情報保護學會論文誌
第18卷 第6(A)號, 2008. 12

차분 전력분석 공격에 안전한 논리 게이트 및 SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수에의 응용

백 유 진†
삼성전자

DPA-Resistant Logic Gates and Secure Designs of SEED and SHA-1

Yoo-Jin Baek†
Samsung Electronics

요 약

차분 전력 분석 공격[8]은 암호시스템에 대한 강력한 부채널 공격 방법 중의 하나이며 마스킹 방법[10]은 이러한 차분 전력 분석 공격에 대한 알고리즘적인 대응 기법의 하나로 잘 알려져 있다. 그러나 마스킹 방법을 산술 덧셈기와 같은 비선형 함수에 적용하는 것은 쉽지 않다. 본 논문은 이러한 마스킹 방법을 산술 덧셈기에 효율적으로 적용하는 새로운 방법을 제안한다. 이를 위해서 본 논문은 먼저 기본 논리 게이트 (AND, OR, NAND, NOR, XOR, XNOR, NOT)에 마스킹 방법을 적용하는 방법을 먼저 제안하고 이러한 기본 게이트들의 조합으로 산술 덧셈기를 구성함으로써 산술 덧셈기에 적용 가능한 새로운 마스킹 방법을 제시한다. 제안된 방법의 응용으로서 본 논문은 SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수를 차분 전력 분석 공격에 안전하게 구현하는 방법과 그 상세한 하드웨어적인 구현 결과를 제시한다.

ABSTRACT

The differential power attack (DPA)[8] is a very powerful side-channel attack tool against various cryptosystems and the masking method[10] is known to be one of its algorithmic countermeasures. But it is non-trivial to apply the masking method to non-linear functions, especially, to arithmetic adders. This paper proposes simple and efficient masking methods applicable to arithmetic adders. For this purpose, we use the fact that every combinational logic circuit (including the adders) can be decomposed into basic logic gates (AND, OR, NAND, NOR, XOR, XNOR, NOT) and try to devise efficient masking circuits for these basic gates. The resulting circuits are then applied to the arithmetic adders to get their masking algorithm. As applications, we applied the proposed masking methods to SEED and SHA-1 in hardware.

Keywords : Differential Power Attack, Masking Method, Logic Gates, Adder, SEED, SHA-1

Ⅰ. 서 론

암호 시스템에 대한 부채널 공격 방법(Side-Channel Attack)은 암호 알고리즘을 물리적인 기기에 구현된 특정 프로그램으로 간주하며, 이러한 프로그램이 연산하는 과정 중에 발생하는 부채널 정보를 이용해서 비밀 정보를 알아내고자 한다. 여기서 부채널 정보란 암호 연산 시 발생하는 전력 소비량의 변화 또는 암호 연산 시간 등의 정보를 의미한다[9].

접수일: 2008년 3월 19일; 수정일: 2008년 7월 7일;
채택일: 2008년 10월 9일
† 주저자, yoojin.baek@samsung.com

차분 전력 분석 공격(Differential Power Attack, 이하 DPA)[8,11]은 부채널 공격 방법의 특수한 형태로서, 암호 연산 중에 발생하는 전력 소비량의 변화를 부채널 정보로 이용한다. DPA가 작동하는 주된 이유 중의 하나는 암호 기기가 소비하는 전력량은 기기 내부에서 수행되는 연산과 밀접한 관련이 있으며, 특히 이러한 연산과 암호 시스템의 비밀 키가 서로 연관되어 있으면 전력 소비량의 변화를 분석해서 해당 비밀 키를 알아낼 수 있다는 점에 있다. 이러한 DPA에 대해서 많은 방어 기법(countermeasure)이 제안되었으며, 그 중에서도 마스킹 방법(masking method)은 가장 강력한 알고리즘적인 방어 기법의 하나로 알려져 있다[10].

마스킹 방법은 어떤 특정한 암호 연산이 수행되기 전에 입력 데이터를 랜덤값을 이용해서 마스킹(masking)함으로써 암호 시스템의 공격자에게는 현재 연산 중인 데이터가 랜덤하게 보이도록 하고자 한다. 그러나 이러한 마스킹 방법을 산술 덧셈기(arithmetic adder)와 같은 비선형 함수에 적용하는 것은 쉽지 않은 문제이며, 본 논문에서는 이러한 산술 덧셈기에 적용 가능한 새로운 마스킹 방법을 제안한다.

