논문 2004-41SD-9-10

SPA 대응 기법을 적용한 이진체 위의 타원곡선 스칼라곱셈기의 하드웨어 구현

(Hardware Implementation of Elliptic Curve Scalar Multiplier over GF(2n) with Simple Power Analysis Countermeasure)

김 현 익*, 정 석 원**, 윤 중 철***

(Hyun Ik Kim, Seok Won Jung, and Joong Chul Yoon)

요 약

본 논문에서는 하드웨어 상에 구현된 암호 프리미티브의 안전성을 위협할 수 있는 부채널 공격의 하나인 단순 전력 분석(Simple Power Analysis)에 대응하는 알고리즘을 제안하고 이를 하드웨어로 구현하고자 한다. 제시하는 알고리즘은 기존에 알려진 대응 알고리즘보다 스칼라 곱셈 방법이 보다 효율적인 장점이 있다. 기존의 대응 알고리즘은 연산의 종속성 때문에 하드웨어의 장점인 병렬 처리 기법을 효율적으로 적용하기 어려운 단점이 존재한다. 이러한 단점을 보완코자 본 논문에서 제시하는 알고리즘은 동작 성능의 저하를 최소화하기 위해 역원 계산 시간 동안 곱셈 및 제곱 연산을 수행할 수 있도록 구성하였다. 또한 하드웨어 기술 언어인 VHDL(VHSIC Hardware Description Language)로 제안 알고리즘을 구현하여 성능 검증을 수행하였으며 이의 활용을 모색하였다. 하드웨어 합성은 Syplify pro7.0을 사용하였으며, 타겟 칩 Xillinx VirtexE XCV2000EFG1156을 대상으로 하였을 때 전체 등가 게이트는 60,608게이트, 최대 동작 주파수는 약 30Mhz로 산출되었다. 본 논문에서 제시한 스칼라 곱셈기는 전자 서명(Digital Signature), 암호화(Encryption) 및 복호화(Decryption), 키 교환(Key Exchange)등의 핵심 연산으로 사용될 수 있을 것으로 보이며, 자원 제약이 심한 Embedded-Micom 환경에 적용하였을 경우, 단순 전력 분석에 안전하면서 효율적인 연산 기능을 제공할 수 있을 것으로 보인다.

Abstract

This paper suggests a new scalar multiplication algorithm to resist SPA, which threatens the security of cryptographic primitive on the hardware recently, and discusses how to apply this algorithm. Our algorithm is better than other SPA countermeasure algorithms aspect to computational efficiency. Since known SPA countermeasure algorithms have dependency of computation. these are difficult to construct parallel architecture efficiently. To solve this problem, our algorithm removes dependency and computes a multiplication and a squaring during inversion with parallel architecture in order to minimize loss of performance. We implement hardware logic with VHDL(VHSIC Hardware Description Language) to verify performance. Synthesis tool is Synplify Pro 7.0 and target chip is Xillinx VirtexE XCV2000EFG1156. Total equivalent gate is 60,608 and maximum frequency is 30Mhz. Our scalar multiplier can be applied to digital signature, encryption and decryption, key exchange, etc. It is applied to a embedded-micom, it protects SPA and provides efficient computation.

Keywords : Side channel attack, Simple Power Analysis, Elliptic Curve Cryptosystem, VHDL, FPGA

* 학생회원, 고려대학교 정보보호대학원
(CIST, Korea University)
** 정회원, 목포대학교 정보보호전공
(Major in Information Security, Mokpo Universiry)
*** 정회원, 삼성전자 시스템 LSI 사업부
(System LSI Division, Samsung Electronics, Co. Ltd)
※ 본 연구는 정보통신부 대학 IT연구센터 육성·지원 사업의 연구결과로 수행 되었습니다.
접수일자: 2004년3월31일, 수정완료일: 2004년9월1일

I. 서 론

최근 정보 통신 이용자의 폭발적인 증가와 다양한 네트워크, 통신 인프라의 발전으로 통신 서비스에 대한 이용자의 요구가 고도화, 다양화 되고 있다. 이에 대응하기 위해 정보보호 기술, 정보 통신 기술의 향상을 위한 연구 개발이 세계적으로 활발하게 이루어지고 있으

며, 기술의 고도화를 지지하기 위해 소프트웨어 및 하드웨어 기술의 연구 개발이 활발하게 진행되고 있다. 특히 IC 카드 및 보안 토큰, ATM기, 보안 라우터 등과 같은 하드웨어 기반의 보안 시스템에 양질의 보안 서비스를 제공하기 위한 연구 및 표준화 작업이 활발하게 진행되고 있다. 암호시스템에 대한 기존의 안전성 평가는 증명 가능한 안전성이나 계산적 안전성과 같은 이론에 기반을 두고 있다.

그러나 최근 암호시스템이 구현된 하드웨어에서 또 다른 형태의 공격이 제기되고 있어 암호시스템의 실질적인 안전성 평가가 중요한 이슈로 떠오르고 있다. 즉 스마트카드, 모바일 폰, PDA 등과 같은 내장된 하드웨어에 암호알고리즘이 구현될 때, 전력 소모량, 알고리즘의 수행시간, 오류주입 및 전자파 방출양 등과 같은 부가정보가 공격자에게 악의적으로 이용될 수 있다.

이러한 형태의 암호시스템에 대한 공격을 부채널 공격(side channel attack)이라고 한다. 1996년 Paul Ko-cher가 암호 프리미티브에 대한 시간공격(timing atta-ck)[9]을 도입하여 최초로 부채널 공격법을 소개하였다. 시간 공격(Timing Attack)은 비밀 키에 의존하여 발생하는 암호학적 연산 시간의 차이를 탐지하여 비밀 키를 유도하는 공격을 말한다.

또한 1999년에 Paul Kocher는 새로운 형태의 부채널 공격법인 전력 분석 공격법(Power Analysis)[10]을 소개하였다. 전력 분석 공격법은 단순 전력 분석법(Simple Power Analysis; SPA)과 차분 전력 분석법(Differential Power Analysis)으로 나뉠 수 있다.

단순 전력 분석법은 암호 프리미티브의 연산 수행 시 소모하는 전력량을 관찰하여 키 값을 유도해 내는 방법이다. 이와는 달리 차분 전력 분석법은 비밀 키에 대해 여러 번의 알고리즘 수행을 거쳐 수집된 전력 소모량 측정치의 차이와 키 사이의 상관관계를 통계적으로 분석하여 키 값을 유도해 내는 방법이다.

이외의 부채널 공격 방법으로 오류주입 공격법(Fault Attack)[19]이 있는데 이는 공격자가 암호 칩 내부의 특정 메모리에 오류를 주입하여 잘못된 연산이 수행되도록 하여 공격자가 원하는 값을 유도해 내는 방식을 일컫는다.

이중 전력 분석 공격법은 실제 환경에서 쉽게 적용될 수 있으며, 대칭키 및 공개키 암호시스템에 동일하게 적용할 수 있어 이에 대한 대응 방안에 대한 연구가 활발히 이루어지고 있다.

본 논문에서는 차세대 공개키 암호알고리즘으로 주목받고 있는 타원곡선 암호알고리즘의 핵심 연산인 스칼라 곱셈 연산에 대해 단순 전력 분석법에 대응하는 알고리즘을 제시하고 이를 하드웨어로 구현하고자 한다. 기존에 알려진 단순 전력 분석 대응 알고리즘은 연산의 종속성 때문에 하드웨어의 장점인 병렬 처리 기법을 적용하기 어려운 단점이 존재한다. 본 논문에서 제시하는 알고리즘은 알고리즘 상의 데이터의 종속성을 제거해서 병렬처리를 효율적으로 가능하게 했으며, 동작 성능의 저하를 최소화하기 위해 유한체 원소의 역원 계산 시간동안 유한체 원소의 곱셈 및 제곱 연산을 수행할 수 있도록 구성하였다. 또한 하드웨어 기술 언어인 VHDL(VHSIC Hardware Description Language)로 제안 알고리즘을 구현하여 성능 검증을 수행하였다. 하드웨어 합성은 Syplify pro7.0을 사용하였으며, 타겟 칩 Xillinx VirtexE XCV2000EFG1156을 대상으로 하였을 때 전체 등가 게이트는 60,608게이트, 최대 동작 주파수는 약 30Mhz로 산출되었다.

