情報保護學會論文誌
第12券 第4號, 2002. 8

부가채널 공격에 안전한 효율적인 타원곡선 상수배 알고리즘

임 채 훈*

Improved Elliptic Scalar Multiplication Algorithms Secure Against Side-Channel Attacks

Chae Hoon Lim*

요 약

실행시간이나 전력 소모량 분석 등과 같은 부가채널 공격에 안전하면서도 보다 효율적인 새로운 타원곡선 상수배 알고리즘을 제안한다. 먼저 기존 방식들을 분석하여 잠재적인 문제점들을 지적하고, 이런 문제점들을 제거할 수 있는 간단한 ±1-이진 부호화 방식을 제안한다. 또한 제안 부호화 방식을 이용한 이진 및 고정 윈도우 알고리즘을 기술하고 안전성과 효율성을 분석한다.

ABSTRACT

Improved algorithms for elliptic scalar multiplication secure against side-channel attacks, such as timing and power analysis, are presented and analyzed. We first point out some potential security flaws often overlooked in most previous algorithms and then present a simple ±1-signed encoding scheme that can be used to enhance the security and performance of existing algorithms. More specifically, we propose concrete signed binary and window algorithms based on the proposed ±1-signed encoding and analyze their security and performance. The proposed algorithms are shown to be more robust and efficient than previous algorithms.

Keyword : *Side-channel attacks, Simple/differential power analysis, Elliptic curve cryptosystems*

Ⅰ. 서 론

타원곡선 암호는 타원곡선상의 연산에서 정의되는 이산대수문제의 어려움을 이용하는 암호시스템으로 기존 공개키 암호에 비해 더 짧은 키 길이로도 유사한 정도의 안전도를 얻을 수 있다는 장점으로 인해 최근 들어 각광 받기 시작한 공개키 암호시스템의 한 부류이다. 특히 타원곡선 암호시스템의 안전도는 키 길이의 증가에 따라 거의 지수 함수적으로 증가하므로 준지수 함수적인 증가를 갖는 기존의 공개키 암호시스템에 비해 장기적으로 기술의 발전에 따른 키 길이의 증가 비율 면에서도 뛰어나 장점을 가지고 있다. 타원곡선 암호는 이러한 장점들로 인해 스마트 카드나 무선 통신 단말기 등과 같이 메모리와 처리능력이 제한된 응용분야에서 특히 효율적으로 사용될 수 있다.

다른 공개키 암호에서와 마찬가지로 타원곡선 암호 분야에서도 구현 성능을 향상시키기 위한 보다 효율적인 연산 알고리즘에 대한 연구가 학계의 주요 연구분야의 하나로 활발히 수행되어 왔다. 그러나 최

* 세종대학교 인터넷학과(chlim@sejong.ac.kr)

근 들어 비밀키 값에 의존하는 알고리즘의 실행시간이나 전력 소모량의 차이, 전자파 방출량 등을 관찰/분석하여 훨씬 쉽게 키 값을 유도해 낼 수 있는 부가채널 공격기법(side channel attacks)들이 개발되어 실질적인 위협으로 부각됨에 따라, 이러한 부가채널 공격에 보다 안전한 연산 알고리즘의 개발이 또한 주요 이슈로 부각되고 있다.[1~8]

전력 소모량 분석(power analysis)에 의한 공격은 특히 스마트카드나 하드웨어 구현 등과 같이 물리적으로 안전하게 설계된 장치에 대해서도 이에 대한 대책을 구현하지 않은 경우 간단히 내장된 비밀키를 유도해 낼 수 있는 매우 강력하고 실질적인 공격방법이다. 전력 소모량 분석은 단 한번의 알고리즘 수행에 대한 전력 소모량 관찰을 통해 키 값을 유도해 내는 단순 전력 소모량 분석(SPA : simple power analysis)과, 같은 비밀키에 대해 여러 번의 알고리즘 수행에서 걸쳐 수집된 전력 소모량 측정치들 사이의 상관관계를 통계적으로 분석하여 키 값을 유도해 내는 보다 강력한 차분 전력 소모량 분석(DPA : differential power analysis)로 나누어진다.

SPA는 단 한번의 알고리즘 수행에 대한 관찰만으로도 관련 키 값을 유도해 낼 수 있으므로 비밀키에 의존하는 거의 모든 공개키 암호 연산(모듈러 멱승, 타원곡선 상수배)에 적용될 수 있다. 따라서 SPA에 취약한 환경에서의 구현 시에는 효율성을 희생하고라도 SPA에 안전한 연산 알고리즘을 사용하는 것이 필수적이다. SPA에 안전하기 위해서는 통상 알고리즘의 전 수행기간 동안 키 값에 무관하게 동일한 순서로 동일한 양의 연산이 일어나도록 해야 한다.

한편 DPA는 고정된 비밀키가 개입된 암호 연산(RSA의 복호화/서명 과정, static-static mode나 ephemeral-static mode의 DH/ECDH에서 공유키 계산 과정 등)의 서로 다른 데이터에 대한 다수의 수행과정에서 비밀키의 특정 비트 값에 의존하는 중간 계산 값과 해당 전력 소모량 사이의 상관관계를 통계적으로 분석하는 공격 방법이다. 따라서 이를 막기 위해서는 중간 계산 값을 예측할 수 없도록 매 수행시 마다 내부적으로 키나 데이터를 랜덤화시켜 알고리즘을 수행하도록 하는 것이 필수적이다. 타원곡선 암호의 경우 매 수행시 마다 비밀키나 타원곡선 점, 혹은 타원곡선 방정식 자체를 내부적으로 랜덤하게 바꾸어 연산을 수행함으로써 대부분의 DPA 공격을 막을 수 있고, 이 과정은 계산하고자 하는 타원곡선 상수배에 비해 그렇게 많은 계산량이 필요치 않으므로 SPA에 안전한 연산 알고리즘은 쉽게 DPA에 안전한 알고리즘으로 바꿀 수 있다.[3,6]

본 논문에서는 먼저 기존의 SPA에 안전한 타원곡선 상수배 알고리즘들을 분석하여 지나치기 쉬운 잠재적인 문제점들을 지적하고, 이런 문제점들을 제거할 수 있는 간단한 ±1-이진 부호화 방식을 제안한다. 또한 제안된 이진 부호화 방식을 이용한 보다 안전하고 효율적인 이진 및 윈도우 알고리즘을 제안하고 그 안전성 및 효율성 분석을 다룬다.

II. 타원곡선 연산

2.1 GF(p)상의 타원곡선 연산

유한체 GF(p)(p>3인 소수)상의 타원곡선은 다음과 같은 Weierstrass 방정식을 만족하는 GF(p)상의 모든 점 (x, y)와 가상의 무한원점(point at infinity) O로 구성된다:

$$E: y^2 = x^3 + ax + b,\ a, b \in \mathrm{GF}(p),\ 4a^3 + 27b^2 \neq 0$$

타원곡선 덧셈군은 무한원점 O를 항등원으로 하여 다음과 같은 덧셈규칙으로 생성된다. 타원곡선상의 점 P_i를 $P_i = (x_i, y_i)$라 두자.

- 역원 : $P = (x, y)$에 대해 $-P = (x, -y)$
- 덧셈(Addition): 두 점 P_1, P_2 $(P_1 \neq \pm P_2)$의 합 $P_1 + P_2 = P_3$ 는 다음과 같이 계산된다:

$$x_3 = \lambda^2 - (x_1 + x_2),$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{where } \lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

- 두 배(doubling) : 점 P_1을 그 자신과 더하는 두 배 연산 $2P_1 = P_3$는 다음과 같이 계산된다:

$$x_3 = \lambda^2 - 2x_1,$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{where } \lambda = \frac{3x_1^2 + a}{2y_1}$$

위의 아핀 좌표계(affine coordinates)에서의 타원곡선 덧셈은 유한체상의 곱셈에 대한 역원의 계

산을 필요로 하는데, 이는 곱셈에 비해 상당히 많은 계산량을 필요로 하므로 곱셈을 좀 더 많이 사용하더라도 역원 계산이 필요없는 사영 좌표계(projective coordinates)가 흔히 사용된다. GF(p)상의 타원곡선에 대해 가장 효율적인 사영 좌표계는 아핀 좌표계와 다음과 같은 동치관계에 따라 변형된 방정식을 사용한다[9]:

$$(x,y)\sim(X,Y,Z) \Leftrightarrow x=\frac{X}{Z^2}, y=\frac{Y}{Z^3},$$
$$E': Y^2=X^3+aXZ^4+bZ^6,\ a,b\in \mathrm{GF}(p)$$

따라서 아핀 좌표계와 사영 좌표계 사이의 점들간의 변환은 다음과 같이 이루어진다(통상 효율성을 위해 γ는 1을 취함):

$$(x,y)\mapsto (\gamma^2x,\gamma^3y,\gamma) \text{ for some } \gamma\in \mathrm{GF(p)},$$
$$(X,Y,Z)\mapsto \left(\frac{X}{Z^2},\frac{Y}{Z^3}\right).$$

사영 좌표계의 타원곡선은 방정식 E'을 만족하는 $(0,0,0)$이 아닌 GF(p)상의 모든 점 (X,Y,Z)들과 무한원점 $(1,1,0)$으로 이루어진다. 사영 좌표계에서의 덧셈규칙은 다음과 같다. 타원곡선상의 사영 좌표계 점 P_i를 $P_i=(X_i,Y_i,Z_i)$라 두자.

