광학 오류 주입 공격에 강인한 몽고메리 지수승 기반 RSA 하드웨어 구현

이동건* · 최용제** · 최두호** · 김민호*** · 김호원****

Hardware Implementation of Optical Fault Injection Attack-resistant
Montgomery exponentiation-based RSA

Dong-geon Lee* · Yong-je Choi** · Doo-ho Choi** · Minho Kim*** · Howon Kim****

이 논문은 2010년도 정부(교육과학기술부)의 재원으로
한국연구재단의 지원을 받아 수행된 연구임(No.2010-0026621).

요 약

본 논문에서는 RSA를 몽고메리 지수승 기반의 하드웨어로 구현함에 있어 광학 오류 주입 공격을 탐지할 수 있는 기술을 제안한다. 본 기법은 몽고메리 곱셈 기반의 연산에서 메모리 입출력에 오류가 주입되었는지 확인하기 위해 무결성 검증 절차를 구현하였으며, 곱셈 연산에는 사용되는 로직에 광학 오류 주입 탐지 기법을 적용함으로써 안전한 지수승 연산을 가능하도록 하였다. 제안한 기법은 다양한 오류에 대하여 안전한 것으로 확인되었으며, 암호화 연산 수행시간에 영향을 미치지 않으며, 전체 면적 대비 3% 미만의 오버헤드로 구현 가능하다.

ABSTRACT

In this paper, we propose a novel optical fault detection scheme for RSA hardware based on Montgomery exponentiation, which can effectively detect optical fault injection during the exponent calculation. To protect the RSA hardware from the optical fault injection attack, we implemented integrity check logic for memory and optical fault detection logic for Montgomery-based multiplier. The proposed scheme is considered to be safe from various type of attack and it can be implemented with no additional operation time and small area overhead which is less than 3%.

키워드

RSA, 몽고메리 지수승, 광학 오류, 오류 탐지, ASIC

Key word

RSA, Montgomery Exponentiation, Optical Fault, Fault Detection, ASIC

* 정회원 : 부산대학교 컴퓨터공학과
** 정회원 : 한국전자통신연구원
*** 정회원 : 공군사관학교 전자전산학과
**** 종신회원 : 부산대학교 컴퓨터공학과 (교신저자, howonkim@pusan.ac.kr)

접수일자 : 2012. 08. 29
심사완료일자 : 2012. 09. 18

Open Access <http://dx.doi.org/10.6109/jkiice.2013.17.1.76>

©This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright © The Korea Institute of Information and Communication Engineering.

I. 서 론

RSA 알고리즘[1]은 공개키 암호방식을 대표하는 알고리즘으로써, 디지털 서명, 인증, 키분배 및 암호화 등 다양한 분야에 걸쳐 활용될 수 있는 알고리즘이다. RSA라는 이름은 알고리즘을 개발한 Rivest, Shamir, Adleman의 첫 글자를 따서 만들어 졌으며, 충분히 큰 정수에 대하여 소인수분해에 대한 어려움에 기인한 암호화 알고리즘이다. RSA 알고리즘은 모듈러 지수승 연산에 기반하고 있으며, 충분한 안전성을 가지기 위해서는 1024비트 이상의 긴 키를 사용해야 하고, 지수승 연산 자체의 복잡성으로 인해 소프트웨어로 구현할 시에 좋은 성능을 얻기가 쉽지 않다.

RSA 연산을 최적화 하는 많은 방법들이 제시되었는데, 반복적인 모듈러 곱셈에서 발생하는 오버헤드를 줄이기 위한 몽고메리 곱셈 기법이 제시되기도 하였으며[2], 중국인의 나머지 정리를 이용한 RSA-CRT 기법[3]이 등장하기도 하였다. RSA 구현에서 몽고메리 곱셈을 이용할 경우, RSA 연산의 기본이 되는 지수승 연산에 있어 몽고메리 방식의 곱셈 및 리덕션(Reduction)을 도입함으로써, 곱셈 중 발생하는 리덕션에 대한 비용을 줄이는데 크게 기여하여 전체 지수승 연산의 효율을 증가시켰다.

하지만, RSA-CRT 기법의 경우 오류 주입 공격에 대한 취약점을 가지고 있는 것으로 알려져 있다[4-6]. 오류 주입 공격은 암호화를 수행중인 칩에 자극을 가하여, 칩이 오동작을 일으키도록 유도한 뒤, 오류가 섞인 암호문을 바탕으로 암호화에 사용된 비밀정보를 알아내는 위협적인 공격이다. 오류만 제대로 주입 가능하다면, 부채널 공격[7]에 비해서 훨씬 적은 수의 암호화 수행을 통해 비밀정보를 알아낼 수 있기 때문에, 매우 위협적인 공격이라고 할 수 있다. 오류를 주입하는 방법에는 대상 암호화 칩의 전압이나 클럭을 변화시키는 방법, 온도를 변화시키는 방법, 강한 빛이나 레이저, X-ray, 이온 빔 등을 주입하는 방법 등이 있다[8]. 그 중 빛이나 레이저 등을 이용해 오류를 주입하는 광학 오류 주입 기법은 그 실효성이 실험을 통해 증명된 바 있으며, 매우 위협적인 공격으로 인식되고 있다[9-11]. 따라서 안전하게 RSA 알고리즘을 구현하기 위해서는 오류 주입 공격을 방지할 수 있는 기술을 설계 단계에서부터

탐재할 필요가 있다.

RSA 암호 알고리즘에 대한 오류 주입 공격 방지 기술에 대한 연구 역시 활발히 진행되었다. RSA-CRT 알고리즘을 개선하여 오류 주입을 방지하려는 노력이 있었으며[12,13], RSA-CRT 알고리즘을 사용하는 프로토콜에 대한 개선도 시도되었다[14]. 하지만, 이러한 노력에도 불구하고, 새로운 공격 방법이 제시되어 여전히 오류 주입 공격으로부터 취약함을 보였다[15]. 본 연구에서는 여러 가지 오류 중에서 광학 오류를 방어할 수 있는 기술을 제안하며, 이전에 제시된 연구들과 달리 알고리즘 적인 접근이 아닌, 칩 제조 단계에서 광학 오류 주입을 탐지할 수 있는 회로를 추가하는 기법을 제시하고자 한다.