선형 덧셈기에 적용 가능한 기존의 마스킹 방법으로는 먼저 부울 마스크(Boolean mask)와 산술 마스크(arithmetic mask) 사이의 변환 알고리즘을 이용하는 방법이 있다[1,2,5]. 부울 마스크와 산술 마스크란 데이터를 마스킹하는 대표적인 두 가지 방법으로, 부울 마스크는 XOR연산을, 산술 마스크는 산술 덧셈 연산을 사용하여 데이터를 마스킹한다. 이 때, 만약 평문이 처음에 부울 마스크로 마스킹되었다고 가정을 하고 암호 연산 중에 산술 덧셈 연산을 만나게 되는 경우, 상기 부울 마스크된 데이터를 산술 마스크된 데이터로 변환할 수 있다면, 산술 덧셈기는 산술 연산에 대해서 선형 연산이기 때문에 산술 마스크된 데이터를 이용해서 산술 덧셈 연산을 수행하는 것은 어렵지 않게 된다. 또한 그 결과 값에 이 후에 등장하는 부울 연산을 적용하기 위해서는 산술 마스크된 데이터를 다시 부울 마스크된 데이터로 변환하는 알고리즘이 필요하게 된다. 따라서 이러한 종류의 마스킹 방법에서 가장 중요한 부문은 부울 마스크된 데이터와 산술 마스크된 데이터 사이의 변환 알고리즘이 되며 상기 논문에서는 이러한 변환 알고리즘을 새롭게 제안한다. 그러나 본 논문에서는 산술 덧셈기를 XOR연산에 대한 비선형 함수로 간주하며 마스크 사이의 어떠한 변환도 고려함이 없이 마스킹 방법을 산술 덧셈기에 직접 적용한다. 이를 위해서 본 논문은 먼저 산술 덧셈기를 기본 게이트들(AND, OR, NAND, NOR, XOR, XNOR NOT)의 조합으로 분해한 후, 마스킹 방법을 각각의 기본 게이트들에 적용함으로써 산술 덧셈기에 대한 마스킹 방법 적용 문제를 해결한다. 따라서 이러한 기본 논리 게이트들에 대해서 마스킹 방법을 효율적으로 적용하는 문제를 본 논문에서는 주로 다루게 된다.

또 다른 방법은 본 논문에서 제안하는 방법과 유사하게 게이트 단위로 마스킹 방법을 적용하는 문제를 다루며, 가령 AES 알고리즘을 DPA에 안전하게 구현하기 위해서 AND 게이트와 OR 게이트에 대한 마스킹 문제를 해결하려고 시도한다. 예를 들어 E. Trichina에 의해서 제안된 방법은[14], 만약 랜덤 비트 r_x, r_y, r_z에 대하여 $x'=x\oplus r_x$, $y'=y\oplus r_y$가 주어진 경우 $((x\wedge y)\oplus r_z,\ r_z)$와 $((x\vee y)\oplus r_z,\ r_z)$를 계산하기 위하여 다음의 등식을 이용한다:

$$(x\wedge y)\oplus r_z = (((r_z\oplus(r_x\wedge r_y))\oplus(r_x\wedge y'))\oplus (r_y\wedge x'))\oplus(x'\wedge y')$$
$$(x\vee y)\oplus r_z = (((\neg r_z\oplus(\neg r_x\vee\neg r_y))\oplus (\neg r_x\vee y'))\oplus(\neg r_y\vee x'))\oplus(x'\vee y').$$

여기서 $\wedge$, $\vee$, $\oplus$는 각각 비트 단위의 AND, OR, XOR 연산을 의미한다. 따라서 AND 게이트에 마스킹 방법을 적용하기 위해서는 4개의 AND 게이트와 4개의 XOR 게이트를, OR 게이트에 마스킹 방법을 적용하기 위해서는 4개의 OR 게이트, 4개의 XOR 게이트 그리고 3개의 NOT 게이트를 필요로 하며 추가적으로 1개의 랜덤 비트 r_z의 생성이 요구된다. 반면에 본 논문에서 제안한 AND 게이트에 대한 마스킹 방법은 4개의 NAND 게이트, 4개의 XOR 게이트를 필요로 하며, OR게이트에 대한 마스킹 방법은 2개의 NOR 게이트, 2개의 NAND 게이트 그리고 4개의 XOR 게이트를 필요로 하지만 어떠한 추가적인 랜덤 비트의 생성도 요구하지 않는다.

기본 논리 게이트에 마스킹 방법을 적용하는 또 다른 방법으로는 MUX 게이트를 이용하는 방법이 있으며[4], 보다 자세하게는 AND 게이트에 마스킹 방법을

적용하기 위해서 6개의 AND 게이트, 3개의 OR 게이트, 그리고 2개의 NOT 게이트를, OR 게이트에 마스킹 방법을 적용하기 위해서 6개의 OR 게이트, 3개의 AND 게이트 그리고 3개의 NOT 게이트를 필요로 한다.

본 논문에서 제안된 마스킹 방법은 부울 연산과 산술 연산을 모두 사용하는 암호 알고리즘을 DPA에 안전하게 구현하는 데에 사용될 수 있다. 이러한 알고리즘의 대표적인 예로는 AES 후보 알고리즘이었던 MARS [15], RC6[16], Twofish[17]와 대한민국 상용 블록 암호 알고리즘 표준인 SEED[7], 그리고 해쉬 함수인 MD5[18], SHA-1[13] 등이 있으며 본 논문은 이 중에서 SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수에 새로운 마스킹 방법을 적용하는 구체적인 방법과 그 자세한 구현 결과를 제시한다.

본 논문은 다음과 같이 구성된다. 먼저 2절은 DPA, 마스킹 방법, 그리고 마스킹 문제를 소개한다. 또한 기본 논리 게이트와 여러 가지 덧셈기에 대한 간단한 소개도 같은 절에서 다루어진다. 3절에서는 마스킹 방법을 기본 논리 게이트 및 full adder, ripple adder 등에 적용하는 새로운 방법들을 설명하고 4절에서는 SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수의 간단한 소개와 이러한 암호 알고리즘에 본 논문에서 제시한 새로운 마스킹 방법을 적용하는 방법 및 그 상세한 구현 결과를 제시한다.

