Pollard-ρ에 기반한 타원곡선 이산대수문제 공격법들의 분석 및 병렬화 구현*

서 병 국†, 이 은 정‡, 이 필 중†
†포항공과대학교 전자전기공학과
‡포항공과대학교 정보통신 연구소

Analysis and Parallelization of Pollard-ρ based Attacks on ECDLP

Byung Kuk Suh†, Eun Jeong Lee‡, Pil Joong Lee†
†Dept. of Electronic and Electrical Eng., POSTECH, Pohang, Korea
‡POSTECH Information Research Laboratories, Pohang, Korea

요 약

암호해독법은 암호시스템의 안전성을 논하는데 필수적이다. 본 논문에서는 ECDLP 공격법인 Pollard-ρ와 그 변형들간의 성능을 유한체 $GF(2^{19}) \sim GF(2^{41})$ 상의 타원곡선에서 측정 비교 하였다. 또한 이 공격법을 네트웍을 통해 10대의 컴퓨터로 병렬처리해 공격시간을 $\frac{1}{10}$로 단축시켰으며 실험 데이타를 토대로 $GF(2^{163})$상에서 공격시간 및 저장용량을 예측하였다.

1 서 론

1985년에 Koblitz [9]와 Miller [13]에 의해서 독립적으로 제안된 타원곡선 암호시스템(ECC, Elliptic Curve Cryptosystem)은 공개키 암호시스템이며 타원곡선 이산대수문제(ECDLP, Elliptic Curve Discrete Logartihm Problem)의 어려움에 기반을 두고 있다. 타원곡선은 수학에서 소수성 판별이나 소인수분해 등의 연구에 유용하게 사용되어 왔었다. ECC의 장점으로는 현재 가장 많이 사용되고 있는 RSA에 비해서 같은 암호학적 안전도를 유지하기 위한 키길이가 현저히 작고 연산이 효율적이라는 점에 있다. 따라서 저장용량 및 대역폭(bandwidth)의 제한이 있는 smart card나 무선통신에 유용하게 사용될 수 있으며 기존의 공개키 암호시스템이 사용된 곳에 모두 적용될 수 있다.

ECC의 암호해독법은 ECDLP에 대한 공격을 의미하며 여러 공격법 중에 가장 효과적인 방법은 Pollard-ρ [16] 라고 알려져 있다. 또한 이러한 암호해독 분야에 네트웍을 통한 병렬 및 분산처리를 이용하는 방법들이 이용되고 있다 [4, 5, 26].

*본 연구는 정보통신연구관리단의 국책기술 개발사업(타원곡선 암호시스템을 이용한 차세대 정보보호 기술의 연구개발)의 지원에 의해 이루어졌다.

본 논문에서는 Teske [25]와 Wiener [27]에 의해서 각각 제안된 Pollard-ρ의 개선 방법을 실험을 통해 서로간의 성능 비교를 하였다. 그리고 [17]의 distinguished point개념을 도입하여 Pollard-ρ에 기반한 방법들이 time-memory trade-off 관계가 아닌 어느 특정한 지점에서 최소의 공격시간이 존재함을 보였다. 또한 ECDLP 공격을 병렬처리로 구현하고 그 공격시간이 프로세서 개수에 따라서 선형적인 속도증가가 됨을 확인하였다.

2 용어정리

이진 유한체(binary finite field) $GF(2^m)$에서 ECC에 사용되는 short Weierstrass 식은 다음과 같다.

$$y^2 + xy = x^3 + ax^2 + b\,, \qquad a, b \in GF(2^m),\ b \neq 0 \tag{1}$$

타원곡선 E는 식 (1)을 만족하는 $(x, y) \in GF(2^m) \times GF(2^m)$과 무한대 점인 $\mathcal{O}$로 구성된다. 본 논문에서 사용될 용어를 정리하면 다음과 같다.