제안하는 기법은 곱셈 연산의 결과 값이 메모리에 저장된 후, 다음 연산을 위해 다시 사용될 때, 메모리 내부를 대상으로 하는 광학 오류 주입이 있었는지를 확인하기 위한 로직을 구현하였으며, 제안한 기법의 효용성은 시뮬레이션을 통해 증명하였다. 또한 곱셈기 내부를 대상으로 하는 공격에 대비하기 위해 광학 오류를 탐지하는 회로를 제안하고 있다.

본 논문의 나머지는 다음과 같이 구성된다. 2장에서는 본 논문에서 제안하는 RSA 하드웨어를 설명하기 위해 필요한 몽고메리 지수승에 대해서 간략히 소개하고, 3장에서는 제안하는 RSA 하드웨어의 설계 방법에 대하여 상세히 설명한다. 4장에서는 구현된 RSA 하드웨어의 구현 결과와 오류 주입 공격 방어 기법에 대한 시뮬레이션 결과를 제시하며, 5장에서 글을 맺는다.

II. 관련 연구

P.L. Montgomery가 제안한 몽고메리 곱셈[2]은 모듈러 곱셈에 있어 곱셈 후에 발생하는 리덕션 연산을 단순화하기 위한 기법이다. 이 연산에서는 리덕션을 단순화하기 위해 모듈러 M 과 서로소이며 2^n 의 형태를 가지는 R 을 선택하고, 곱셈의 입력이 되는 x, y 를 $\tilde{x} = x \cdot R \pmod{M}$, $\tilde{y} = y \cdot R \pmod{M}$ 의 형태로 변환하여 몽고메리 도메인으로 변환하여, $\tilde{x} \cdot \tilde{y} \cdot R^{-1} = x \cdot R \cdot y \cdot R \cdot R^{-1} = x \cdot y \cdot R \pmod{M}$ 과 같이 연산을 한다. 이와 같은 방법으로 리덕션에 소요되는 비용을 비

트 쉬프트 형태의 연산으로 단순화 할 수 있다. 다정도 곱셈 형태의 몽고메리 곱셈 알고리즘은 알고리즘 1에 나타나있다.

알고리즘1. $MontMul(x, y, M, R)$ - 다정도 몽고메리 곱셈

$$\begin{aligned} \text{입력: } & x = (x_{n-1} \dots x_1 x_0)_b, y = (y_{n-1} \dots y_1 y_0)_b, \\ & M = (M_{n-1} \dots M_1 M_0)_b, R = b^r, 0 \leq x, y < M, \\ & \gcd(M, b) = 1, M' = -M^{-1} \pmod{b} \\ \text{출력: } & x \cdot y \cdot R^{-1} \pmod{M} \end{aligned}$$

1. $A \leftarrow 0$ where $A = (a_n a_{n-1} \dots a_0)_b$
2. for $i = 0$ to n
 - 2.1. $u_i \leftarrow (a_0 + x_i y_0)M \bmod b$
 - 2.2. $A \leftarrow (A + x_i y_i + u_i M)/b$
3. if $A \geq M$ then $A \leftarrow A - M$
4. return A as result.

위의 다정도 몽고메리 곱셈 기법을 활용하여, RSA 알고리즘에서 사용되는 지수승 연산을 다음과 같은 알고리즘으로 표현할 수 있다.

알고리즘2. $MontExp(x, e, M, R)$ - 몽고메리 지수승 연산 알고리즘

입력: $1 \leq x < m$ 을 만족하는 밑수 x , 길이 t 인 지수 e
 의 이진 표현, l 의 길이를 가지는 모듈러스 M ,
 $R = 2^l$, $M = -M^{-1} \pmod{2}$
 출력: $x^e \pmod{M}$

1. $R' = R^2 \bmod M$
2. $x' = \text{MontMul}(x, R', M, R)$
3. $A = R \bmod M$
4. for $i = t$ to 0
 - 4.1. $A = \text{MontMul}(A, A, M, R)$
 - 4.2. If $e_i = 1$ then $A = \text{MontMul}(A, x', M, R)$
5. $A = \text{MontMul}(A, 1, M, R)$
6. return A as result

III. 오류 주입에 강인한 몽고메리 곱셈 기반 RSA 연산 모듈 설계

이 장에서는 제안하는 **RSA** 연산 모듈을 어떻게 설계하였는지에 대하여 설명하고자 한다. 본 제안에서는 몽고메리 지수승 연산 모듈을 구현하기 위해 알고리즘 1의 몽고메리 곱셈 연산 모듈을 구현하였으며, 2장의 알고리즘 2의 순서에 따라 동작을 수행한다. 또한 오류가 주입되었음을 확인하기 위해 연산에 필요한 값이 저장되는 메모리 블록과 곱셈 연산 부분에 오류 주입 공격 방지 기법을 적용하였다.

3.1. 전체 회로 구조

그림 1은 제안하는 RSA 하드웨어의 블록도를 보여주고 있다. 제안하는 하드웨어는 외부로부터 연산에 필요한 메시지 M , 지수 E , 몽고메리 곱셈의 입력에 사용되는 X 및 Y , 그리고 곱셈의 결과가 저장되는 A 를 포함한 5개의 메모리와, 몽고메리 곱셈 알고리즘 중, 덧셈 연산을 담당하는 부분, 리덕션 연산을 위한 모듈, 그리고 전체 회로를 제어하는 제어로직으로 구성된다. 각각의 메모리는 32비트 단위로 주소를 지정하며, 64개의 주소를 가진다.

그림 1. 제안하는 RSA 하드웨어의 블록도
Fig. 1 Block diagram of proposing RSA hardware

그림 2. 각 메모리간의 데이터 이동 순서
Fig. 2 Sequence of data movement between memories

즉, 각각 2048비트의 피연산자를 저장할 수 있으며, 본 RSA 하드웨어는 선택에 따라 512, 1024, 2048 비트의 지수승 연산이 가능하다. 본 논문에서는 오류 주입 공격 방어책이 구현되지 않은 일반적인 구현과 방어 기법이 구현된 두 가지 버전을 구현하였으며, 광학 오류 주입 공격 탐지에 필요한 별도의 로직은 빗살무늬로 표시하였다. 오류 주입 공격 탐지 기술은 크게 두 가지로 나눈다. 첫 번째는 메모리에 데이터가 입력되고 출력할 때 RAX(Rotated and Accumulated XOR)라는 무결성 검증 코드를 이용하여 오류를 탐지하는 기법이며, 두 번째는 몽고메리 곱셈 연산에 필요한 로직에 광학 오류 주입을 감지하는 특수한 덧셈기를 이용하는 기술이다.

