본 논문에서는 C언어 이용 시 더미 연산에 사용되는 변수의 형태를 4가지로 분류하고, 변수의 형태를 달리 사용하여 하이딩 대응기법을 적용한 경우 모두에 대해 더미 연산을 구분할 수 있는 신규 취약점을 제시한다. 그리고 이에 대한 대응기법을 제시한다.

ABSTRACT

When cryptographic algorithms are implemented to provide countermeasures against the side-channel analysis, designers frequently employ the combined countermeasures between the first-order masking scheme and hiding schemes. Their combination can be enough to offer security and efficiency.

However, if dummy operations can be distinguished from real operations, an attacker can extract the secret key with lower complexity than the intended attack complexity by the designer inserting the dummy operations. In this paper, we categorize types of variables used in a dummy operation when C language is employed. Then, we present the novel vulnerability that can distinguish dummy operations for all cases where the hiding schemes are applied using different types of variables. Moreover, the countermeasure is provided to prevent the novel vulnerability.

Keywords: Side-Channel Analysis, Countermeasure, Hiding Scheme, Dummy

Received(11. 30. 2018), Modified(02. 18. 2019), Accepted(02. 18. 2019)

* This work was supported as part of Military Crypto Research Center(UD170109ED) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

† 주저자, n_seeu@kookmin.ac.kr
‡ 교신저자, christa@kookmin.ac.kr(Corresponding author)

I. 서 론

부채널 분석(Side-Channel Analysis, SCA)은 암호 알고리즘이 실제 디바이스에서 수행될 때 발생되는 연산 시간, 소비 전력, 전자파 방출, 소리 등의 부가적인 정보를 이용하여 비밀 정보를 찾아내는 분석 방법이다. 대표적인 방법으로는 단순 전력 분석(Simple Power Analysis, SPA)[11], 차분 전력 분석(Differential Power Analysis, DPA)[11]과 상관 전력 분석(Correlation Power Analysis, CPA)[1] 등이 있다. 단순 전력 분석은 한 개의 파형만으로 비밀 정보를 획득하는 공격이고 차분 전력 분석과 상관 전력 분석은 다수의 전력 파형과 통계적인 방법을 이용한 공격 방법이다. 비밀 정보에 대한 누출의 원인은 부채널 정보가 중간 값과 연산에 의존하여 발생하기 때문이다. 평문 또는 암호문이 알려진 경우 공격자는 비밀 정보를 추측함으로써 중간 값의 계산이 가능하다.

부채널 분석에 대한 대응기법의 목표는 부채널 정보와 계산 가능한 중간 값 사이의 연관성을 제거하는 것이다. 부채널 분석 대응기법은 마스크(Mask)라는 랜덤 값을 부가함으로써 중간 값을 랜덤하게 변경하여 공격자가 중간 값을 추측할 수 없도록 하는 마스킹 기법(Masking Scheme)[6, 13]과 공격 대상 연산의 발생 시점을 랜덤하게 변경하는 하이딩 기법(Hiding Scheme)[10]으로 구분된다.

사물인터넷(Internet of Things, IoT) 환경과 금융 IC 카드 등의 임베디드 장비에 암호 알고리즘을 부채널 분석에 안전하도록 구현하는 경우, 일반적으로 1차 마스킹 기법과 하이딩 기법이 혼합하여 사용된다[10, 12, 14]. 2차 이상의 고차 마스킹 기법을 사용하면 암호 알고리즘의 동작 시간이 AES[9] 암호 알고리즘의 경우 대응기법을 적용하지 않은 구현에 비해 13배 이상으로 증가하여 가용성을 만족하지 못하기 때문이다[4]. 하지만 1차 마스킹 기법은 2차 차분 전력 분석 및 2차 상관 전력 분석에 의해 키 분석이 가능하여 충분한 안전성을 제공하지 못한다. 따라서 효율성 관점에서 1차 마스킹 기법에 하이딩 기법을 추가적으로 적용하여 가용성과 안전성을 모두 제공한다.

랜덤 더미 연산 삽입 기법(Random Insertion of Dummy Operations Scheme)은 하이딩 기법의 일종으로 암·복호화와는 무관한 연산인 더미 연산을 정상적인 연산 중간에 삽입하여 공격 대상이 되는 연산의 수행 시점을 랜덤하게 변경하는 대응기법이다. 여기서 중요한 점은 부채널 정보 상에서 더미 연산을 실제 연산과 구분할 수 없도록 유사하게 구성하는 데에 있다. 하이딩 기법의 또 다른 대응기법으로는 셔플링 기법(Shuffling Scheme)이 있다. 셔플링 기법은 연산 순서를 랜덤하게 섞는 방법으로 랜덤 더미 연산 삽입 기법과 동일하게 공격 대상이 되는 연산의 수행 시점을 숨기는 것을 목적으로 하고 있다.