p	소수이며 암호학적으로 안전하게 쓰일 수 있을만한 크기
q	$q = p^m$, $m \geq 1$
E	유한체 $GF(q)$에서 식 (1)을 만족하는 타원곡선
#E	E의 위수(order) 이며 E에 속하는 점들의 개수
n	#E를 나누는 소수 n
G	위수 n을 갖는 E의 점이다. 보통 타원곡선 암호시스템은 G로 생성되는 cyclic group $< G >$에서 구현된다.
Q	E의 한 점, 보통 $Q = lG$, $0 \leq l < n$
$Q_1 + Q_2$	E에서 정의되는 타원곡선 점들의 덧셈
lQ	Q를 $l \in \mathbb{Z}_n$번 덧셈 하는 것$(\underbrace{Q + Q + \ldots + Q}_{l})$을 상수배(scalar multiplication)로 표기한다.
$\in_R$	'$x \in_R S$'는 집합 S에서 원소 x를 랜덤하게 선택한다는 의미
$\|n\|$	정수 n의 bit 크기

3 타원곡선 이산대수문제(ECDLP)

유한체 상의 이산대수문제(DLP, Discrete Logarithm Problem)와 타원곡선 상의 이산대수문제(ECDLP, Elliptic Curve Discrete Logarithm Problem) 는 다음과 정의 된다.

- **DLP** 유한체 $GF(q)$가 주어져 있고 $g, h \in GF(q)^*$ 에 대해서 $g^l = h \in GF(q)$를 만족하는 $0 \leq l < q - 1$을 찾는 문제
- **ECDLP** 유한체 $GF(q)$위에서 정의된 타원곡선 E$(GF(q))$와 소수 위수 n을 갖는 타원곡선 위의 점 $G \in$ E$(GF(q))$ 로 생성되는 부분그룹 $< G >$가 주어져 있을 때 점 $Q \in < G >$에 대해서 $lG = Q$를 만족하는 $0 \leq l < n$을 찾는 문제

DLP와 ECDLP를 비교하면 전자는 $g^l (= \underbrace{g \cdot g \cdots g}_{l})$과 같은 곱하기 기호를 사용하고 후자는 $lG (= \underbrace{G + G + \ldots + G}_{l})$와 같이 더하기 기호를 사용하는 연산자의 차이를 제외하고는 개념상으로는 유사

한 문제이다. DLP는 index-calculus 방법으로 subexponential 시간 안에 공격이 가능하다 [19]. 그러나 ECDLP는 index-calculus 방법으로 subexponential 시간 안에 풀릴 수 없음이 밝혀졌다 [22]. 지금까지 알려진 ECDLP 공격을 소개하고 각각의 방법에 필요한 계산량과 저장용량을 기준으로 간략히 정리하겠다. 자세한 알고리즘은 참고문헌에 소개되어 있다.

1. baby-step giant-step algorithm [21]

   Exhaustive search에서의 time-memory trade-off 관계가 적용된다. 즉, 기억용량을 늘리면 공격에 필요한 시간이 짧아지고, 기억용량을 줄이면 공격에 필요한 시간이 길어진다. 기억용량은 $O(\sqrt{n})$, 계산량 역시 $O(\sqrt{n})$만큼 든다.

2. Pohlig-Hellman algorithm [15]

   이 알고리즘은 먼저 그룹의 위수 n을 $\prod_{i=1}^{t} p_i^{\lambda_i}$로 소인수분해하고 $i = 1, \ldots, t$에 대해서 $l \bmod p_i^{\lambda_i}$를 계산한 뒤 CRT(Chinese Remainder Theorem)를 이용하여 $l \bmod n$을 구한다. 계산 복잡도는 $O(\sum_{i=1}^{t} \lambda_i(\log n + \sqrt{p_i}\log p_i))$ 정도 되므로 #E가 smooth(작은 소수들의 곱으로 이루어짐)할 때 효과적인 방법이다.

3. Pollard-ρ algorithm [16]

   이 알고리즘은 baby-step giant-step 알고리즘의 랜덤화된 버전이다. 계산량은 $\sqrt{\frac{\pi n}{2}}$, $O(\sqrt{n})$으로서 baby-step giant-step과 비슷하나, 저장용량을 거의 필요로 하지 않는다는 점에서 장점 때문에 ECDLP를 해결하는 가장 좋은 방법으로 여겨지고 있다.

