정 보 보 호 학 회 지
제 30 권 제1호, 2020. 2

전력분석공격에 대한 하드웨어 마스킹 대응기법 동향

백 유 진*

요 약

암호시스템에 대한 부채널분석 공격은 기기에 대한 훼손이 없이 비교적 저렴한 비용으로 적용이 가능한 반면에 공격 결과는 보안에 매우 치명적일 수 있다. 따라서 암호시스템의 구현에 있어서 성능적인 측면과 함께 부채널분석 공격에 대한 안전성 역시 반드시 고려되어야 한다. 본 논문에서는 부채널분석 공격 대응기법 중에서 하드웨어 기반 마스킹 기법의 연구 동향에 대해서 알아보고자 한다.

Ⅰ. 서 론

스마트 카드, IoT(Internet of Things) 장치, 스마트 폰, PC 등과 같은 정보 기기가 개인 정보, 금융 정보 등과 같은 민감한 정보를 저장하는 경우가 일상화되면서 해당 기기를 목표로 하는 보안 침해 사고에 대한 우려가 커지고 있고 이런 우려에 대응하기 위해 다양한 암호시스템이 사용되고 있다. 하지만 부채널분석 공격(Side-Channel Attack)에 대한 고려 없이 암호시스템을 구현하게 되면 보안과 관련된 또 다른 문제가 발생할 수 있다.

부채널분석 공격은 기기를 훼손함이 없이 비교적 저렴한 비용으로 적용이 가능한 반면에 공격 결과는 보안에 매우 치명적일 수 있음이 알려져 있다. 따라서 암호시스템의 구현에 있어서 성능적인 측면과 함께 부채널분석 공격에 대한 안전성 역시 반드시 고려되어야 한다. 본 논문에서는 부채널분석 공격에 대한 대응기법 중에서 하드웨어 기반 마스킹 기법의 연구 동향에 대해서 알아보고자 한다.

Ⅱ. 전력분석 공격과 그 대응기법

2.1. 전력분석 공격

일반적으로 암호시스템에 대한 공격은 크게 수학적 공격법과 부채널분석 공격법으로 분류가 가능하다. 이 중에서 수학적 공격법은 암호 알고리즘을 비밀키를 인자로 가지는 수학적인 함수로 간주하며 고정된 비밀키에 대하여 평문·암호문 쌍이 주어진 경우 해당 비밀키를 복구할 때 필요한 연산량을 기준으로 암호 알고리즘의 안전성을 평가한다. 이러한 수학적 공격법의 예로는 차분 공격, 선형 공격, 대수적 공격 등이 있다 ([1]). 반면에 부채널분석 공격법은 암호 알고리즘을 물리적인 기기에서 동작하는 실제 프로그램 또는 회로로 간주하며 암호 연산 과정 중에 발생하는 연산시간량, 전력소비량, 전자기장 방사량, 오류가 주입된 계산 결과값 등의 부채널 정보를 분석하여 기기 내에 저장된 비밀키를 복구한다 ([2,3,4,5]). 그리고 전력분석 공격(Power Analysis Attack)은 암호 기기의 전력 소비 패턴을 분석함으로써 비밀 정보를 알아내는 부채널분석 공격법의 한 종류이다.

전력분석 공격은 일반적으로 데이터 수집 단계 및 데이터 분석 단계라는 2 단계를 거쳐 공격이 진행된다 ([3]). 먼저 데이터 수집 단계에서는 랜덤하게 (또는 공격자에 의해 임의로) 선택된 평문(또는 암호문)을 이용해서 암호화(또는 복호화) 연산을 수행한 후 해당 연산에 대한 소비전력 파형을 수집한다. 이후 데이터 분석 단계에서는 비밀키(의 일부분)에 대해서 그 값을 예측한 후 예측값과 입력된 평문(또는 암호문)을 이용하여 내부 연산값을 계산한다. 그리고 이 계산값의 유효성을 수집된 전력소비 파형을 이용하여 검증하는 과정을 반복함으로써 결국 비밀키 전체값을 복구하게 된다.

전력분석 공격이 가능한 이유는 다음과 같다. 먼저 스마트 카드 IC와 같은 저전력 기기의 경우 자체 전원

* 우석대학교 정보보안학과(교수, yoojin.baek@gmail.com)

을 사용하지 않고 대신에 외부에서 전력을 공급받는 경우가 많으며 이 경우 외부 전력 공급량을 모니터링하게 되면 실제 암호 기기가 소비하는 전력량을 매우 정확하게 측정할 수 있는데 바로 이 점이 전력분석 공격이 가능한 이유가 된다. 또한 스마트 카드 IC와 같은 저전력 기기의 경우 전력 소비량을 줄이기 위해서 암호 연산 도중에 암호 연산과 관계가 없는 (CPU를 포함한) 주변 기기의 동작을 정지시키는 경우가 많으며 이 경우 암호 연산만이 소비하는 전력량을 매우 정확하게 측정할 수 있고 따라서 이러한 기기에 전력분석 공격이 수월하게 적용 가능하다. 이 점은 특히 PC나 스마트폰과 같은 고사양 기기에 대해서 전력분석 공격의 적용이 쉽지 않은 이유를 설명해주는데, 가령 PC의 경우 암호 연산 도중에 백그라운드에서 다양한 응용 프로그램이 동작을 하는 경우가 많으며 이러한 응용프로그램의 동작은 암호 연산만의 전력소비량 측정에 많은 장애물로 작용하기 때문이다. 그리고 이런 이유로 전력분석 공격은 스마트 카드 IC 등과 같은 저전력 기기에 주로 성공적으로 적용이 되었다. 하지만 전력분석 공격과 매우 유사한 전자기장분석 공격의 경우, 정보 기기의 layout 정보 등을 이용하여 암호 연산기의 정확한 위치를 식별할 수 있고 이를 이용해서 암호 연산만의 전자기장 방사량을 비교적 정확하게 측정하는 것이 가능하기 때문에 PC나 스마트폰 등에도 공격이 잘 적용된다고 알려져 있다([4]).

전자 기기는 내부에 저장된 값에 따라 저장과 처리에 필요한 전력소비량 패턴이 다르며 전력분석 공격은 이러한 패턴의 차이를 이용한다. 예를 들어 내부 비트값의 변화가 없는 경우, 즉 비트 0이 계속 유지되거나 비트 1이 계속 유지되는 경우와 비트가 변화하는 경우, 즉 비트 0이 비트 1로 바뀌거나 비트 1이 비트 0으로 바뀌는 경우는 전력소비량 비교를 통해 구분할 수 있으며 바로 이 점이 전력분석 공격을 가능하게 하는 또 다른 이유가 된다.