본 논문에서는 랜덤 더미 연산 삽입 기법과 셔플링 기법이 적용된 AES 암호 알고리즘의 소비 전력 파형에서 더미 연산의 구현 방법에 따른 실제 연산과 더미 연산의 구분 가능성을 보인다. 더미 연산의 구분이 가능함에 따라 공격자는 설계자가 의도한 셔플링 복잡도보다 낮은 셔플링 복잡도로 차분 전력 분석이나 상관 전력 분석 등의 부채널 분석을 수행할 수 있다. 가령 n개의 Sbox 연산에 최대 d개의 더미 연산을 이용하여 랜덤 더미 연산 삽입 기법과 셔플링 기법을 적용한다면 셔플링 복잡도는 $\frac{1}{n+d}$이지만 더미 연산이 구분 가능함에 따라 $\frac{1}{n}$으로 감소한다. 이를 필요 파형 수 관점에서 다시 서술하면, 대응기법이 적용되지 않은 알고리즘의 경우 부채널 분석에 필요한 파형 수가 α라고 가정하였을 때, 랜덤 더미 연산 삽입 기법과 셔플링 기법을 적용하였을 경우 1차 차분 전력 분석 및 1차 상관 전력 분석에 필요한 파형 수는 $\alpha \times (n+d)^2$이다[15]. 하지만 더미 연산의 구분이 가능해지면 이론적으로 $\alpha \times n^2$으로 감소된다. 예를 들어, $n = d = 16$인 경우에 75%의 감소율을 보인다. 이 결과는 부채널 분석에 매우 치명적이다.

또한 더미 연산의 실제 연산과의 구분 가능성을 막기 위한 C언어 레벨에서의 대응기법을 제시한다. 본 논문에서 제시하는 대응기법을 적용하였을 때, 약 49.98%의 구분 성공률을 보여 랜덤 더미 연산 삽입 기법으로 의도한 안전성을 충분히 제공함을 확인할 수 있다.

본 논문의 구성은 2장에서 하이딩 기법과 분석 대상 알고리즘과 더미 연산에 사용되는 변수의 종류들에 대해 설명한다. 3장에서는 더미 연산의 구분 가능성과 이의 실험적 결과를 설명하고 4장에서 대응기법을 제시한다. 그리고 5장에서 결론을 맺는다.

II. 배경 지식

2.1 하이딩 기법(Hiding Scheme)

하이딩 기법[15]의 목표는 부채널 정보 파형과 예상되는 중간 값 간의 독립성을 부여하는 것이다. 이를 제공하는 대표적인 소프트웨어 대응기법은 랜덤 더미 연산 삽입 기법과 셔플링 기법이 있다.

더미 연산은 암·복호화와는 무관한 의미 없는 연산을 뜻한다. 랜덤 더미 연산 삽입 기법은 암호 알고리즘에서 특정 연산이 수행되는 중에 더미 연산을 랜덤하게 삽입하여 공격자가 목표로 하는 연산이 수행되는 시점을 랜덤하게 해준다. 랜덤 더미 연산 삽입 기법의 중요한 점은 암호 알고리즘 수행 중 발생하는 부채널 정보(소비 전력, 전자파 방출 등) 관점에서 더미 연산을 실제 연산과 유사하게 구성하는 것이다.

랜덤 더미 연산 삽입 기법과 달리, 셔플링 기법은 연산의 수행 순서를 랜덤하게 섞는 방법이다. 셔플링 기법 또한 목적은 공격자가 목표로 하는 연산의 수행 시점을 랜덤하게 함에 있다.

2.2 변수 종류(Type of Variables)

C언어 레벨에서의 변수 종류를 3가지로 분류할 수 있다. 변수가 해당 함수 내의 지역 변수(Local variable)로 선언되는 경우, 소스 코드의 전역 변수(Global variable)로 선언하는 경우 그리고 다른 함수의 지역 변수를 해당 함수의 인자(Function argument)로 받아오는 경우가 있다. 더미 연산에 사용되는 변수를 함수 인자로 받아오는 경우 실제 연산에 사용되는 변수와 상이한 함수 인자(Different function argument)로 받아오는 경우와 동일한 함수 인자(Same function argument)로 받아오는 경우를 고려할 수 있다.

Fig. 1.은 더미 연산에 사용되는 변수의 형태의 예이다. 전역 변수를 사용하는 경우는 F1 함수이며 D_GV라고 명명된 전역 변수를 더미 연산에 사용한다. 지역 변수를 사용하는 경우는 F2 함수이고 D_LV라고 명명된 지역 변수를 사용하여 더미 연산을 수행한다. F3 함수는 상이한 함수 인자를 사용하는 경우로 D_DFA라고 명명된 인자를 사용하여 더미 연산을 수행하고 F4 함수는 동일한 함수 인자를 사용하는 경우이며 더미 연산과 실제 연산 모두 SFA라고 명명된 함수 인자를 사용하여 연산을 수행한다.

Encryption.c

```
char D_GV[16];

void main() {
    char r_in[16], d_in[16], u_in[32];

    for (int i = 0; i < 16; i++) {
        u_in[i] = r_in[i];
        u_in[i + 16] = d_in[i];
    }

    F1(r_in);
    F2(r_in);
    F3(r_in, d_in);
    F4(u_in);
}

void F1(char R[]) {
    for (int i = 0; i < 16; i++) {
        R[i] = Sbox[R[i]];
        D_GV[i] = Sbox[D_GV[i]];
    }
}

void F2(char R[]) {
    char D_LV[16];

    for (int i = 0; i < 16; i++) {
        R[i] = Sbox[R[i]];
        D_LV[i] = Sbox[D_LV[i]];
    }
}

void F3(char R[], char D_DFA[]) {
    for (int i = 0; i < 16; i++) {
        R[i] = Sbox[R[i]];
        D_DFA[i] = Sbox[D_DFA[i]];
    }
}

void F4(char SFA[]) {
    for (int i = 0; i < 32; i++)
        SFA[i] = Sbox[SFA[i]];
}
```

Fig. 1. Classification according to variable types

2.3 분석 대상

더미 연산의 구분 가능성에 대한 실험 환경 및 실험 대상의 구현 방법에 대해 서술한다. 랜덤 더미 연산 삽입 기법과 셔플링 기법을 적용한 AES 암호 알

Table 1. Pseudo code of the subbytes function in AES with the dummy operations and the shuffling scheme.