본 논문은 타원곡선 암호알고리즘의 기본적인 개념과 기존에 제시된 대응 알고리즘의 분석, 그리고 제안하는 알고리즘의 장점과 연산의 효율성 및 하드웨어 아키텍처와 동작 성능에 대한 고찰로 구성되어 있다.

Ⅱ. 타원곡선 암호 알고리즘

타원곡선을 이용한 암호시스템의 적용은 1985년 Neal Koblitz[8]와 Victor Miller[12]에 의해서 처음 제안되었다.

타원곡선은 유한체의 표수 p가 2인 경우와 3보다 큰 경우로 나뉘어 질 수 있다. 본 논문에서는 표수가 2인 경우에 대한 것만 살펴볼 것이다. 그러나 아래에 설명하는 스칼라 곱셈과 그 알고리즘은 표수가 3보다 큰 경우에도 적용이 가능하다.

표수가 2인 유한체 $F=GF(2^n)$에 대하여 타원곡선 E는 다음과 같이 정의된다.

$$E: y^2 + xy = x^3 + ax^2 + b \quad (1)$$

여기에서 $a, b \in F$ 이다. 이에 대한 타원곡선 군을 아래 식과 같이 정의한다.

$$E(F)=\{(x, y)|y^2+xy = x^3+ax^2+b\} \cup \{O\} \quad (2)$$

여기에서 O는 무한원점이다. 이때 E(F)는 점들의 덧셈에 대하여 덧셈 군(additive group)을 형성하게 되고 무한원점 O는 덧셈의 항등원이 된다. 타원곡선 위의 점

P=(x,y)에 대한 정수 k와의 스칼라 곱셈(Scalar Multiplication) [k]P는 점 P를 k번 더하는 것으로 정의한다.

$$[k]P = P + P + \cdots + P \quad (3)$$

타원곡선 암호시스템의 안전성은 P와 [k]P가 주어졌을 때 k를 알아내는 것이 어렵다는 사실에 바탕을 두고 있으며, 이를 ECDLP(Elliptic Curve Discrete Logari-thm Problem)라 한다[5].

스칼라 곱셈 방법의 종류는 크게 이진 방법(binary method), 변형된 m-ary 방법, sliding window 방법[1] 등이 있다. 그 중 이진 방법은 사전계산 과정을 거치지 않기 때문에 적은 면적으로 스칼라 곱셈기를 구현할 수 있는 장점이 있다. 따라서 본 논문에서 이진 방법을 이용한 스칼라 곱셈 방법을 단순전력 분석법인 SPA (Simple Power Analysis)에 강한 구조로 변형하여 구현에 적용한다.

이진 스칼라 곱셈 방법은 스칼라 k를 이진수로 전개한 후 한 비트씩 보아가며 연산을 하는 방법으로 다음 알고리즘 1과 같다.

[알고리즘 1] 이진 연산 방법

Input: 점 P, m 비트 정수 $k=(k_{m-1}k_{m-2}\ldots k_0)_2$, $k_{m-1}, k_{m-2}, \ldots, k_0 \in \{0,1\}$

Output: Q = [k]P

---

1. Q ← O
2. For j = m - 1 down to 0
   2.1 Q ← [2]Q
   2.2 If k_j = 1 then Q ← Q + P
3. Retrun Q

이진 연산 방법에는 k를 읽는 순서에 따라 RL(Right-to-left) 방식과 LR(Left-to-Right)방식이 있으며 알고리즘 1은 LR 방식에 의한 스칼라 곱셈 방법을 나타내고 있다. 알고리즘 1은 k의 해밍 웨이트(Hamming weight)를 W라 할 때 m번의 두 배 연산과 W번의 덧셈 연산이 수행된다.

타원곡선 스칼라 곱셈의 핵심 연산인 타원곡선 점 덧셈 연산과 점 두 배 연산 공식은 유한체 원소들의 사칙연산으로 표현된다. 그런데 타원곡선 위의 점을 아핀좌표계(affine coordinates)로 표현하느냐 사영좌표계(pro-jective coordinates)로 표현하느냐에 따라 스칼라 곱셈 계산 공식이 달라진다. 타원곡선의 점을 사영좌표계로 표현하면 타원곡선 스칼라 곱셈공식 중에 유한체 원소에 대한 역원 계산이 필요하지 않은 장점이 있다. 그러나 유한체 원소의 역원 계산이 곱셈 연산보다 7배 보다 빠른 경우 아핀좌표계를 사용하는 것이 더 효율적임은 알려진 사실이다[11].

본 논문에서는 효율적인 역원 연산 알고리즘인 Montgomery 형태 알고리즘을 부채널 공격에 강한 구조로 변형하여 적용하였으며, 이에 따른 연산 효율의 저하를 최소화하도록 구조를 설계하였다. 따라서 본 논문에서는 아핀좌표계를 사용한다.

다음은 아핀좌표계를 사용하여 이진체 $GF(2^n)$ 위에 정의된 타원곡선 스칼라 곱셈의 핵심 연산인 타원곡선 점 덧셈 연산과 두 배 연산을 정리한 것이다[5].

i) 덧셈 연산인 경우

서로 다른 두 점 $P_0 = (x_0, y_0)$, $P_1 = (x_1, y_1)$에 대해 $P_2 = P_0+P_1 = (x_2, y_2)$ 일 때,

$$\lambda = \frac{y_0 + y_1}{x_0 + x_1} \text{ 라 하면}$$

$$x_2 = \lambda^2 + \lambda + x_0 + x_1 + a,$$

$$y_2 = (x_0 + x_2)\lambda + x_2 + y_0 \quad (4)$$

이다.

ii) 두 배 연산인 경우

$P_0 = (x_0, y_0)$, $P_1 = 2P_0 = (x_1, y_1)$ 일 때,

$$\lambda = x0 + \frac{y_0}{x_0} \text{ 라 하면}$$

$$x_1 = \lambda 2 + \lambda + a,$$

$$y_1 = (x_0 + x_1)\lambda + x_1 + y_0 \quad (5)$$

이다.

두 배 연산 및 덧셈 연산에 필요한 연산회수를 유한체 원소의 역원 연산(I), 곱셈 연산(M), 멱승 연산(S)으로 계산하여 표로 정리하면 표 1과 같다.

표 1. 덧셈 및 두 배에 필요한 유한체 연산 회수
Table 1. Number of field arithmetic for doubling & addition.

연산 \ 체	$GF(2^n)$
타원곡선 점 덧셈	1I + 2M + 1S
타원곡선 점 두 배	1I + 2M + 1S

Ⅲ. 단순 전력 분석 방법(Simple Power Analysis)

단순 전력 분석법(Simple Power Analysis; SPA)은 비밀 키에 따라 하드웨어에 탑재된 암호 프리미티브의 동작 수행 시 소모하는 전력량을 직접 관측하여 해당 시스템을 공격하는 방법이다. 즉 비밀 키의 비트 정보에 따른 전력 소모량을 구분하여 정보를 도출하는 분석 방법이다.

단순 전력 분석법(Simple Power Analysis)은 Paul Kocher[10]가 제안 하고 이를 최초로 DES에 적용하였다. 스마트카드 내에서 DES 알고리즘 연산 수행 시, 키 스케줄 내부에 순환 이동(rotate shift)하는 28비트 키 레지스터와 DES의 치환(permutation)부분의 상관성을 이용하여 키 비트가 '0'일 때와 '1'일 때의 치환 과정에 해당하는 소비 전력의 차이를 관측하여 단순 전력 분석을 수행하였다.