- 역원 : $P=(X,Y,Z)$에 대해 $-P=(X,-Y,Z)$
- 덧셈(Addition): 두 점 P_1, $P_2(P_1\neq\pm P_2)$의 합 $P_1+P_2=P_3$는 다음과 같이 계산된다:

$$A=X_1Z_2^2+X_2Z_1^2,\ Z_3=BZ_1Z_2,$$
$$B=X_1Z_2^2-X_2Z_1^2,\ X_3=D^2-AB^2,$$
$$C=Y_1Z_2^3+Y_2Z_1^3,\ Y_3=[D(AB^2-2X_3)-B^3C]/2.$$
$$D=Y_1Z_2^3-Y_2Z_1^3,$$

만일 $Z_2=1$이면, 즉 P_2가 아핀 좌표계의 점이라면 $P_2=(x_2,y_2,1)$이라 둘 때 위의 계산은 다음과 같이 단순화된다:

$$A=X_1+x_2Z_1^2,\quad Z_3=BZ_1,$$
$$B=X_1-x_2Z_1^2,\quad X_3=D^2-AB^2,$$
$$C=Y_1+y_2Z_1^3,\quad Y_3=[D(AB^2-2X_3)-B^3C]/2.$$
$$D=Y_1-y_2Z_1^3,$$

타원곡선 덧셈에 필요한 계산량은 12M+4S(M : multiplication, S : squaring)이며, 만일 $Z_2=1$이라면(대부분의 상수배 알고리즘에서 고정된 점이나 사전계산 점들은 효율성을 위해 아핀 좌표계를 사용함) 필요한 계산량은 8M+3S로 줄어든다.

- 두 배(doubling): 점 P_1을 그 자신과 더하는 두배 연산 $2P_1=P_3$는 다음과 같이 계산된다:

$$A=3X_1^2+aZ_1^4,\quad Z_3=BZ_1,$$
$$B=2Y_1,\quad X_3=A^2-2C,$$
$$C=B^2X_1,\quad Y_3=A(C-X_3)-B^4/2.$$

두 배 연산에 필요한 계산량은 4M+6S이며, 만일 $a=-3$이면 $A=3(X_1-Z_1^2)(X_1+Z_1^2)$과 같이 계산하여 2번의 제곱셈을 줄일 수 있다(대부분의 표준에서 효율성을 위해 $a=-3$으로 선택하고 있음).

아핀 좌표계의 점을 사영 좌표계로 옮긴 후 상수배 연산을 수행하여 그 결과를 다시 아핀 좌표계로 변환해 주어야 하며, 이 과정에서 I+3M+S의 계산량이 필요하다(I: field inversion).

2.1.1 Montgomery 상수배를 위한 타원곡선 연산

Montgomery 상수배 알고리즘(Ⅳ.3절 참조)은 일정한 차를 갖는 두 점을 연속적으로 갱신시키는 과정을 통해 상수배를 계산한다. 이런 경우는 두 점의 차가 항상 일정하다는 사실을 이용하여 x-좌표들만 계산함으로써 상수배를 구할 수 있다. 최종적으로 계산된 두 점의 x-좌표와 알려진 차로부터 y-좌표 역시 복구해 낼 수 있다.

두 점의 차 $P_2-P_1=P=(x,y)$와 P_1, P_2의 x-좌표 x_1, x_2만이 주어지는 경우 $P_1+P_2=P_3$의 x-좌표 x_3와 P_1의 y-좌표 y_1은 다음과 같이 계산될 수 있다[4]:

$$x_3=\begin{cases}\dfrac{(x_1x_2-a)^2-4b(x_1+x_2)}{x(x_2-x_1)^2} \text{ or} \\ \dfrac{2(x_1+x_2)(x_1x_2+a)+4b}{(x_2-x_1)^2}-x, & \text{if } P_1\neq P_2\\ \dfrac{(x_1^2-a)^2-8bx_1}{4(x_1^3+ax_1+b)}, & \text{if } P_1=P_2\end{cases}$$
$$y_1=\frac{y^2+x_1^3+ax_1+b-(x-x_1)^2(x+x_1+x_2)}{2y}$$

위 공식의 사영 좌표계 표현은 표준적인 $x=X/Z$, $y=Y/Z$의 변환공식을 사용하여 다음과 같이 나타낼 수 있다. $P_i=(X_i, Y_i, Z_i)$, $P_2-P_1=P=(X, Y, Z)$라 두자.

- 덧셈(Addition) : $P_1+P_2=P_3$의 X-좌표와 Z-좌표는 다음과 같이 계산된다(아핀 좌표계에서의 첫 번째 공식 이용):

$$Z_3=X(X_1Z_2-X_2Z_1)^2,$$
$$X_3=Z((X_1X_2-aZ_1Z_2)^2-4bZ_1Z_2(X_1Z_2+X_2Z_1))$$

두 점의 차 P가 아핀 좌표계로 표현되는 경우 $P=(x, y, 1)$이라 둘 때 위 식은 다음과 같이 단순화된다:

$$Z_3=x(X_1Z_2-X_2Z_1)^2,$$
$$X_3=(X_1X_2-aZ_1Z_2)^2-4bZ_1Z_2(X_1Z_2+X_2Z_1)$$

Montgomery 덧셈에 필요한 계산량은 9M+2S이며, 만일 $Z=1$이면(대부분 두 점의 차 P는 고정된 기본점이므로 이 경우에 해당) 8M+2S, 또한 $a=-3$(혹은 작은 수)이면 7M+2S로 줄어든다.

- 두 배(doubling) : $2P_1=P_3$의 X-좌표와 Z-좌표는 다음과 같이 계산된다:

$$Z_3=4(X_1Z_1(X_1^2+aZ_1^2)+bZ_1^2\cdot Z_1^2),$$
$$X_3=(X_1^2-aZ_1^2)^2-8bZ_1^2\cdot X_1Z_1.$$

필요한 계산량은 6M+3S이며, 만일 $a=-3$(혹은 작은 수)이면 5M+3S로 줄어든다.

2.2 GF(2^m)상의 타원곡선 연산

유한체 GF(2^m)상의 타원곡선은 다음과 같은 Weierstrass 방정식을 만족하는 GF(2^m)상의 모든 점 (x, y)와 가상의 무한원점(point at infinity) O로 구성된다:

$$E:\ y^2+xy=x^3+ax^2+b,\ a, b\in \mathrm{GF}(2^m),\ b\neq 0$$

타원곡선 덧셈군은 무한원점 O를 항등원으로 하여 다음과 같은 덧셈규칙으로 생성된다.

- 역원 : $P=(x, y)$에 대해 $-P=(x, x+y)$
- 덧셈(Addition) : 두 점 P_1, $P_2(P_1\neq\pm P_2)$의 합 $P_1+P_2=P_3$는 다음과 같이 계산된다:

$$x_3=\lambda^2+\lambda+x_1+x_2+a,$$
$$y_3=\lambda(x_1+x_3)+x_3+y_1,\ \text{where } \lambda=\frac{y_2+y_1}{x_2+x_1}$$

- 두 배(doubling) : 점 P_1을 그 자신과 더하는 두 배 연산 $2P_1=P_3$는 다음과 같이 계산된다:

$$x_3=\lambda^2+\lambda+a,$$
$$y_3=\lambda(x_1+x_3)+x_3+y_1,\ \text{where } \lambda=x_1+\frac{y_1}{x_1}$$

GF(2^m)상에서의 타원곡선에 대한 사영 좌표계로는 다음과 같은 동치관계에 따라 변형된 방정식이 연산에 가장 효율적이다[10] :

$$(x, y)\sim(X, Y, Z)\ \Leftrightarrow\ x=\frac{X}{Z}, y=\frac{Y}{Z^2},$$
$$E':\ Y^2+XYZ=X^3Z+aX^2Z^2+bZ^4.$$

따라서 아핀 좌표계와 사영 좌표계 사이의 점들간의 변환은 다음과 같이 이루어진다(통상 효율성을 위해 γ는 1을 취함) :

$$(x, y)\mapsto(\gamma x, \gamma^2 y, \gamma)\ \text{for some}\ \gamma\in \mathrm{GF}(2^m),$$
$$(X, Y, Z)\mapsto\left(\frac{X}{Z}, \frac{Y}{Z^2}\right).$$

사영 좌표계의 타원곡선은 방정식 E'을 만족하는 $(0,0,0)$이 아닌 GF(2^m)상의 모든 점 (X, Y, Z)와 무한원점 $(1,0,0)$으로 이루어진다. 사영 좌표계에서의 덧셈규칙은 다음과 같다.

- 역원 : $-P=(X, Y+XZ, Z)$ for $P=(X, Y, Z)$
- 덧셈(Addition) : 두 점 P_1, $P_2(P_1\neq\pm P_2)$의 합 $P_1+P_2=P_3$는 다음과 같이 계산된다:

$$A_0=X_2Z_1,\quad D=B_0+B_1,\quad H=DF,$$
$$A_1=X_1Z_2,\quad E=Z_1Z_2,\quad X_3=D^2+H+G$$
$$B_0=Y_2Z_1^2,\quad F=CE,\quad I=C^2A_0E+X_3,$$
$$B_1=Y_1Z_2^2,\quad Z_2=F^2,\quad J=C^2B_0+X_3,$$
$$C=A_0+A_1,\quad G=C^2(F+aE^2),\quad Y_3=HI+Z_3J.$$

만일 $Z_2=1$이면, 즉 P_2가 아핀 좌표계의 점이라면, $P_2=(x_2, y_2, 1)$이라 둘 때 위의 계산은 다음과 같이 단순화된다:

$$A=X_1+x_2Z_1, \quad E=BC,$$
$$B=Y_1+y_2Z_1^2, \quad X_3=B^2+D+E,$$
$$C=AZ_1, \quad F=X_3+x_2Z_3,$$
$$D=A^2(C+aZ_1^2), \quad G=X_3+y_2Z_3,$$
$$Z_3=C^2, \quad Y_3=EF+Z_3G$$

필요한 계산량은 14M+6S이며, 만일 $Z_2=1$이면 10M+4S로 줄어든다. 대부분의 표준에서 권고하는 타원곡선에서는 효율성을 위해 $a=0$이나 1로 택하는데, 이 경우는 각각 13M+6S, 9M+4S의 계산량이면 된다.

- 두 배(doubling) : 점 P_1을 그 자신과 더하는 두 배 연산 $2P_1=P_3$는 다음과 같이 계산된다:

$$Z_3=X_1^2\cdot Z_1^2,$$
$$X_3=X_1^4+bZ_1^4,$$
$$Y_3=bZ_1^4\cdot Z_3+X_3(aZ_3+Y_1^2+Z_1^4).$$

여기에 필요한 계산량은 5M+5S이며, 만일 $a=0$이나 1이면 4M+5S로 줄어든다.