전력분석 공격이 가능한 또 다른 이유는 공격이 진행되는 방식과 관련이 있다. 전술하였듯이 전력분석 공격 과정 중 데이터 분석 단계는 입력된 평문(또는 암호문)과 기기에 저장된 비밀키의 예측값을 이용하여 내부 연산값을 계산하게 되는데 이러한 과정의 성공적인 수행을 위해서는 평문(또는 암호문)의 정확한 값을 공격자가 알고 있다는 가정이 필요하다. 따라서 만약 이 가정이 성립하지 않는 경우, 즉 공격자가 평문(또는 암호문)에 대한 정보를 알지 못하면 전력분석 공격의 난이도가 상당히 올라가거나 공격 자체가 불가능해지며, 본 논문의 주요 주제인 마스킹 기법은 바로 이러한 가정이 성립하지 못하게 함으로써 전력분석 공격을 방어하게 된다.

전력분석 공격은 일반적으로 단순 전력분석 공격(Simple Power Analysis Attack, 이하 SPA)과 차분 전력분석 공격(Differential Power Analysis Attack, 이하 DPA)으로 구분된다([3]). 먼저 SPA는 공격자가 암호 연산을 한 번 또는 수차례 구동시키면서 수집한 전력소비량의 시간에 따른 변화 양상을 주로 시각적으로 해석하는 방식으로 공격을 진행한다. 예를 들어 RSA 공개키 암호시스템에서 모듈라 지수승 연산을 위해 사용되는 모듈라 제곱 연산과 모듈라 곱셈 연산은 전력 소비 패턴이 매우 상이하다고 알려져 있으며 따라서 그 패턴을 분석하면 두 연산을 쉽게 구별할 수 있고 이를 통해 RSA 암호시스템의 복호화 연산에 사용되는 비밀 지수값을 SPA를 통해 상당히 정확하게 복구할 수 있다. 반면에 DPA는 SPA에 비해 더 강력한 공격 방법으로 알려져 있으며 암호 연산을 여러 차례 구동시킨 후 얻을 수 있는 전력소비량 정보를 다양한 신호처리 방법을 사용하여 분석하고 이를 통해 암호기기 내부에 저장된 비밀 정보를 알아낸다.

2.2. 마스킹 기법

전력분석 공격에 대응하기 위해서 다양한 기법이 제안되었으며 그 중에서 마스킹 기법(masking method)은 블록암호 알고리즘에 대한 대표적인 DPA 대응기법의 하나이다([6]). 마스킹 기법은 메시지에 대한 암·복호화 연산을 수행하기 전에 난수를 이용하여 평문을 마스킹하고 마스킹된 데이터에 대해서 암·복호화 연산을 수행하면 외부에서 관찰 가능한 소비 전력 패턴과 연산 중간값과의 관련성을 축소 또는 제거할 수 있다는 사실에 기반을 둔다.

유한체 K_1, K_2, 함수 $f : K_1 \rightarrow K_2$, 입력 마스크 차수 d_{in}, 출력 마스크 차수 d_{out}이 주어지고 XOR 연산 $\oplus$이 K_1, K_2의 덧셈 연산이라고 했을 때, 함수 f에 (d_{in}, d_{out})-차 (불)마스킹 기법을 적용하는 과정은 다음과 같다: 임의의 $x \in K_1$에 대하여

1) $x_1, x_2, ..., x_{d_{in}} \in K_1$을 랜덤하게 선택한다.
2) $x_0 = x \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_{d_{in}}$을 계산한다. ($(x_0, x_1, ..., x_{d_{in}})$는 x의 d_{in}-차 마스크 또는 함수 $z = f(x)$의 d_{in}-차 입력 마스크라고 한다)
3) $(x_0, x_1, ..., x_{d_{in}})$을 사용하여 $z_0 \oplus z_1 \oplus ... \oplus z_{d_{out}} = f(x)$를 만족시키는 $z_0, z_1, ..., z_{d_{out}} \in K_2$를 계산한다. ($(z_0, z_1, ..., z_{d_{out}})$은 $z = f(x)$의 d_{out}-차 마스크 또는 함수 $z = f(x)$의 d_{out}-차 출력 마스크라고 한다)
4) 마스킹 기법이 안전하다는 것은 $z_0, z_1, ..., z_{d_{out}}$의 계산 과정에서 원래의 입력값인 x에 대한 (어떠한) 정보의 노출도 없어야 함을 의미한다.

마스킹 기법에서 만약 $d_{in} = d_{out} = d$이면 (d_{in}, d_{out})-차 마스킹 기법을 줄여서 d-차 마스킹 기법이라고도 한다. 일반적으로 입력 마스크 차수 d_{in}과 출력 마스크 차수 d_{out}은 마스킹 기법의 안전성과 관련이 있으며 d_{in}과 d_{out}이 커지면 마스킹 기법을 공격하기 위해 공격자가 투입해야 하는 자원의 양은 증가하는 경향이 있다. 또한 마스킹 기법에서 각 출력 마스크 z_i는 입력 마스크 $(x_0, x_1, ..., x_{d_{in}})$로부터 계산 가능하며 이 계산을 담당하는 함수를 f_i로 표기하면 $(f_0, f_1, ..., f_{d_{out}})$를 함수 $z = f(x)$의 마스킹 회로라고 한다. 따라서 $i = 0, 1, ..., d_{out}$에 대하여 $f_i : K_1^{d_{in}+1} \rightarrow K_2$이고 $z_i = f_i(x_0, x_1, ..., x_{d_{in}})$가 성립한다.

마스킹 기법을 선형 함수 또는 affine 함수에 적용하는 것은 그리 어렵지 않다. 예를 들어 $f : K_1 \rightarrow K_2$가 선형 함수이고 입력값 x에 대한 d-차 마스크 $(x_0, x_1, ..., x_d)$가 주어졌을 때 $z_i, i = 0, 1, ..., d$를 $z_i = f(x_i)$라고 하면 $z_0 \oplus z_1 \oplus ... \oplus z_d = f(x)$이고 f의 계산은 x_i만 사용하며 따라서 $f(x_i)$의 계산 과정에 x에 대한 어떠한 정보의 노출도 발생하지 않기 때문에 $(f, f, ..., f)$는 f의 안전한 마스킹 회로 구현이 된다. 하지만 마스킹 기법을 적용하고자 하는 함수가 블록 암호 알고리즘의 S-box처럼 (XOR 연산에 대한) 비선형 함수인 경우 마스킹 기법을 적용하는 것은 쉽지 않다고 알려져 있다. 그리고 이 문제를 하드웨어적으로 해결하기 위하여 게이트 레벨 마스킹 기법(gate-level masking method)이 제안되었다.