대표적인 공개키 알고리즘인 RSA는 모듈러 멱승 연산을 핵심 연산으로 사용한다. 모듈러 멱승 연산은 일반적으로 간단한 제곱-곱셈(square-multiply) 연산으로 구현되며, 제곱과 곱셈 명령의 수행 시, 각 연산에 해당하는 전력 소모량을 측정하여 전체 멱승 연산 시 키의 해밍 웨이트(Hamming weight)에 따른 전력 소모량을 측정하여 공격에 이용할 수 있다[14].

마찬가지로 타원곡선 암호시스템에서 스칼라 곱셈이 이진 연산 방법에 의해서 계산된다면 단순 전력 분석법인 SPA가 적용 가능하다[2].

알고리즘 1에서 비밀 키 k_i = 1인 경우 두 배와 덧셈 연산이 차례로 수행된다. 반면에 k_i = 0일 경우 두 배 연산만이 일어난다. 이로부터 공격자는 덧셈과 두 배 연산의 전력 소모량을 추출할 수 있으며, 전체 연산 수행 시 발생하는 전력 소모량을 분석함으로써 비밀 키 k에 따른 연산을 구분 할 수 있게 된다. 이를 통해 공격자는 비밀 키에 대한 정보를 재구성할 수 있게 된다. 따라서 SPA에 대한 취약점은 비밀 키 또는 중간 계산 값에 의존하는 조건 분기 문의 수행 여부 때문에 발생한다.

이에 대한 대응 방법의 하나는 최적화된 구현으로 소비전력 신호를 감소시키는 것이다. 소비전력 신호를 감소시키는 방법에 한계가 있을 경우, 개발자는 구현 단계에서 공격자가 중요한 정보를 산출하지 못하도록 잡음를 첨가할 수도 있다. 또한 암호문과 무관하게 명령어의 순서를 바꾸는 방법이나 소비 전력이 비슷한 명령어들을 이용한 구현방법으로도 방지가 가능하다[10].

다른 방안으로는 암호 연산의 비밀 키에 대한 의존성을 제거하는 방법이다. 예를 들면 알고리즘 1에서 덧셈과 두 배의 연산 양이 동일하도록 스칼라 곱셈 알고리즘을 설계하여 의존성을 제거할 수 있다.

본 논문에서는 알고리즘 측면에서의 대응 방안을 모색하였다. SPA를 고려한 알고리즘의 연구는 크게 Double-and-Add always 방법과 덧셈과 두 배 연산이 통합된 공식을 이용한 연구가 진행되었다. Double-and-Add always 방법에 대한 기존 연구 결과에 대해 간단히 살펴보도록 하겠다.

1. Double-and-Add-always 방법

Double-and-Add-always 방법은 조건 분기 문에서 비밀 키의 값이 0일 때에도 덧셈 연산을 무의미 연산(dummy operation)으로 추가하여 비밀 키에 따른 연산의 종속성을 없애 SPA를 막는 방법이다.

추가된 연산을 바탕으로 알고리즘 2[2], 알고리즘 3, 알고리즘 4[4] 및 알고리즘 6[15]을 아래와 같이 설계할 수 있으며, Montgomery-ladder을 이용한 곱셈 방식[13]을 설명할 수 있다. 편의상 타원곡선 점의 덧셈은 ECADD로, 두 배 연산은 ECDBL로 표기하도록 하겠다.

가. 이진 Double-and-Add-always 방법

알고리즘 2와 알고리즘3은 비밀 키에 상관없이 덧셈 연산 및 두 배 연산을 루프마다 동일하게 함으로써 연산량의 차이를 없앴다. 따라서 키 비트의 정보와 상관없이 전력 소모량을 균일하게 분포시킬 수 있기 때문에 SPA에 대응할 수 있게 된다. 하지만 알고리즘 2에서는 단계 2.1과 2.2에서 같은 변수 Q[0]를 사용함으로 두 연산 사이에 종속성이 존재한다.

[알고리즘 2] double-and-add-always 이진 left-to-right 알고리즘

input:점 P, m 비트 정수 $k=(k_{m-1}k_{m-2}\ldots k_0)_2$, $k_{m-1}, k_{m-2}, \ldots, k_0 \in \{0,1\}$

output: Q = [k]P

1. Q[0] ← P
2. for i = m-2 down to 0
   2.1 Q[0] ← ECDBL(Q[0])
   2.2 Q[1] ← ECADD(Q[0], P)
   2.3 Q[0] ← Q[k_i]
3. Return(Q[0])

따라서 두 배 연산 중 Q[0]의 x좌표에 대한 연산이

종료되어야 덧셈 연산을 위한 역원 연산이 수행될 수 있다. 하지만 알고리즘 3은 알고리즘 2가 가지고 있는 변수의 종속성이 없어 덧셈에서 역원 계산을 끝내고 나면 바로 두 배에서 역원 연산을 수행할 수 있는 장점이 있다. 그러나 이도 다음 루프 때 두 배에 사용되는 Q[0]가 덧셈에 사용되므로 데이터의 종속성이 존재한다.

[알고리즘 3] double-and-add-always 이진 right-to-left 알고리즘

input: 점 P, m 비트 정수 $k=(k_{m-1}k_{m-2}\ldots k_0)_2$, $k_{m-1}, k_{m-2}, \ldots, k_0 \in \{0,1\}$
output: Q = [k]P

1. Q[0] ← P, Q[1] ← O
2. for i = 0 to m - 1
   2.1 Q[2] ← ECADD(Q[0], Q[1])
   2.2 Q[0] ← ECDBL(Q[0])
   2.3 Q[1] ← Q[1+k_{i-1}]
3. return (Q[1])

나. 부호가 있는 Double-and-Add-always 방법

기존 Addition-Subtraction 알고리즘은 이진 알고리즘과는 달리 비밀 키 k의 각 비트를 0, 1, -1로 표현하는 방법이다. 이 방법은 이진 알고리즘보다 덧셈 연산량을 평균 m/2에서 m/3으로 줄임으로써 연산의 효율을 높인다. 알고리즘 4는 Addition-Subtraction 알고리즘을 비밀 키에 의존하지 않고 덧셈 연산을 동일하게 수행함으로써 전력 소모량을 균일하게 하여 SPA를 방어한다. 그 대신 기존 알고리즘에 비하여 덧셈 연산 량이 2m/3+1로 증가하며, 알고리즘 2 및 3과 마찬가지로 덧셈 연산 시 변수 Q[0]에 대한 종속성이 발생한다. 또한 키에 대한 인코딩 과정이 추가되어야 하므로 부가적인 로직이 필요하다.

[알고리즘 4] 부호가 있는 이진 double-and-add- always left-to-right 알고리즘

input: 점 P, m+1 비트 정수 $k=(k_m k_{m-1}\ldots k_0)_2$, $k_m, k_{m-1}, \ldots, k_0 \in \{0,1,-1\}$
output: Q = [k]P

1. Q[0] ← P
2. P[0] ← P, P[1] ← P, P[-1] ← -P
3. for i = m-2 down to 0
   3.1 Q[0] ← ECDBL(Q[0])
   3.2 Q[1] ← ECADD(Q[0], P[k_i])
   3.3 Q[-1] ← Q[1]
   3.4 Q[0] ← Q[$|k_{i-1}|$]
4. return (Q[0])

다. Montgomery-ladder

[13]에서는 비표준 형식인 Montgomery 타원곡선

$$EM : By^2 = x^3 + Ax^2 + x \quad (6)$$

에 대한 타원곡선 스칼라 곱셈 방법을 제시하였다.