4. Distributed version of Pollard-ρ algorithm [26]

   Van Oorschot 와 Wiener는 Pollard-ρ 알고리즘이 병렬처리화(parallelized) 될 수 있음을 보였다. 만일 m 개의 processor를 이용하여 병렬화 된다면 계산량은 $\sqrt{\frac{\pi n}{2}}/m$, $O(\sqrt{n})$ 로서 선형적인 속도 개선(linear speed-up)을 볼 수 있다.

5. MOV attack [14]

   Menezes, Okamoto, Vanstone은 일반적인 타원곡선 $E(GF(q))$에 대해서 ECDLP 문제가 유한체 $GF(q^B)$ $(B \geq 1$, B는 정수) 상의 DLP 문제로 바뀔 수 있음을 보였다. 만일 이러한 경우 B가 작다면 유한체에서의 공격법인 index-calculus를 이용해서 subexponential 시간 안에 문제를 풀 수 있다. 이에 따라 [1, 2]에서는 $B \geq 20$을 만족해야 안전성을 보장할 수 있다고 권고하였다. 하지만 초특이 타원곡선(supersingular curve)인 경우는 $B \leq 6$이라고 알려져 있기 때문에 효과적인 MOV attack이 가능하다고 할 수 있다.

6. 변칙 타원곡선(anomalous curves)에 적용되는 공격

   변칙 타원곡선은 $\#E(GF(q)) = q$ 인 특징을 갖고 있다. Semaev [20], Smart [23], Satoh와 Araki [18]는 $q = p^m$, $p \neq 2$인 변칙 타원곡선에 대해서 $O((\log q)^3)$의 계산 복잡도로 공격이 가능함을 보였다.

4 Pollard-ρ와 그의 개선된 변형들

4.1 Pollard-ρ

ECDLP를 공격하는데 효과적인 Pollard-ρ는 점 $G, Q\,(=lG)$가 주어졌을 때 logarithm l을 찾는 방법이다. 이 Pollard-ρ는 일반적으로 작은 정수 r에 대해서 군(group)이 r개의 disjoint하고 크기가 비슷한 집합 $S_1, \ldots, S_r$ 로 나눌 수 있을 때 사용할 수 있다.

먼저 타원곡선 점들의 집합을 동일한 크기의 S_1, S_2, S_3의 3개로 나눈다. 이를 효과적으로 하는 방법은 § 5.1에서 다룬다. 다음과 같이 정의하는 함수

$$f_P(Z) = \begin{cases} 2Z & \text{if } Z \in S_1 \\ Z+G & \text{if } Z \in S_2 \\ Z+Q & \text{if } Z \in S_3 \end{cases} \tag{2}$$

를 *iterating* 함수라 부르고 식 (2)의 계산 회수를 *iteration* 회수라 한다. $f_P(Z)$를 통해서 타원곡선 점 Z_i와 $a_i, b_i \in \mathbb{Z}_n^*$ 이 세 가지로 구성된 (Z_i, a_i, b_i)-tuple이 매 iteration 마다 만들어지며 다음 식을 만족한다.

$$Z_i = a_i G + b_i Q \tag{3}$$

따라서 $f_P(Z)$와 식 (3)을 종합하면 $i+1$번 째의 tuple을 구하는 식을 다음과 같이 정리 할 수 있다.

$$(Z_{i+1}, a_{i+1}, b_{i+1}) = \begin{cases} (2Z_i, 2a_i, 2b_i) & \text{if } Z_i \in S_1 \\ (Z_i+G, a_i+1, b_i) & \text{if } Z_i \in S_2 \\ (Z_i+Q, a_i, b_i+1) & \text{if } Z_i \in S_3 \end{cases}$$