3.2. 지수승 연산 수행 절차

그림 1에서 나타난 것과 같이 메모리는 M, E, X, Y, A 의 다섯 개의 메모리로 구성된다. 연산에 필요한 입력 값인 모듈러스 M 과 암호화에 사용되는 키인 지수값 e 는 각각 메모리 M, E에 저장된다. 몽고메리 곱셈기의 두 개

의 입력으로 사용되는 값은 각각 X, Y에 저장되며, 곱셈 연산의 결과는 A 메모리에 저장된다.

본 논문에서 제안하는 RSA 하드웨어의 연산 순서는 2장의 알고리즘 2에 따라 동작하며, 이를 그림2에 나타내었다. 연산이 시작되기 전에, M, E 메모리에 각각 모듈러스 M 과 지수값 e 가 입력된다. 또한 평문 값이 X 메모리에 입력된다. $R^2 \bmod M$ 값의 경우 R과 M 값이 매 시행에서 고정된 값으로 지정되면, 이를 매번 계산할 필요가 없다. 본 구현에서는 미리 계산된 값을 사용할 경우 그림 2의 피연산자 입력 단계에서 이 값을 외부로부터 입력 받을 수 있으며, 그렇지 않을 경우에는 Step-1 단계에서 리덕션 모듈을 통해 이 값을 계산할 수 있다. 즉 피연산자 입력 단계에서 사전에 계산된 $R^2 \bmod M$ 을 입력할 경우 Step-1의 시간 비용을 줄일 수 있다. Step-2에서는 $x' = \text{MontMul}(x, R, M, R)$ 를 계산하여 메모리 A에 저장하고, 이후의 과정에서는 평문 X를 사용하지 않기 때문에, 계산 결과인 x' 을 메모리 A로부터 X로 복사한다. 다시 한 번 리덕션 모듈을 통해

Step-3의 $A = R \pmod{M}$ 을 수행하고 이 결과를 Y 메모리에 저장한다. 이후에는 Step-4의 for loop를 수행하게 되는데, 이진 형태의 지수값 e 를 왼쪽 쉬프트 연산을 통해 지수값 e 의 MSB(Most Significant Bit)가 0인지 1인지를 판단하여 이에 따라 동작이 수행되게 된다. 처음 for loop를 시작하면, MSB가 1이 될 때 까지 왼쪽 쉬프트를 수행한 뒤에 해당 비트에서부터 loop가 수행되게 된다. for loop에서는 Step-4.1의 Doubling 연산을 수행한 뒤, 지수값 e 의 현재 비트를 확인하여 값이 1이면, Step-4.2의 곱셈 연산을 수행하고, 이를 메모리 A에 저장한 후 Y에 복사한다. 이 과정은 e 의 모든 비트를 처리할 때까지 반복되며, for loop를 빠져나온 뒤 Step-5의 $A = \text{MontMul}(A, 1, M, R)$ 를 수행하고 나면 최종 지수승 연산의 결과가 메모리 A에 저장되게 된다.

3.3. 광학 오류 주입 탐지 기법

본 논문에서 제안하는 오류 주입 탐지 기법은 메모리와 몽고메리 곱셈 모듈에 각각 적용되었다. 메모리 부분에 광학 오류가 주입되어 메모리에 들어있는 값에 변화가 있었는지를 체크하기 위해, 메모리에 값이 입력되는 시점에서 입력 값에 대한 일종의 서명 값을 저장해 두고, 메모리에서 다시 저장된 값을 출력하여 사용할 때 다시 이 서명 값을 생성하여 값들이 입력될 시점과 변화가 있었는지를 검사한다. 이를 위해 본 논문에서는 RAX(Rotated and Accumulated XOR)라는 기법을 제안한다. RAX 기법을 설명하기 전에 이해를 돕기 위해 우선 AX(Accumulated XOR)에 대한 설명을 먼저 제시한 후에 RAX 기법을 설명하고자 한다. 또한 곱셈 모듈 내에는 연산 도중 광학 오류가 주입되었는지를 판단하기 위해 이를 탐지하기 위한 특별한 로직을 추가하였다. 본 RSA 연산 모듈은 오류가 주입되었을 경우에 동작을 멈추고, 암호문을 출력하지 않도록 제작되어 있다. 메모리와 곱셈 모듈에 적용된 오류 탐지 기법에 대하여 각각을 자세히 설명하면 다음과 같다.

3.3.1. 메모리에서의 오류 탐지 기법

본 구현에서는 메모리 부분에 주입된 광학 오류를 탐지하기 위해 메모리에 입력된 값과 출력된 값이 서로 동일한지를 체크하기 위한 로직을 탑재하였다. 이를 위해 메모리에서 입출력이 발생할 시에 무결성 체크를 위한 코드를 각각 생성하여 입력시에 생성된 코드와 출력시

생성된 코드가 같은지를 비교하는 방법으로 오류 주입 탐지 회로를 구현하였다. 본 몽고메리 지수승 연산기는 512, 1024, 2048 비트의 지수승 연산을 지원하지만, 설명에서는 1024비트 연산을 기준으로 설명하고자 한다.

본 몽고메리 기반 지수승 연산기에서 사용되는 1024 비트의 모든 피연산자는 32비트 단위로 나누어져 0번 주소에는 LSB(Least Significant Block)가 저장되며, 가장 상위 주소인 31번지에는 MSB(Most Significant Block)가 저장된다. 이 값들이 연산에 사용될 때는 LSB에서 MSB 순서로 메모리에서 읽은 후 연산에 사용되며, 저장시에도 동일한 순서로 메모리에 저장된다. 이렇게 메모리 입출력 순서가 항상 동일한 특성을 이용하면 메모리에 입력되는 값과 출력되는 값에 대한 무결성 검증 코드를 생성하여 광학 오류 주입 탐지 기법에 적용할 수 있다.

무결성 검증 코드 생성을 위해서 해쉬 함수나 CRC(Cyclic Redundancy Code) 등의 기술이 사용될 수 있지만, 이를 구현하기 위한 하드웨어적인 비용이 크기 때문에, 본 논문에서는 단순한 방법으로 무결성 검증 코드를 생성할 수 있는 RAX라는 기법을 제안한다.

본 구현에서 제안하는 RAX를 설명하기 전에 기반이 되는 AX기법을 먼저 설명하고자 한다. AX기법은 메모리에 저장된 피연산자를 32비트 단위로 모두 XOR 한 것으로, 메모리 입출력이 일어날 때 데이터가 하나씩 출력될 때 마다 XOR하여 레지스터에 누적 시키는 방법으로 동일한 연산을 수행할 수 있으며, 이를 알고리즘 3에 보였다.