Okeya와 Sakurai는 Montgomery 형태 타원곡선에서 무작위화된 사영좌표 점을 사용하여 전력 분석 공격에 안전한 스칼라 곱셈 방법을 제시 하였다[18]. Okeya-Sakurai의 스칼라 곱셈 방법은 비밀 키의 한 비트 당 10.2번의 유한 체 곱셈을 필요로 하며, 이 방법은 단순 전력 분석법 SPA와 차분 전력 분석법 DPA에 대응하는 스칼라 곱셈 알고리즘 중 가장 빠른 것으로 알려져 있다. 그러나 Montgomery 형태 타원곡선은 타원곡선군의 원소의 개수가 4로 나누어져야 한다는 제약사항이 존재하며, 이는 표준곡선에 적용이 되지 않는 단점이 있다.

Izu와 Takagi는 Montgomery 형태 타원곡선에 적용되었던 덧셈 체인인 Montgomery-ladder를 일반 타원곡선 형태인 Weierstass 형태 타원곡선에 적용하였으며, 알고리즘 5와 같이 Weierstass 형태 타원곡선에서 y좌표를 계산하지 않고 타원곡선 위의 점에 대한 덧셈, 두 배 연산을 가능하게 하는 방법을 제안하였다[6].

[알고리즘 5] Montgomery-ladder

input: 점 P, m 비트 정수 $k=(k_{m-1}k_{m-2}\ldots k_0)_2$, $k_{m-1}, k_{m-2}, \ldots, k_0 \in \{0,1\}$
output: Q = [k]P

1. Q[0] ← P, Q[1] ← 2P
2. For i = m-2 down to 0
   2.1 Q[2] ← ECDBL(Q[k_i])
   2.2 Q[1] ← ECADD(Q[0], Q[1])
   2.3 Q[0] ← Q[2 - k_i]
   2.4 Q[1] ← Q[1 + k_i]
3. Return (Q[0])

라. 윈도우 기반 Double-and-ADD-always 방법

[15]에서 Möller는 SPA 공격을 방어하기 위해서 윈도우 방법 알고리즘을 변형한다. 이를 위해서 윈도우 크기를 w라 할 때, 비밀 키 $k=\sum_{i=0}^{n} k_i 2^{wi}$를 새로운 비밀 키 $d=\sum_{i=0}^{n(+1)} d_i 2^{wi}$로 변환 시킨다. 여기에서 $k_i \in \{0, 1, \ldots, 2^w-1\}$ 이고 $d_i \in \{-2^w, 1, \ldots, 2^w-1\}$ 이며, n(+1)의 의

미는 어떤 경우 한 워드가 더 생긴다는 것을 뜻한다. 이 방법은 키를 윈도우 단위로 적을 때 0이 없는 것으로 표현하는 방법이다.

알고리즘 6을 적용 시, 사전 계산 값에 대한 저장 공간과 키 부호화 과정의 추가로 인하여 부가적인 로직 소모가 이루어진다. 따라서 하드웨어 구현 시 더 많은 면적이 필요한 단점이 있다.

[알고리즘 6] 변형된 윈도우 방법

input: 점 P, $d = \sum_{i=0}^{n} d_i 2^{wi}$

output: Q = [d]P

[사전 계산 단계]

1. P[1] ← P
2. for i=2 to $2^w - 1$
   P[i] ← ECADD(P[i-1], P)
3. P[-2^w] ← -ECDBL(P[2^{w-1}])

[실제 계산 단계]

1. Q[0] ← P[d_l]
2. for i = n - 1 down to 0
   2.1 for j = 1 to w
   Q[0] ← ECDBL(Q[0])
   2.2 Q[0] ← ECADD(Q[0], P[d_i])
3. return (Q[0])

2. SPA에 강한 스칼라 곱셈 방법 제안

앞에서 설명했듯이 알고리즘 2, 3, 4를 이용한 SPA 대응 방안의 경우, 덧셈 연산과 두 배 연산 중 변수에 대한 종속성이 존재한다. 이는 유한체 원소에 대한 곱셈기와 역원기를 병렬로 사용할 때 연산 효율이 떨어지는 단점이 된다. Okeya와 Sakurai 방법은 타원곡선 군의 원소의 개수가 4의 배수가 되어야 한다는 제약 사항이 존재하기 때문에 표준 곡선에 적용할 수 없다. 윈도우 기반의 대응 알고리즘은 사전 계산 과정과 비밀키에 대한 인코딩이 필요하기 때문에 하드웨어 구현 시 면적이 커지게 되는 단점이 있다. 본 논문에서는 위에서 나열한 알고리즘의 단점을 보완하고 표준 곡선에 적용하기 적합한 SPA 대응 알고리즘을 알고리즘 7과 같이 제안하고자 한다.

제안된 알고리즘은 알고리즘 3을 기반으로 하고 있으며 각 루프마다 두 배 연산과 덧셈 연산이 동일하게 수행되기 때문에 비밀 키의 비트 정보에 따른 연산양의 차이가 발생하지 않는다. 따라서 SPA에 대응할 수 있게 된다.

[알고리즘 7] 제안 알고리즘

input: 점 P, m 비트 정수 k=$(k_{m-1}k_{m-2}\cdots k_0)_2$,
$k_{m-1}, k_{m-2}, \ldots, k_0 \in \{0,1\}$

output: Q = [k]P

1. Q[0] ← O, Q[1] ← O, R[0] ← O, R[1] ← P
2. For i=0 to m-1
   2.1 R[0] ← R[1]
   2.2 R[1] ← ECDBL(R[1])
   2.3 Q[1] ← ECADD(Q[0] + R[0])
   2.4 Q[0] ← Q[k_i]
3. return(Q[0])

Okeya와 Sakurai 방법의 스칼라 곱셈방법이 SPA와 DPA를 고려한 스칼라 곱셈 알고리즘 중 가장 빠르지만 특정 곡선에만 적용 가능하기 때문에 NIST[17]와 SECG[3]에서 제시한 표준 곡선에 적용할 수 없다. 하지만 본 논문에서 제시한 알고리즘은 모든 표준 곡선에 적용가능 하다.

또 다른 장점은 기존의 대응 알고리즘과는 달리 알고리즘 7의 단계 2.2와 단계 2.3 사이에 사용되는 변수의 종속성이 없다. 따라서 단계 2.2의 두 배 연산에 필요한 유한체 원소의 역원 연산이 끝난 후 단계 2.3의 덧셈 연산에 필요한 유한체 원소의 역원 연산을 바로 수행할 수 있으며, 다음 루프의 두 배 연산도 이전 루프의 덧셈 연산의 데이터(R[0])와 연관성이 없다. 따라서 단계 2.3의 역원 연산이 수행되는 동안 단계 2.2에서 필요한 곱셈, 제곱 연산을 수행함으로써 추가 연산으로 인해 지연되는 동작 성능의 저하를 최소화 한다.

표 3과 표 4는 본 논문에서 제시한 알고리즘과 알고리즘 3의 연산을 각각 정리한 표이다. 이때 역원 연산과 곱셈 연산은

$$\text{inversion cycle} \approx 4(\text{multiplication cycle}) \quad (7)$$

이라고 가정한다. 곱셈이 c 클록 사이클(clock cycle)이 소요된다면 역원은 4c 사이클이 필요하다. 곱셈기를 사용하여 제곱을 구한다면 덧셈과 두 배는 각각 3번의 곱셈과 1번의 역원이 필요하다. 이를 정리하면 표 2와 같다. 타원곡선 위의 점에 대한 연산 공식에는 유한체 원소의 덧셈이 포함되어 있으나 이는 각 비트의 배타적 논리합(XOR)으로 쉽게 구현되므로 효율성 분석은 유한체 원소의 곱셈과 역원 시간으로 계산한다.

알고리즘 3은 표 4에 의해 덧셈에서 3번의 곱셈과 두 배에서 한 번의 곱셈 시간이 없어지므로 효율성을 분석하면 다음과 같다.