게이트 레벨 마스킹 기법은 다음과 같은 사실에 기반을 둔다:

- 임의의 회로는 기본 게이트인 AND, OR, XOR, NAND, NOR, XNOR, NOT 게이트의 합성 함수(composite function)로 표현 가능하다.
- 임의의 회로를 AND, OR, XOR, NAND, NOR, XNOR, NOT 게이트의 합성 함수로 표현한 후 각각의 기본 게이트에 마스킹 기법을 적용한 결과는 원래 회로에 대한 하나의 마스킹 기법 구현이 된다.

따라서 게이트 레벨 마스킹 기법의 주요 연구 주제는 기본 게이트인 AND, OR, XOR, NAND, NOR, XNOR, NOT 게이트에 대한 효율적이고 안전한 마스킹 기법의 적용이 된다. 그리고 모든 기본 게이트는 NAND 게이트만으로 구현 가능하기 때문에 결국 게이트 레벨 마스킹 기법에서는 NAND 게이트 또는 (NAND 게이트와 NOT 게이트의 합성인) AND 게이트에 적용 가능한 효율적이고 안전한 마스킹 기법의 설계를 주로 다루게 된다. 예를 들어 논문 [7]은 추가적인 랜덤 비트(r)를 사용하여 AND 게이트 $z = f(x, y) = xy$에 적용한 다음과 같은 마스킹 기법을 제시하였다: 입력 x, y의 1차 마스크 $(x_0, x_1), (y_0, y_1)$과 랜덤 비트 r에 대하여 AND 게이트의 출력 마스크 (z_0, z_1)는 다음과 같이 계산된다

$$z_0 = (((r \oplus x_0 y_0) \oplus x_0 y_1) \oplus x_1 y_0) \oplus x_1 y_1$$
$$z_1 = r$$

2.3. 글리치 공격과 하드웨어 마스킹 기법

글리치 공격 이전에 제안된 게이트 레벨 마스킹 기법 대부분은 해당 게이트의 모든 입력 신호가 동시에 게이트에 도착한다는 가정에 기반을 두었다. 하지만 실제로

게이트의 입력 신호들은 여러 가지 이유로 서로 다른 시간에 도착한다. 예를 들어 신호가 게이트를 통과하게 되면 게이트 지연 시간이 발생할 수 있고 또한 신호가 거쳐야 하는 path의 길이에 따라 역시 게이트 도달 시간이 다를 수 있다. 그리고 이러한 게이트 지연과 path 길이에 따른 게이트 도달 시간의 편차는 가장 많이 사용되는 반도체 공정인 CMOS 공정에서 매우 일반적인 현상이며 이러한 현상을 완전히 제거하는 것은 매우 어려운 작업이라고 알려져 있다.

입력 신호가 서로 다른 시간에 게이트에 도착하게 되면 해당 게이트의 출력값은 한 클록 싸이클 내에서 안정화되기 전까지 여러 번 변하게 되는데 이러한 현상을 글리치(glitch)라고 하며 이러한 글리치 현상은 실제 회로가 소비하는 전력량에 많은 영향을 주게 된다. 그리고 글리치 현상과 전력소비량 사이의 관계를 이용하여 암호 기기 내에 저장된 비밀 정보를 알아내는 전력분석 공격법을 글리치 공격이라고 한다([8,9,10]). 일반적으로 전자 회로는 값을 유지할 때보다 값이 변할 때 더 많은 전력을 소비하고 글리치란 결국 회로의 출력값이 여러 번 변하는 현상을 의미하기 때문에 글리치가 발생하지 않는 경우에 비해 글리치가 발생할 때 더 많은 전력을 소비하게 된다. 게다가 글리치 패턴은 회로의 입력값에 강하게 의존하게 되는데 이러한 의존성이 바로 글리치 공격이 가능한 기본 이유가 된다. 글리치 공격은 특히 하드웨어 마스킹 기법에 더 심각한 영향을 준다고 알려져 있는데 그 이유는, 소프트웨어 마스킹 기법의 경우 바이트 혹은 워드 단위로 마스킹 회로를 구현하고 따라서 글리치 특성이 전체적으로 상쇄되는 경향이 있지만 하드웨어 마스킹 기법의 경우에는 비트 단위의 마스킹 회로를 구현하기 때문에 글리치 특성을 좀 더 명확하게 분석할 수 있기 때문이다.

전력분석 공격은 일반적으로 차수가 커짐에 따라 그 난이도가 올라가는데 여기서 전력분석 공격의 차수(order)란 공격에 사용되는 통계학적 moments의 종류 또는 전력분석 공격에 사용되는 전력소비량 측정 시점의 개수를 의미한다. 예를 들어 평균을 이용하는 전력분석 공격을 1차 전력분석 공격, 분산을 이용하는 공격을 (univariate) 2차 전력분석 공격이라고 정의한다. 또한 1개의 특정 시점에서 수집된 전력소비량을 이용한 전력분석 공격을 1차 전력분석 공격, 서로 다른 2개의 시간 시점에서 수집된 전력소비량을 이용한 전력분석 공격을 (multivariate) 2차 전력분석 공격이라고 한다. 그리고 글리치 공격 이전에는 d-차 전력분석 공격에 대응하기 위해서는 일반적으로 d-차 마스킹 기법, 즉 입력 신호를 $d+1$개의 마스크로 분해하는 방법이 충분한 안전성을 제공해줄 것으로 예상이 되었다. 하지만 글리치 공격이 제안된 이후로 하드웨어 마스킹 기법과 소프트웨어 마스킹 기법은 전혀 다른 분야로 분화가 되었으며 결론적으로 소프트웨어 마스킹 기법에서는 d - 차 전력분석 공격에 대한 대응은 d-차 마스킹 기법의 적용으로 충분히 가능하다고 여겨지고 있지만 하드웨어 마스킹 기법에 관한 여러 연구에서는 d - 차 전력분석 공격에 대응하기 위해서 입력 신호를 $(d+1)$개보다 더 많은 마스크로 분해하는 것이 필요할 것이라고 예측하고 있다([17,19,20]). 그리고 더 많은 마스크를 사용하면 회로 면적과 연산 시간 등의 증가를 초래하기 때문에 결국 하드웨어 마스킹 기법은 소프트웨어 마스킹 기법에 비해 그 구현에 있어서 더 많은 계산 자원을 필요로 할 것이라고 예측되고 있다.

Ⅲ. 글리치 공격에 대한 대응기법

글리치 공격이 제안된 이후로 다양한 하드웨어 기반 대응 기법이 제안되었으며 다음은 이러한 대응 기법의 몇 가지 예이다.