알고리즘3. $AX(D)$ - Accumulated XOR 무결성 검증 코드 생성

입력 : $D = (d_{1023} \dots d_1 d_0)_b$

출력 : $R = (r_{31} \dots r_1 r_0)_b$

1. $R \leftarrow 0$

2. for $i=0$ to 31

2.1. $R \leftarrow R \oplus (d_{(32i+31)} \dots d_{(32i)})_b$

3. return R as result

AX 연산을 이용하게 되면, 해쉬 함수나 CRC기법에 비해 훨씬 단순한 로직으로 에러를 검출할 수 있다는 장

점이 있다. 하지만, 주입된 광학 오류로 인해 기존의 비트가 반대로 출력된다고 가정하였을 때, XOR 연산의 특성상 메모리의 동일한 비트위치에 짝수 개의 오류가 발생했다면, 짝수 번째의 오류로 인해 홀수 번째의 오류가 상쇄되어 오류를 검출하지 못하는 단점을 가진다. 이러한 단점을 해결하기 위해 본 논문에서는 알고리즘 4에 제시된 순환 형태의 AX, 즉 RAX 기법을 제안한다. 이 방법은 AX 기법에 Rotate 연산을 추가함으로써 AX 기법이 가진 짝수개 오류 주입에 대한 취약점을 극복하도록 하였다.

알고리즘 4. $RAX(D)$ - Rotated and Accumulated

XOR 무결성 검증 코드 생성

입력 : $D = (d_{1023} \dots d_1 d_0)_b$

출력 : $R = (r_{31} \dots r_1 r_0)_b$

4. $R \leftarrow 0$

5. for $i=0$ to 31

5.1. $R \leftarrow ROL(R \oplus (d_{(32i+31)} \dots d_{(32i)})_b)$

6. return R as result

이와 같은 방법을 이용하면, 특정 비트의 오류가 순환 연산에 의해 주변의 비트로 이동하며 RAX 연산을 수행하게 되고, 이후의 연산에서 이 오류가 RAX 값에 영향을 미치지 때문에, 오류가 발견되지 못할 확률이 AX에 비해 상대적으로 적으며, 이는 시뮬레이션을 통해 검증하였다. 여러 종류의 오류 모델에 대한 시뮬레이션 결과는 IV에서 보였다.

그림 3은 RAX 기법을 기반으로 메모리에 주입된 오류를 탐지하기 위한 회로를 보여주고 있다. 이 회로는 데이터가 입력될 때, 그리고 출력될 때 각각의 데이터를 이용해 RAX 무결성 검증 코드를 생성할 수 있도록, 입출력 값을 누적하여 XOR 시켜 저장할 수 있는 회로이다. 그리고 데이터가 저장될 시점에 계산된 RAX 값과 데이터가 다시 불러올 시점에 계산된 RAX 값을 비교할 수 있는 회로가 있으며, 이를 통해 광학 오류가 주입되었는지를 판단한다.

그림 3의 회로를 이용하여 광학 오류를 탐지하는 절차는 그림 4에 나타나 있다. 광학 오류 탐지 절차는 입출력 시퀀스가 시작하면서 이루어진다.

그림 3. 순환 누적 XOR 연산 방식의 오류 탐지 회로
Fig. 3 Fault detection circuit based on rotated and accumulated XOR

시작은 무결성 검증 코드가 저장되는 레지스터 (RAX_MEM, RAX_NEW, RAX_OLD)들을 초기화 하면서 시작된다. 몽고메리 지수승 연산이 시작되면, 반복적으로 곱셈 연산을 수행하게 된다. 피연산자들은 LSB에서 MSB 순으로, 32비트 단위로 불러와 사용되게 된다. 이 때 RAX_MEM에는 사전에 메모리에 데이터가 입력될 때 계산된 RAX 코드가 저장되어 있다. 연산을 위해 메모리의 데이터가 연산기로 로드 될 때에 메모리에서 출력되는 데이터를 이용해 RAX_OLD에 저장된 코드를 계산한다. 이와 동시에 메모리에서 읽어온 데이터를 이용해 곱셈 연산이 수행되며, 곱셈 연산의 결과가 메모리에 저장되면서 이 값들에 대한 코드가 RAX_NEW에 저장된다. 메모리에서 읽어온 값이 이전에 입력된 값과 같은지를 체크하려면 RAX_MEM과 RAX_OLD에 저장된 코드가 서로 같은지를 확인한다. 두 값이 같다면 오류가 주입되지 않은 것이며, RAX_NEW에 있는 새로 메모리에 입력될 값의 RAX 코드를 RAX_MEM에 갱신하여, 다음 데이터를 읽어서 사용할 때 활용할 수 있도록 한다.

3.3.2. 곱셈 연산 모듈 내의 광학 오류 주입 탐지 기법

메모리에서만 오류가 주입 되었는지를 탐지한다면, 연산중에 발생한 오류에 대해서는 탐지를 할 수 없다.

그림 4. 메모리 내 광학 오류 주입 탐지 순서도
Fig. 4 Flow chart of optical fault detection in memory

본 제안기법에서는 연산중에 오류가 주입되었는지를 확인하기 위해 몽고메리 곱셈에 사용되는 덧셈기 중 일부에 그림 5와 같은 광학 오류 주입 탐지 회로를 구성하였다. 이 회로는 레이저와 같은 광원이 1.1eV 이상의 에너지로 칩상으로 주사될 때, 양공의 형태로 흡수되는 원리를 이용한 것이다[16]. 광학 오류 주입 탐지 회로를 구성하기 위해 몽고메리 곱셈기를 구성하는 덧셈기 중 일부에 그림 5와 같이 덧셈 로직 외에 추가로 두 개의 인버터를 배치하였다. 광학 오류 주입을 감지하는 부분은 두 개의 인버터 사이의 짧은 선으로 표시한 전송선이며, 이 전송선은 곱셈기 내부에 주입되는 광학 오류를 잘 감

지할 수 있도록 곱셈기 로직 내부에 배치되어 연결되어야 한다.