최종적으로는 사영 좌표계의 점을 아핀 좌표계의 점으로 변환해 주어야 하며, 이 과정에는 I+2M+S의 계산량이 요구된다.

2.2.1 Montgomery 상수배를 위한 타원곡선 연산

GF(2^m)상의 타원곡선에 대해서도 마찬가지로 두 점의 x-좌표와 그 차만으로 두 점의 합을 구하는 공식을 유도해 낼 수 있다. 두 점의 차 $P_2-P_1=P=(x,y)$와 P_1, P_2의 x-좌표 x_1, x_2만이 주어지는 경우 $P_1+P_2=P_3$의 x-좌표 x_3와 P_1의 y-좌표 y_1은 다음과 같이 계산될 수 있다[11] :

$$x_3=\begin{cases} x+\dfrac{x_1x_2}{x_1^2+x_2^2}, & \text{if } P_1\neq P_2 \\ x_1^2+\dfrac{b}{x_1^2}, & \text{if } P_1=P_2. \end{cases}$$

$$y_1=\frac{(x+x_1)((x+x_1)(x+x_2)+x^2+y)}{x}+y$$

위 공식의 사영 좌표계 표현은 표준적인 $x=X/Z$, $y=Y/Z$의 변환을 사용, 다음과 같이 나타낼 수 있다. $P_2-P_1=P=(X,Y,Z)$라 두자.

- 덧셈(Addition) : $P_1+P_2=P_3$의 (X_3, Z_3)는 다음과 같이 계산된다:

$$A=(X_1Z_2+X_2Z_1)^2,$$
$$Z_3=AZ,$$
$$X_3=AX+Z(X_1Z_2\cdot X_2Z_1).$$

두 점의 차 P가 아핀 좌표계로 표현되는 경우 $P=(x,y,1)$라 둘 때 위 식은 다음과 같이 단순화된다:

$$Z_3=(X_1Z_2+X_2Z_1)^2,$$
$$X_3=xZ_3+X_1Z_2\cdot X_2Z_1.$$

여기에 필요한 계산량은 6M+S이며, $Z=1$인 경우는 4M+S로 줄어든다.

- 두 배(doubling) : $2P_1=P_3$의 (X_3, Z_3)는 다음과 같이 2M+4S의 계산량으로 구할 수 있다:

$$X_3=X_1^4+bZ_1^4,$$
$$Z_3=X_1^2\cdot Z_1^2$$

2.3 계산량 비교

각 유한체상의 타원곡선에서 두 점의 덧셈과 두 배 연산에 필요한 계산량을 아핀 및 사영 좌표계, 일반적인 덧셈 및 x-좌표만을 계산하는 Montgomery 방식을 위한 덧셈 각각에 대해 [표 1]과 [표 2]에 정리하였다.

타원곡선 점의 x-좌표만을 계산하는 Montgomery 연산은 Montgomery 방식의 상수배 알고리즘에서만 사용 가능한 만큼 기본 연산의 효율성 비교는 결국 타원곡선 상수배 알고리즘과 결부시켜 생각해야 한다. Montgomery 상수배 알고리즘은 상수의 각 비트마다 한번씩의 타원곡선 덧셈과 두 배 연산이 반드시 필요하지만, 일반적인 타원곡선 상수배 알고리즘에서는 사전계산 테이블을 이용하여 타원곡선 덧셈 수를 크게 줄일 수 있다.

GF(2^m)상의 타원곡선의 경우 사영 좌표계를 사용하는 Montgomery 연산의 효율성이 매우 뛰어나 Montgomery 상수배 알고리즘이 알려진 모든

〔표 1〕 아핀 좌표계에서의 타원곡선 연산량 (I,M,S) 비교

타원곡선 유한체	General		Montgomery	
	Add	Double	Add	Double
GF(p)	(1,2,1)	(1,2,2)	(1,3,1)	(1,4,2)
GF(2^m)	(1,2,1)	(1,2,1)	(1,2,2)	(1,1,1)

〔표 2〕 사영 좌표계에서의 타원곡선 연산량 (M,S) 비교

타원곡선 유한체	General		Montgomery	
	Add	Double	Add	Double
GF(p)	(12,4)	(4,6)	(9,2)	(6,3)
(Z=1)	(8,3)		(8,2)	
(+a=-3)	(8,3)	(4,4)	(7,2)	(5,3)
GF(2^m)	(14,6)	(5,5)	(6,1)	(2,4)
(Z=1)	(10,4)		(4,1)	
(+a=0,1)	(9,4)	(4,5)	(4,1)	(2,4)

타원곡선 상수배 알고리즘 중에서도 가장 효율적인 방법의 하나로 꼽힌다. 또한 스마트카드나 하드웨어 구현 등과 같이 메모리가 부족하거나 값비싼 환경에서도 임시 저장공간을 적게 사용하는 Montgomery 방식이 보다 유리할 수 있다.

반면 Montgomery 연산은 GF(p)상의 타원곡선에서는 그렇게 효율적이지는 않아 사전 계산 테이블을 이용하여 덧셈의 수를 줄일 수 있는 대부분의 윈도우 계열 알고리즘이 훨씬 효율적이다. 다만 타원곡선 덧셈과 두 배 연산이 동일한 빈도로 사용되는 이진 알고리즘의 경우는 사영 좌표계의 Montgomery 방식이 조금은 더 효율적이다(GF(p)상의 연산에서 제곱은 일반적인 곱셈에 비해 약 20%정도 빠르며 (S≈0.8M), GF(2^m)의 경우 제곱은 곱셈의 15~20% 정도 (S≈0.2M)밖에 시간이 걸리지 않는다).[12]

Ⅲ. 단순 및 차분 전력 소모량 분석

3.1 단순 전력 소모량 분석(SPA)

전력 소모량 분석은 암호 알고리즘의 구현상의 취약점을 이용하는 대표적인 공격방법으로, 기본적으로 한 순간의 전력 소모량은 그 순간 실행되는 명령어와 데이터에 밀접히 관련되어 있다는 사실에 기반한다. 암호 알고리즘에 대한 공격에서는 비밀키가 개입된 연산의 전력 소모량이 비밀키와 밀접한 상관관계를 갖는 경우 단순한 전력 소모량의 관찰만으로도 해당 비밀키를 쉽게 알아낼 수 있다. 특히 공개키 암호는 기본 연산 자체가 상당히 많은 컴퓨팅 자원을 소모하여 단위 연산들을 구분하는 것이 용이한 만큼 구현시 세심한 주의를 기울이지 않는다면 전력 소모량 분석에 특히 취약할 수 있다. 알고리즘 자체의 암호분석을 위해 엄청난 시간과 노력을 투자하는 것에 비하면 이러한 부가채널 공격은 간단한 측정 장비만으로도 쉽게 공격이 가능한 하므로 구현과정에서의 안전성 확보가 무엇보다도 중요함을 알 수 있다.

예를들어 점 P와 k비트의 정수 d가 주어졌을 때 타원곡선 상수배 $Q=dP$를 계산하는 표준적인 방법인 이진 상수배 알고리즘을 생각해 보자. 다음과 같은 상수 d의 이진 전개를 바탕으로,

$$d=d_{k-1}2^{k-1}+\cdots+d_1 2+d_0, \quad d_i\in\{0,1\}$$

이진 상수배 알고리즘은 d의 최상위 비트부터 아래로 처리해 내려가는 Left-to-Right 알고리즘과 최하위 비트부터 위로 처리해 올라가는 Right-to-Left 알고리즘으로 나눌 수 있다.

Algorithm BL0 : Binary L-R algorithm
INPUT: d, P
OUTPUT: Q = dP (d_{k-1}=1로 가정)
1: Q ← P
2: for i=k-2 to 0
3: Q ← 2Q
4: if d_i=1, Q ← Q + P
5: return Q

Algorithm BR0 : Binary R-L algorithm
INPUT: d, P
OUTPUT: Q = dP
1: Q ← O, T ← P
2: for i=0 to k-1
3: if d_i=1, Q ← Q + T
4: T ← 2T
5: return Q

위의 두 알고리즘은 모두 d_i의 비트 값이 1일 때만 타원곡선 덧셈을 수행한다. II장에서 보았듯이 타원곡선 덧셈과 두 배 연산은 전혀 다른 방식과 순서로 계산되며 요구되는 계산량에서도 큰 차이가 있으므로 SPA로 쉽게 구분될 수 있다. 따라서 위의

두 알고리즘은 한번의 수행과정에 대한 전력 소모량 관찰만으로도 타원곡선 덧셈의 수행 여부를 바탕으로 쉽게 비밀키 d를 알아낼 수가 있다. SPA에 안전하기 위해서는 d_i의 비트 값에 무관하게 항상 동일한 연산이 동일한 순서로 일어나도록 해야한다. 아래에 기술된 두 알고리즘이 이런 목적으로 알고리즘 BL0/BR0를 변형시킨 것이다.[3,4]

Algorithm BL : Modified Algorithm BL0
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← P /* d_{k-1} = 1로 가정 */
2: for i=k-2 to 0 step -1
3: Q[0] ← 2Q[0]
4: Q[1] ← Q[0] + P
5: Q[0] ← Q[d_i]
6: return Q[0]

Algorithm BR : Modified Algorithm BR0
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← P, Q[1] ← O
2: for i=0 to k-1 step +1
3: Q[2] ← Q[0] + Q[1]
4: Q[0] ← 2Q[0]
5: Q[1] ← Q[1+d_i]
6: return Q[1]

알고리즘 BL/BR에서는 타원곡선 덧셈이 각 키 비트 d_i의 값에 무관하게 항상 수행된다. 즉 d_i=0인 경우도 덧셈은 수행하되 다음 단계에서 그 결과를 이용하지는 않는다(dummy addition). 단계 5에서의 변수 대입연산이 d_i의 비트 값에 의존하여 수행되도록 되어 있으나 실제로는 대입연산이 일어나지 않도록 구현할 수도 있다. 원래의 알고리즘에 비해 전력 소모량 분석에 안전하도록 변형된 알고리즘은 훨씬 더 많은 계산량이 필요함을 알 수 있다. 따라서 안전하면서도 보다 효율성이 뛰어난 알고리즘의 개발이 중요한 과제로 부각되고 있다.