Ⅱ. 기초 지식

2.1. DPA와 마스킹 방법

Kocher 등에 의해서 제안된 암호 시스템에 대한 DPA[8]는 암호 연산 중에 발생하는 전력 소비량의 변화를 이용해서 암호 기기에 저장되어 있는 비밀 정보를 알아낸다. DPA가 작동하는 가장 큰 이유 중의 하나는 전력 소비량은 암호 기기 내의 내부 상태와 강한 관련성이 있으며 따라서 이러한 전력 소비량을 분석하면 암호 기기 내에서 수행되는 연산의 종류와 관련된 키 정보를 알아 낼 수 있다는 데에 있다. 특히, 본 논문에서는 전력 소비량은 연산 데이터의 해밍 웨이트(Hamming weight)와 비례한다는 전력 소비 모델을 채용하며 또한 각각의 샘플 시간에서의 전력 소비 정보의 통계량을 이용해서 암호 시스템을 공격하는 일차 DPA(first-order DPA)를 고려 대상으로 한다[11]. 따라서 이 후에 DPA라 함은 일차 DPA을 의미한다.

DPA는 먼저 암호화 하고자 하는 평문 집합을 추정된 키 비트를 이용해서 두 개의 부분 집합으로 분류한다. 이 경우 만약 추정된 키 비트가 올바를 경우 해당 전력 소비 정보와 연산 데이터 사이에는 통계적으로 강한 상관관계를 갖게 된다. 반면에 추정이 틀린 경우에는 두 데이터 사이의 관계는 랜덤하게 보이게 된다. 따라서 이러한 관계를 이용하면 비밀 키에 대한 유용한 정보를 얻게 된다.

DPA에 대한 가장 강력한 알고리즘적인 대응 방법 중의 하나로 다음과 같은 마스킹 방법이 잘 알려져 있다: 먼저 평문을 랜덤 데이터를 이용해서 마스킹된 데이터로 변환한 후(마스킹 단계), 이 마스킹된 데이터를 이용해서 암호 연산을 수행한다. 마지막으로 결과 값에 대해서 역 마스킹 변환을 수행하면 원하는 암호문을 얻게 된다.

일반적으로 마스킹 방법을 암호 알고리즘에 적용하기 위해서는 다음과 같은 두 종류의 마스크 기법이 주로 사용된다.

정의 1 k 비트열 x와 랜덤 비트열 r, 그리고 $x'=x \oplus r$에 대하여 (x', r)을 x의 (랜덤) 부울 마스크 (Boolean mask)라 정의한다.

정의 2 k 비트열 x와 랜덤 비트열 r, 그리고 $x'=x+r \bmod 2^k$에 대하여 (x', r)을 x의 (랜덤) 산술 마스크 (arithmetic mask)라 정의한다.

마스킹 방법을 암호 연산에 적용하기 위해서는 다음과 같은 마스킹 문제(masking problem)의 해결이 요구된다: 주어진 함수 $f:\{0,1\}^k \rightarrow \{0,1\}^n$에 대하여 다음을 만족시키는 함수 $F:\{0,1\}^{2k} \rightarrow \{0,1\}^{2n}$를 계산하시오.

1) 주어진 $x \in \{0,1\}k$의 임의의 마스크 (x', r)에 대하여 F는 $f(x)$의 랜덤 마스크를 출력한다. 즉 랜덤 비트열 $s \in \{0,1\}^n$에 대하여 $F(x', r) = (f(x) \oplus s, s)$ 또는 $F(x', r) = (f(x)+s \bmod 2^n, s)$이다.
2) F의 계산 중에 발생하는 모든 중간 결과 값은 x와 서로 상관관계가 없다.

가령, f가 선형 함수인 경우에 함수 $F(x', r) = (f(x'), f(r))$는 다음과 같은 이유로 f에 대한 마스킹 문제의 간단한 해결 방법을 제공한다: $x'=x\oplus r$에 대하여 $f(x')\oplus f(r) = f(x\oplus r)\oplus f(r) = f(x)$. 또 다른 예로서 만약 g가 아핀 함수(affine function)인 경우, 즉 $n\times k$ 행렬 A와 벡터 $b\in\{0,1\}^n$에 대하여 $g(x) = Ax\oplus b$인 경우에 함수 $G(x', r) = (g(x'), Ar)$가 g에 대한 마스킹 문제의 간단한 해결 방법이 됨은 같은 방식으로 증명될 수 있다.

그러나 일반적으로 마스킹 방법을 비선형 함수에 적용하는 것은 그다지 용이하지 않다. 특히 만약 암호 알고리즘이 부울 연산과 산술 연산(특히 산술 덧셈기)을 모두 사용하고 평문이 부울 마스크로 마스킹된 경우에는 산술 연산은 $\oplus$에 대하여 비선형 함수가 된다. 따라서 이 경우 해당 마스킹 문제는 다음과 같이 재정의될 수 있다.