위 식을 이용해 시작점 $Z_0 = a_0 G + b_0 Q$, $a_0, b_0 \in_R \mathbb{Z}_n^*$를 구한 뒤 $(Z_i, a_i, b_i) - \text{tuple}, (i = 0, 1, \ldots)$ 시퀀스를 계산한다. 타원곡선 점은 유한하므로 이러한 시퀀스의 Z_i는 회람문자 ρ처럼 원을 그리며 $Z_{j \neq i}$와 반드시 만나 충돌한다(collision). 충돌이 생기면 $Z_i = Z_j$, $(i \neq j)$를 만족하고 식 (3)에 따라 $a_i G + b_i Q = a_j G + b_j Q$을 만족하므로 logarithm $l\,(Q = lG)$을 다음과 같이 구할 수 있다.

$$l = \frac{a_i - a_j}{b_j - b_i} \bmod n$$

이 알고리즘은 시퀀스의 모든 tuple들을 저장하는데 $O(\sqrt{n})$ 정도의 저장 복잡도와 그 tuple들 사이에서 충돌쌍을 찾는데 $O(\sqrt{n})$정도의 계산 복잡도가 있다. 이 결과는 baby-step giant-step algorithm과 계산 및 저장 복잡도에서 같다. 하지만 저장용량을 Floyd's cycle finding algorithm [7]을 이용해서 아예 필요없게 만들거나 § 5.2에서 소개되는 구분점을 이용하여 크게 줄일 수 있다.

실제 구현시에는 충돌이 생길 때 까지의 계산량(타원곡선 덧셈 연산) 또는 iteration 회수를 예측할 필요가 있는데 Lemma 1을 통해 이론적으로 $\sqrt{\frac{\pi n}{2}}$가 됨을 알 수 있다.

Lemma 1 ([6, 26]) 총 개수가 n개 있는 원소의 집단에서 임의로 골랐을 때 중복되기 전까지의 원소의 개수를 random variable X라고 했을 때 X의 평균값은 다음과 같다.

$$\mathrm{E}(X) \simeq \sqrt{\frac{\pi n}{2}}$$

4.2 Pollard-ρ 개선 I: iterating함수

Teske는 고전적인 Pollard-ρ의 iterating 함수 $f_P(Z)$를 개선함으로써 기존의 것 보다 더 좋은 결과를 낼 수 있다고 밝혔다 [25]. Teske는 $GF(p)$, $p > 3$상의 타원곡선 $y^2 = x^3 + ax + b$에서 $f_P(Z)$를 3가지 유형으로 변형하여 테스트 한 후 그 중 가장 좋은 $f_T(Z)$를 제안하였다 [25]. 참고로 본 논문에서는 Teske가 DLP로 표기한 식을 ECDLP에 적합한 식으로 옮겨 적었다.

이 방법은 타원곡선 점들의 집합을 3이 아닌 20개의 동일한 영역($S_1, S_2, \ldots, S_{20}$)으로 나눈다. $f_T(Z)$는 § 4.1와는 다르게 임의로 선택된 곱셈상수 $m_1, n_1, \ldots, m_{20}, n_{20} \in_R \mathbb{Z}_n^*$을 통해 $f_T(Z)$값의 랜덤성(random walk)에 좋은 영향을 미치도록 하였다. $f_T(Z)$는 다음과 같다.

$$f_T(Z) = Z + m_k G + n_k Q, \quad Z \in S_k, k = 1, \ldots, 20 \tag{4}$$

식 (3), (4)를 만족하는 (Z_i, a_i, b_i)-tuple 을 구하는 식은 다음과 같다.

$$(Z_{i+1}, a_{i+1}, b_{i+1}) = \begin{cases} (Z_i + m_1 G + n_1 Q, a_i + m_1, b_i + n_1) & \text{if } Z_i \in S_1 \\ (Z_i + m_2 G + n_2 Q, a_i + m_2, b_i + n_2) & \text{if } Z_i \in S_2 \\ \vdots & \\ (Z_i + m_{20} G + n_{20} Q, a_i + m_{20}, b_i + n_{20}) & \text{if } Z_i \in S_{20} \end{cases}$$