- 글리치 특성이 없는 새로운 공정 기술 또는 셀 구조
- 다자 계산 방법과 비밀 분산 기법의 결합
- Hazard 대수
- 마스킹 회로에 적절한 방식으로 레지스터 또는 latch를 추가
- G-equivariant 게이트
- 임계치 구현 기반 마스킹 기법

글리치 공격에 대한 첫 번째 대응 방법으로는 글리치 자체가 발생하지 않는 공정 기술이나 셀을 새로 개발하는 것이 될 수 있으며 dual-rail precharge logic이 대표적인 예가 된다([11]). dual-rail precharge logic의 기본 원리는 먼저 각 비트를 2개의 비트, 즉 비트 0은 01로, 비트 1은 10으로 확장한 후 이렇게 확장된 비트들에 필요한 연산을 적용하기 전에 precharge 단계를 추가함으

로써 글리치 발생을 방지하는 것이다. 하지만 공정 기술이나 셀을 새로 개발하는 것은 많은 비용이 필요하다는 문제점이 있다.

다자 계산(Multi-Party Computation) 방법과 비밀 분산 기법(Secret Sharing Scheme)을 결합하여 글리치 공격에 대응할 수도 있다. 여기서 다자 계산이란, 가령 n명의 참여자 $U_i(1 \le i \le n)$가 비밀 정보 $Z_i(1 \le i \le n)$를 각각 보유하고 있다고 가정했을 때 특정 함수 f에 대하여 함수값 $f(Z_1, ..., Z_n)$를 비밀 정보 Z_i의 노출 없이 계산하는 방법에 관한 것으로 특히 일부 악성 참여자들이 서로 모의를 하더라도 계산 과정이 왜곡되거나 방해를 받지 않아야 한다([12]). Shamir의 비밀 분산 기법은 특정 비밀 정보를 다수의 참여자에게 분산시키는 방법에 관한 기술로 참여자 중 일부가 서로 악의적인 모의를 하더라도 해당 비밀 정보가 외부로 노출되지 않아야 함을 요구한다([13]). 그리고 다자 계산 방법과 비밀 분산 기법을 결합하면 다수의 참여자가 비밀 정보를 분산 저장한 후 그 비밀 정보의 노출 없이도 분산 저장된 비밀 정보에 대한 특정 연산값을 계산할 수 있으며 BGW 프로토콜은 이러한 계산을 위해 제안된 대표적인 프로토콜로 알려져 있다([14]). 특히 BGW 프로토콜을 사용하면 글리치 공격에 대한 대응도 가능한데, 가령 프로토콜의 참여자 각각을 입력 마스크 중의 하나를 저장하고 있는 암호 모듈이라고 가정하고 BGW 프로토콜을 적용하면 임의의 유한체 연산에 대한 마스킹 기법 구현이 가능하다([15]). 하지만 이렇게 다자 계산 방법과 비밀 분산 기법을 결합하여 마스킹 회로를 구현하게 되면 연산 오버헤드가 많이 발생하여 하드웨어 구현에 적합하지 않게 된다.

글리치 공격에 대한 대응 기법으로 알려진 또 다른 방법으로 hazard 대수(algebra)가 있다. 여기서 hazard란 글리치 현상을 의미하는 또 다른 용어로서 hazard 대수는 글리치 현상을 모델링한 수학적 개념이다([16]). 하지만 극소수 연구자만이 hazard 대수에 관한 연구를 진행하고 있기 때문에 아직은 관련 연구 결과가 미흡한 실정이다.

글리치 공격이 가능한 가장 큰 이유는 게이트 입력 신호가 서로 다른 시간에 도착한다는 점에 있다. 따라서 마스킹 회로 중간에 적절한 레지스터(또는 latch)를 추가하게 되면 입력 신호의 도달 시간의 편차를 어느 정도 줄일 수 있고 따라서 글리치 효과를 축소할 수 있다. 따라서 하드웨어 회로에 레지스터 등을 추가하는 것은 글리치 공격 방어에 어느 정도 기여는 할 수 있으나 완벽하게 글리치 공격을 방어하지는 못한다. 따라서 이 방법은 글리치 공격에 대한 다른 대응 기법의 보완 방법으로 주로 사용되며 특히 이후에 설명할 임계치 구현 기반 마스킹 기법은 그 특성상 계산 중간에 적절한 레지스터를 추가함으로써 좀 더 향상된 글리치 공격에 대한 안전성을 제공하게 된다([17]).

글리치가 발생하게 되면 게이트의 출력 신호는 안정화되기 전까지 값이 계속 변하는 toggling 현상이 발생하게 되고 toggling의 횟수에 따라 게이트가 실제 소비하는 전력량은 달라지게 된다. G-equivariant 게이트는 게이트 입력 신호의 도착 순서에 상관없이 toggling 개수가 동일하면 글리치 공격을 방어할 수 있다는 원리에 기반을 둔 마스킹 회로를 의미한다. 실제로 논문 [18]에서는 AND 게이트 $z = f(x,y) = xy$에 대한 입력 x, y의 2차 마스크 $(x_0, x_1, x_2), (y_0, y_1, y_2)$가 주어졌을 때 다음과 같이 주어진 f의 마스킹 회로 (f_0, f_1, f_2)는 G-equivariant함을, 즉 입력 신호 $x_0, x_1, x_2, y_0, y_1, y_2$가 어떤 순서로 f_i에 도착하더라도 각 출력 마스크 z_i의 toggling 패턴은 모두 동일함을 보여주었다:

$$f_0(x_0, x_1, x_2, y_0, y_1, y_2) = x_0y_0 \oplus x_2y_2 \oplus x_2y_0$$
$$f_1(x_0, x_1, x_2, y_0, y_1, y_2) = x_0y_1 \oplus x_1y_2 \oplus x_0y_2$$
$$f_2(x_0, x_1, x_2, y_0, y_1, y_2) = x_1y_1 \oplus x_1y_0 \oplus x_2y_1$$

Ⅳ. 임계치 구현 기반 마스킹 기법

4.1. Correctness, Non-Completeness and Uniformness

임계치 구현 기반 마스킹 기법(Threshold Implementation, 이하 TI)는 글리치 공격에 대한 대응 기법이면서 1차 DPA에 대한 증명 가능한 안전성까지 제공해준다는 사실 때문에 많은 주목을 받고 있는 대표적인 하드웨어 기반 마스킹 기법으로 correctness, non-completeness, uniformness라는 3가지 특성을 만족시켜야 한다([17,19,20]).

TI가 만족시켜야 하는 특성 중에서 correctness는 TI 뿐만 아니라 일반적인 마스킹 기법이 만족시켜야 하는

기본적인 성질로서, 함수 $z=f(x)$의 마스킹 회로 $(f_0, f_1, ...)$이 주어진 경우 임의의 입력 x와 x의 임의의 마스크 $(x_0, x_1, ...)$에 대하여 다음이 성립되어야 함을 요구한다:

$$\bigoplus_i f_i(x_0, x_1, ...) = f(x)$$

즉 correctness란 마스킹 회로의 출력이 원함수의 출력 마스크와 동일해야 함을 의미한다.