산술 덧셈기에 대한 마스킹 문제 랜덤 비트열 r, $s\in\{0,1\}^k$와 $x' = x\oplus r$, $y' = y\oplus s$를 만족시키는 $(x', r), (y', s) \in \{0,1\}^{2k}$ 에 대하여 $((x+y \bmod 2^k)\oplus t, t)$ 를 계산하시오. 단 t는 랜덤 비트열이며, 계산 도중의 중간 결과 값과 x, y는 서로 상관관계가 없어야 한다.

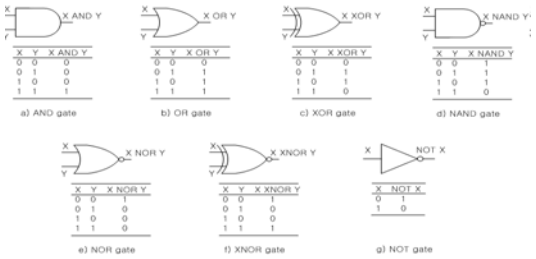
상기 마스킹 문제를 해결하는 새로운 알고리즘은 3절에 제시되며 이 알고리즘을 위해서 본 논문에서는 먼저 산술 덧셈기를 기본 논리 게이트 AND, OR, NAND, NOR, XOR, XNOR, NOT의 조합으로 분해한다. 그리고 각 논리 게이트에 대한 마스킹 문제의 해결법을 각각 제시하고 그 해결법을 이용해서 산술 덧셈기에 대한 마스킹 방법을 도출한다. 물론 이러한 해결 방식은 매우 일반적이기 때문에 어떠한 비선형 함수에도 적용 가능하다.

2.2. 논리 게이트

임의의 조합 논리 회로(combinational logic circuit)는 기본 논리 게이트를 이용하여 구현될 수 있다. 이러한 기본 논리 게이트의 예, 그리고 그 각각의 진리표 및 표식 법은 [그림 1]에 주어진다.

2.3. Full Adder와 Ripple Adder

여러 비트로 구성된 데이터에 대해서 산술 덧셈 연산을 수행하기 위해서는 각 비트 위치에서의 캐리(carry)를 고려해야 한다. 이러한 캐리 연산을 고려한 기본 덧셈기로 full adder가 사용된다. full adder는 X, Y, CIN의 3비트를 입력으로 받아 다음을 만족시키는 두 비트 S, $COUT$을 출력한다:

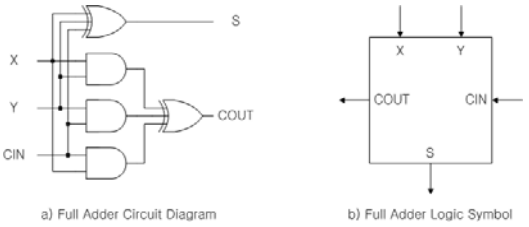


[그림 1] 기본 논리 게이트

$$S = X\oplus Y\oplus CIN$$

$$COUT = (X\wedge Y)\oplus(X\wedge CIN)\oplus(Y\wedge CIN)$$

full adder를 수행하는 회로와 그 표식 법은 [그림 2]에 주어진다.

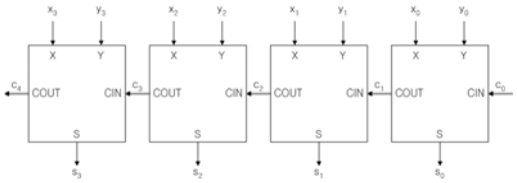


[그림 2] Full Adder

두 개의 n-비트열에 대한 덧셈 연산을 수행하기 위해서는 상기 full adder를 반복적으로 이용한 ripple adder가 사용되어 질 수 있다. [그림 3]은 4-비트 ripple adder를 묘사한다. 여기서 첫 번째 캐리 비트 c_0는 일반적으로 0으로 주어진다[3].

Ⅲ. 마스킹 방법이 적용된 논리 회로

산술 덧셈기를 포함한 모든 논리 회로는 기본 논리 게이트들의 조합으로 분해 될 수 있기 때문에 본 논문에서는 먼저 이러한 기본 논리 게이트들에 적용 가능한 마스킹 방법을 고려한다. 그러나 이러한 기본 논리



[그림 3] 4-비트 Ripple Adder

게이트 중에서 XOR 게이트는 선형 함수이고 NOT 게이트는 아핀 함수이다. 따라서 이러한 게이트들에 마스킹 방법을 적용하는 것은 어렵지 않다. 또한, NAND, NOR, XNOR 게이트는 각각 AND, OR, XOR 게이트와 NOT 게이트의 합성 함수이기 때문에 (물론 모든 논리 게이트는 NAND 게이트로 구성될 수 있지만, 이론 전개의 용이성을 위하여 본 논문에서는 AND, OR 게이트를 기본 게이트로 가정한다.) 이 절에서는 주로 AND, OR 게이트에 마스킹 방법을 적용하기 위한 다음과 같은 문제의 해결법을 다루게 된다.

AND 게이트에 대한 마스킹 문제 랜덤한 $r, s \in \{0,1\}$와 $x' = x \oplus r$, $y' = y \oplus s$에 대하여 (x', r), (y', s)가 주어진 경우, 랜덤 비트 t에 대하여 $((x \wedge y) \oplus t, t)$를 계산하시오. 단 계산 도중의 중간 결과 값과 x, y는 서로 상관관계가 없어야 한다.