Input	st_in[16], dm_in[16], order[32]
Output	st_out[16], dm_out[16]

```
for i←0 to 31 do
  switch order[i] do
    case 0
      st_out[0]←Sbox[st_in[0]]
      break
    case 1
      st_out[1]←Sbox[st_in[1]]
      break
                    ⋮
    case 16
      dm_out[0]←Sbox[dm_in[0]]
      break
    case 17
      dm_out[1]←Sbox[dm_in[1]]
      break
                    ⋮
end for
```

고리즘을 C언어를 사용하여 구현한 경우를 대상으로 한다. 대응기법을 정확하게 적용하기 위해 어셈블리(Assembly) 언어를 사용하여야 하지만, 설계자의 편의성과 다양한 구현 환경을 고려하였을 때 C언어가 널리 사용된다. 그러므로 본 논문에서는 C언어를 사용하여 구현된 암호 알고리즘을 대상으로 하며 랜덤 더미 연산 삽입 기법과 셔플링 기법을 swtich-case 문을 사용하여 적용한다.

더미 연산에 사용된 변수의 형태로 2.2절에서 소개한 것처럼 더미 연산에 사용되는 변수의 종류에 따라 4가지 방법을 고려한다.

Table 1.은 AES 암호 알고리즘의 Subbytes 연산에 하이딩 대응기법이 적용된 의사코드이다. 삽입된 더미 연산의 개수를 실제 연산의 개수와 동일하게 16개로 고정하였으며 구현 방식은 셔플링 된 연산 순서가 저장되어 있는 order 변수를 사용하여 switch-case 문으로 구성하였다. st_in과 st_out은 각각 실제 연산의 입·출력을 저장하는 변수이며 dm_in과 dm_out은 더미 연산의 입·출력을 저장하는 변수이다. 단, 동일한 함수 인자를 사용하는 구현의 경우 st_in과 st_out의 길이를 2배 들리고 전반부와 후반부를 각각 실제 연산과 더미 연산의 입·출력을 저장하는 변수로 사용한다.

III. 신규 취약점

본 장에서는 더미 연산과 실제 연산을 구분할 수 있는 신규 취약점을 보여준다. 이전 장에서 설명한 것과 같이 더미 연산에 사용된 변수의 형태에 따른 4가지 구현 방법에 대해 각각 실험을 수행한다. 각각의 구현 방법은 Table 1.에서 dm_in과 dm_out 변수의 선언 형태만 다르기 때문에 자세한 소스 코드는 생략한다. 단, 동일한 함수 인자를 사용하는 구현의 경우 dm_in과 dm_out 대신에 st_in과 st_out을 사용하며 인덱스는 switch-case문의 인덱스와 동일하다.

하이딩 대응기법이 적용된 AES암호 알고리즘을 XMEGA128D4-U[7] 칩에 C언어로 구현하였으며 칩이 동작하는 동안 발생하는 소비 전력을 Chipwhisperer-Lite(CW1173)[8]로 측정하여 실험을 진행하였다.

더미 연산과 실제 연산을 구분하기 위한 기준으로 BCDC(Bounded Collision Detection Criterion)[5] 값을 이용한다. 두 집단 간의 유사도를 의미하는 BCDC는 다음과 같이 정의된다.

$$BCDC(T_1, T_2) = \frac{1}{\sqrt{2}} \times \frac{\sigma_{(T_1 - T_2)}}{\sigma_{(T_1)}} \qquad (1)$$

T_1은 BCDC 값 계산의 기준이 되는 참조 영역이고 T_2는 그 대상이 되는 영역이다. $\sigma_{(T_1)}$은 T_1의 표준편차, $\sigma_{(T_1 - T_2)}$는 T_1과 T_2의 차이의 표준편차이다. BCDC 값이 0에 가까울수록 T_1과 T_2와의 유사도가 큼을 의미한다.

3.1 지역 변수(Local Variable)

더미 연산에 사용되는 dm_in, dm_out 변수를 AES 암호 알고리즘의 Subbytes 함수 내부의 지역 변수로 설정한 경우의 소비 전력 파형의 일부는 Fig. 2.와 같다. 파형 전체에서 32개의 Sbox 연산의 구분이 가능하다. 본 논문에서는 가독성을 위해 32개 중 일부만을 그림으로 나타낸다. 소비 전력 파형의 'D'로 표시된 영역은 더미 Sbox 연산이 수행되는 부분이고 'R'로 표시된 영역은 실제 Sbox 연산이 수행되는 부분이다.