표 2. 덧셈 및 두 배에 필요한 연산 싸이클
Table 2. Computation cycle for addition and doubling.

덧셈 싸이클	3c + 4c
두 배 싸이클	3c + 4c

표 3. 제안 알고리즘의 연산 표
Table 3. Arithmetic sequence of proposed algorithm7.

loop	cycle	Doubling $R[1] \leftarrow 2R[1]$	$R[1]$	Addition $Q[1] \leftarrow Q[0] + R[0]$	$Q[1]$	$R[0]$
$i=1$	1c	Inversion				
	2c					
	3c					
	4c					$R[0] \leftarrow R[1]$ $(R[0] = P)$
	5c	Multiplication		Inversion		
	6c	Squaring	new $R[1]_x$ of $R[1]$			
	7c	Multiplication	new $R[1]_y$ of $R[1]$			
	8c					
$i=2$	9c	Inversion		Multiplication		
	10c			Squaring	new $Q[1]_x$ of $Q[1]$	
	11c			Multiplication	new $Q[1]_y$ of $Q[1]$	
	12c					$R[0] \leftarrow R[1]$ $(R[0] = 2P)$
	13c	Multiplication		Inversion		
	14c	Squaring	new $Q[0]_x$ of $Q[0]$			
	15c	Multiplication	new $Q[0]_y$ of $Q[0]$			
	16c					
$i=3$	17c	Inversion		Multiplication		
	18c			Squaring	new $Q[1]_x$ of $Q[1]$	

알고리즘 3의 전체 연산 싸이클

$$= m(\text{덧셈 싸이클} - 3c) + m(\text{두 배 싸이클} - c)$$
$$= m(4c + 6c) \text{ 싸이클} \quad (8)$$

제안 알고리즘은 [표 4]에 의해 각 루프 단계마다 오직 2번만의 역원 계산 시간만 존재하며 마지막 루프 연산 시 3번의 곱셈이 필요하므로 효율성을 분석하면 다음과 같다.

제안 알고리즘의 전체 연산 싸이클

$$= m(24c) + 3c \text{ 싸이클} \quad (9)$$

따라서 알고리즘 3에 비해 본 논문에서 제시한 알고리즘은 2mc-3c 만큼의 연산량을 줄일 수 있다.

Ⅳ. 하드웨어 구조

SPA를 방지하는 제안 알고리즘은 타원곡선 스칼라 곱셈을 위해 최대 m번의 두 배 연산과, m번의 덧셈 연산이 필요하다. 두 배 연산 및 덧셈 연산은 1번의 유한체 원소의 역원 연산과 2번의 곱셈 연산, 1번의 제곱 연산이 필요하다. 본 논문은 면적의 최소화를 위해 제곱 연산을 곱셈기를 사용하는 구조로 채택하였다. 또한 제안 스칼라 곱셈기는 WTLS 5번, 3번 표준곡선을 지원하는 구조로 설계 되었다.

표 4. 알고리즘 3의 연산 표
Table 4. Arithmetic sequence of algorithm 3.

Iter-ation	cycle	Addition $Q[2] \leftarrow Q[1] + Q[0]$	$Q[2]$	Doubling $Q[0] \leftarrow 2Q[0]$	$Q[0]$
$i=0$	1c	Inversion			
	2c				
	3c				
	4c				
	5c	Multiplication		Inversion	
	6c	Squaring	new $Q[2]_x$ of $Q[2]$		
	7c	Multiplication	new $Q[2]_y$ of $Q[2]$		
	8c				
	9c			Multiplication	
	10c			Squaring	new $Q[0]_x$ of $Q[0]$
$i=1$	11c	Inversion		Multiplication	new $Q[0]_y$ of $Q[0]$
	12c				
	13c				
	14c				
	15c	Multiplication		Inversion	
	16c	Squaring	new $Q[2]_x$ of $Q[2]$		
	17c	Multiplication	new $Q[2]_y$ of $Q[2]$		
	18c				
	19c			Multiplication	

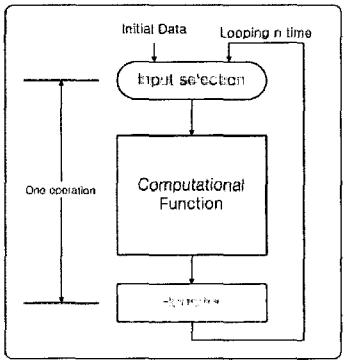

그림 1. 루프 구조
Fig. 1. Loop architecture.

스칼라 곱셈기의 연산 모듈로 사용되는 역원기, 곱셈기는 자원 효율성을 가지는 루프 아키텍쳐 설계 기법을 적용하여 구현하였다[16]. 즉 연산기는 하나의 연산 모듈을 이용하여 m번의 연산을 반복 수행함으로 전체 연산을 수행 한다. 이는 자원 활용 측면에서 적은 면적을 이용하여 기능 구현을 할 수 있기 때문에 적용하였으며, 자원 제약이 심한 임베디드 환경에 활용할 수 있도록 하였다.

1. 역원기 아키텍처(Inversion Architecture)

가. 역원 알고리즘

몽고메리 알고리즘에 따른 역원 계산은 두 단계로 나뉠 수 있다[21]. 첫 단계의 최종 결과 값은 입력 값 a(x)에 대하여 결과 값 $a(x)^{-1}x^k$을 산출하게 된다. $a(x)^{-1}x^k$ 값은 다음과 같은 합동식으로 쉽게 $a(x)^{-1}$ 값을 얻어낼 수 있다.

$$a(x)^* = a(x)^{-1}x^k + a_0p(x) \bmod p(x) = a(x)^{-1}x^k \quad (10)$$

여기에서 a_0는 $a(x)^{-1}x^k$의 상수항이다. 따라서 $a(x)^*$의 상수항은 0이 된다. 그러므로 다음과 같은 결과를 얻을 수 있다.

$$a(x)^{-1}x^{k-1} = a(x)^{-1}\frac{x^k}{x} = a(x)^*\frac{1}{x} \quad (11)$$

상기 결과 식은 단순히 비트 이동을 수행할 수 있는 하드웨어 로직으로 구현할 수 있으며 k번의 수행 시간 후 $a(x)^{-1}$의 결과 값을 산출할 수 있다.

[알고리즘 8]
input $a(x)^{-1}x^k$와 k
output $a(x)^{-1}$

1. $c(x) \leftarrow a(x)^{-1}x^k$
2. For i = k-1 to 0
   2.1 $c(x) \leftarrow (c(x) + c_0 p(x))/x$
3. return(c(x))

[21]에서 제시한 역원 알고리즘과 알고리즘 8을 이용하여 $a(x)^{-1}$을 구할 수 있지만, 시간 공격(Timing Atta-ck)에 취약해질 수 있다. 왜냐하면 [21]에서 제시한 역원 알고리즘은 k의 값이

$$\deg(a(x)) \le k \le \deg(p(x))+\deg(a(x))+1 \quad (12)$$

의 범위를 가지며, 이는 입력 데이터 a(x)에 따라 알고리즘의 루프 회수가 달라짐을 뜻한다. 따라서 입력 데이터에 따라 연산 수행 시간이 달라져 시간 공격(Timing attack)에 취약해질 수 있다.

본 논문에서는 기존의 몽고메리 역원 알고리즘에 알고리즘 9의 단계 3.2를 추가하여 m 값이 하나씩 줄어드는 while 문 수행 때마다 x의 값이 한번씩 곱해지게 하여 시간 공격에 대응하고자 하였다. 즉 모든 입력에 대하여 동일한 연산 수행 시간이 소요된 후 $a(x)^{-1}x^{2m}$의 결과 값이 산출되도록 역원 알고리즘을 변형 하였다. 따라서 알고리즘 8은 $k=2^m$으로 동작한다. 그러므로 알고리즘 9를 수행하는데 2m 클록 그리고 알고리즘 8을 수행하는 데 2m 클록해서 역원 계산에 필요한 총 시간은 4m 클록만큼 시간이 소요되게 되며, 시간공격에 강한 구조를 가지게 된다.