그림 5. 덧셈기 내부의 광학 오류 탐지 회로
Fig. 5 optical fault detection circuit for adder

위의 논리 회로로 광학 오류를 감지하는 원리는 다음과 같다. 일반적으로 Reset 신호는 회로 내부의 플립플롭을 초기화하기 위해 사용되며, 일반적으로 논리값 1에서 0으로 변화하거나 논리값 0인 상태를 감지하여 플립플롭을 초기화시킨다. 이후에는 논리값 1을 유지해야 칩 내부의 모든 플립플롭이 정상적으로 동작을 한다. 본 회로에서는 전송선에 레이저를 통해 광학 오류를 주입하면 논리 값이 1로 변하는 상황을 가정하였다. 그림 5의 플립플롭은 초기에 리셋 신호에 의해 0으로 초기화 되어있다. RSASStart 신호가 1로 변하여 연산이 시작되었을 때, 인버터 사이의 전송선에 레이저로 인한 광학적 오류가 주입이 되면, 첫 번째 인버터를 통해 출력되는 논리값 0이 양공의 형태로 흡수된 레이저의 에너지로 인해 논리값 1로 바뀌게 된다. 이 값은 두 번째 인버터에 의해서 논리값 0으로 출력이 되며, 이후 오류 주입이 끝나게 되면, 다시 1로 바뀌어 출력되게 된다. 이 값의 변화는 플립플롭의 CLK 핀으로 입력이 되어 오류 주입으로 인한 rising edge로 인해 RSASStart 신호 값인 1이 capture 되도록 한다. 이와 같은 방법으로 광학 오류 주입을 탐지 가능하다.

3.3.3. ASIC 구현에서의 광학 오류 주입 탐지 회로 배치

몽고메리 곱셈을 위한 덧셈 연산 내부에는 알고리즘 1의 Step-2.2에 있는 $A + x_i y + u_i M$ 연산을 32비트 digit 단위로 수행하기 위해 많은 덧셈기가 사용되는데, 이들 중 일부 덧셈기에 오류 탐지 회로가 추가된 덧셈기를 배치

하여 몽고메리 곱셈에 필요한 덧셈 로직과 함께 배치 및 라우팅 하였다.

그림 6. 몽고메리 곱셈 로직 내부의 광학 오류 주입 탐지 회로 배치

Fig. 6 Placement of optical fault detection circuit in Montgomery multiplier logic

본 구현에서는 총 64개의 광학 오류 주입 탐지 회로가 추가된 덧셈기를 사용하였으며, 그림 6에 여러 개의 광학 오류 탐지 회로로부터 나온 출력을 오류 탐지에 사용하는 회로를 나타내었다. 각각의 오류 탐지 회로에서 나온 출력은 모두 AND 연산을 거쳐 플립플롭에 입력이 되며, 이는 하나의 회로라도 논리값 0을 출력하면 플립플롭의 입력으로 0이 입력되게 하기 위함이다.

오류 탐지 회로 자체는 그림 5에서 보이듯이 덧셈 연산과는 논리적으로는 전혀 관계없는 회로이며, 이로 인해 설계 프로그램은 임의로 최적화를 통해 오류 주입 탐지 회로를 설계에서 제거하기도 하므로, 이를 방지하기 위한 Constraints를 사전에 설정하였다.

그림 7. Constraints 적용 전의 오류 탐지 회로 분포
Fig. 7 Distribution of fault detection circuit before applying grouping constraints

오류 탐지 회로의 효과가 극대화되기 위해서는, 몽고메리 곱셈 로직 내부에 골고루 흩어져 배치되어야 한다. 셀 배치 및 라우팅 진행시 덧셈기와 동일한 모듈에 들어 있는 오류 주입 탐지회로는 각각의 덧셈기 내부에 배치 될 것으로 기대할 수 있지만, 실제 셀 배치 및 라우팅에서 설계 프로그램은 시간 Constraints 위주로 이를 수행하며, 오류 탐지회로가 골고루 흩어지지 않을 수가 있다. 그림 7은 아무런 Constraints 없이 배치 및 라우팅을 수행한 결과이며, 서로 가까이 배치된 오류 탐지 회로를 하얀색 동그라미로 묶어서 표시하였다. 그림의 점선으로 표시된 셀은 몽고메리 곱셈기에 속하는 셀이며, 그 외의 부분은 곱셈과 관련 없는 회로이다. 오류 탐지 회로가 곱셈에 필요한 셀들이 차지하는 영역 중에 가운데 위쪽으로 집중되어 배치된 것을 확인할 수 있다.

그림 8. 그룹핑 Constraints 적용 후 배치
Fig. 8 Distribution of fault detection circuit after applying grouping constraints

이러한 현상을 방지하기 위해 본 구현에서는 오류 탐지 회로와 덧셈기 회로를 그룹핑하는 Constraints를 설정하여, 같은 모듈 내에 있는 덧셈기 로직과 오류 탐지 회로가 최대한 가까이 배치 및 라우팅 될 수 있도록 하였다. 그 결과 그림 8과 같이 광학 오류 탐지 회로가 곱셈기 내부에 골고루 흩어져서 배치된 것을 확인할 수 있었다.

IV. 구현 결과

이 장에서는 III장에서 제안한 광학 오류 방지 기법을 적용하여 RSA 알고리즘을 하드웨어로 구현하였을 때의 성능을 제시하고자 한다. 제안하는 오류 주입 탐지 기법을 구현함에 있어 추가로 필요한 시간과 공간적인 비

용을 제시하기 위해, 오류 주입 탐지 기법이 적용되지 않은 몽고메리 곱셈만을 적용한 RSA 연산 모듈과 III장에서 제시된 광학 오류 탐지 기법을 적용한 연산 모듈을 각각 구현하였다. 본 논문에서는 전자를 M-RSA, 후자를 MC-RSA라고 부를 것이다. 제안하는 설계는 Verilog HDL을 이용해 RTL level로 설계 되었으며, 각각 ASIC을 target으로 합성 및 레이아웃 되었다.

4.1. ASIC 구현 결과

M-RSA와 MC-RSA는 Magnachips/Hynix의 0.18um 공정을 이용해 ASIC으로 제작이 되었으며, 합성, 배치 및 라우팅에는 각각 Synopsys 사의 Design Compiler B-2008.09-SP5 버전과 Astro Z-2007.03-SP12 버전이 사용되었다. 목표 동작 주파수는 동일하게 15MHz로 설정하여 합성되었다. 본 구현에서는 5개의 메모리 구현을 위해 메모리 마크로 블록을 사용하지 않고, Standard Library에 있는 셀만으로 합성 되었다.

표 1에 M-RSA와 MC-RSA가 ASIC으로 구현되었을 때의 결과를 비교하였다.