3.2 차분 전력 소모량 분석(DPA)

알고리즘 BL/BR이 단순 전력 소모량 분석에 대해서는 안전한 것으로 보이지만 보다 정교한 차분 분석 공격에는 여전히 취약하다. DPA는 SPA로는 구분이 안되는 랜덤한 잡음처럼 보이는 전력 소모량 측정치들을 통계적인 방법으로 분석하여 잡음신호를 제거하고 원하는 신호를 추출해 내는 기법을 사용한다. 따라서 DPA는 서로 다른 데이터를 사용한 다수의 알고리즘 실행과정에서 수집된 전력 소모량 측정치들을 필요로 한다.

일반적으로 DPA 공격은 비밀키에 의존하는 알려진 특정 중간 계산 값과 그 과정에서 소모되는 전력량 사이의 상관관계를 분석하여 관련 키 비트를 추정한다. 예를들어 알고리즘 BL에서 두 번째 최상위 비트 d_{k-2}는 $4P$의 이진 전개에서의 특정 비트(예를 들어 최하위 비트)와 이때의 전력 소모량 사이의 상관관계를 분석하여 결정할 수 있다. 만일 $d_{k-2}=0$이면 다음 단계의 두 배 연산은 $Q=4P$를 계산할 것이므로 $4P$의 특정 비트는 이 때의 전력 소모량과 밀접히 관련되어 있을 것이지만, $d_{k-2}=1$인 경우는 $4P$가 절대 계산되지 않으므로(대신 $Q=6P$가 계산됨) 전력 소모량은 $4P$의 특정 비트와는 아무런 상관관계가 없을 것이다.[3]

좀 더 구체적으로 l개의 서로 다른 기본점 P_i로 알고리즘 BL를 l번 수행하여 $Q_i=dP_i$ $(1\le i\le l)$를 계산하고, 각 실행에서 측정된 전력 소모량을 시간의 함수로 $C_i(t)$라 두자. 그리고 $4P_i$의 선택된 특정 비트를 s_i라 두자. 그러면 전력 소모량 $C_i(t)$와 특정 비트 s_i 사이의 상관관계는 다음과 같은 함수 $g(t)$로 계산될 수 있다:

$$g(t)=\langle C_i(t)\rangle_{1\le i\le l \mid s_i=1}-\langle C_i(t)\rangle_{1\le i\le l \mid s_i=0}$$

$4P_i$가 계산되는 순간을 t_1이라 두면 $C_i(t_1)$은 s_i와 밀접한 상관관계를 가질 것이다. 따라서 만일 $d_{k-2}=0$이면 $s_i=1$인 $4P_i$들의 평균 전력 소모량과 $s_i=0$인 $4P_i$들의 평균 전력 소모량 사이에 차이가 있을 것이므로 $g(t)$는 $t=t_1$인 순간 피크를 보일 것이다. 그러나 $d_{k-2}=1$인 경우는 $4P_i$가 계산되지 않으므로 $4P_i$의 특정 비트는 $t=t_1$인 순간의 전력 소모량과 아무런 관계가 없으므로 $g(t)$는 단순한 잡음 수준으로 아무런 피크도 나타나지 않을 것이다. 이런 과정을 각 키 비트에 대해 반복하면 전체 비밀키 d를 알아낼 수 있다.

위와 같은 DPA 공격은 키에 의존하는 중간 계산 결과를 예측할 수 있는 한 어떤 타원곡선 상수배 알고리즘에 대해서도 적용 가능함을 쉽게 알 수 있다. 그러나 이런 유형의 DPA 공격은 $Q=dP$의 계

산에서 비밀키 d나 기본점 P를 내부적으로 매번 랜덤하게 바꾸어서 실행시킴으로써 쉽게 막을 수 있고,[3,6] 이런 과정에 필요한 계산량은 전체 계산량에 비해 무시할 수 있을 정도이므로, 일단 SPA에 안전한 계산 알고리즘이 개발되면 DPA 공격에 대한 대응책 중의 하나를 적용시켜 이를 쉽게 전력 소모량 분석에 안전한 알고리즘으로 바꿀 수 있다.

Ⅳ. 기존 타원곡선 상수배 알고리즘 분석

4.1 이진 상수배 알고리즘

Ⅲ.1절의 알고리즘 BL/BR은 SPA에 대한 안전성 측면에서 간과하기 쉬운 잠재적인 문제점을 안고 있다. 알고리즘 BL에서 최상위 비트를 항상 1로 가정한 것과 알고리즘 BR에서 무한원점 O와의 덧셈의 구현이 그것이다. 우리는 정수 d를 점 P의 위수 n보다 작은 수로 랜덤하게 선택했을 때 항상 최상위 비트를 1로 가정할 수는 없으며, 만일 이를 가정한다면 우리는 이미 한 비트의 비밀키에 대한 정보를 잃게 되는 것이다. 또한 만일 0인 최상위 비트들을 모두 무시하고 최초로 1이 되는 비트부터 계산을 시작한다면 SPA에 의해 쉽게 무시되는 0의 수를 알아낼 수 있으므로 0인 최상위 비트의 수만큼 비밀키의 크기가 줄어들게 된다. 이는 랜덤한 d에 대해 평균적으로 약 2비트의 키가 노출되는 결과를 초래한다.

마찬가지로 알고리즘 BR에서 최하위 비트가 0인 경우 만일 단계 3에서 O와의 덧셈을 다른 덧셈과 동일하게 처리하지 않는다면, 즉 통상의 구현에서처럼 Q←P+O를 단지 Q←P와 같이 점의 복사나 할당으로 처리해 버린다면, 알고리즘 BL에서와 마찬가지로 SPA에 의해 d의 0인 최하위 비트 수를 쉽게 알아낼 수 있다. 따라서 이 경우도 평균적으로 약 2비트의 키 값이 노출되는 결과를 초래한다. 이런 문제는 알고리즘 BR이 $d_i=0$인 경우에도 타원곡선 덧셈을 수행하지만 그 결과를 이용하지 않고 버리기 때문에 발생하는 것으로 일반적으로 많이 사용하는 더미 연산(dummy operation)의 추가로 SPA에 대한 안전성을 확보하는 것이 때로는 취약점을 가질 수 있음을 보여준다.

위와 같은 잠재적인 보안상 허점들을 피하기 위해서는 비밀키 d를 항상 k비트로 고정시켜 다루고, 대신 0인 비트들에 대응하는 무한원점 O와의 덧셈을 다른 덧셈과 구분할 수 없도록 동일한 연산으로 특수 처리해 주어야 한다. 아래 두 알고리즘이 이런 가정을 바탕으로 지적된 문제점을 해결할 수 있도록 알고리즘 BR과 BL을 약간 수정한 것이다.

Algorithm BL′ : Modified BL
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← O, Q[1] ← P
2: for i=k-1 to 0 step -1
3: Q[0] ← 2Q[d_i]
4: Q[1] ← Q[0] + P
5: return Q[d_0]

Algorithm BR′ : Modified BR
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← O, Q[1] ← O, Q[2] ← P
2: for i=0 to k-1 step +1
3: Q[d_i] ← Q[d_i] + Q[2]
4: Q[2] ← 2Q[2]
5: return Q[1]

알고리즘 BL′는 본질적으로는 알고리즘 BL과 크게 다른 것이 없으나 대입연산을 제거하여 코드를 좀 더 단순화시켰다. 알고리즘 BR′는 알고리즘 BR에 비해 약간의 변경이 일어난 것으로 덧셈 단계에서 $d_i=0$인 경우는 $Q[0]$를, $d_i=1$인 경우는 $Q[1]$을 갱신시킴으로써 최종적으로는 $Q[1]=dP$, $Q[0]=d'P$가 계산됨을 알 수 있다(d'는 d에 대한 1의 보수).

R-L 종류의 알고리즘에서는 덧셈과 두 배 연산이 동시에 병렬 처리될 수 있으므로 아핀 좌표계 연산의 경우 더 나은 성능을 제공할 수 있다(사영 좌표계에서는 매 번 변하는 점의 덧셈을 수행하므로 Z값 중의 하나가 1이 될 수가 없어 오리려 성능이 훨씬 떨어진다). 특히 하드웨어 구현의 경우는 칩 면적을 더 사용하여 속도를 두 배로 높일 수 있다.

소프트웨어적인 측면에서도 만일 아핀 좌표계에서 연산을 한다면 시간이 가장 많이 걸리는 유한체 역원 계산을 병렬로 처리하여 성능을 높일 수 있다. 즉 두 수 x, y의 역원은 $z=(xy)^{-1}$를 먼저 계산한 후 $x^{-1}=zy$, $y^{-1}=zx$와 같이 계산함으로써 2번의 역원 계산 대신 1번의 역원 계산과 3번의 곱셈을 사용하여 구할 수 있다. 일반적으로 t개의 수에 대한

병렬 역원 계산은 1번의 역원 계산과 3(t-1)번의 곱셈으로 수행할 수 있으므로, 한 번을 제외한 나머지 역원 계산을 3번의 곱셈으로 맞바꿀 수 있다[13, Algorithm 10.3.4]. 유한체 역원 계산은 프로세서나 유한체 종류, 데이터의 길이 등에 따라 상당한 차이가 있으나 유한체 곱셈에 비해 수 십배까지 차이가 날 수 있으므로 대부분의 소프트웨어 구현에서 위의 병렬처리 기법은 훨씬 나은 성능을 제공한다.

4.2 고정 윈도우 알고리즘

이진 상수배 알고리즘 BL은 사전 계산 테이블을 이용하여 쉽게 고정 윈도우 알고리즘으로 확장시킬 수 있다. 윈도우 크기를 w라 두면 2^w보다 작은 모든 수에 대한 점 P의 배수들을 사전 계산해 두고 d의 상위부터 w비트씩 처리해 내려가는 것이다.