Fig. 2.의 'BCDC reference'로 표시된 부분은 실제 Sbox 연산의 소비 전력 파형 중 일부분으로 BCDC 값 계산 식 (1)의 참조 영역인 T_1으로 설정한 부분이다. 참조 영역 T_1과 동일한 길이의 대상 영역 T_2를 파형의 처음부터 1포인트 씩 이동시켜가며 계산하였다. 그 결과의 일부분은 Fig. 3.과 같다. 실제 Sbox 연산이 수행된 곳에서는 0에 가까운 BCDC 값을 보인다. 전체 BCDC 값 중 하위 16개를 Fig. 3.의 빨간색 원으로 표시하였고 각각의 하위 값은 실제 Sbox 연산이 수행된 시점에서 계산되었다. AES 암호 알고리즘의 Subbytes 함수를 1백만 번 수행한 소비 전력 파형에서 더미 연산을 구분한 결과, 99.92%의 성공률로 구분이 가능하였다.

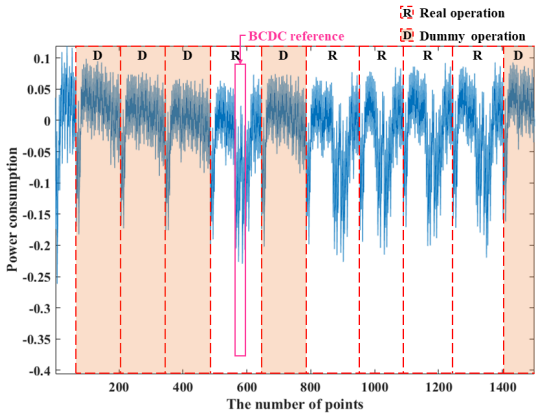


Fig. 2. The power consumption trace about the Subbytes operation of AES with the 16 local variable dummy operations and the shuffling scheme.

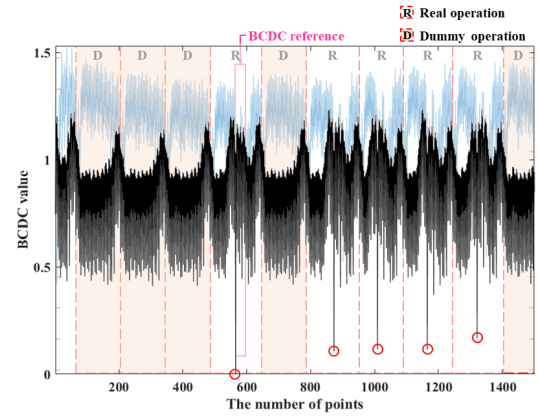


Fig. 3. The BCDC value trace of the Fig. 2.

Table 2. Assembly code of the Subbytes operation when the dummy operations are used as a local variable.

Real operation	st_out[0]←Sbox[st_in[0]]
	
	movw r28, r14
	ld r30, Y
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	movw r30, r16
	st Z, r24
	rjmp .+268

Dummy operation	dm_out[0]←Sbox[dm_in[0]] Not performed.

따라서 지역 변수를 사용하여 더미 연산을 구현하였을 때, BCDC 값을 이용하면 실제 연산과 구분이 가능하다.

이와 같이 더미 연산이 실제 연산과 구분이 되는 근거로 C언어로 구현된 암호 알고리즘이 컴파일(Compile) 과정을 거친 어셈블리 코드를 제시한다.

Table 2.에서 보이는 바와 같이 더미 연산을 지역 변수로 사용하여 구현한 경우에서는 어셈블리 코드 상에서 더미 Sbox 연산에 해당하는 코드를 찾지 못하였다. Chipwhisperer-Lite(CW1173)가 제공하는 컴파일러(WinAVR 4.3.3)가 최적화 과정에서 Subbytes 함수의 출력과 무관한 연산으로 판단하여 제거한 것으로 추정된다. 그럼에도 Subbytes 함수의 소비 전력 파형이 32개의 부분으로 구분될 수 있는 이유는 switch-case 문에서 참조하고 있는 order 배열은 최적화 대상이 아니기 때문이다. 최적화 되지 않은 배열을 사용하는 연산 또한 최적화 대상이 아니기 때문에 내부 연산이 수행되지 않더라도 switch-case 문으로 이루어진 반복문은 수행된다.

3.2 전역 변수(Global Variable)

더미 연산에 전역 변수를 사용한 경우의 소비 전력 파형의 일부분은 Fig. 4.와 같다. 이 경우 또한 실제 Sbox 연산 중 일부분인 'BCDC reference'로 표시된 부분을 참조 영역 T_1으로 설정하여 BCDC 값을 계산하였다. 계산된 BCDC 값의 일부분은 Fig. 5.와 같으며 실제 Sbox 연산이 수행된 시점

('R'로 표시된 영역)에서 더미 Sbox 연산이 수행된 시점('D'로 표시된 영역)보다 낮은 BCDC 값을 보인다. 그러나 여섯 번째 Sbox 연산은 더미 연산임에도 불구하고 하위 16개의 BCDC 값을 포함하고 일곱 번째 Sbox 연산은 실제 연산임에도 하위 16개의 BCDC 값을 포함하지 않는다. 따라서 BCDC 값을 이용하여 더미 연산을 구분하는데 오류가 존재한다. 이와 같이 잘못 판단하는 경우가 존재하지만, AES 암호 알고리즘의 Subbytes 함수를 1백만 번 수행한 소비 전력 파형에서 더미 연산을 구분한 결과 93.76%의 성공률로 더미 연산을 구분할 수 있었다. 따라서 전역 변수를 사용하여 더미 연산을 구현한 경우 또한 실제 연산과 구분이 가능하다.