연산 수행 시간의 경우 MC-RSA는 광학 오류 주입 탐지 기법이 적용되어 있음에도 불구하고, M-RSA와 같은 시간에 연산을 수행함을 표를 통해서 확인할 수 있다. 이는 제안하는 광학 오류 주입 기법이 지수승 연산과는 별도로 독립적으로 동작하기 때문에 가능하다. 제안하는 기법은 몽고메리 연산에 필요한 $R^2 \bmod M$ 값을 사전에 계산된 값을 E 메모리에 입력하여 사용하거나, 연산 로직으로 하여금 $R^2 \bmod M$ 값을 계산하도록 할 수 있다. $R^2 \bmod M$ 계산에는 131757 클럭 사이클이 필요하다.

전체 연산 수행 시간에 있어서 가장 영향을 많이 미치는 것은 *MontMul*의 실행횟수이다. 지수승 연산 (*MontExp*)의 경우 알고리즘 2에 보였듯이 지수인 E 를 이진수로 표현하였을 때 1인 비트의 개수에 따라 *MontMul*의 수행 횟수가 달라져 전체 연산 시간에 영향을 미치게 된다. *MontMul*의 경우 알고리즘 1에 보였듯이 주요 연산인 Step-2가 $n=1024$ 회를 반복하게 되고, 연산 결과를 X 혹은 Y 레지스터로 복사하는데 32 클럭 사이클이 소모되어, 기타 A 와 M 의 비교를 위해 필요한 시간과 조건 체크를 위한 과정의 시간 비용을 α 라고 할 때,

표 1. M-RSA와 MC-RSA의 ASIC 구현 결과 비교
Table. 1 ASIC implementation result between M-RSA and MC-RSA

설계		M-RSA	MC-RSA
Frequency(MHz)		15	15
Clock Cycles	E_1	R^2 미입력 141141 R^2 입력 9384	R^2 미입력 141141 R^2 입력 9384
	E_2	R^2 미입력 2520320 R^2 입력 2388563	R^2 미입력 2520320 R^2 입력 2388563
Latency(ms)	E_1	R^2 미입력 9.4 R^2 입력 0.6	R^2 미입력 9.4 R^2 입력 0.6
	E_2	R^2 미입력 168 R^2 입력 159.2	R^2 미입력 168 R^2 입력 159.2
Area (GE)	조합회로	연산로직 20285 메모리 38794 제어로직 1116 총 합 60195	연산로직 20287 메모리 40454 제어로직 1130 총 합 61871
	순차회로	연산로직 2155 메모리 55839 제어로직 597 총 합 58591	연산로직 2156 메모리 57605 제어로직 596 총 합 60357
	총 합	118786	122228
	전력소모(mW)	22.58	22.35

연산 실행 시간은 지수승 E 에 따라 달라지기 때문에, 시간 성능 측정에는 PKCS #1[20]에 제시되어 있는 테스트 벡터(M, X, E)와, 연산에 필요한 최장 시간을 도출하기 위해 별도로 E_2 를 사용하였으며, 이들 테스트 벡터를 표 2에 나타내었다.

[illegible]

소화 가능하다. 따라서 가장 비중을 많이 차지하는 연산인 *MontMul* 만을 고려하였을 때, 알고리즘 2의 Step-2와 함께 총 8번이 수행된다. 알고리즘 2에서 Step-1의 $R^2 \bmod M$ 계산을 제외하고, *MontMul* 이외에 필요한 시간을 β 라고 하면, 대략 $8(1056 + \alpha) + \beta$ 의 시간이 필요하며, 이는 $8448 + 7\alpha + \beta$ 이다. $R^2 \bmod M$ 값을 미리 계산하여 지수승 연산 모듈에 공급할 경우 9384 클럭 사이클이 필요함을 시뮬레이션을 통해 확인하였으며, $R^2 \bmod M$ 값을 미리 입력하지 않을 경우 141141 클럭 사이클이 필요하다. E_2 를 사용할 경우 Step-4.1과 4.2를 모두 1024번씩 수행하므로, 총 2049번의 *MontMul*을 수행하게 되어, 수행시간은 $2049(1056 + \alpha) + \beta$ 로 $2163744 + 2049\alpha + \beta$ 이다. 시뮬레이션에서는 $R^2 \bmod M$ 값을 미리 계산하여 입력할 경우 총 2388563 클럭사이클이 필요함을 확인할 수 있었다.

4.2. 메모리에 대한 광학 오류 주입 공격 시뮬레이션

본 시뮬레이션은 메모리 내부와 메모리로 연결되는 버스 상에 오류가 가해진 상황을 가정하여 시행되었으며, 오류가 주입되었을 때 값이 반전되는 Bit Flip 모델과, 값이 1 혹은 0으로 고정되는 stuck-at-1과 stuck-at-0 오류에 대하여, 진행되었으며, 홀수개 및 짝수개의 일시적 오류와 영구적인 오류에 대하여 시뮬레이션을 진행하였다. 이와 같은 광학 오류의 주입 형태에 따른 모델을 그림 9에 나타내었다.

그림 9. 광학 오류 주입 위치에 대한 모델
Fig. 9 Model for the optical fault injecting position

이는 각 주소 당 32비트의 데이터가 저장되는 메모리에 대한 모델로써, 그림의 가로 축은 하나의 메모리 주소에 저장된 각 비트들을 나타내며, 세로축은 각 메모리의 주소를 나타낸다. 광학 오류는 메모리를 구성하는 셀에 주입 될 수도 있으며, 메모리로 연결되는 버스 상에 주입 될 수도 있다. 셀의 특정 위치에 오류가 주입된다고 가정하면, 1~9의 모델은 메모리의 특정 주소에 하나의 비트에 대해서만 오류가 주입된 상황을 나타내며, 10~18의 모델은 연속된 주소의 메모리의 동일한 비트 위치에 걸쳐 2개의 비트에 오류가 주입된 상황을 나타낸다. 19~21의 모델은 모든 메모리 주소의 특정 1개 비트에 모두 오류가 주입된 상황을 가정하였다.

그림 9는 전체 메모리 셀에서의 어떤 셀 위치에 오류가 주입될 것인지를 나타낼 수도 있지만, 메모리 데이터 버스의 특정 비트 상에 주입된 오류에 대하여 주입 시간에 따른 모델로 바라볼 수도 있다. 예를 들면, 1번 오류의 경우 29번 비트 정도의 버스 위치에 하위 주소의 데이터가 입출력 되는 특정 시점에만 주입된 오류로 바라볼 수 있다. 같은 맥락으로 19번 오류의 경우 버스의 29번 비트 위치에 계속적으로 주입된 오류로 볼 수 있다.