Algorithm W : Fixed window algorithm
INPUT: d, P
OUTPUT: Q = dP
1: T[0] ← P, T[j] ← jP ($1 \le j \le 2^w-1$)
2: r ← k mod w; if r=0, set r ← w
3: j ← $d_{k-1} \ldots d_{k-w+1}d_{k-w}$
4: Q[0] ← T[j]
5: for i=k-w-1 to r+w-1 step -w
6: Q[0] ← 2^wQ[0]
7: j ← $d_i \ldots d_{i-w}d_{i-w+1}$
8: Q[1] ← Q[0] + T[j]
9: if j≠0, Q[0] ← Q[1]
10: Q[0] ← 2^rQ[0]
11: j ← $d_{r-1} \ldots d_1d_0$
12: Q[1] ← Q[0] + T[j]
13: if j≠0, Q[0] ← Q[1]
14: return Q[0]

알고리즘 BL에서와 마찬가지로 알고리즘 W에서도 해당 윈도우 값이 0일 때도 동일하게 타원곡선 덧셈을 수행해야 하므로 $T[0]$에 임의의 점(위에서는 P)을 저장하여 덧셈은 수행하되 단계 9에서 보듯이 이 경우는 계산 결과를 전혀 이용하지 않는다. 마지막 윈도우는 w비트가 안될 수 있으므로 특수처리해 주어야 한다.

알고리즘 W에서도 첫 윈도우가 0인 경우 무한원점과의 덧셈에 대한 특수 처리가 필요하다. 이런 가정하에 위 알고리즘은 통상적인 SPA 공격에 안전한 것처럼 보이지만 DPA 기술을 이용한 공격에 취약할 수 있다. 즉 비록 윈도우 값에 상관없이 동일한 연산이 일어나지만 타원곡선 덧셈단계에서 더해지는 사전계산 점들이 비밀키 값에 의존하므로 덧셈 과정의 전력 소모량 분석을 통해 키 값을 유도해 내는 공격이 가능할 수도 있다.[14] 이는 타원곡선 상수배 과정에서 사전계산 점들이 반복 사용되기 때문에 발생되는 문제로, 통상의 DPA 공격과는 달리 단 한 번의 타원곡선 상수배 과정에 대한 전력 소모량 측정만으로 분석이 가능하며, 따라서 최초에 한 번만 점 P를 랜덤하게 만드는 통상적인 DPA 방어책은 전혀 도움이 되지 않는다. Randomized algorithm을 사용하는 것이 하나의 대응책이 될 수 있으나,[7,8] 효율성이 낮을뿐더러 기존의 효율적인 윈도계열 알고리즘을 이용할 수 없다는 단점이 있다. VI.1절에서 윈도우 알고리즘에서 적용 가능한 대응책을 생각해 본다.

4.3 Montgomery 상수배 알고리즘

$Q=dP$를 계산하는 Montgomery 알고리즘은 d의 최상위 비트부터 매 비트마다 덧셈과 두 배 연산을 통해 두 점을 연속적으로 갱신시켜 나가되 그 차가 항상 P를 유지하도록 한다. 이 방법은 Montgomery에 의해 타원곡선기반의 소인수분해 알고리즘을 최적화시키는 과정에서 처음 사용된 이후 최근 들어 그 효율성이나 부가채널 공격에 대한 안전성 등 여러 장점들로 학계의 주목을 받아온 알고리즘이다.[4,5,11]

Algorithm M : Montgomery algorithm
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← P, Q[1] ← 2P /* d_{k-1} = 1 */
2: for i=k-2 to 0 step -1
3: Q[1] ← Q[1] + Q[0]
4: Q[2] ← 2Q[d_i]
5: Q[0] ← Q[2-d_i]
6: Q[1] ← 2Q[1+d_i]
7: return Q[0]

알고리즘 M은 각 단계에서 d_i의 값에 따라

$(Q[0], Q[1]) = (jP, (j+1)P)$로부터 $d_i = 0$이면 $(2jP, (2j+1)P)$를, $d_i = 1$이면 $((2j+1)P, 2(j+1)P)$를 계산하여 최종적으로는 $(Q[0], Q[1]) = (dP, (d+1)P)$가 계산됨을 알 수 있다. for문의 매 반복과정에서 두 점 $Q[0], Q[1]$의 차는 항상 P로 일정하게 유지되므로, 두 점의 차와 x-좌표들만으로 덧셈과 두 배 연산을 할 수가 있고, 또한 필요하다면 최종적인 $Q[0], Q[1]$의 x-좌표들과 그 차인 P로부터 $Q[0]$의 y-좌표 역시 복구할 수 있다(Ⅱ장 참조). Montgomery 알고리즘은 사영 좌표계에서만 더 나은 성능을 제공하며, 이 경우 $Q[0]$의 y-좌표 복구는 $Q[0], Q[1]$의 x-좌표들의 아핀 좌표계 변환($x_0 \leftarrow X_0/Z_0, x_1 \leftarrow X_1/Z_1$)과 동시에 수행하여 성능을 높일 수 있다. 즉 각각에 필요한 유한체 역원 계산을 병렬로 수행하여 한 번의 역원만 계산하도록 하는 것이다. 이 과정에 필요한 계산량은 GF(p)상의 타원곡선의 경우 I+12M+S, GF(2^m)상의 타원곡선의 경우 I+11M+S이다.

알고리즘 M 역시 d의 최상위 비트를 항상 1로 가정하고 기술하였으나 실제로는 0인 경우의 처리방안을 제공해 주어야 한다. 아래 알고리즘 M′는 이런 목적으로 알고리즘 M을 수정한 것으로 또한 대입연산을 제거하고 좀 더 간단한 코드로 기술하였다. 앞에서와 마찬가지로 무한원점 O와의 덧셈은 더미 연산을 통해 다른 덧셈과 동일하게 처리해 주어야 한다.

Algorithm M′: Modified Montgomery
INPUT: d, P
OUTPUT: Q = dP
1: Q[0] ← O, Q[1] ← P
2: for i=k-1 to 0 step -1
3: Q[0] ← Q[0] + Q[1]
4: Q[1] ← 2Q[$d_i \oplus d_{i+1} \oplus 1$]
5: return Q[d_0]

원래의 알고리즘에서 d_i의 값에 따른 변수 교환 대신 키 비트의 패러티 변화에 따라 타원곡선 두 배 연산을 Q[0]나 Q[1] 중 어디에 대해 수행할 지를 결정하며 최종적으로는 d_0의 값에 따라 Q[0]나 Q[1]을 출력한다. Montgomery 알고리즘의 동작원리를 자세히 살펴보면 위의 변형된 알고리즘이 원래의 알고리즘과 동일한 연산을 수행함을 쉽게 알 수 있다.

Ⅴ. 제안 타원곡선 상수배 알고리즘

5.1 이진 부호화 기법

이진수를 각 비트가 -1, 0 혹은 1의 값을 갖도록 이진 부호화시켜 가능한 0의 수를 최대화시키도록 코딩하는 방법은 타원곡선 덧셈군과 같이 역원의 계산이 쉬운 경우 상당한 성능 향상을 얻을 수 있으므로 많은 연구가 진행되어온 분야이다. 그러나 SPA에 안전한 연산 알고리즘을 위해서는 키의 비트 값에 무관하게 동일한 양의 계산이 일어나도록 하되 효율을 높이는 것이 목표이므로 이러한 이진 부호화 코딩은 크게 도움이 되지 않는다.

본 논문에서는 $Q = dP$의 계산시 비밀키 d를 -1이나 1의 비트값만 갖도록 부호화시키는 방법을 고려한다(표기의 편의상 -1을 $\hat{1}$으로 두자). 즉 임의의 연속된 0비트 후에 1로 끝나는 0^j1과 같은 비트열을 동일한 값을 갖는 $1\,\hat{1}^j$로 대체시키는 것이다. 이러한 코딩은 계산시간이나 전력 소모량 면에서 키 값에 따라 거의 차이가 나지 않을 것이므로 매 번 필요할 때마다 실시간으로 수행할 수도 있다. 그러나 이와 같은 직접적인 코딩 대신 보다 간단히 회전이동만으로 구현하는 방법을 제안한다.

점 P의 소수 위수 n보다 작은 k비트의 랜덤한 상수 $d = \sum_{i=0}^{k-1} d_i 2^i$, $d_i \in \{0, 1\}$에 대한 타원곡선 상수배 $Q = dP$를 계산한다고 하자. 먼저 상수 d가 짝수인 경우 소수 위수 n을 더해 줌으로써 항상 홀수로 만들 수 있음을 상기하자. 상수 d의 ±1-이진 부호화된 값을 $\hat{d}$이라 두고, 표기의 편의상 $\hat{d}$에서 비트 0을 -1의 값을 갖는 것으로 재정의 하자. 또한 d가 짝수인 경우 n을 더해 주면 d의 비트 길이가 하나 증가하여 $(k+1)$-비트가 될 수 있으므로 이진 부호화 후의 $\hat{d}$은 원래 d의 홀짝에 상관없이 항상 $(k+1)$-비트 길이로 간주한다. 그러면 $\hat{d}$은 다음과 같이 간단히 d를 오른쪽으로 1비트 회전이동 시킨 것과 동일함을 알 수 있다 :

$$\hat{d} = \mathrm{ROR}(k+1, d, 1) = \sum_{i=0}^{k} \hat{d}_i 2^i$$

여기서 $\mathrm{ROR}(k+1, d, 1)$은 d를 $(k+1)$-비트 정수로 간주하여 이를 오른쪽으로 1비트 회전이동시킨 것을 의미한다. d=1011001에 대해 이를 7비트 및

8비트로 ±1-이진 부호화 시키는 경우의 예를 들어 보면 다음과 같다:

$$d = 1\ 0\ 1\ 1\ 0\ 0\ 1$$

$$\hat{d} = 1\ 1\ \hat{1}\ 1\ 1\ \hat{1}\ \hat{1} \quad (7-\text{bit encoding})$$

$$= 1\ 1\ 0\ 1\ 1\ 0\ 0 = \mathrm{ROR}(7, d, 1)$$

$$= 1\ \hat{1}\ 1\ \hat{1}\ 1\ 1\ \hat{1}\ \hat{1}\ (8-\text{bit encoding})$$

$$= 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0 = \mathrm{ROR}(8, d, 1)$$

이 ±1-이진 부호화 기법의 장점은 k비트의 d를 항상 최상위 비트가 1인 $(k+1)$-비트 길이의 부호화된 값으로 바꿀 수 있다는 것이다. 이 성질은 앞의 이진 알고리즘들에서 최상위 비트가 0인 경우 무한원점과의 덧셈을 특수하게 처리해야 하는 불편을 없앨 수 있다. 또한 키 비트가 0인 경우에도 더미 연산 대신 타원곡선 뺄셈을 수행하므로 더미 연산의 사용에 따른 잠재적인 취약성을 피할 수 있다.