TI가 만족시켜야 할 두 번째 성질인 non-completeness는 TI를 대표하는 특성이며 함수 $z=f(x)$의 마스킹 회로 $(f_0, f_1, ...)$가 1차 non-completeness를 만족시킨다는 의미는 f의 각 출력 마스크 $z_i = f_i(x_0, x_1, ...)$의 계산에 모든 x_j가 사용되는 것이 아니라 그 중에서 적어도 하나는 사용되지 않는다는 것이다. 좀 더 일반적으로 $(f_0, f_1, ...)$가 d-차 non-completeness를 만족시킨다는 것은 출력 마스크 중에서 임의로 d개를 선택했을 때 이 d개 출력 마스크 계산에 적어도 하나의 입력 마스크가 사용되지 않음을 의미한다.

일반적으로 선형함수에 대하여 non-completeness를 만족시키는 마스킹 회로를 설계하는 것은 어렵지 않다. 예를 들어 XOR 게이트 $z=f(x,y)=x\oplus y$에 대하여 x,y의 1차 마스크 $(x_0, x_1), (y_0, y_1)$가 주어졌을 때 다음과 같이 계산하는 f의 마스킹 회로 (f_0, f_1)는 각 f_i의 계산에 x_i, y_i를 사용하지 않기 때문에 1차 non-completeness를 만족시킴을 알 수 있다:

$$f_0(x_0, x_1, y_0, y_1) = x_1 \oplus y_1$$
$$f_1(x_0, x_1, y_0, y_1) = x_0 \oplus y_0$$

선형 함수와는 달리 비선형 함수에 대해서 non-completeness를 만족시키는 효율적이고 안전한 마스킹 기법을 설계하는 것은 상당히 어렵다고 알려져 있으며 따라서 non-completeness를 만족시키는 비선형 함수에 대한 안전하고 효율적인 마스킹 회로의 설계는 상당히 중요한 연구 주제가 된다. 이와 관련하여 논문 [17]에서는 다음과 같이 주어진 AND 게이트 $z=f(x,y)=xy$에 대한 1차 non-complete 마스킹 회로 (f_0, f_1, f_2)를 제안하였다:

$$f_0(x_0, x_1, x_2, y_0, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1$$
$$f_1(x_0, x_1, x_2, y_0, y_1, y_2) = x_2y_2 \oplus x_2y_0 \oplus x_0y_2$$
$$f_2(x_0, x_1, x_2, y_0, y_1, y_2) = x_0y_0 \oplus x_0y_1 \oplus x_1y_0$$

또한 논문 [20]은 대수적 차수(algebraic degree)가 t인 비선형 함수에 대하여 입력 마스크의 개수가 $td+1$, 출력 마스크의 개수가 $\binom{td+1}{t}$인 d-차 non-complete 마스킹 회로를 만드는 일반적인 방법을 제안하였으며 다음은 (대수적 차수가 2인) AND 게이트 $z=f(x,y)=xy$에 대한 2차 non-complete 마스킹 회로의 예이다:

$$z_0 = x_1y_1 \oplus x_0y_1 \oplus x_1y_0$$
$$z_1 = x_2y_2 \oplus x_0y_2 \oplus x_2y_0$$
$$z_2 = x_3y_3 \oplus x_0y_3 \oplus x_3y_0$$
$$z_3 = x_0y_0 \oplus x_0y_4 \oplus x_4y_0$$
$$z_4 = x_1y_2 \oplus x_2y_1$$
$$z_5 = x_1y_3 \oplus x_3y_1$$
$$z_6 = x_4y_4 \oplus x_1y_4 \oplus x_4y_1$$
$$z_7 = x_2y_3 \oplus x_3y_2$$
$$z_8 = x_2y_4 \oplus x_4y_2$$
$$z_9 = x_3y_4 \oplus x_4y_3$$

TI의 마지막 성질인 uniformness는 간단히 말해서, 마스킹이 적용되기 전의 함수 출력 값의 확률분포가 마스킹이 적용된 후에도 동일하게 유지가 될 뿐만 아니라 동일한 출력 값에 대한 출력 마스크의 확률은 모두 동일해야 함을 요구한다. 예를 들어 AND 게이트 $z=f(x,y)=xy$의 출력 비트에 대한 확률 분포는 다음 표 1과 같이 주어진다

그리고 AND 게이트에 대해서 아래의 (1)처럼 주어진 마스킹 회로의 출력 비트 (z_0, z_1, z_2)에 대한 확률 분포는 표 2에 주어진다.

〔표 1〕 AND 게이트의 출력비트 확률분포

출력비트	0	1
확률	3/4	1/4

〔표 2〕 마스킹 회로 (1)의 출력비트 확률분포

출력비트	000	011	101	110
확률	21/64	9/64	9/64	9/64
출력비트	001	010	100	111
확률	5/64	5/64	5/64	1/64

$$\begin{aligned} z_0 &= x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \\ z_1 &= x_2y_2 \oplus x_2y_0 \oplus x_0y_2 \\ z_2 &= x_0y_0 \oplus x_0y_1 \oplus x_1y_0 \end{aligned} \quad (1)$$

표 2가 보여주듯이 원래의 출력비트 0에 해당하는 마스크 000, 011, 101, 110의 확률이 동일하지 않기 때문에 결국 (1)에 주어진 마스킹 회로는 uniform하지 않다는 것을 알 수 있다. 반면에 아래의 (2)에 주어진 AND 게이트에 대한 마스킹 회로의 출력비트 확률분포는 표 3에 주어진다. ()

$$\begin{aligned} z_0 &= x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus r \\ z_1 &= x_2y_2 \oplus x_2y_0 \oplus x_0y_2 \oplus s \\ z_2 &= x_0y_0 \oplus x_0y_1 \oplus x_1y_0 \oplus r \oplus s \end{aligned} \quad (2)$$

식 (2)에서 r, s는 랜덤하게 선택된 비트를 의미한다. 표 3이 보여주듯이 출력비트 0에 해당하는 마스크 000, 011, 101, 110 각각의 확률은 모두 48/256로 동일하고 출력비트 1에 해당하는 마스크 001, 010, 100, 111 각각의 확률은 모두 16/256로 동일하다. 또한 출력비트 0에 해당하는 마스크 각각의 확률 48/256은 출력비트 1에 해당하는 마스크 각각의 확률 16/256의 3배가 되는데 이는 표 1에 제시된 AND 게이트의 출력비트 확률분포가 마스킹이 적용된 후에도 유지가 됨을 의미하고 결론적으로 마스킹 회로 (2)는 uniformness를 만족시킴을 알 수 있다.