다음에서 Bit Flip, Stuck-at-1, Stuck-at-0 모델에 대한 시뮬레이션 결과를 각각 제시한다. 각 메모리에 입력되는 값은 랜덤하게 설정을 하였으며, 각 모델에 대한 실험은 모두 동일한 랜덤 테스트 벡터를 이용하여 시뮬레이션 되었으며, 1024비트의 데이터가 32비트씩 32번 입력되는 상황을 10,000,000번 반복하여, 주입된 광학 오류를 제대로 탐지해 내는지를 시뮬레이션 하였다.

4.2.1. Bit Flip 모델에 대한 시뮬레이션 결과

표 3에 Bit Flip 모델에 대한 광학 오류 주입 시뮬레이션 결과를 보였다. AX 모델을 사용하게 되면, Bit Flip 모델의 경우 특정 비트에 오류가 주입되면, 오류가 주입된 비트에 XOR 연산의 형태로 AX 연산 결과에 반영이 된다. 이 때, 한번 더 같은 위치에 오류가 주입되게 되면, 이전에 입력되었던 오류가 상쇄된다. 이렇게 오류가 상쇄되는 현상을 방지하기 위해 RAX는 오류가 주입된 위치가 이동할 수 있도록 연산의 결과를 비트 순화 이동 시킨 후 다시 XOR을 취하는 방법을 사용하였다. 표 2에서 나타나듯이 AX의 경우 짝수개 오류 및 영구적 오류의 경우 전혀 탐지를 못하지만, RAX의 경우 모든 경우에서 오류를 탐지해 내는 것을 확인할 수 있다.

표 3. AX Bit Flip 모델에 대한 광학 오류 주입 시뮬레이션 결과
Table. 3 Optical fault injecting simulation result for the AX bit model

오류 주입 모델	단일 오류									짝수개 오류									영구적 오류		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
AX 오류 탐지율	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0	0	0
RAX 오류 탐지율	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

4.2.2. Stuck-at 모델에 대한 시뮬레이션 결과

표 4에 Stuck-at 모델에 대한 시뮬레이션 결과를 보았다. Stuck-at 모델의 경우 오류가 주입 되었지만, 이가 오류로 반영되지 않는 경우가 있다. 예를 들어 원래 값이 1인 비트에 대해서 Stuck-at-1 오류가 발생할 경우에는 오류가 주입되지 않은 것과 같다. 이러한 이유로 Stuck-at 모델의 경우 실제 오류라고 할 수 있는 경우의 수만을 셀하여 실제 오류 주입 횟수라는 제목으로 표에 나타내었다. AX의 경우 단일 오류에 대해서는 완벽하게 탐지해 내지만, 짝수 오류(2번 연속적 오류) 및 영구적 오류에 대해서는 확률적으로 탐지가 가능하다. 이는 Stuck-at-1 모델을 예로 들어 설명하면, 다른 주소의 동일 비트에 두 번의 오류가 주입된다고 가정했을 때, 원래의 비트 값이

(1, 1) 경우에는 오류가 주입되었는지 아닌지를 판단할 수 없으며, 실제 오류가 주입되어도 값에는 영향을 미치지 않는다. 이 경우를 제외하면, (0, 0), (0, 1), (1, 0) 인 세 가지 상황이 존재하게 되는데, 이 중에 (0, 0)인 경우에는 두 개의 비트에 오류가 XOR되어 AX 계산에 반영되면, 이 결과가 상쇄되기 때문에, 오류로 인식이 되지 않는다. 값이 (0, 1), (1, 0)인 경우에만 오류로 인식되며, 메모리에 입력되는 값이 충분히 랜덤하게 골고루 입력된다면, 이 상적인 경우 약 1/3의 확률로 오류를 감지해 낼 수 있다. 오류 2개(짝수개의 오류)를 통해 테스트한 경우(모델 10~18)에는 성공률이 약 66.7%에 근접한 것을 확인할 수 있다. 영구적 오류의 경우 오류가 주입 되는 위치의 비트가 모든 주소에서 1인 경우에는 오류가 주입되었는지 아

표 4. Stuck-at 모델에 대한 오류 주입 시뮬레이션 결과

Table. 4 Optical fault injecting simulation result for the stuck-at model.

(1) Stuck-at-1 모델의 경우 오류가 주입된 위치의 비트가 원래 1인 경우 이는 오류로서 인식되지 않으며, 10,000,000번의 실험 중 이러한 경우를 제외한 횟수를 나타냄. 2) Stuck-at-0 모델의 경우 역시 원래 0인 비트의 경우 오류가 주입되어도 이를 오류로 인식하지 않으며, 이러한 경우를 제외한 횟수를 나타냄)

Fault 모델		Stuck-at-1 모델			Stuck-at-0 모델		
		실제오류 주입 횟수1	AX 오류 탐지율	RAX 오류 탐지율	실제오류 주입 횟수2	AX 오류 탐지율	RAX 오류 탐지율
단일 오류	1	4869734	100	100	5130266	100	100
	2	4599373	100	100	5400627	100	100
	3	5679847	100	100	4320153	100	100
	4	5014808	100	100	4985192	100	100
	5	4562852	100	100	5437148	100	100
	6	4469812	100	100	5530188	100	100
	7	5008140	100	100	4991860	100	100
	8	4539108	100	100	5460892	100	100
	9	5361917	100	100	4638083	100	100
짝수 오류	10	4930320	66.17	100	4930320	65.92	100
	11	5464556	77.58	100	5464556	64.9	100
	12	5153069	65.98	100	5153069	70.18	100
	13	5209389	67.56	100	5209389	69.47	100
	14	4885550	68.98	100	4885550	62.61	100
	15	7596973	81.34	100	5235773	65.19	100
	16	5071364	66.59	100	5071364	68.02	100
	17	4000416	58.62	100	4000416	55.74	100
	18	3845655	57.58	100	3845655	53.66	100
영구적 오류	19	4476679	44.77	100	4476679	44.77	100
	20	5289175	52.89	100	5289175	52.89	100
	21	4946239	49.46	100	4946239	49.46	100

난지에 대한 사실을 알 수 없다. 이 경우를 제외하고, 나머지 경우 중에서 해당 비트의 0의 개수가 짝수가 되는 경우에는 오류가 주입된다 하더라도 오류로 판단하지 못하며, 나머지 경우에만 오류를 제대로 판단할 수 있다. 0의 개수가 짝수가 되는 경우는 홀수가 되는 경우보다 1번 작으며(모든 비트가 1인 경우를 제외했기 때문에), 확률은 절반에 가깝다.