[알고리즘 9] 시간공격에 강한 변형된 몽고메리 역원 알고리즘
input: a(x), p(x) where $\deg(a(x)) \le \deg(p(x))$
output: $a(x)^{-1}x^{2m}$

1. $u(x) \leftarrow p(x)$, $v(x) \leftarrow a(x)$, $r(x) \leftarrow 0$, $s(x) \leftarrow 1$
2. $k \leftarrow 0$
3. while($k < 2m$)
   3.1 if($v(x) \neq 0$)
      3.1.1 if($u_0=0$) $u(x) \leftarrow u(x)/x$, $s(x) \leftarrow x \cdot s(x)$
      3.1.2 elsif ($v_0=0$) $v(x) \leftarrow v(x)/x$, $r(x) \leftarrow x \cdot r(x)$
      3.1.3 elsif ($\deg(u(x) > \deg(v(x))$) then
            $u(x) \leftarrow (u(x)+v(x))/x$
            $r(x) \leftarrow r(x)+s(x)$
            $s(x) \leftarrow x \cdot s(x)$
      3.1.4 else $v(x) \leftarrow (v(x) + u(x))/x$
            $s(x) \leftarrow s(x)+r(x)$
            $r(x) \leftarrow x \cdot s(x)$
   3.2 else
      3.2.1 if($r_m=1$) $r(x) \leftarrow r(x)+p(x)$
      3.2.2 else $r(x) \leftarrow x \cdot r(x)$
   3.3 $k \leftarrow k+1$
4. if($\deg(r(x)) = \deg(p(x))$) $r(x) \leftarrow r(x)+p(x)$
5. return($r(x) = a(x)^{-1}x^{2m}$)

나. 설계 및 동작 기능

역원기의 동작은 WTLS 3번 곡선 또는 WTLS 5번 곡선에 한정하여 설명하겠다. 역원기의 동작은 start 신호의 활성화(1)로 시작되며, s, r, degU 레지스터는 각각 000...001(164 bit), 000...000(164 bit), 100...000(164 bit)로 초기값이 설정된다. 다음 클록에 U 레지스터와 V 레지스터에 각각 법(modulus)과 역원을 구하고자 하는 원소를 저장한다. 연산 종료와 함께 done 신호가 활성화 되며, 결과 값의 출력이 이루어지며 상위 모듈에서 결과 값을 읽어가게 된다. 그림 2의 회로도는 2m 클록 싸이클 동안은 몽고메리 역원 계산에 필요한 동작이 수행되며, 그 후 2m 싸이클 동안은 몽고메리 잉여(residue) x^{2m}을 제거하기 위한 연산이 수행된다. 즉 알고리즘 8과 알고리즘 9의 연산을 같은 회로 상에서 수행할 수 있는 아키텍처로 설계하였다. WTLS 3번 곡선 또는 WTLS 5번 곡선에 대해 연산 수행 시간은 총 652 클록이 소요된다.

그림 2의 Zero Check Logic은 알고리즘 9의 단계 3.1을 수행하는 로직으로 V 레지스터의 값이 0인 가를

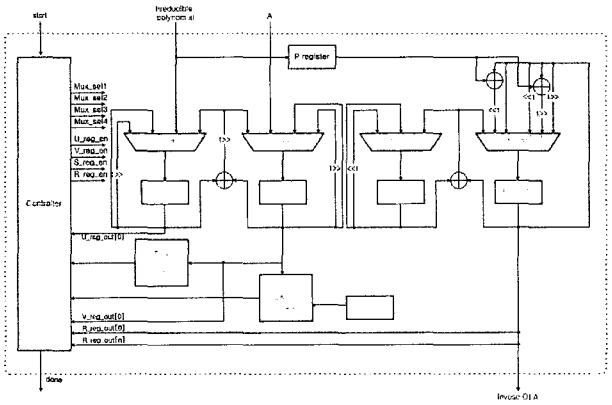

그림 2. 역원기 설계 블록
Fig. 2. Design block of inverter.

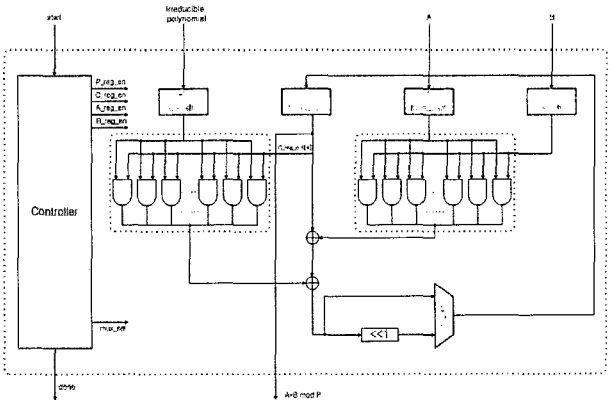

그림 3. 곱셈기 설계 블록
Fig. 3. Design block of multiplier.

확인하는 로직이다. Degree Check Logic은 알고리즘 9의 단계 3.1.3을 수행하는 로직이다. 그림 2의 제어부는 Zero Check Logic의 출력값과 Degree Check Logic의 출력값, U 레지스터의 최하위 값 그리고 V 레지스터의 최하위 값으로 알고리즘 9의 분기 조건을 판단한다. 이 조건 판단에 따라 MUX의 입력 신호들이 결정되며, MUX 선택 신호에 의해 선택된 입력 값들은 레지스터에 저장되게 된다. MUX의 입력들은 역원 연산을 위한 덧셈 연산 및 나눗셈 연산, 그리고 곱셈 연산이 모두 수행된 결과 값이다. 유한 체 덧셈은 비트 단위의 XOR 연산, 곱셈 연산 및 나눗셈 연산은 좌우 비트 이동으로 수행된다.

2. 곱셈기 아키텍처(Multiplier Architecture)

가. 곱셈 알고리즘

곱셈 알고리즘 역시 입력 데이터의 해밍웨이트에 상관없이 일정한 연산 수행 시간이 소요되는 'shift-and-add'[20] 알고리즘을 채택하여 이를 구현에 사용하였다.

[알고리즘 10] shift-and-add 곱셈 알고리즘
input: a(x), b(x), p(x)
output: a(x)b(x) mod p(x)
1. $c(x) \leftarrow 0$
2. For i = m-1 to 0
   2.1 $c(x) \leftarrow c(x) \cdot x$
   2.2 $c(x) \leftarrow c(x) + c_m \cdot p(x) + b_i a(x)$
3. return(c(x))

나. 설계 및 동작 기능

곱셈기의 동작은 start 신호의 활성화로 시작되며 다음 클록에 외부 입력 값을 모듈 내부의 데이터 레지스터 A, B, P에 병렬로 저장하며 결과 값을 저장하는 레지스터 C는 초기 값 0으로 설정된다. 단 B 입력 데이터는 클록 당 1 비트(LSB)씩 출력하는 쉬프트 레지스터로 구현하였다.

P 레지스터의 출력 값과 중간 계산 결과 값 및 최종 연산 결과 값을 저장하는 C 레지스터의 최상위 한 비트 출력을 입력으로 가지는 AND 게이트 모듈은 C에 저장된 값이 P에 저장된 기약다항식보다 큰 최고차항이 존재할 경우, P에 저장된 값으로 나눗셈 연산을 수행하기 위한 로직이다. 또한 A 레지스터의 출력과 B레지스터의 출력을 입력으로 하는 AND 모듈은 유한체 곱셈 연산을 수행한다. 즉 B값의 i번째 비트와 전체 A 값의 곱 $A*B_i$를 계산하게 된다. 두 개의 AND 게이트 모듈에서 출력되는 결과는 첫 번째 XOR 게이트를 통해 곱해진 값과 중간 계산 값(C)과의 유한 체 덧셈 연산이 수행되며, 두 번째 XOR 게이트를 통해 기약다항식으로 나누는 연산이 수행된다. 결과 값은 중간 연산 결과 값을 저장하는 C 레지스터에 좌측으로 한 비트 이동 시킨 후 저장되며, 곱셈 연산 수행이 종료되면 done 신호가 활성화되며 상위 모듈에서 결과 값을 저장하게 된다.