마스킹 회로가 uniformness를 만족시켜야 하는 이유는 마스킹 회로의 안전성 증명과 관련이 있다. 논문

〔표 3〕 마스킹 회로 (2)의 출력비트 확률분포

출력비트	000	011	101	110
확률	48/256	48/256	48/256	48/256
출력비트	001	010	100	111
확률	16/256	16/256	16/256	16/256

[19]에 의하면, non-complete한 마스킹 회로의 입력 마스크가 uniform한 확률분포를 따르면 해당 마스킹 회로는 글리치 공격과 1차 DPA에 대하여 안전함을 증명할 수 있다. 따라서 가령 A라는 마스킹 회로의 출력이 B라는 마스킹 회로의 입력으로 사용된다고 했을 때 만약 A 마스킹 회로의 출력이 uniform하고 B 마스킹 회로가 non-complete하면 B 마스킹 회로는 글리치 공격 및 1차 DPA에 대한 안전성을 수학적으로 증명할 수 있다. 하지만 A 마스킹 회로의 출력이 uniform하지 않으면 B 마스킹 회로가 non-complete하더라도 B 마스킹 회로의 안전성 증명에 [19]의 결과를 적용할 수 없다. 결국 uniformness와 non-completeness를 결합하면 마스킹 회로의 안전성 증명이 매우 수월해지는 장점을 가지게 된다.

일반적으로 선형 함수에 대해서 uniformness를 만족시키는 마스킹 회로를 설계하는 것은 어렵지 않다. 하지만 비선형 함수에 대해서 uniformness를 만족시키는 마스킹 회로를 설계하는 것은 용이하지 않다. 그리고 비선형 함수에 대하여 uniform한 마스킹 회로를 구현하더라도 구현된 회로는 선형 함수에 대한 uniform한 마스킹 회로에 비해 훨씬 더 많은 연산을 사용하게 된다.

마스킹 회로를 uniform하게 만드는 일반적인 방법으로는 다음 2가지 방법이 제안되었다:

- 랜덤비트를 사용하는 방법
- 마스크의 개수를 증가시키는 방법

먼저 랜덤비트를 적절하게 사용하면 마스킹 회로를 uniform하게 만들 수 있는데 그 예로 (2)에서 제시된 AND 게이트 마스킹 회로가 있다. 실제로 (1)에서 제시된 AND 게이트 마스킹 회로가 uniform하지 않지만 여기에 2개의 랜덤 비트 r, s를 (2)와 같이 사용하면 해당 회로는 uniform하게 되며 이 방법은 다른 마스킹 회로에 대해서도 유사하게 적용 가능하다.

마스킹 회로를 uniform하게 만드는 또 다른 방법으로 마스크의 개수를 증가시키는 방법이 있다. 예를 들어 입력 마스크의 개수가 3이고 랜덤 비트를 사용하지 않으면서 non-complete하고 uniform한 AND 게이트 마스킹 회로를 만드는 방법은 없다고 알려져 있다. 하지만 만약 입력 마스크의 개수를 4로 증가시키면 non-completeness와 uniformness를 동시에 만족시키는

AND 게이트에 대한 마스킹 회로는 가능한데 다음은 이러한 회로의 한 예가 된다:

$$\begin{aligned} z_0 &= (x_2 \oplus x_3)(y_1 \oplus y_2) \oplus y_1 \oplus y_2 \oplus \\ &\quad y_3 \oplus x_1 \oplus x_2 \oplus x_3 \\ z_1 &= (x_0 \oplus x_2)(y_0 \oplus y_3) \oplus y_0 \oplus y_2 \oplus \\ &\quad y_3 \oplus x_0 \oplus x_2 \oplus x_3 \\ z_2 &= (x_1 \oplus x_3)(y_0 \oplus y_3) \oplus y_1 \oplus x_1 \\ z_3 &= (x_0 \oplus x_1)(y_1 \oplus y_2) \oplus y_0 \oplus x_0 \end{aligned} \quad (3)$$

(3)이 non-completeness를 만족시킨다는 것은 z_i의 계산에 x_i, y_i가 사용되지 않는다는 사실로부터 알 수 있고, uniformness를 만족시킨다는 것을 보이는 것은 x_i, y_j의 랜덤한 선택에 따른 (z_0, z_1, z_2, z_3)의 확률분포를 계산하는 것으로 충분하다.

4.2. CMS와 DOM

글리치 공격 및 1차 DPA에 대한 증명 가능한 안전성을 제공해 준다는 장점에도 불구하고 TI는 몇 가지 단점을 가지고 있으며 대표적으로는 TI가 사용하는 마스크의 개수와 관련이 있다. 앞 절에서 소개하였듯이 대수적 차수가 t인 비선형함수에 대하여 입력 마스크의 개수가 $td+1$이고 출력 마스크의 개수가 $\binom{td+1}{t}$이며 d차 non-completeness를 만족시키는 마스킹 회로가 TI를 대표하는 회로로 알려져 있다. 하지만 이런 방식의 마스킹 회로는 $d+1$개의 마스크를 사용하는 소프트웨어 마스킹 기법에 비해 훨씬 더 많은 마스크를 사용하며 따라서 마스킹 회로의 구현에 필요한 연산량의 증가와 그로 인한 칩면적 증가를 불러오게 된다. 그리고 CMS(Consolidating Masking Scheme)는 기존에 제안된 TI의 마스크 개수를 줄이기 위해 제안된 또 다른 마스킹 기법이다([21]).

CMS는 대수적 차수가 $t(>1)$인 비선형 함수에 대하여 d차 DPA에 대응하기 위해 $td+1$개의 입력 마스크를 사용하는 기존의 TI와는 달리 $d+1$개의 입력 마스크만을 사용하여 마스킹 회로를 구성한다. 그리고 이렇게 입력 마스크의 개수를 줄이게 되면 이를 구현하는 회로의 면적은 줄어드는 장점이 있지만 다른 한편으로 CMS의 출력 마스크의 uniformness를 위해 기존 TI에 비해 더 많은 랜덤 비트를 사용해야 한다는 단점이 있다.