OR 게이트에 대한 마스킹 문제 랜덤한 $r, s \in \{0,1\}$와 $x' = x \oplus r$, $y' = y \oplus s$에 대하여 (x', r), (y', s)가 주어진 경우, 랜덤 비트 t에 대하여 $((x \vee y) \oplus t, t)$를 계산하시오. 단 계산 도중의 중간 결과 값과 x, y는 서로 상관관계가 없어야 한다.

상기 두 문제에 대한 해결 방법으로 본 논문은 다음과 같은 2개의 알고리즘을 제안한다.

알고리즘1 (AND게이트에 적용된 마스킹 방법)

입력: $x'(=x \oplus r)$, r, $y'(=y \oplus s)$, $s \in \{0, 1\}$

출력: 랜덤 비트 t에 대하여 $((x \wedge y) \oplus t, t)$

1. $a \leftarrow \neg(x' \wedge y')$, $b \leftarrow \neg(y' \wedge r)$;
2. $c \leftarrow \neg(x' \wedge s)$, $d \leftarrow \neg(r \wedge s)$;
3. $a \leftarrow a \oplus y'$, $b \leftarrow b \oplus y'$;
4. $a \leftarrow a \oplus c$, $b \leftarrow b \oplus d$;
5. Return (a, b).

알고리즘2 (OR게이트에 적용된 마스킹 방법)

입력: $x'(=x \oplus r)$, r, $y'(=y \oplus s)$, $s \in \{0, 1\}$

출력: 랜덤 비트 t에 대하여 $((x \vee y) \oplus t, t)$

1. $a \leftarrow \neg(x' \vee y')$, $b \leftarrow \neg(r \vee s)$;
2. $c \leftarrow \neg(x' \wedge s)$, $d \leftarrow \neg(r \wedge y')$;
3. $a \leftarrow a \oplus c$, $b \leftarrow b \oplus d$;
4. Return (a, b).

상기 알고리즘이 AND 및 OR 게이트에 대한 마스킹 문제의 해결 방법을 제공해 준다는 것을 증명하기 위해서는 다음의 보조 정리들이 필요하다.

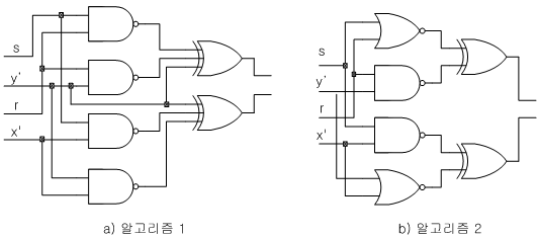
보조 정리 1. 알고리즘 1의 출력 값 (a, b)는 $x \wedge y$의 랜덤 부울 마스크이다.

증명. 다음의 등식을 이용하면 (a, b)가 $x \wedge y$의 부울 마스크임을 알 수 있다.

$$\begin{aligned} a \oplus b &= (y' \oplus \neg(x' \wedge y') \oplus \neg(x' \wedge s)) \oplus \\ &\quad (y' \oplus \neg(r \wedge y') \oplus \neg(r \wedge s)) \\ &= (x' \wedge y') \oplus (x' \wedge s) \oplus (r \wedge y') \oplus (r \wedge s) \\ &= (x' \oplus r) \wedge (y' \oplus s) = x \wedge y \end{aligned}$$

또한 다음을 이용하면 (a, b)는 랜덤 마스크임을 알 수 있다: 주어진 $\alpha, \beta, \gamma \in \{0,1\}$에 대하여 r과 s를 랜덤하게 선택하면 $Pr(b=\alpha | x=\beta, y=\gamma)=1/2$이다.

보조 정리 2. 알고리즘 1의 중간 결과 값의 확률 분포는 x, y에 독립이다.



[그림 4] AND 게이트 및 OR 게이트에 대한 마스킹 방법

증명. 알고리즘 1의 중간 결과 값은 다음과 같은 6개가 존재한다: $\neg(x' \wedge y')$, $\neg(x' \wedge s)$, $\neg(y' \wedge r)$, $\neg(r \wedge s)$, $y' \oplus \neg(x' \wedge y')$, $y' \oplus \neg(y' \wedge r)$. 만약 r과 s를 랜덤하게 선택하면, 임의로 주어진 $\alpha, \beta, \gamma \in \{0,1\}$에 대하여 우리는 다음과 같은 등식을 얻을 수 있다:

$$
\begin{aligned}
& Pr(\neg(x' \wedge y') = \gamma | x = \alpha, y = \beta) \\
&= Pr(\neg(x' \wedge s) = \gamma | x = \alpha, y = \beta) \\
&= Pr(\neg(r \wedge y') = \gamma | x = \alpha, y = \beta) \\
&= Pr(\neg(r \wedge s) = \gamma | x = \alpha, y = \beta) \\
&= Pr(y' \oplus \neg(x' \wedge y') = \gamma | x = \alpha, y = \beta). \\
&= Pr(y' \oplus \neg(r \wedge y') = \gamma | x = \alpha, y = \beta) \\
&= \begin{cases} \frac{1}{4} & \text{if } \gamma = 0 \\ \frac{3}{4} & \text{if } \gamma = 1 \end{cases}
\end{aligned}
$$

알고리즘 2에 대해서도 다음과 같은 비슷한 보조 정리를 얻을 수 있다.