V. 결 론

본 논문에서는 효율적으로 지수승 연산을 수행하면서도 광학 오류 주입 공격을 탐지해 오류 발생시 암호문을 출력하지 않는 안전한 RSA 모듈을 구현하는 방법을 제안하였다. 효율적인 지수승 연산을 위해 몽고메리 방식의 곱셈 및 지수승 연산을 활용하였으며, 광학 오류 주입 공격 탐지를 위해서 각각 메모리와 곱셈 모듈에 사용되는 덧셈 로직에 광학 오류 주입 탐지 기법을 구현하였다. 메모리에서의 광학 오류 주입 공격 탐지를 위해서는 메모리에 값이 입력 될 때와 그 값이 다시 사용될 때 각각 RAX라고 하는 간단한 로직을 필요로 하는 무결성 검증 코드 생성 기법을 이용하여 광학 오류 주입 여부를 판단한다. 연산 로직에서는 레이지 주입시에 발생하는 에너지를 검출할 수 있는 로직을 연산 로직 곳곳에 배치하여 광학 오류를 탐지할 수 있도록 하였다. 광학 오류 주입 탐지 기법을 적용하지 않은 구현과 비교하였을 때, 광학 오류 탐지 기법으로 인한 시간 적인 오버헤드는 존재하지 않으며, 동일한 주파수의 클럭을 사용할 경우 동일한 시간에 연산 결과를 출력한다. 또한 ASIC 구현시에는 3% 정도의 매우 작은 면적 오버헤드를 가진다.

참고문헌

- [1] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM vol. 21, no. 2, pp. 120-126, 1978.
- [2] P. L. Montgomery, "Modular Multiplication without Trial Division", Mathematics of Computation, vol. 44, pp. 519-521, 1985.
- [3] C. Couvreur, J. Quisquater, "Fast decipherment algorithm for RSA public-key cryptosystem", Institution of Engineering and Technology IET, Electronics Letters, vol. 18, no. 21, pp. 905-907, October, 1982.
- [4] D. Boneh, R. A. DeMillo, R. J. Lipton, "On the importance of checking cryptographic protocols for faults", EUROCRYPT 1997, LNCS, col. 1223, pp. 37-51, 1997.
- [5] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimbalu and T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults", In Proceeding of the 1997 Security Protocols Workshop, 1997.
- [6] H. Park, K. S. Bae, S. J. Moon, D. H. Choi, Y. S. Kang, J. C. Ha, "A New Fault Cryptanalysis on Montgomery ladder Exponentiation Algorithm", ICIS-2009, vol. 2, pp. 896-899, 2009.
- [7] S. Mangard, E. Oswald, and T. Popp, "Power Analysis Attacks: Revealing the Secrets of Smart Cards", Springer Verlag, 2007.
- [8] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerers apprentice guide to fault attacks", Workshop on Fault Diagnosis and Tolerance in Cryptography in association with DSN 2004 - The International Conference on Dependable Systems and Networks, pp. 330-342, 2004.
- [9] S. Skorobogatov and R. Anderson, "Optical Fault Injection Attack", Workshop on Cryptographic Hardware and Embedded Systems-CHES'02, LNCS 2523, pp. 2-12, 2002.
- [10] M. Schmidt and M. Hutter, "Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results", Proceedings of the 15th Austrian Workshop on Microelectronics, pp. 61-67, October, 2007.
- [11] 박제훈, 문상재, 하재철, "CRT-RSA 암호시스템에 대한 광학적 오류 주입 공격의 실험적 연구", 정보보호학회논문지, 제 19권, 제 3호, pp. 51-59, 2009.
- [12] A. Shamir, "How to check modular exponentiation", In presented at the rump session of EUROCRYPT 1997, May, 1997.

- [13] A. Shamir, "Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks" US Patent 5991415, November, 1999.
- [14] S. Yen, S. Kim, S. Lim, and S. Moon, "RSA speedup with Chinese Remainder Theorem Immune Against Hardware Fault Cryptanalysis", IEEE Transaction on Computer, vol. 52, no. 4, pp. 461-472, April, 2003.
- [15] S. Yen, D. Kim, and S. Moon, "Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection", FDTC-06, LNCS 4236, pp. 53-61, Springer-Verlag, 2006.
- [16] K. T. Tan, S. H. Tan and S. H. Ong. "Functional failure analysis on analog device by optical beam induced current technique", In Physical & Failure Analysis of Integrated Circuits 1997, pp. 296-301, 1997.

저자소개

이동건(Dong-geon Lee)

2009년 부산대학교 정보컴퓨터
공학부 졸업(학사)
2011년 부산대학교 컴퓨터공학과
졸업(석사)

2011년 ~ 현재 부산대학교 컴퓨터공학과 박사과정
재학중

※ 관심분야: 정보보호, 부채널공격, 오류주입공격,
VLSI Design

최용제(Yong-je Choi)

1996년 8월 전남대학교 전자공학과
졸업
1999년 2월 전남대학교 전자공학과
석사

1999년 2월 ~ 8월 전남대학교 전자통신연구소
인턴연구원

1999년 8월 ~ 현재 한국전자통신연구원 사이버융합
보안연구단 선임연구원

※ 관심분야: 보안프로세서 설계, 부채널 분석시스템,
RFID/USN 보안

최두호(Dooho Choi)

1994년 성균관대학교 수학과 졸업
(학사)

1996년 한국과학기술원(KAIST)
수학과 석사(이학석사)

2002년 한국과학기술원(KAIST) 수학과 석사
(이학박사)

2002년 1월 ~ 현재 한국전자통신연구원(ETRI)

사이버융합보안연구단 책임연구원/팀장

※ 관심분야: 암호학, 부채널 분석, RFID/USN 보안

김민호(Minho Kim)

2006년 8월 Oregon 주립대
공학박사

2008년 12월 ~ 현재 공군사관학교
전자전산학과 부교수

※ 관심분야: 암호학, 정보보호, 보안프로토콜, 컴퓨터
포렌식

김호원(Howon Kim)

1993년 경북대학교
전자공학과 학사

1995년 포항공과대학교
전자전기공학과 공학석사

1999년 포항공과대학교 전자전기공학과 공학박사

1998년 ~ 2008년 한국전자통신연구원(ETRI)

정보보호연구단 선임연구원/팀장

2008년 ~ 현재 부산대학교 정보컴퓨터공학부 부교수

※ 관심분야: 스마트그리드 보안, RFID/USN 보안,
PKC 암호, VLSI, Embedded system 보안, IoT