5.2 타원곡선 덧셈과 뺄셈의 통합구현

본 논문에서는 ±1-이진 부호화 기법을 이용한 타원곡선 상수배 알고리즘을 다루고 있다. 따라서 부가채널 공격에 안전하기 위해서는 타원곡선의 덧셈과 뺄셈이 수행시간이나 전력 소모량 등에 있어서 서로 구분 될 수 없도록 동일한 코드로 구현되어야 한다.

II장에서 살펴 보았듯이 $GF(2^m)$상의 타원곡선에서 사영 좌표계를 이용하는 경우를 제외하면 타원곡선 덧셈과 뺄셈의 차이는 단지 y-좌표에 대한 한 번의 유한체 덧셈/뺄셈 뿐이다. 이것은 대부분의 경우 큰 차이를 주지는 않겠지만 안전성을 위해서는 구현 시 타원곡선 덧셈과 뺄셈을 동일한 코드로 통합 구현하는 것이 바람직하며 이는 간단히 구현될 수 있다. 사영 좌표계를 사용하는 $GF(2^m)$상의 타원곡선에서는 아핀 좌표계와 사영 좌표계 사이의 변환공식에 따라 추가로 한번의 유한체 곱셈이 더 요구되어 성능이 떨어질 수 있으나, 타원곡선 상수배를 위한 이진 알고리즘이나 윈도우 알고리즘에서 타원곡선 덧셈은 항상 고정된 혹은 사전 계산된 점들과의 덧셈이고 이들은 통상 효율성을 위해 아핀 좌표계에서 계산된 값을 사용하므로 역시 문제가 되지 않는다.

혹자는 DPA에 대한 대응책으로 사영 좌표계에서 타원곡선 점을 Randomization시키는 방법을 사용하면 고정된 점이나 사전 계산된 점들이 아핀 좌표계로 표현되더라도 결국은 Z-좌표가 랜덤한 값으로 바뀌므로 타원곡선 덧셈시 효율이 떨어진다고 주장하지만 이는 사실이 아닐 수 있다. 자세한 것은 VI.1절에서 다룬다.

본 논문에서 제안된 타원곡선 상수배 알고리즘에서는 표기의 편의상 다음과 같은 일반화된 타원곡선 연산을 사용하기로 한다:

$$\mathrm{ECADD}(Q, P, s) = \begin{cases} P+Q & \text{if } s=1, \\ P-Q & \text{if } s=0, \end{cases}$$

$$\mathrm{ECDBL}(Q, w) = 2^w Q.$$

즉 함수 ECADD는 플래그 비트 s에 따라 두 점에 대한 덧셈 혹은 뺄셈을 수행한다. 이는 s에 상관없이 Q의 y-좌표에 대한 음수 변환을 수행한 후 s=0인 경우 변환된 값을, 그렇지 않은 경우는 원래의 값을 사용하여 나머지 계산을 수행하면 될 것이다. ECDBL은 두 배 연산의 반복 횟수 w를 입력으로 받아 w번의 두 배 연산을 수행하는 연산이다.

5.3 이진 상수배 알고리즘

알고리즘 SBL과 SBR은 알고리즘 BL과 BR의 부호화 버전으로 d의 ±1-이진 부호화 과정을 포함하여 기술되어 있다 (이진 부호화된 값에서 비트 0은 -1을 의미함을 상기하자). 기본적으로 각 키 비트마다 두 배 연산과 타원곡선 덧셈/뺄셈을 한번씩 수행한다.

Algorithm SBL : Signed binary L-R
INPUT: d, P, n
OUTPUT: Q = dP
1: if d is *even*, d ← d+n
2: d ← ROR(k+1,d,1)
3: Q ← P
4: for i=k-1 to 0 step -1
5: Q ← ECDBL(Q, 1)
6: Q ← ECADD(Q, P, d_i)
7: return Q

Algorithm SBR : Signed binary R-L
INPUT: d, P, n
OUTPUT: Q = dP
1: if d is even, d ← d+n
2: d ← ROR(k+1,d,1)
3: Q ← O, T ← P
4: for i=0 to k-1 step +1
5: Q ← ECADD(Q, T, d_i)
6: Q ← ECDBL(T, 1)
7: return Q

각 알고리즘에서 첫 두 단계는 상수 d에 대한 이진 부호화 과정으로 우선 d가 짝수인 경우는 홀수로 만들어 d를 항상 $(k+1)$-비트의 홀수로 간주하여 ±1-이진 부호화시킨다. 나머지의 상수배 과정에서는 d_i의 각 비트마다 항상 타원곡선 덧셈/뺄셈과 두 배 연산이 한번씩 수행된다. d_i의 비트 값은 단지 ECADD 내에서 타원곡선 덧셈을 수행할 지 혹은 뺄셈을 수행할 지만을 알려줄 뿐 두 연산이 통합 구현되는 한 항상 동일한 연산이 일어나게 된다.

알고리즘 SBL/SBR에서는 상수 d가 항상 최상위 비트가 1인 고정 길이로 부호화되어 사용되므로 알고리즘 BL/BR에서 존재하던 문제점들이 모두 자연스럽게 해결된다. 즉 최상위 비트가 항상 1이므로 알고리즘 BL에서처럼 첫 부분에서 무한원점과의 덧셈을 특수 처리해 주어야 할 필요가 없으며, 알고리즘 SBR에서도 알고리즘 BR에서처럼 비록 처음에는 무한원점과의 덧셈이 수행되지만 더미 연산을 사용하지 않으므로 이 무한원점과의 덧셈을 다른 덧셈과 동일하게 처리해야 할 필요가 없다. 전체적인 계산량은 알고리즘 BL/BR과 거의 차이가 없으나 계산과정에서 필요한 임시변수의 수를 하나씩 줄일 수 있어 메모리가 제한된 환경에서 더 효율적이다.

Montgomery 알고리즘 역시 ±1-이진 부호화 코딩을 이용하면 원래의 알고리즘에서 존재하던 무한원점과의 덧셈에 대한 특수처리 부담을 덜 수 있다. 알고리즘 M′와는 달리 여기서는 타원곡선 두 배 연산을 Q[0], Q[1] 중 어이에 대해 수행할 지는 이전 비트에서 현재 비트로의 부호 변화에 따라 결정한다. 계산과정 동안 Q[0]가 항상 홀 수배의 점을 유지하고 d가 홀 수이므로 최종 결과는 Q[0]가 된다.

Algorithm SM : Signed Montgomery
INPUT: d, P, n
OUTPUT: Q = dP
1: if d is even, d ← d+n
2: d ← ROR(k+1,d,1)
3: Q[0] ← P, Q[1] ← 2P
4: for i=k-1 to 1 step -1
5: Q[0] ← Q[0] + Q[1]
6: Q[1] ← 2Q[$d_i \oplus d_{i-1} \oplus 1$]
7: Q[0] ← Q[0] + Q[1]
8: return Q[0]

5.4 고정 윈도우 알고리즘

통상 SPA 공격은 스마트카드나 휴대용 이동 단말기 혹은 하드웨어 구현 등 리소스가 매우 제한된 환경에서 실질적인 위협이 되는 만큼 구현시 사용 가능한 메모리에 제약이 있을 수 있다. 그러나 하드웨어 기술의 발달로 이러한 제약은 점점 줄어들고 있으며, 저가의 스마트카드와 같은 매우 제한된 환경이라 하더라도 이진 상수배 알고리즘에서 필요로 하는 임시 메모리보다는 좀 더 여유가 있는 경우가 대부분이다. 따라서 가능한 임시 저장소를 적게 사용하면서도 효율성을 높일 수 있는 알고리즘을 개발하는 것은 대단히 중요하다. 이진 알고리즘의 확장인 윈도우 계열 알고리즘은 이용 가능한 임시 메모리의 용량에 따라 사전 계산 테이블의 크기를 조절할 수 있으므로 다양한 구현 환경에서 사용될 수 있다. 우선 SPA에 안전하기 위해서는 가변 윈도우를 사용할 수는 없으므로 고정 윈도우 알고리즘을 고려한다. 아래의 알고리즘 SW는 이진 상수배 알고리즘 SBL의 w-비트 고정 윈도우 버전을 기술한 것이다.