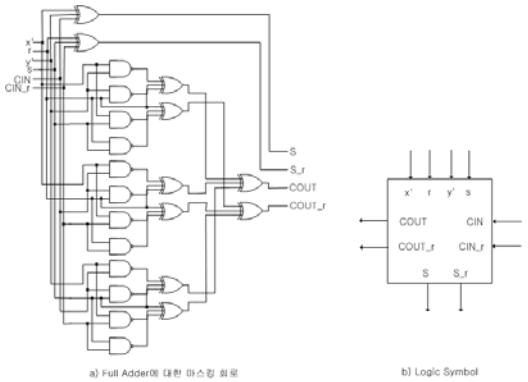
보조 정리 3. 알고리즘 2의 출력 값 (*a, b*)는 $x \vee y$의 랜덤 부울 마스크이다.

보조 정리 4. 알고리즘 2의 중간 결과 값의 확률 분포는 *x, y*에 독립이다.

상기의 보조 정리들로부터 알고리즘 1과 2는 AND 게이트와 OR 게이트에 대한 마스킹 문제의 완벽한 해결방법을 제공함을 알 수 있으며 그 해당 회로는 [그림 4]에 주어진다.

이제, 이러한 기본 논리 게이트에 대한 마스킹 방법을 이용하여 여러 가지 산술 덧셈기에 대한 마스킹 알고리즘을 설계할 수 있다. 가령, [그림 2]에 제시된 full adder는 3개의 AND 게이트와 2개의 3-입력 XOR 게이트로 구성되며, 따라서 이런 게이트들을 [그림 4]에 묘사된 회로와 선형 함수에 대한 기본적인 마스킹 회로로 대체하면 [그림 5]와 같은 full adder에 대한 마스킹 방법이 구해질 수 있다. 물론 full adder를 구현하는 회로는 여러 가지가 있으며 그 각각의 회로에 대한 마스킹 방법 또한 달라질 수 있음은 자명하다.

마지막으로 full adder에 대한 마스킹 회로는 ripple adder에 대한 마스킹 회로를 생성하는 데 사용될 수 있



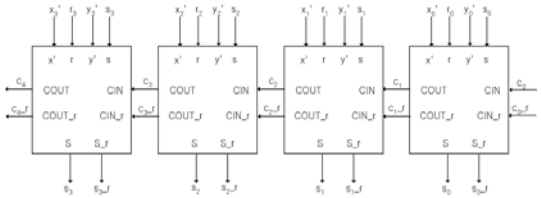
[그림 5] Full Adder에 대한 마스킹 회로 및 Logic Symbol

다. 가령, [그림 6]은 4-비트 ripple adder에 대한 마스킹 회로를 나타내며 이러한 ripple adder 마스킹 회로는 다음 절에 주어지는 것과 같이 암호 알고리즘을 DPA에 안전하게 구현하는 데에 사용될 수 있다.

Ⅳ. 응 용

SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수를 DPA에 안전하게 구현하기 위해서 본 논문은 다음과 같은 방법을 사용한다. 먼저 각 알고리즘은 암호/해쉬 연산을 수행하고자 하는 평문/메시지와 해당 데이터에 상응하는 크기의 랜덤 데이터를 입력받은 후 평문/메시지를 상기 랜덤 데이터를 이용해서 부울 마스크된 데이터로 변환한다. 이 때 입력받은 랜덤 데이터는 알고리즘 외부의 랜덤 수 생성기(Random Number Generator)를 이용해서 생성된 후 해당 알고리즘에 안전하게 입력된 것으로 간주된다. 이 후 연산 도중에 선형 함수나 아핀 함수를 만나게 되면 2장 1절에 제시된, 각 함수에 대한 적당한 마스킹 방법을 마스킹된 데이터에 적용한다. 또한 만약 산술 덧셈기와 같은 비선형 함수를 만나게 되면 그 함수를 기본 논리 게이트로 분해한 후 각각에 알맞은 마스킹 방법을 적용한다. 마지막으로 그 결과 데이터에 역 마스킹 변환을 적용함으로써 원래의 암호문 또는 해쉬값을 얻게 된다. 이 경우 역 마스킹 변환은 마지막에 한 번만 수행되며 따라서 마스킹을 위한 랜덤 데이터는 각 평문 또는 메시지 블록에 대해서 하나만이 사용된다.

본 논문에서는 상기 방식으로 SEED와 SHA-1 알고



[그림 6] 4-비트 Ripple Adder에 대한 마스킹 회로

리즘을 DPA에 안전하게 하드웨어적으로 구현을 하였다. 그 구현 결과물에 대해서 Cadence NC-Verilog를 이용해서 시뮬레이션을 수행하였으며, Synopsys Design-Compiler와 Samsung 0.18μm 공정[12]을 사용해서 합성을 하였다. 그 상세한 구현 결과는 [표 1]과 [표 2]에 제시된다.