3. 스칼라 곱셈기 아키텍쳐

역원기 모듈, 곱셈기 모듈, 덧셈 및 두 배 연산 모듈로 구성되는 스칼라 곱셈기는 비밀키 k를 이용하여 kP를 계산하는 것이다. 본 논문에서 제시한 스칼라 곱셈 연산기는 역원 연산 수행 시간이 곱셈 연산 수행 시간보다 긴 점을 이용하여 연산의 효율을 높이고자 하였다. 두 배 연산에 필요한 역원 연산 수행이 종료된 후, 역원기는 덧셈 연산에 필요한 역원 연산을 바로 수행하며, 이 시간 동안 두 배 연산은 곱셈과 멱승 연산을 수행할 수 있다. 따라서 단순 전력 분석을 방지하는 알고리즘 적용으로 인한 성능 저하를 최소화 하고자 하였

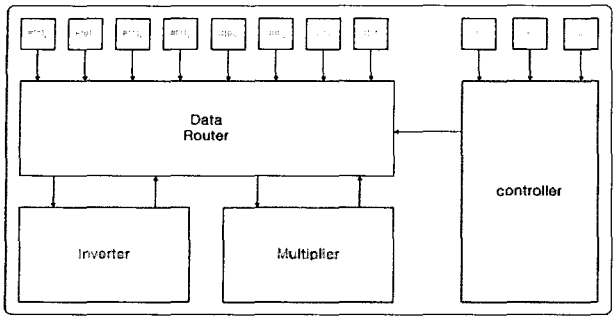

그림 4. 스칼라 곱셈기
Fig. 4. Scalar multiplier.

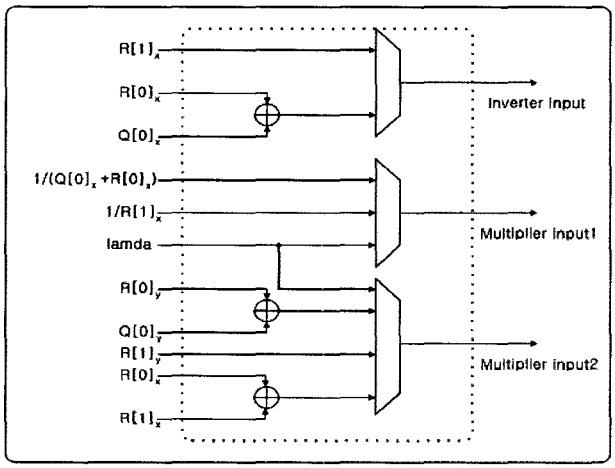

그림 5. 역원기, 곱셈기 입력 라우터
Fig. 5. Input router of inverter and multiplier.

다. 그림 4는 제안하는 알고리즘을 수행하도록 설계된 스칼라 곱셈기의 전체 구조이며, 효율성 증대를 위해 역원기와 곱셈기를 병렬로 운용하는 구조이다.

스칼라 곱셈기의 제어부는 그림 5와 같이 구성된다. 제어부는 각 처리 상태에 따라 데이터 라우터의 동작 제어를 위한 오피코드(opcode) 생성 및 역원기, 곱셈기 제어 신호를 생성한다. 오피코드(opcode)에 따른 데이터 라우터의 동작 기능을 정리하면 표 5와 같다.

표 5. 오피 코드에 따른 데이터 라우터의 동작 기능
Table 5. Functional operation of data router according to the opcode.

opcode	동작 기능
00000	Register file reset
00001	Load P to R[1]
00010	Load R[1] to R[0]
00011	Start inverter for Doubling($1/x_0$)
00100	Load inverse result(Doubling)
00101	Start inverter for Addition ($1/(x_0+x_1)$)
00110	Start multiplier for y_0/x_0 (Doubling)
00111	Load λ(Doubling)
01000	Start multiplier for λ^2(Doubling)
01001	Load λ^2(Doubling)
01010	Start multiplier for $\lambda(x_0+x_1)$ (Doubling)
01011	Load $\lambda(x_0+x_1)$ (Doubling)
01100	Load 2*R[1] to R[1]
01101	Load R[0] to Q[1]
01110	Load inverse result(Addition)
01111	Start multiplier for λ(Addition)
10000	Load λ(Addition)
10001	Start for multiplier for λ^2(Addition)
10010	Load λ^2(Addition)
10011	Start multiplier for $\lambda(x_1+x_2)$(Addition)
10100	Load $\lambda(x_1+x_2)$(Addition)
10101	Load R[0] to Q[1]
10110	Load Q[0] to Q[0]
10111	Load Q[1] to Q[0]
11111	Idle State

V. 구현 결과 및 결론

1. 테스트 검증 및 구현 결과

하드웨어 합성은 Synplicity사의 Synplify Pro 7.0을 이용하였으며, Xillinx사의 Foundation 3.1i 툴을 통해 Place & Route, Functional 및 Timing 시뮬레이션을 수행하였다. 테스트 벡터는 WTLS 5번, 3번 곡선의 도메인 파라메터 및 k값을 사용하였다. 아키텍쳐 기능 테스트 및 디버깅을 위해 테스트 벡터 생성기를 C로 코딩하며 결과값 및 중간 연산값을 시뮬레이션 테스트의 검증값으로 활용하였다(그림 6 참조). 본 구현에 사용된 테스트 벡터를 정리하면 다음과 같다.

[곡선 형태: WTLS 5]
유한체 크기: 163
기약다항식: $x^{163}+x^8+x^2+x+1$
타원곡선 E : $y^2 + xy = x^3 + ax^2 + b$
파라미터 a:72546B5435234A422E0789675F432C89435DE5242
P_x: AF69989546103D79329FCC3D74880F33BBE803CB
P_y: 1EC23211B5966ADEA1D3F87F7EA5848AEF0B7CA9F
K: 03a41434aa99c2ef40c8495b2ed9739cb2155a1e0d
(단, 파라미터 값은 16진수 표현임)

그림 7과 그림 8은 곱셈기와 역원기의 Functional 시뮬레이션 결과를 나타내고 있다. reset 신호에 따라 내부 레지스터 파일의 초기화가 이루어지며 start 신호 입력으로 연산이 시작된다. 연산 종료 시, done 신호가 활성화 되며 상위 모듈에서 결과 값을 읽어 가게 된다.

그림 9는 스칼라 곱셈기의 Timing 시뮬레이션 결과를 나타내고 있으며 정확히 동작함을 검증하였다.

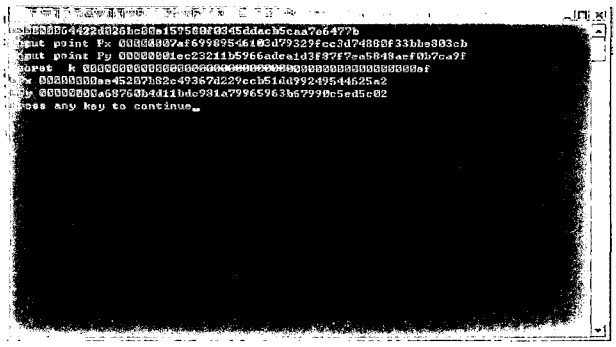

그림 6. 테스트 벡터 생성기
Fig. 6. Test vector generator.