Table 3.은 이 경우의 어셈블리 코드이다. 실제

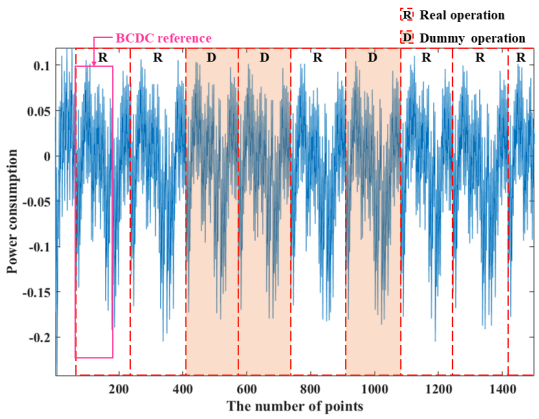


Fig. 4. The power consumption trace about the Subbytes operation of AES with the 16 global variable dummy operations and the shuffling scheme.

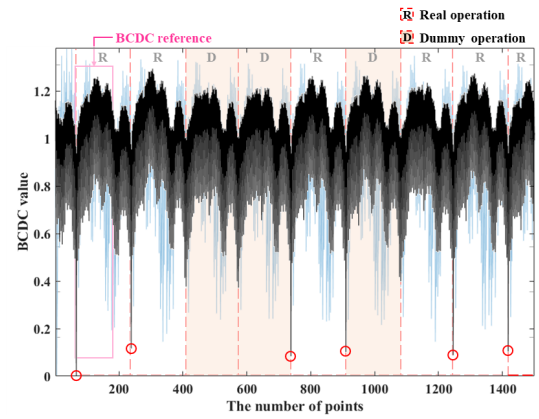


Fig. 5. The BCDC value trace of the Fig. 4.

Table 3. Assembly code of the Subbytes operation when the dummy operations are used as a global variable.

Real operation	st_out[0]←Sbox[st_in[0]]
	
	movw r28, r14
	ld r30, Y
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	movw r30, r16
	st Z, r24
	rjmp .+556

Dummy operation	dm_out[0]←Sbox[dm_in[0]]
	
	lds r30, 0x2231
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	sts 0x21B0, r24
	rjmp .+268

연산과 더미 연산에 사용된 어셈블리 명령어와 어셈블리 코드 구성이 다름을 확인할 수 있다.

3.3 상이한 함수 인자(Different Function Argument)

실제 Sbox 연산에 사용되는 변수는 Subbytes 함수의 인자로 입력되기 때문에 이와 동일하게 더미 Sbox 연산에 사용되는 dm_in과 dm_out 변수도 Subbytes 함수의 인자로 입력되도록 구성하였다 (Subbytes(st_in, st_out, dm_in, dm_out)). st_in 및 st_out과 dm_in 및 dm_out의 형태가 함수 인자로 동일하나 다른 변수를 사용하기 때문에 이를 상이한 함수 인자라고 명명한다. 이의 소비 전력 파형의 일부분은 Fig. 6.과 같다. 상이한 함수 인자를 사용한 구현 방식의 경우에서도 실제 Sbox 연산 중 일부분을 'BCDC reference'로 설정하여 T_1으로 사용하였다. BCDC 값을 계산한 결과 Fig. 7.과 같이 실제 연산이 더미 연산에 비해 더 작은 BCDC 값을 가지며 1백만 개의 파형을 수집하여 구분한 결과 99.86%의 성공률을 보였다.

Table 4.의 어셈블리 코드 상에서도 사용된 어셈블리 명령어와 어셈블리 코드의 구성이 다름을 확인할 수 있다. 따라서 실제 연산에 사용된 변수와 동일

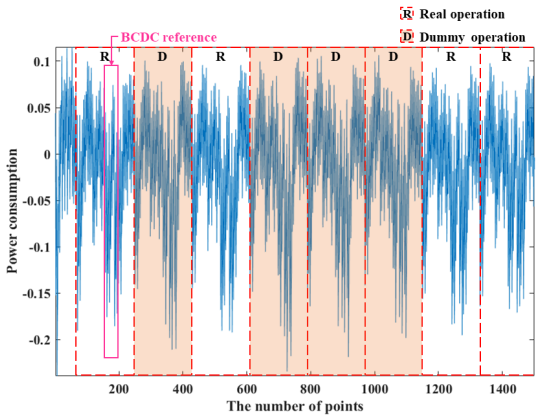


Fig. 6. The power consumption trace about the Subbytes operation of AES with the 16 different function argument dummy operations and the shuffling scheme.

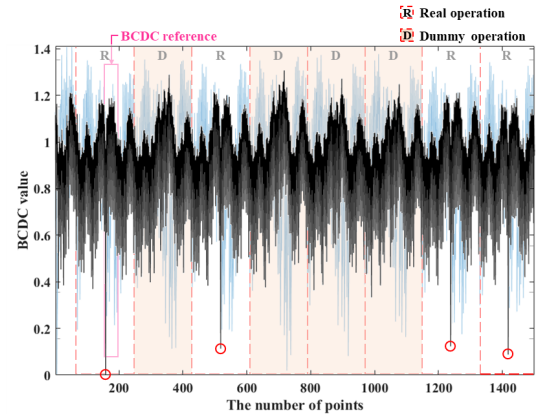


Fig. 7. The BCDC value trace of the Fig. 6.