Algorithm SW : Fixed signed window
INPUT: d, P, n
OUTPUT: Q = dP
1: if d is even, d ← d+n
2: d ← ROR(k+1,d,1)
3: r ← k mod w; if r=0, set r ← w
4: T[j] ← (2j+1)P $(0 \le j < 2^{w-1})$
5: j ← $d_{k-1}d_{k-2} \ldots d_{k-w+1}$
6: Q[0] ← T[j]
7: for i=k-w to r+w-1 step -w
8: Q[0] ← ECDBL(Q[0], w)

```
 9:      e ← d_{i-1}d_{i-2} ... d_{i-w+1}
10:      j ← (d_i=1 ? e : e')
11:      Q[1] ← ECADD(Q[0], T[j], d_i)
12: Q[0] ← ECDBL(Q[0], r)
13: e ← d_{r-2}d_{r-3} ... d_0
14: j ← (d_{r-1}=1 ? e : e')
15: Q[1] ← ECADD(Q[0], T[j], d_{r-1})
16: return Q[0]
```

알고리즘 SW의 사전계산 테이블 $T[\,]$에는 w비트 윈도우의 경우 2^w보다 작은 홀 수배의 점들이 저장된다. 따라서 고정 윈도우 알고리즘 W에 비해 사전계산 점을 위한 임시 메모리를 정확히 반만을 이용해 동일한 성능을 제공할 수 있으므로 훨씬 효율적이다. 비밀키 d가 ±1-이진 부호화되어 있으므로 윈도우 값의 최상위 비트로 부호를 식별하고 나머지 비트들로 테이블의 인덱스를 계산할 수 있다. 예를 들어 w=3인 경우 각 윈도우 값 w_i에 따라 더해 주어야 할 점 P의 배수를 보면 다음과 같다:

$T[0]=P, T[1]=3P, T[2]=5P, T[3]=7P$

$w_i=$ 0 0 0 → −7 → $-T[3]$
0 0 1 → −5 → $-T[2]$
0 1 0 → −3 → $-T[1]$
0 1 1 → −1 → $-T[0]$
1 0 0 → +1 → $+T[0]$
1 0 1 → +3 → $+T[1]$
1 1 0 → +5 → $+T[2]$
1 1 1 → +7 → $+T[3]$

윈도우 값의 최상위비트가 0인 경우는 음수이므로 타원곡선 뺄셈을 해 주어야 하고 이때의 해당 테이블 인덱스는 윈도우 값의 나머지 비트들의 1의 보수와 같다. 최상위 비트가 1인 경우는 나머지 비트들을 바로 테이블의 인덱스로 삼을 수 있다(단계 10, 14: j ← (d_i=1 ? e : e')은 d_i=1이면 e를 d_i=0이면 e'을 j에 할당함을 의미).

Ⅳ. 안전성 및 효율성 분석

6.1 안전성 분석

이 절에서는 제안 알고리즘의 DPA 공격에 대한 안전성을 살펴본다. DPA 공격은 동일한 비밀키가 사용된 다수의 서로 다른 기본점에 대한 타원곡선 상수배 과정에서 비밀키에 의존하는 특정 중간 계산 결과와 해당 전력 소모량 사이의 상관관계를 통계적으로 분석하여 비밀키를 추출해 내는 공격 방법이다. 이러한 DPA 공격에 대한 대책으로 효율적이어서 가장 인기가 있는 것은 사영 좌표계에서 기본점을 Randomization시키는 방법과 타원곡선 동형사상을 이용하여 타원곡선 방정식 자체를 Randomization 시키는 방법이 있다.[3,6] 이런 Randomization은 대부분의 경우 SPA에 안전한 알고리즘의 수행 전에 한번만 일어나면 되므로 계산상의 부담은 거의 없다. 본 논문에서는 윈도우 알고리즘의 DPA 안전성을 다루어야 하는 만큼 사영 좌표계에서 기본점을 Randomization시키는 방법을 예로 들어 설명하기로 한다.

예를들어 이진 알고리즘 SBL에서 GF(p)상의 사영 좌표계를 이용하여 $Q=dP$를 계산하는 경우를 고려해 보자. 우선 GF(p)에서 γ를 랜덤하게 선택하여 아핀 좌표계의 기본점 $P=(x,y)$를 사영 좌표계의 기본점 $P'=(\gamma^2x,\gamma^3y,\gamma)$로 변환시킨 후 알고리즘 SBL을 이용하여 사영 좌표계에서 $Q'=d'P$를 계산하고 이를 다시 아핀 좌표계로 변환시킨 Q를 출력하면 된다. 이 때 알고리즘 SBL의 단계 6의 타원곡선 덧셈은 Z-좌표가 1이 아닌 점 P'을 더해야 하므로 타원곡선 덧셈당 4M+S만큼의 계산량이 증가하게 된다(약 25% 증가). 그러나 단계 3에서 랜덤하게 변환된 점 P'으로 Q를 초기화시켜 사용하는 한 덧셈단계에서는 P' 대신 아핀 좌표계 점 P를 사용하더라도 DPA에 대한 안전성에는 전혀 문제가 없다. Q가 랜덤한 값으로 초기화되어 사용되는 한 이 후 계산되는 모든 중간 계산결과들도 랜덤한 값으로 생각할 수 있고, 랜덤한 Q와 알려진 아핀 좌표계 점 P를 더하는 연산은 각 비밀키 비트값에 무관하게 동일하게 수행되며 그 계산결과도 전혀 예측할 수 없기 때문이다. 결론적으로 대부분의 사람들이 가정하는 것처럼 이진 알고리즘 BL이나 SBL, 그리고 Montgomery 알고리즘 M이나 SM에서 사영 좌표계 점의 Randomization 기법을 사용하면 고정된 점 P의 Z-좌표가 1이 아니므로 타원곡선 덧셈에서 계산량이 증가되어 불리하다는 주장은 사실이 아님을 알 수 있다.

그러나 고정 윈도우 알고리즘에서는 문제가 다르다. 이 경우는 더해지는 사전 계산점이 비밀키 값에 의존하며, 해당 타원곡선 덧셈 내의 알려진 사전 계산점의 좌표값과 관련된 유한체 곱셈들이 해당 비밀키 비트들과의 상관관계를 노출시킬 수 있기 때문이

다(참고문헌 [14]에서는 RSA의 윈도우 알고리즘 구현에서 사전 계산된 값과의 모듈러 곱셈에 대한 전력 소모량 분석을 상세히 다루고 있다). 예를들어 160비트의 유한체를 사용하고 윈도우 크기를 4로 잡는 경우 대략 40번의 타원곡선 덧셈이 필요하며 동일한 사전 계산점이 평균적으로 대략 40/16=2.5번 사용될 것이므로 사전계산 과정과 40번의 덧셈 과정 중의 유한체 곱셈에 대한 전력 소모량 측정치들간의 상관관계 분석을 통해 비밀키를 유도해 내는 것이 가능할 수도 있다. 비록 RSA에서 고정된 수와의 모듈러 곱셈보다는 타원곡선 암호에서 고정된 점과의 덧셈이 전력 소모량과의 상관관계가 작으므로 이 공격이 얼마나 효과적일 지는 의문이나 분명 공격의 가능성이 있는 만큼 이에 대한 대책을 세우는 것이 필요하다.

가장 간단한 방법은 사전 계산점을 사용할 때마다 사전계산 테이블 전체를 Randomization시키는 것이나 효율성이 전혀 없으므로 여기서는 타원곡선 덧셈내의 랜덤 변수들을 이용하여 고정된 좌표값을 마스킹시키는 방법을 생각해 본다. 먼저 Ⅱ.1절에서 보듯이 GF(p)상의 타원곡선 덧셈에서 아핀 좌표계의 점 $P_2=(x_2, y_2, 1)$가 관련된 유한체 곱셈은 $x_2Z_1^2$과 $y_2Z_1^3$뿐이다. 이들을 다음과 같이 계산해 보자.

$$T_0=y_2Z_1=(y_2+Z_1)Z_1-Z_1^2,$$
$$T_1=y_2Z_1^2=T_0Z_1^2,$$
$$x_2Z_1^2=(x_2+T_0)Z_1^2-T_1$$

전력 소모량의 차이는 주로 반도체에서 0→1,1→0과 같은 상태변화의 수(스위칭 카운트)에 의존하며 따라서 연산자의 해밍 웨이트(Hamming weight)가 주원인이다. Z_1은 매 타원곡선 덧셈마다 랜덤하게 변하는 값으로 볼 수 있고, 고정된 값과 랜덤한 값의 덧셈에 대한 전력 소모량은 그 고정된 값과 상관관계가 거의 없다고 볼 수 있다. 따라서 위와 같이 내부의 랜덤변수를 이용하여 고정된 수와의 곱셈을 Randomize시키는 것은 계산량을 거의 증가시키지 않고도(유한체 덧셈만 4번 더 사용) DPA를 막을 수 있는 매우 효과적인 방법임을 알 수 있다.

GF(2^m)상의 타원곡선 연산에서도 마찬가지 방법으로 고정된 수와의 곱셈을 내부적으로 Randomize 시킬 수 있다. Ⅱ.2절에서 보듯이 여기서는 고정된 원소와의 곱셈을 4번 수행해야 한다: x_2Z_1, $y_2Z_1^2$, x_2Z_3, y_2Z_3. GF(2^m)상의 연산에서는 제곱은 곱셈에 비해 15 - 20% 정도의 부하밖에 걸리지 않으므로 가장 간단한 방법은 제곱을 두 번 (Z_1^4, Z_3^2) 더 사용하여 xZ를 $(x+Z)Z+Z^2$과 같이 계산하는 것이다. 타원곡선 덧셈과정을 좀 더 세밀히 살펴보면 한 번의 제곱(Z_3^2)만 더 사용하면 가능함을 알 수 있다.

6.2 효율성 분석

Ⅳ장에서 살펴본 각 타원곡선 상수배 알고리즘들의 대략적인 성능 비교를 통해 가장 효율적인 알고리즘 및 좌표계를 알아보자. [표 3]과 [표 4]에 가각 아핀 좌표계와 사영 좌표계 연산에서 비밀키의 각 비트당 요구되는 계산량과 임시 메모리 공간의 수(하나의 좌표값, 즉 하나의 유한체 원소 저장을 위한 임시 저장소를 단위로)를 정리해 보았다. 물론 여기서 임시 저장소는 상수배 알고리즘의 수행만을 위한 것이며 실제로 타원곡선 파라미터의 저장이나 타원곡선 덧셈을 위해서는 훨씬 더 많은 저장공간이 필요하다. 이 표는 타원곡선 상수배 $Q=dP$를 계산하는 여러 알고리즘의 대략적인 성능비교를 위해 상수 d의 비트당 요구되는 계산량만을 나타낸 것이다. 정확한 계산량을 구하기 위해서는 좌표계 변환이나 y-좌표 복구, 사전 계산량 등을 고려하여 총 계산량을 산출해야 한다. 물론 윈도우 알고리즘의 경우는 메모리 공간이 문제가 되지 않는다면 사전 계산량을 고려하여 최적 윈도우 크기를 결정해야 하지만, 본 논문에서 관심이 있는 SPA에 대한 안전성이 성능에 비해 우선순위를 갖는 제약된 환경에서는 그렇게 큰 윈도우 크기를 허용하지 않는 경우가 대부분이므로 상세히 다루지 않는다(160비트 근처의 타원곡선의 경우 최적 윈도우 크기는 5임).