4.1 SEED 블록 암호 알고리즘

SEED 알고리즘은 Feistel 형태의 블록 암호 알고리즘으로, 16 라운드로 구성되며 128 비트 길이의 블록 및 키를 사용하며 현재 대한민국 표준 상용 블록 암호 알고리즘으로 선정되었다.[7] 또한 SEED는 차분 공격법(differential cryptanalysis), 선형 공격법(linear cryptanalysis)과 같은 수학적인 암호 알고리즘 공격법에 강인하며 이러한 강인성의 주 원인 중의 하나는 알고리즘 내부에서 사용되는 32 비트 산술 덧셈기의 사용에 있음이 알려져 있다. SEED 알고리즘의 대략적인 구조는 다음과 같다. 먼저 128 비트 입력 평문은 2개의 64 비트 부분 블록으로 나누어지며 그 중에서 오른쪽 64 비트 부분 블록과 키 생성 알고리즘으로부터 생성된 64비트 라운드 키는 라운드 함수 F의 입력 데이터가 된다. 이러한 F의 입력 평문은 두 개의 32 비트 부분 블록 *(C, D)*로 분해 된 후 그 결과 값 *(C', D') = F(C, D)*는 다음과 같은 식에 의해서 계산 된다:

$$
\begin{aligned}
C' &= G[G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} + \\
&\quad (C \oplus K_{i,0})] + G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\}] + \\
&\quad G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} + (C \oplus K_{i,0})], \\
D' &= G[G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} + \\
&\quad (C \oplus K_{i,0})] + G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\}].
\end{aligned}
$$

상기 수식에서 사용된 덧셈 기호는 모듈로 2^{32} 산술 덧셈을 의미하며, 함수 G는 다음과 같이 정의된 두 함수의 합성 함수이다: 첫 번째 함수는 유한체 $GF(2^8) \simeq GF(2)[x]/(x^8+x^6+x^5+x+1)$ 상에서 정의된 두개의 단항식 x^{247}, x^{251}에 해당하는 8×8 S-box로 구성되고, 다른 함수는 16개의 8 비트 부분 블록에 대한 치환 함수(permutation)이다.

본 논문에서 제안된 기본 논리 게이트에 대한 마스킹 방법은 상기 F함수의 덧셈 연산과 G함수의 S-box 연산에 적용될 수 있다. SEED의 입력 데이터는 먼저 128 비트 랜덤 데이터에 의해서 부울 마스크된 형태로 변환되며 본 논문에서 제안된 방법에 의하면 이러한 부울 마스크 형태의 입력 데이터는 암호 연산을 거치더라도 오류 없이 원래의 암호문으로 복원될 수 있으며 또한 암호 연산 중에 발생하는 전력 소비 변화가 노출되더라도 연산의 종류나 키 정보가 노출되지 않는다.

본 논문의 SEED 알고리즘 구현은 속도보다는 전력 소비와 칩 면적을 최소화하도록 설계가 되었으며 각 라운드당 7 클록 사이클로 동작한다. 그 자세한 구현 결과는 [표 1]에 제시된다.

[표 1] SEED 구현 결과

	Gate Count	Critical Path	Throughput
SEED without masking	9.5K	32 ns	33.96 Mbps @ 30 MHz
SEED with masking	16.2K	63 ns	11.32 Mbps @ 10 MHz

4.2 SHA-1 해쉬 함수

SHA-1 해쉬 함수는 2^{64} 비트 이하의 메시지를 160 비트의 해쉬 값으로 변환한다. 이를 위해서, SHA-1은 메시지에 특수한 패딩 규칙(padding rule)을 적용하여 그 결과 값이 512 비트의 배수 길이를 갖는 데이터가 되게 하고, 이러한 각각의 512 비트 부분 블록에 순차적인 연산을 적용한다. 좀 더 자세히 설명하면, 먼저 패딩된 데이터를 n개의 512 비트 부분 블록으로 분해하고 그 각각의 부분 블록을 M_1, ..., M_n이라고 하면 다음과 같은 연산 과정이 각각의 부분 블록에 순차적으로 적용한다.

1. M_i를 16개의 워드 크기 데이터 *W[0], ..., W[15]* 로 분해한다.
2. $16 \le i \le 79$에 대하여, *W[t] = (W[t-3]⊕W[t-8] ⊕W[t-14]⊕W[t-16]<<<1)*를 계산한다.
3. *(A, B, C, D, E)←(H1, H2, H3, H4, H5)*;
4. *t*=0에서 79까지 다음 연산을 수행한다.
   4.1 *temp←(A<<<5)+fi(B,C,D)+E+W[t]+yi*;
   4.2 *(A, B, C, D, E)←(temp, A, B<<<30, C, D);*
5. *(H1, H2, H3, H4, H5)←(H1+A, H2+B, H3+C, H4+D, H5+E);*

여기서, <<<는 비트 단위의 회전함수(rotation)를, 덧셈 기호 +는 모듈로 2^{32}에서의 산술 덧셈을, 그리고 함수 f_i와 y_i는 SHA-1에서 미리 정의된 함수와 상수를 의미한다.

비록 해쉬 함수 연산 자체는 어떠한 비밀 키를 필요로 하지 않지만 비밀 키를 사용하는 HMAC 함수[6]의 원시 함수(primitive function)로 사용된다. HMAC 함수는 비밀 키와 해쉬 함수를 사용해서 메시지의 무결성 검증(integrity check)에 사용되는 암호 함수이다. 즉, 비밀 키 k, 메시지 $text$ 그리고 암호학적 해쉬 함수 H에 대하여 HMAC 함수는 다음과 같이 계산된다: *HMAC(k, text)=H(k⊕opad, H(k⊕ipad, text))*. 여기서 $ipad$와 $opad$는 미리 정의된 상수이다.