하게 더미 연산을 함수 인자를 사용하여 구성한 경우도 더미 연산의 구분이 가능하다.

3.4 동일한 함수 인자(Same Function Argument)

3.3절에서 더미 연산에 사용된 변수를 실제 연산에 사용된 변수와 동일한 형태인 함수 인자로 구성하였지만 더미 연산의 구분이 가능하였다. 따라서 본 절에서는 변수의 형태뿐만 아니라 동일한 배열을 사용한 구현 방법을 고려한다. dm_in과 dm_out 변수를 사용하지 않고 기존에 16개의 배열로 구성된 st_in과 st_out 변수를 32개의 배열로 확장하여 전반부 16개의 배열은 실제 Sbox 연산에 사용하고 후반부 16개의 배열은 더미 Sbox 연산에 사용한다.

Table 4. Assembly code of the Subbytes operation when the dummy operations are used as a function argument.

Real operation	st_out[0]←Sbox[st_in[0]]
	
	movw r30, r16
	ld r24, Y
	mov r30, r24
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	movw r30, r22
	st Z, r24
	rjmp .+585

Dummy operation	dm_out[0]←Sbox[dm_in[0]]
	
	ld r30, Y
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	st X, r24
	rjmp .+268

이러한 방식을 동일한 함수 인자라고 명명한다. 이와 같은 구현 방식의 소비 전력 파형의 일부분은 Fig. 8.과 같고 실제 연산 중 일부분을 'BCDC reference'로 설정하여 T_1으로 사용한다. BCDC 값을 계산한 결과의 일부분은 Fig. 9.와 같다. 전역 변수를 사용한 경우처럼 첫 번째 수행된 Sbox는 더미 연산이지만 하위 16개의 BCDC 값 포함하여실제 Sbox 연산으로 판단되었다. 이와 같은 잘못 판단하는 사례가 존재하지만 1백만 개의 소비 전력 파형을 수집하여 구분하였을 때 73.72%의 성공률로 구분이 가능하였다. 실제 연산과 더미 연산의 구분이 불가능한 경우는 50%의 성공확률을 가져야하기 때문에 동일한 함수 인자를 사용한 구현 또한 더미 연산의 구분이 일부 가능하다.

Table 5.는 본 구현 방식의 어셈블리 명령어로 실제 연산과 더미 연산의 명령어 구성은 동일하다. 그럼에도 불구하고 위의 실험처럼 더미 연산의 구분이 일부 가능한 이유로는 st_in에 저장된 메모리 값을 불러오는 부분에서 참조하는 메모리 주소 상의 차이점이 원인으로 예상된다. 실제 연산은 st_in 배열의 전반부를 사용하기 때문에 ldd 명령어가 Y부터 Y+15까지를 참조하지만 더미 연산은 Y+16부터

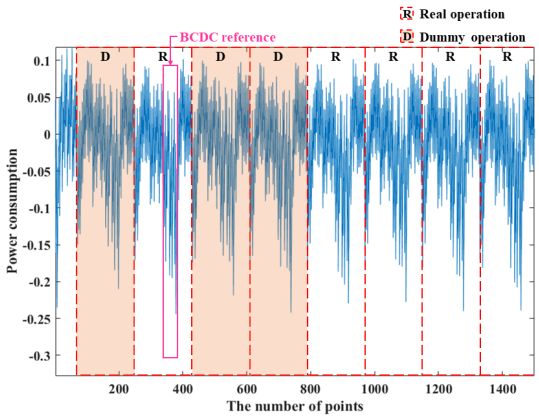


Fig. 8. The power consumption trace about the Subbytes operation of AES with the 16 same function argument dummy operations and the shuffling scheme.

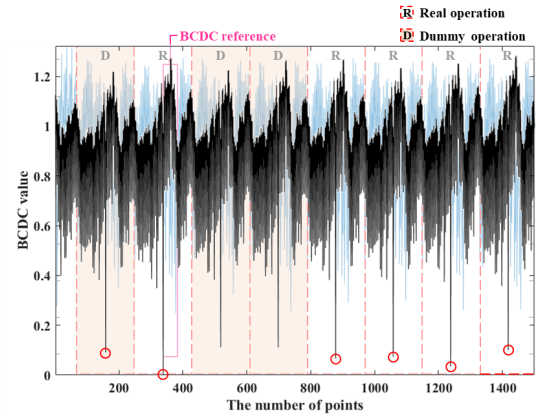


Fig. 9. The BCDC value trace of the Fig. 8.

Y+31까지를 참조한다. 메모리 주소를 이동하는 부분에서 더미 연산이 실제 연산보다 더 많은 이동을 요하기 때문에 소비 전력이 다르다고 판단된다.

더미 연산에 사용되는 변수의 형태에 따른 더미 연산의 구분 성공률은 Table 6.과 같다.

IV. 대응기법

3장에서 더미 연산을 지역 변수, 전역 변수, 상이한 함수 인자와 동일한 함수 인자로 사용한 경우 모두 더미 연산이 실제 연산과 구분이 가능함을 확인하였다. 본 장에서는 암호 알고리즘 설계자가 C언어를 사용하는 경우에서 더미 연산의 구분 가능성을 제거

Table 5. Assembly code of Subbytes operation when dummy operations are used as same function argument.