이 방법이 효과적이려면 Pollard-ρ의 $f_P(Z)$와 $f_T(Z)$에 소요되는 계산량이 적어도 같아야 한다. § 4.1를 참조하면 $f_P(Z)$는 $(Z_{i+1}, a_{i+1}, b_{i+1})$-tuple 계산시 Z_{i+1}에 타원곡선 덧셈 한 번 나머지 a_{i+1} 또는 b_{i+1} 계산시 모듈러 덧셈 한 번의 연산이 필요하다. $f_T(Z)$에서는 Z_{i+1} 계산시 $m_k G + n_k Q$을 사전 계산하여 타원곡선 덧셈 한 번으로 만들고 나머지 a_{i+1}, b_{i+1} 계산은 m_k 또는 n_k를 0으로 만들어 a_{i+1}와 b_{i+1} 2개 중 1개만 바뀌게 하면 $f_P(Z)$와 계산량이 동일하다. 따라서 $f_T(Z)$가 충돌쌍을 찾을 때 까지의 iteration의 회수를 줄인다면 전체적으로 공격시간 단축의 효과를 가져온다. Teske는 실험결과로 다음과 같이 iteration 회수를 비교하였다.

1. Pollard-ρ, $f_P(Z)$: $\simeq 1.596\sqrt{n}$
2. Teske에 의한 개선, $f_T(Z)$: $\simeq 1.292\sqrt{n}$

Lemma 1에 의한 이론적인 수치가 $\sqrt{\frac{\pi n}{2}} \simeq 1.253\sqrt{n}$ 이며 이 값에 가장 근접한 $f_T(Z)$가 가장 좋은 공격 방법임을 알 수 있다.

4.3 Pollard-ρ 개선 II: 검색면적

Wiener는 Pollard-ρ의 공격 속도를 향상시키기 위하여 iterating 함수를 개선하는 대신 검색해야 할 타원곡선 점들을 줄이는 2가지 방법을 제안했다 [27].

4.4 임의의 곡선에 적용되는 빠른 공격

$f_P(Z_i)$를 통해 계산한 점 Z_{i+1}에 대해서 $-Z_{i+1}$을 계산하여 iteration 한 번 마다 점 2개를 검색할 수 있으므로 전체적으로 검색해야할 면적을 전체의 $\frac{1}{2}$로 줄일 수 있다. 음수 점 계산은 $E(GF(p^m))$의 한 점 $Q = (x, y)$에 대해서 $-Q = (x, x + y)$, $(p = 2)$, $-Q = (x, -y)$, $(p > 3)$ 과 같이 매우 간단해 $-Q$계산에 부담이 없다. 따라서 공격시간은 Lemma 1에 의해서 전체의 $\frac{1}{\sqrt{2}}$로 줄어든다. 이 방법을 정리하면 다음과 같다.

1. iterating 함수 $f_P(Z)$를 이용해서 (Z_i, a_i, b_i)-tuple을 구한다.

2. $-Z_i$를 구하여 $Z_i, -Z_i$ 중 y 좌표의 binary representation 값이 작은 점을 선택한다.

3. 만일 $-Z_i$가 선택되었다면 식 $Z_i = a_iG + b_iQ$를 만족할 수 있도록, $a_i \rightarrow -a_i$로 $b_i \rightarrow -b_i$로 각각 교체한다.

4.5 Subfield Curves에 적용되는 빠른 공격

Frobenius endomorphism [10, 12, 27]을 이용하면 *subfield curve* $GF(2^{ed})$에 대해서 검색면적을 $\frac{1}{d}$ 만큼 줄일 수 있다.