CMS는 다음과 같은 4단계를 통해 마스킹 회로를 구현한다([21]);

- Nonlinear 단계: 이 단계에서는 마스킹 회로의 구현에 필요한 모든 비선형 항을 계산한다. 예를 들어 AND 게이트 $z = xy$에 대하여 2차 DPA에 대응하는 마스킹 회로 구현을 위해서 이 단계에서는 3개의 마스크로 구성된 입력 $(x_0, x_1, x_2), (y_0, y_1, y_2)$로 부터 $x_i y_j (i, j = 0,1,2)$ 를 계산한다.
- Linear 단계: 이 단계에서는 마스킹 회로의 non-completeness 특성을 훼손하지 않는 범위에서 이전 Nonlinear 단계에서 계산된 항들에 대한 XOR 합을 구한다. 예를 들어 AND 게이트 $z = xy$에 대한 2차 DPA 대응 마스킹 회로 구현을 위해서 이전 Nonlinear단계에서 계산한 $x_i y_j$ 항들의 일부 항을 XOR 연산으로 결합하는 단계가 Linear 단계에 해당하지만 이 경우 $x_i y_j$ 항들 중 어떤 2개 항도 XOR 연산으로 결합되면 2차 non-completeness 특성이 훼손되기 때문에 이 예에 대해서 Linear 단계에서 할 수 있는 작업은 없다고 할 수 있다.
- Refreshing 단계: 이 단계는 이전 Linear 단계에서 계산한 항들이 이후 계산 단계의 입력으로 사용될 경우 랜덤 비트의 추가를 통해 uniformness를 만족시키도록 하면서 동시에 마스킹 회로의 correctness가 훼손되지 않도록 만든다. 예를 들어 AND 게이트 $z = xy$에 대한 2차 DPA 대응 마스킹 회로 구현의 Linear 단계에서 계산한 항들을 $z_0 = x_0 y_0$, $z_1 = x_0 y_1$, $z_2 = x_0 y_2$, $z_3 = x_1 y_0$, $z_4 = x_1 y_1$, $z_5 = x_1 y_2$, $z_6 = x_2 y_0$, $z_7 = x_2 y_1$, $z_8 = x_2 y_2$라고 할 때 Refreshing 단계에서는 9개의 랜덤 비트 $r_i, i = 0, 1, ..., 8$를 사용하여 다음을 계산한다: $z_0' = z_0 \oplus r_8 \oplus r_0$, $z_1' = z_1 \oplus r_0 \oplus r_1$,

$z_2' = z_2 \oplus r_1 \oplus r_2$, $z_3' = z_3 \oplus r_2 \oplus r_3$, $z_4' = z_4 \oplus r_3 \oplus r_4$, $z_5' = z_5 \oplus r_4 \oplus r_5$, $z_6' = z_6 \oplus r_5 \oplus r_6$, $z_7' = z_7 \oplus r_6 \oplus r_7$, $z_8' = z_8 \oplus r_7 \oplus r_8$. 그리고 이 때 계산된 $(z_0', ..., z_8')$는 uniformness와 correctness를 동시에 만족시킨다는 것을 어렵지 않게 보일 수 있다.

- Compression 단계: 이 단계는 마스킹 회로의 입력 마스크의 개수와 출력 마스크의 개수가 다를 경우 이를 같게 만들기 위해서 이전 단계에서 계산된 항들에 적절한 XOR 연산을 수행하는 단계이다. 예를 들어 AND 게이트 $z = xy$에 대한 2차 DPA 대응 마스킹 회로 구현을 위해 Refreshing 단계에서 계산한 $(z_0', ..., z_8')$에 대하여 이 단계는 다음과 같은 계산을 수행하여 출력 마스크의 개수를 3개로 줄이게 된다: $z_0'' = z_0' \oplus z_1' \oplus z_2'$, $z_1'' = z_3' \oplus z_4' \oplus z_5'$, $z_2'' = z_6' \oplus z_7' \oplus z_8'$. 이 때 한 가지 주의할 점은 CMS의 Refreshing 단계와 Compression 단계가 한 클록에 실행이 되면 글리치 현상 때문에 안전성에 문제가 생길 수 있다는 것이다. 따라서 CMS에서는 이를 방지하기 위해서 Refreshing 단계와 Compression 단계 사이에 레지스터 등을 추가하여 Refreshing 단계와 Compression 단계가 같은 클록에서 실행되는 것을 방지하고 결국 글리치 공격에 대응하게 된다.

이렇게 정의된 CMS는 Refreshing 단계에서 출력 마스크의 uniformness를 위해 많은 랜덤 비트를 사용할 것을 요구한다는 단점이 있다. 가령 앞에서 예로 든 AND 게이트 $z = xy$에 대한 2차 DPA에 대응하는 마스킹 회로는 CMS 구현을 위해 Refreshing 단계에서 9개의 랜덤 비트를 사용한다. 그리고 이러한 랜덤 비트의 사용은 결국 또 다른 칩면적의 증가를 초래하게 된다. 좀 더 자세하게 설명하면, 마스킹 회로에 랜덤 비트를 사용하기 위해서는 2가지 방법이 가능한데 하나는 암호 기기 자체에 난수발생기(Random Number Generator)를 구현하는 것이고 다른 하나는 필요한 만큼의 랜덤 비트를 외부에서 생성한 후 이를 암호 기기 내에 저장한 후 사용하는 것이다. 하지만 난수발생기의 구현은 또 다른 회로 면적의 증가를 초래하게 되고, 랜덤 비트를 저장해서 사용하는 경우에도 이를 위한 레지스터 등의 구현이 필요하기 때문에 결국 두 방법 모두 원치 않은 칩면적의 증가가 발생하게 된다. 따라서 TI나 CMS에서 uniformness를 만족시키기 위해 사용하는 랜덤 비트의 개수를 최대한 줄이게 되면 그만큼 효율적인 마스킹 회로 구현이 가능해지며 이를 위해 제안된 마스킹 회로 구현 방법이 바로 DOM(Domain Oriented Masking)이다([22]).

DOM의 구현을 위해서는 Domain이라는 개념이 필요하다. 실제로 DOM에서는 입력 마스크의 인덱스(index)가 같으면 같은 Domain에 있다고 정의하는데 예를 들어 AND 게이트 $z = xy$에 대하여 2차 DPA에 대응하는 마스킹 회로 구현을 위해서 3개의 마스크로 구성된 입력 $(x_0, x_1, x_2), (y_0, y_1, y_2)$이 주어졌다고 가정하면 x_i와 y_i는 모두 동일한 인덱스 i를 사용하기 때문에 같은 Domain에 있다고 하고, 만약 $i \neq j$이면 x_i와 y_j는 다른 인덱스를 사용하기 때문에 다른 Domain에 있다고 정의한다. 이렇게 Domain이라는 개념을 정의한 후 DOM은 먼저 마스킹 회로의 구현을 위해 CMS의 Nonlinear 단계처럼 $x_i y_j$ 항을 계산한다. 그리고 CMS가 Refreshing 단계에서 uniformness를 위해 모든 $x_i y_j$에 랜덤 비트를 더해 주었다면 DOM은 다른 Domain에 있는 항들의 곱 $x_i y_j (i \neq j)$에 대해서만 랜덤 비트를 더해 주며 따라서 필요한 랜덤 비트의 개수가 줄어드는 장점을 가지게 된다 (보다 자세한 랜덤 비트의 추가 방법은 논문 [22] 참조). 또한 CMS에서는 Refreshing 단계와 Compression 단계 사이에 모든 항들을 저장할 수 있는 레지스터의 추가를 요구하는 반면에 DOM은 다른 Domain에 있는 항들의 곱 $x_i y_j (i \neq j)$과 관련된 항들만 레지스터로 저장할 것을 요구하기 때문에 CMS에 비해 구현되는 레지스터의 크기가 줄어드는 장점을 보여준다. 결론적으로 DOM은 랜덤 비트의 사용을 줄이고 레지스터의 크기를 줄임으로써 CMS에 비해 더욱 효율적인 마스킹 회로의 구현을 가능하게 한다.