Real operation	st_out[0]←Sbox[st_in[0]]
	
	ldd r30, Y
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	adiw r26, 0x00
	st X, r24
	sbiw r26, 0x00
	rjmp .+556

Dummy operation	st_out[16]←Sbox[st_in[16]]
	
	ldd r30, Y+16
	ldi r31, 0x00
	subi r30, 0xF0
	sbci r31, 0xDF
	ld r24, Z
	adiw r26, 0x00
	st X, r24
	sbiw r26, 0x00
	rjmp .+268

Table 6. Distinguish success rate according to types of variable used in dummy operations.

Types of variable used in dummy operations	Distinguish success rate
Local variable	99.92%
Global variable	93.76%
Difference function argument	99.86%
Same function argument	73.72%

하는 대응기법을 제시한다. 3.4절에서 더미 연산에 사용된 변수를 실제 연산에 사용된 변수와 동일한 배열로 구성하였지만 참조 메모리 주소의 차이로 더미 연산의 구분이 일부 가능하였다. 따라서 본 장에서는 랜덤한 메모리 주소를 참조하도록 하는 대응기법을 제시한다.

기존의 방식은 st_in에 실제 연산과 더미 연산에 사용될 값이 순차적으로 저장하고 order 변수에 저장된 순서에 따라 연산을 수행하는 방식이다. 메모리 주소의 차이 문제를 해결하기 위해 order 변수에 저

Table 7. Pseudo code of the new countermeasure.

Input	st_in[32], order[32]
Output	st_out[32]

```
for i←0 to 31 do
  temp_in[i]←st_in[order[i]]
end for

for i←0 to 31 do
  temp_out[i]←Sbox[temp_in[i]]
end for

for i←0 to 31 do
  st_out[order[i]]←temp_out[i]
end for
```

장된 순서에 따라 실제 연산 및 더미 연산에 사용될 값을 temp_in에 미리 저장하고 순차적으로 Sbox 연산을 수행한다. 이후에 셔플링 기법이 다른 연산에 영향을 미치지 않기 위해 Sbox 연산의 출력 값을 다시 정상적인 순서로 바꾸어서 st_out에 저장하는 방식이다. 이를 의사코드로 나타내면 Table 7.과 같다.

본 대응기법의 전력 파형의 일부분은 Fig. 10.과 같고 실제 연산 중 일부분인 'BCDC reference'를 참조 영역 T_1으로 설정하여 BCDC 값을 계산한 결과는 Fig. 11.과 같다. 첫 번째, 네 번째, 다섯 번째, 여섯 번째, 여덟 번째, 열 번째와 열두 번째 수행된 Sbox 연산은 더미 연산이지만 실제 연산으로 판단하였고 세 번째, 아홉 번째와 열여덟 번째는 실

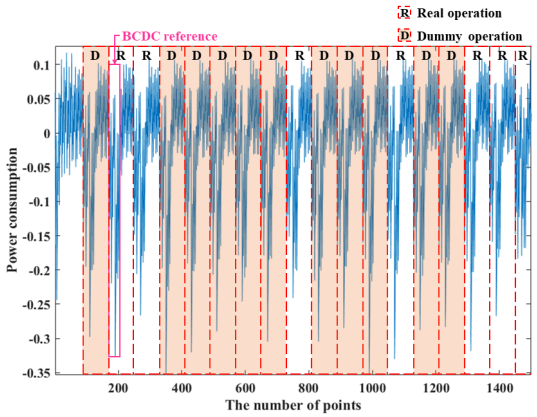


Fig. 10. The power consumption trace about the new countermeasure.

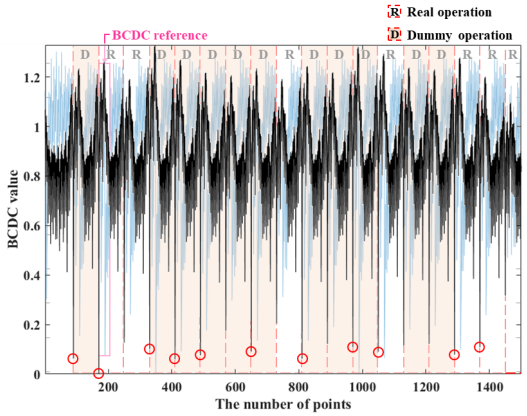


Fig. 11. The BCDC value trace of the Fig. 10.

제 연산이지만 더미 연산으로 판단하였다. 이처럼 잘못 판단하는 사례가 다수이며 1백만 개의 소비 전력 파형을 수집하여 실험한 결과 49.98%의 성공률을 보이므로 본 구현 방식은 더미 연산의 구분이 불가능하다.

위와 같은 실험적인 결과뿐만 아니라 Table 8.과 같이 컴파일된 어셈블리 코드 상에서도 더미 연산과 실제 연산이 동일한 어셈블리 명령어 구성을 사용함을 확인하였다. 또한 상이한 메모리 주소를 사용하는 연산이 없음을 확인하여 3.4절에서 보이는 문제점을 해결하였다. 따라서 셔플링 기법을 배열 저장 순서에 적용할 때, 더미 연산과 실제 연산을 부채널 정보를 사용하여 구분할 수 없다. 설계자가 더미 연산과 셔플링 기법을 사용하여 암호 알고리즘에 하이딩 대응기법을 적용하는 경우에 더미 연산과 실제 연산에 사용되는 변수를 동일한 배열을 사용하며 연산 수행 순

Table 8. Assembly code of the new countermeasure.

	temp_out[0]←Sbox[temp_in[0]]

Real & Dummy operation	movw r26, r22
	add r26, r18
	adc r27, r19
	movw r30, r20
	add r30, r18
	adc r31, r19
	ld r30, Z
	ldi r31, 0x00
	subi r30, 0xF0
	sbcd r31, 0xDF
	ld r24, Z
	st X, r24

서가 아닌 배열 저장 순서에 셔플링 기법을 적용하여야 구분 가능성을 최소화하고 의도한 공격 복잡도를 충분히 제공할 수 있다.