본 논문에서는 512 비트 메시지에 대해서 해쉬 연산을 수행하는 SHA-1 함수의 덧셈 연산에 새롭게 제안된 마스킹 방법을 적용하며 이를 위해서 Verilog-HDL을 이용한다. 그 구현 결과는 512 비트 메시지에 대해서 80 클록 사이클로 동작을 하며 그 자세한 구현 결과는 [표 2]에 제시된다.

[표 2]에 제시된 구현 결과에서 주목해야 할 점은 SEED의 경우와는 다르게 마스킹 방법이 적용된 SHA-1의 경우 마스킹 방법이 적용되지 않은 SHA-1에 비해서 거의 3배의 칩 면적 증가가 있다는 점이다. 이러한 결과의 주요 원인으로는, SHA-1함수는 비트 단위의 OR 함수, AND함수와 더불어 산술 덧셈기를 그 내부 연산으로 빈번하게 사용하고 있다는 점에 있을 것이다.

[표 2] SHA-1 구현 결과

	Gate Count	Critical Path	Throughput
SHA-1 without masking	8.8K	12.2 ns	512 Mbps @ 80 MHz
SHA-1 with masking	24.82K	42.7 ns	128 Mbps @ 20 MHz

Ⅴ. 결 론

본 논문에서는 일차 DPA에 대한 가장 강력하고 효율적인 방어 기법의 하나로 알려진 마스킹 방법을 다루었다. 특히 본 논문에서는 여러 가지 기본 논리 게이트들과 산술 덧셈기에 적용 가능한 마스킹 방법을 새로 제안하였으며 본 논문에서 제안된 방법은 SEED 블록 암호 알고리즘과 SHA-1 해쉬 함수를 DPA에 안전하게 구현하는 데에 사용되었다.

참고문헌

[1] J. Coron and L. Goubin, On boolean and arithmetic masking against differential power analysis, *Springer-Verlag*, Proc. of CHES 2000, LNCS vol. 1965, pp. 231-237, 2000.

[2] J. Coron and A. Tchulkine, A New Algorithm for Switching from Arithmetic to Boolean Masking, *Springer-Verlag*, Proc. of CHES 2003, LNCS vol. 2779, pp. 89-97, 2003.

[3] M. Ercegovac and T. Land, Digital Arithmetic, *Morgan Kaufmann Publishers*, 2004.

[4] J. Golić and R. Menicocci, Universal Masking on Logic Gate Level, Electronics Letters 40 (9), pp. 526-527, 2004.

[5] L. Goubin, A sound method for switching between boolean and arithmetic masking, *Springer-Verlag*, Proc. of CHES 2001, LNCS vol. 2162, pp. 3-15, 2001.

[6] H. Krawczyk, M. Bellare, and R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, 1997.

[7] Korea Information Security Agency (KISA), SEED Algorithm Specification, available at http://www.kisa.or.kr.

[8] P. Kocher, J. Jaffe, and B. Jun, Differential pow-

er analysis, Springer-Verlag, Proc. of Crypto '99, LNCS vol. 1666, pp. 388-397, 1999.

[9] P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, *Springer-Verlag*, Proc. of Crypto '96, LNCS vol. 1109, pp. 104-113, 1996.

[10] T. Messerges, Securing the AES finalists against power analysis attacks, *Springer-Verlag*, Proc. of FSE 2000, LNCS vol. 1978, pp. 150-165, 2000.

[11] T. Messerges, Using Second-Order Power Analysis to Attack DPA Resistant Software, *Springer-Verlag*, Proc. of CHES 2000, LNCS vol. 1965, pp. 238-251, 2000.

[12] Samsung smart-card library (smart130).

[13] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2, Anouncing the Secure Hash Standard, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf, 2002.

[14] E. Trichina, Combinational Logic Design for AES Subbyte Transformation on Masked Data, Cryptology ePrint Archive, Report 2003/236, 2003, Available at http://eprint.iacr.org.

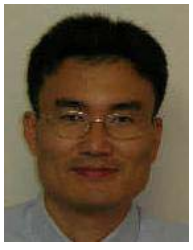
[15] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas Jr., L. O'Connor, M. Peyravi, D. Stafford, and N. Zunic, MARS-a candidate cipher for AES, IBM Corporation, June 1998, AES submission.

[16] R.L. Rivest, M.J.B. Robshaw, R. Sidney, and V.L. Yin, The RC6 Block Cipher, 1998, AES submission.

[17] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, Twofish: A 128-Bit Block Cipher, June 1998, AES submission.

[18] R. Rivest, The MD5 message digest algorithm, RFC 1321, April 1992.

<著者紹介>



백 유 진 (Yoo-Jin Baek) 정회원
1997년 2월 : 서울대학교 수학과 졸업
1999년 2월 : 서울대학교 수학과 이학 석사
2003년 3월 : 서울대학교 수학과 이학 박사
2003년 3월~6월 : KAIST 박사후 연구원
2003년 7월~현재 : 삼성전자
<관심분야> 정보보호, 부채널 공격