[표 3] 아핀 좌표계에서의 비트당 계산량 비교(storage는 |p| 혹은 m 비트 메모리 개수)

Alg.	storage	EC/GF(p)	EC/GF(2^m)
BL	4	2I+4M+3S	2I+4M+2S
BR	6	I+7M+3S	I+7M+2S
M/SM	4	I+10M+3S	I+6M+3S
SBL	2	2I+4M+3S	2I+4M+2S
SBR	4	I+7M+3S	I+7M+2S
W	$2^{w+1}+4$	$(1+\frac{1}{w})$ (I+2M+S)+S	$(1+\frac{1}{w})$ (I+2M+S)
SW	2^w+2	$(1+\frac{1}{w})$ (I+2M+S)+S	$(1+\frac{1}{w})$ (I+2M+S)

〔표 4〕 사영 좌표계에서의 비트당 계산량 비교(storage는 |p| 혹은 m 비트 메모리 개수, 괄호 ()안은 타원곡선 상수 a=-3(GF(p))이거나 a=0 혹은 1 (GF(2^m))인 경우의 계산량)

Alg.	storage	EC/GF(p)	EC/GF(2^m)
BL	6	12M+9S (12M+7S)	15M+9S (13M+9S)
BR	9	16M+10S (16M+8S)	19M+11S (17M+11S)
M/SM	4	14M+5S (12M+5S)	6M+5S (6M+5S)
SBL	3	12M+9S (12M+7S)	16M+9S (13M+9S)
SBR	6	16M+10S (16M+8S)	20M+11S (18M+11S)
W	$2^{w+1}+6$	$(4+\frac{8}{w})$M $+(6+\frac{3}{w})$S	$(5+\frac{11}{w})$M $+(5+\frac{4}{w})$S
SW	2^w+3	$((4+\frac{8}{w})$M $+(4+\frac{3}{w})$S)	$((4+\frac{9}{w})$M $+(5+\frac{4}{w})$S)

아핀 좌표계의 연산에서는 가장 시간이 많이 걸리는 연산이 유한체상의 역원 계산이므로 가능한 역원 계산을 최소화시킬 수 있도록 덧셈이나 두 배 연산이 병렬을 처리할 수 있을 때는 역원 연산 대신 곱셈을 많이 사용하도록 하였다(Ⅲ.1절 마지막 문장 참조). 일반적으로 소프트웨어 구현에서는 대부분의 프로세서에서 역원 계산이 곱셈에 비해 훨씬 값비싼 연산이므로 아핀 좌표계는 거의 사용되지 않는다. 반면 하드웨어 구현의 경우 통산 역원 계산 회로를 구현하는 경우가 많고 이때는 계산량이 적고 병렬처리가 가능한 알고리즘 BL/SBL이나 GF(2^m)상의 타원곡선의 경우 Montgomery 알고리즘 등이 유리할 것이다.

사영 좌표계에서는 표 4에서 보듯이 이진 상수배 알고리즘 부류에서는 어느 타원곡선에서나 Montgomery 알고리즘이 가장 우수한 성능을 보여준다. 특히 GF(2^m)상의 타원곡선의 경우 임시 메모리가 충분하고 SPA를 고려하지 않더라도 대부분의 프로세서에서 Montgomery 알고리즘이 가장 효율적인 알고리즘의 하나이므로 당연히 선택의 최우선 순위를 갖는다. GF(p)상의 타원곡선에 대해서도 이진 알고리즘의 경우 Montgomery 알고리즘이 조금은 더 효율적이다. 대략 S≈0.8M으로 가정하면, M : SBL=18M : 19.2M(a=-3인 타원곡선의 경우 16M : 17.6M). 그러나 Montgomery 알고리즘의 경우는 메모리가 충분하다고 하더라도 윈도우 알고리즘과 같은 확장이 불가능하다.

사용 가능한 임시 메모리에 좀 더 여유가 있는 경우라면 작은 크기의 윈도우에 대해 윈도우 알고리즘 SW를 고려해 볼 수 있다. 예를들어 윈도우 크기가 2인 경우 추가로 2개 정도의 타원곡선 점을 저장할 임시 메모리만 있으면 되고(총 7개의 유한체 원소를 위한 저장공간 필요), 대략 비트당 8M+7.5S≈14M (8M+5.5S≈12.4M if a=-3) 정도의 계산량이 소요된다. Montgomery 알고리즘에 비해 3개 정도의 임시 메모리를 더 사용하여 20%이상 성능을 향상시킬 수 있음을 알 수 있다. 윈도우 크기 3의 경우는 유한체 원소를 위한 저장장소가 11개로 증가하고 대신 비트당 계산량은 약 12.3M(10.7M if a=-3)으로 줄어든다. 이 경우는 Montgomery 알고리즘에 비해 30% 이상의 성능향상을 얻을 수 있다.

사전계산 점들은 아핀 좌표계의 값들로 저장하면 되므로 160비트 타원곡선의 경우 w=2이면 대략 140바이트, w=3이면 220바이트 정도의 추가 메모리가 필요하다. 다른 필요한 임시 메모리까지 고려하더라도 대략 512바이트 정도면 윈도우 크기 2나 3인 알고리즘은 충분히 구현할 수 있을 것이다.

Ⅶ. 결 론

본 논문에서는 기존에 제안된 전력 소모량 분석에 안전한 타원곡선 상수배 알고리즘들의 잠재적인 문제점들을 지적하고, ±1-이진 부호화 엔코딩을 이용하는 보다 안전하고 효율적인 알고리즘을 제안하였다. 제안 방식은 더미 연산을 사용하지 않고도 균일한 실행패턴을 얻을 수 있으며, 기존 방식들에서 존재하는 잠재적인 문제점들도 자연스럽게 해결하였다. 또한 고정 윈도 알고리즘은 사전계산 점들을 반복 사용하는 특성으로 인해 한번의 전력 소모량 측정만으로도 DPA 공격에 취약할 수 있는데, 이는 기존의 DPA 대응책으로는 막는 것이 불가능하였다. 본 논문에서는 이에 대한 한 대응책으로 타원곡선 덧셈내의 랜덤 변수들을 이용하여 유한체 곱셈을 랜덤화시키는 방법을 제안하였다. 따라서 제안된 부호화 엔코딩 기반의 윈도우 알고리즘은 알려진 SPA/DPA 공격에 안전할 뿐더러, 이용 가능한 임시 메모리의 용량에 따라 적절한 윈도우 크기를 선택적으로 사용하여 성능을 높일 수 있으므로 매우 유용할 것으로 생각된다.

참 고 문 헌

[1] C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems," *Advances in Cryptology-Crypto'96*, LNCS 1109, Springer-Verlag, pp. 104~113, 1996.

[2] C. Kocher, "Differential power analysis," *Advances in Cryptology-Crypto'99*, LNCS 1666, S.V., pp. 388~397, 1999.

[3] J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," *Cryptographic hardware and embedded systems - CHES'99*, LNCS 1717, S.V., pp. 292~302, 1999.

[4] T. Izu and T. Takagi, "A fast parallel elliptic curve multiplication against side channel attacks," *Public Key Cryptography-PKC'02*, LNCS 2274, S.V., pp. 280~296, 2002.

[5] K. Okeya and K. Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attacks," *Cryptographic hardware and embedded systems - CHES 2001*, LNCS 2271, S.V., pp. 178~190, 2001.

[6] M.Joye and C.Tymen, "Protections against differential analysis for elliptic curve cryptography - An algebraic approach," *Cryptographic hardware and embedded systems - CHES 2001*, LNCS 2271, S.V., pp. 377~390, 2001.

[7] C. D. Walter, "Mist: Am efficient, randomized exponentiation algorithm for resisting power analysis," *Cryptographic hardware and embedded systems-CHES 2001*, LNCS 2271, S.V., pp. 53~66, 2001.

[8] E. Oswald and M.Aigner, "Randomized addition-subtraction chains as a countermeasure against power attacks," *Cryptographic hardware and embedded systems-CHES 2001*, LNCS 2271, S.V., pp. 39~50, 2001.

[9] IEEE, "P1363: Standard specifications for public key cryptograpgy," D13, 1999.

[10] J. Lopez and R. Dahab, "Improved algorithms for elliptic curve arithmetic in GF(2^m)," Selected Areas in Cryptography-*SAC'98*, LNCS 1556, S.V., pp. 201~212, 1999.

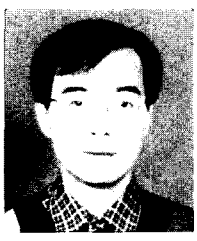
[11] J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over GF(2^m) without precomputation," *Cryptographic hardware and embedded systems-CHES'99*, LNCS 1717, S.V., pp. 316~327, 1999.

[12] C. H. Lim and H. S. Hwang, "Fast implementation of elliptic curve arithmetic in GF(p^n)," *Public Key Cryptography-PKC 2000*, LNCS 1751, S.V., pp. 405~421, 2000.

[13] H. Cohen, *A course in computational number theory*, GTM 138, 3rd edition, S.V., 1996.

[14] C. D. Walter, "Sliding windows succumbs to big mac attack," *Cryptographic hardware and embedded systems-CHES 2001*, LNCS 2271, S.V., pp. 286~299, 2001.

〈著者紹介〉



임 채 훈 (Chae Hoon Lim) 정회원

1989년 2월 : 서울대학교 전자공학과 졸업
1992년 2월 : 포항공과대학교 전자전기공학과 석사
1996년 2월 : 포항공과대학교 전자전기공학과 박사
1996년 3월~2002년 2월 : (주)퓨처시스템 암호체계센터
2002년 3월~현재 : 세종대학교 인터넷학과 조교수
〈관심분야〉 암호 알고리즘/프로토콜 설계/분석, 인터넷 보안