Ⅴ. 결 론

최근 인공지능, 블록체인, 클라우드 컴퓨팅, 빅데이

터, IoT 기술이 사회 구조를 혁명적으로 변화시킬 수 있는 기술로 간주되며 많은 사회적인 관심을 받고 있다. 하지만 이러한 기술의 사용에 있어서 개인 정보, 금융 정보 등과 같은 중요 정보의 누출 사고를 미연에 방지하지 못하면 기술 발전의 장점을 충분히 누릴 수 없게 된다. 그런 의미에서 암호 기술은 혁신적인 ICT 기술 발전의 기반이 된다고 할 수 있다.

암호 알고리즘은 그 구현에 있어서 속도, 메모리 사용량 등과 같은 성능적인 측면 뿐만 아니라 부채널분석 공격에 대한 안전성 측면 역시 고려되어야 한다. 실제로 부채널분석 공격에 대한 대응을 고려하지 않고 암호 알고리즘을 구현할 경우 그 결과물은 보안상의 문제를 해결하는 데에 무용지물이 될 우려가 있기 때문에 CC(Common Criteria), EMVCo, UnionPay 등과 같은 보안인증평가는 부채널분석 공격법에 대한 대응 기술의 구현을 중요 평가 항목으로 간주한다.

본 논문에서는 부채널분석 공격에 대한 대응기법 중에서 하드웨어 기반 마스킹 기법의 연구 동향에 대해서 알아보았으며 특히 블록 암호 알고리즘을 DPA 및 글리치 공격에 안전하게 구현하기 위해서 제안된 대표적인 마스킹 방법인 TI의 여러 가지 특성에 대해서 살펴보았다.

하드웨어 기반 부채널분석 공격 대응기법은 그 구현 방식에 따라 하드웨어 장치 제조비용 및 안전성에 많은 영향을 주게 된다. 따라서 이에 대한 최신 기술을 습득하는 것은 국산 하드웨어 암호모듈의 경쟁력을 향상시킬 수 있을 것을 예상되기 때문에 지속적인 연구가 필요하다.

참 고 문 헌

[1] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[2] P. Kocher, Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS, and Other Systems, CRYPTO 96, LNCS 1109, pp. 104-113, 1996.

[3] P. Kocher, J. Jaffe and B. Jun, Differential Power Analysis, CRYPTO 99, LNCS 1666, pp. 388-397, 1999.

[4] J. Quisquater and D. Samyde, ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards, E-smart 2001, LNCS 2140, pp.200-210, 2001.

[5] D. Boneh, R. Demillo and R. Lipton, On the Importance of Checking Cryptographic Protocols for Faults, EUROCRYPT 97, LNCS 1233, pp. 37-51, 1997.

[6] T.S. Messerges, Securing the AES Finalists against Power Analysis Attacks, FSE 2000, LNCS 1978, pp. 150-164, 2000.

[7] E. Trichina, T. Korkishko and K.H. Lee, Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results, AES Conference, LNCS 3373, pp. 113-127, 2003.

[8] S. Mangard and K. Schramm, Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations, CHES 2006, LNCS 4249, pp. 76-90, 2006.

[9] S. Mangard, T. Popp and B.M. Gammel, Side-Channel Leakage of Masked CMOS Gates, CT-RSA 2005, LNCS 3376, pp. 351-365, 2005.

[10] S. Mangard, N. Pramstaller and E. Oswald, Successfully Attacking Masked AES Hardware Implementations, CHES 2005, LNCS 3659, pp. 157-171, 2005.

[11] T. Popp and S. Mangard, Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints, CHES 2005, LNCS 3659, pp. 172?186, 2005.

[12] A.C.-C. Yao, How to Generate and Exchange Secrets, Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pp. 162?167, 1986.

[13] A. Shamir, How to Share a Secret, Commun. ACM 22(11), pp. 612?613, 1979.

[14] M. Ben-Or, S. Goldwasser and A. Wigderson, Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computation, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 1?10, 1988.

[15] T. Roche and E. Prouff, Higher-Order Glitches Free Implementation of the AES Using Secure Multi-Party Computation Protocols, Journal of Cryptographic Engineering 2(2), pp. 111-127, 2012.

[16] J. Brzozowski and Z. Esik, Hazard Algebras, Formal Methods in System Design 23(3), pp. 223-256, 2003.

[17] S. Nikova, C. Rechberger and V. Rijmen, Threshold Implementations against Side-Channel Attacks and Glitches, ICICS 2006, LNCS 4307, pp. 529-545, 2006.

[18] Y.J. Baek and D.H. Choi, AND 게이트에 대한 2차 G-equivariant 로직 게이트 및 AES 구현에의 응용, Journal of the Korea Institute of Information Security & Cryptology 24(1), pp. 221-227, 2014.

[19] S. Nikova, V. Rijmen and M. Schlaffer, Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches, Journal of Cryptology 24(2), pp. 292-321, 2011.

[20] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov and V. Rijmen, Higher-Order Threshold Implementations, ASIACRYPT 2014, LNCS 8874, pp. 326-343, 2014.

[21] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs and I. Verbauwhede, Consolidating Masking Schemes, CRYPTO 2015, LNCS 9215, pp. 764-783, 2015.

[22] H. Gross, S. Mangard and T. Korak, Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order, IACR Cryptology ePrint Archive, 2016.

〈저자소개〉

백 유 진(Yoo-Jin Baek)

종신회원

1997년 2월 : 서울대학교 수학과 졸업

1999년 2월 : 서울대학교 수학과 이학석사

2003년 2월 : 서울대학교 수리과학부 이학박사

2003년 3월~2003년 6월 : KAIST 박사후 연구원

2003년 7월~2013년 3월 : 삼성전자 책임연구원

2013년 3월~현재 : 우석대학교 정보보안학과 부교수

<관심분야> 부채널분석 공격, 정보 보안