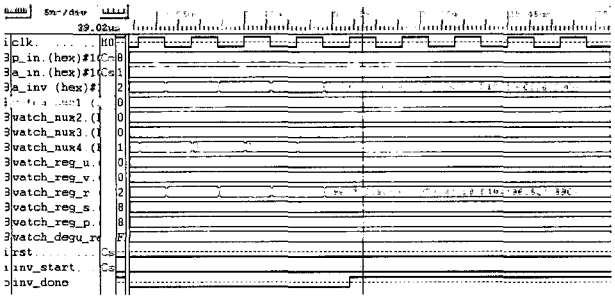

그림 7. 역원기 기능 시뮬레이션
Fig. 7. Functional simulation of inverter.

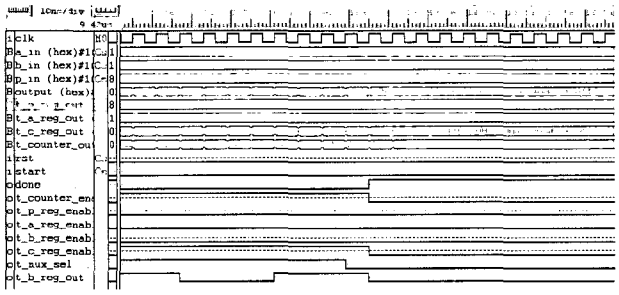

그림 8. 곱셈기 기능 시뮬레이션
Fig. 8. Functional simulation of multiplier.

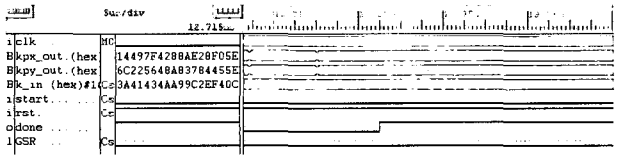

그림 9. 스칼라 곱셈기 타이밍 시뮬레이션
Fig. 9. Timing simultation of scalar multiplier.

표 6. XCV2000EFG1156 하의 타이밍 시뮬레이션 후 동작 성능
Table 6. Performance in timing simulation under XCV2000 EFG1156.

최대 동작 주파수	30.777MHz
전체 등가 게이트	60,608
슬라이스 활용 율	16%

본 구현 결과는 Xillinx virtexE XCV2000EFG1156을 대상으로 하였으며 타이밍 시뮬레이션 결과 얻은 추정 동작 성능은 표 6와 같다.

2. 결 론

본 논문은 하드웨어로 구현된 암호 프리미티브의 안전성을 위협할 수 있는 부채널 공격 방법 중의 하나인 단순 전력 분석인 SPA를 방어하면서 기존에 제시된 알고리즘 보다 효율적으로 연산을 수행할 수 있는 알고리즘을 제안 하였으며 이를 하드웨어로 구현하여 성능을 검증하였다.

제시한 알고리즘은 데이터의 종속성을 없앰으로써 무의미 연산(dummy operation)으로 발생하는 연산상의 속도 저하를 최소화 하였다. 구현 시 이를 고려하여 역원 처리 시간 동안 곱셈 및 제곱 연산을 처리할 수 있는 병렬 처리 기법을 적용하여 가장 효율적인 성능을 낼 수 있도록 하였다.

루프 구조 및 시리얼 구조를 바탕으로 역원기 및 곱셈기를 구현함으로써 필요한 면적의 크기를 가능한 줄였으며, 역원 연산 알고리즘을 시간 공격에 강한 구조로 변형하여 이를 구현에 적용하였다.

스칼라 곱셈기의 성능은 전체 등가 게이트 약 6만 게이트에 최대 동작 주파수는 30Mhz로 산출되었다. 이는 스마트카드나 무선 통신 장비 등 Embedded-Micom 환경에 적용 가능하며, 전자 서명(Digital Signature), 키 교환(Key Exchange), 암,복호화 등의 응용에 활용될 수 있을 것으로 보인다.

참 고 문 헌

[1] I.F.Blake, G.Seroussi, N.P Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[2] J.S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems", *CHES 1999*, LNCS 1717, pp.292-302, Springer -Verlag, 1999.

[3] Standards for efficient cryptography-SEC2: Reco -mmended elliptic curve cryptography domain parameters, 2000. Available from http://www.secg.org

[4] J.C. Ha, S.J. Moon, "Randomized Sigend-Scalar Multiplication of ECC to Resist Power Attacks", *CHES 2002*, LNCS 2523, pp.551-563, 2002.

[5] *IEEE standard specifications for public-key cryptography*, IEEE Std 1363-2000, 2000.

[6] T. Izu, T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel

Attack", *PKC 2002*, LNCS 2274, pp.280-296, 2002.
[7] D.E. Knuth. *The Art of Computer Programming 2-Semi-numerical Algorithms*, Addison-Wesley, 1981.
[8] N. Koblitz, "Elliptic curve cryptosystems", *Mathe -matics of Computation*, number 48, pp. 203-209, 1987.
[9] P. Kocher, "Timing attacks on implementation of Diffie-Hellman, RSA, DSS and other systems", *CRYPTO'96*, LNCS 1109, pp.104-113, 1996.
[10] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", *CRYPTO'99*, LNCS 1666, pp.388-397, 1999.
[11] J. Lopez and R. Dahab. "Fast multiplication on elliptic curves over GF(2m) without precomputa -tion", *CHES'99*, pp. 316-327, LNCS 1717, 1999.
[12] V.S. Miller, "Use of elliptic curve in cryptogra -phy", *CRYPTO'85*, LNCS 218, pp.417-426, 1986.
[13] P.L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of factorizations", *Math. Comp.* 48, pp.243-264, 1987.
[14] T.S, Messerges, E A. Dabbish, R. H. Sloan, "Power Analysis Attacks of Modular Exponentia -tion in Smartcards", *CHES'99*, LNCS 1717, p144-157, 1999.
[15] B.. Möller, "Securing Elliptic Curve Point Multip -lication against Side-Channel Attacks", *ISC 2001*, LNCS 2200, pp.324-334, 2001.
[16] *Mitsubishi Electronic Advance*, Cryptography Edition(vol 100), December 2002.
[17] *Digital Signature Standard(DSS)*, FIPS PUB 186-2
[18] K. Okeya, K. Sakurai, "Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack", *INDOCRYPT 2000*, LNCS 1997, pp.178-190, 2000.
[19] D. B. Richard, A. Demillo, R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", *EUROCRYPT'97*, LNCS 1233, pp.37 -51, 2002.
[20] R. Schroeppel, H. Orman, S. O'Malley, *Fast Key Exchange with Elliptic Curve Systems*, TR-95 -03, University of Arizona, Comp. Sciences De part., 1995.
[21] E. Savas and C. K. Koc. "Architecture for unified filed inversion with applications in elliptic curve cryptography", *ICECS 2002*, pp. 1155-1158, IEEE, 2002.

저 자 소 개

김 현 익(학생회원)
2002년 2월 고려대학교 전자공학과 학사 졸업.
2004년 8월 고려대학교 정보보호 대학원 석사 졸업.
<주관심분야: 암호칩 설계, 스마트카드 보안>

정 석 원(정회원)
1991년 고려대학교 수학과 학사
1993년 고려대학교 수학과 석사
1997년 고려대학교 수학과 박사
2002년 3월~2004년 2월 고려대학교 정보보호대학원 연구조교수
2004년 3월~현재 목포대학교 정보보호전공 전임강사
<주관심분야: 공개키 알고리즘, 암호칩 설계, 스마트카드 보안, 디지털방송보안>

윤 중 철(정회원)
1993년 고려대학교 수학과 학사
1995년 고려대학교 수학과 석사
1999년 고려대학교 수학과 박사
2003년~현재 삼성전자 System LSI 사업부 책임연구원
<주관심분야: 공개키 알고리즘, 암호칩 설계, 스마트카드 보안>