V. 결 론

본 논문에서는 부채널 대응기법으로 하이딩 대응기법을 사용할 때 주로 사용되는 더미 연산의 구현 방식에 따라 실제 연산과의 구분 가능성을 제시한다. 설계자가 C언어를 사용하여 부채널 대응기법이 적용된 암호 알고리즘을 구현한 경우에 대해 더미 연산에 사용된 변수를 지역 변수, 전역 변수, 상이한 함수 인자와 동일한 함수 인자 형태로 선언하였을 때 더미 연산이 실제 연산과 소비 전력 파형에서 구분이 가능하여 설계자가 의도한 공격 복잡도를 충분히 제공하지 못할 수 있음을 보인다. 이를 막기 위한 구현 방식으로 실제 연산과 더미 연산에 사용되는 변수를 동일한 변수로 사용하며 셔플링 기법을 연산 수행 순서가 아닌 배열 저장 순서에 적용하여야 한다.

References

[1] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," Cryptographic Hardware and Embedded Systems, CHES'04, LNCS 3156, pp. 1-15, Aug. 2004.

[2] F.X. Standaert, B. Gierlichs, and I. Verbauwhede, "Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected cmos device," International Conference of Information Security and Cryptology, ICISC'08, LNCS 5461, pp. 253-267, 2008.

[3] F.X. Standaert, T.G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," Advanced in Cryptology, EUROCRYPT'09, LNCS 5479, pp. 443-461, 2009.

[4] H. Kim, S. Hong, and J. Lim, "A fast and provably secure higher-order masking of AES S-box," Cryptographic Hardware and Embedded System, CHES'11, LNCS 6917, pp. 95-107, 2011.

[5] I. Diop, P.Y. Liardet, Y. Linge, and P. Maurine, "Collision based attacks in practice," 2015 Euromicro Conference on Digital System Design, DSD'15, pp. 367-374, 2015.

[6] L. Goubin and J. Patarin, "DES and differential power analysis - the duplication method," Cryptographic Hardware and Embedded System, CHES'99, LNCS 1717, pp. 158-172, 1999.

[7] ATxmega128D4 - 8-bit AVR "atxmega 128d4 specification" Microcontrollers, http://www.microchip.com/wwwproducts/en/ATxmega128D4, 2016.

[8] ChipWhisperer® - NewAE Technology Inc., "chipwhisperer" https://newae.com/tools/chipwhisperer/, 2017.

[9] V. Rijmen and J. Daemen, "Advanced encryption standard," Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, 2001.

[10] P. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," Applied Cryptography and Network Security, ACNS'06, LNCS 3989, pp. 239-252, 2006.

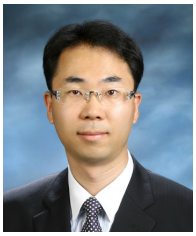
[11] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advanced in Cryptology, CRYPTO'99, LNCS 1666, pp. 388-397, Aug. 1999.

[12] S. Tillich and C. Herbst, "Attacking state-of-the-art software countermeasures - a case study for AES," Cryptographic Hardware and Embe-

dded System, CHES'08, LNCS 5154, pp. 228-243, 2008.
[13] S. Chari, C. Justla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," Advances in Cryptology, CRYPTO'99, LNCS 1666, pp. 384-412, 1999.
[14] S. Tillich, C. Herbst, and S. Mangard, "Protecting AES software implementations on 32-bit processors against power analysis," Applied Cryptography and Network Security, ACNS'07, LNCS 4521, pp. 141-157, 2007.
[15] S. Mangard, E. Oswald, and T. Poop, "Power analysis attacks: Revealing the secrets of smart cards," Springer, 2008.

〈저자소개〉

이 종 혁 (JongHyeok Lee) 학생회원
2017년 2월: 국민대학교 수학과 학사
2017년 3월~현재: 국민대학교 금융정보보안학과 석박사통합과정
〈관심분야〉 부채널 분석 및 대응법 설계, 대칭키 암호 알고리즘, 스마트 카드 보안, 오류 주입 공격



한 동 국 (Dong-Guk Han) 종신회원
1999년 2월: 고려대학교 수학과 학사
2002년 2월: 고려대학교 수학과 이학석사
2005년 2월: 고려대학교 정보보호대학원 공학박사
2004년 4월~2005년 4월: 일본 Kyushu Univ., 방문연구원
2005년 4월~2006년 4월: 일본 Future Univ.-Hakodate, Post.Doc.
2006년 6월~2009년 2월: 한국전자통신연구원 정보보호연구단 선임연구원
2009년 3월~현재: 국민대학교 정보보안암호수학과 정교수
〈관심분야〉 공개키 암호시스템 안전성 분석 및 고속 구현, 부채널 분석 및 대응법 설계, IoT 정보보호 기술