Remark. Frobenius endomorphism φ in $\mathsf{E}(GF(2^{ed}))$는 다음과 같이 정의되며

$$\varphi : (x, y) \longmapsto (x^{2^e}, y^{2^e})$$

$\varphi(Q) = \lambda Q$ 인 정수 λ가 존재한다. λ는 $\lambda^2 - t\lambda + 2^e \equiv 0 \pmod{n}$을 만족한다. 여기서 $t = 2^e + 1 - \#\mathsf{E}(GF(2^e))$이다.
또한, $Q = (x, y)$, $x, y \notin GF(2^{ef}), 1 \leq f \leq d-1$에 대하여 다음을 만족하는 equivalance class를 생성한다.

$$\begin{aligned} \varphi(Q) &= \lambda Q \neq Q, \\ \varphi^2(Q) &= \lambda^2 Q \neq Q, \\ &\vdots \\ \varphi^{d-1}(Q) &= \lambda^{d-1} Q \neq Q, \\ \varphi^d(Q) &= \lambda^d Q = Q \end{aligned} \qquad (5)$$

□

위의 Remark를 통해 전체 검색면적을 줄일 수 있는 방법은 다음과 같다 [26].

1. iterating 함수 $f_P(Z_i)$를 이용해서 $(Z_{i+1}, a_{i+1}, b_{i+1})$ tuple 을 구한다.

2. $2d$ 개의 점 $\pm\varphi^j(Z_i)$, $0 \leq j \leq d-1$ 를 구한다.

3. 2의 $2d$개 중 양수인 d개의 $\varphi^j(Z_i)$ 점에서 x 좌표의 binary representation이 가장 작은 점을 선택한다.

4. 3에서 선택한 $\varphi^j(Z_i)$값과 그 음수 값 $-\varphi^j(Z_i)$중에서 y 좌표의 binary representation이 작은 것을 선택한다.

5. 선택된 점에 따라서 $Z_i = a_iG + b_iQ$ 를 만족할 수 있도록 $a_i \rightarrow \pm\lambda^j a_i$, $b_i \rightarrow \pm\lambda^j b_i$로 변환한다.

공격을 위한 검색면적은 § 4.4에 의해 $\frac{1}{2}$, Frobenius endomorphism에서 $\frac{1}{d}$ 줄어들어 총 전체의 $\frac{1}{2d}$로 감소한다. 따라서 공격시간은 Lemma 1에 의해 $\frac{1}{\sqrt{2d}}$만큼 감소한다. 위의 공격법을 *Koblitz(Anomalous Binary Curve)* [10]에 적용시키면 $e = 1, d = m$이 되어 공격시간을 전체의 $\frac{1}{\sqrt{2m}}$만큼 감소 시킬 수 있어 효과가 극대화 된다. 효과적인 구현을 위해 λ^j의 값들은 사전 계산하여 사용한다. 또한 $\varphi(Q)$의 계산은 제곱으로 이루어져 있으므로 정규기저(normal basis)를 이용하면 단순한 cyclic shift로 빠른 계산이 가능하다.

5 구현시 효과적인 방법들

5.1 타원곡선 점들의 균등 분할

Pollard-ρ에 기반한 ECDLP 공격법에서는 무엇보다도 $\mathrm{E}(GF(q))$를 균등하게 분할하여 $S_1, \ldots, S_r$, ($r \in \{3, 20\}$) 중 특정한 S_r에 iterating 함수의 계산이 집중되지 않도록 하는 것이 중요하다.

[8]의 multiplicative 해쉬함수 이론을 이용해서 다음과 같이 $\mathrm{E}(GF(q))$를 S_i들로 분할할 수 있다 [25].

$\mathrm{E}(GF(q))$에서 $A = (\sqrt{5}-1)/2$를 자리수 $2 + \lfloor \log_{10}(qr) \rfloor$까지 계산한다. $u^*(Q)$함수를 다음과 같이 정의한다.

$$u^* : \mathrm{E}(GF(q)) \mapsto [0,1), \quad Q = (x,y) \mapsto \begin{cases} Ay - \lfloor Ay \rfloor & \text{if } Q \neq \mathcal{O}, \\ 0 & \text{if } Q = \mathcal{O}, \end{cases}$$

또한 $u^*(Q)$를 이용하여 $u(Q)$함수를 정의하면

$$u : \mathrm{E}(GF(q)) \mapsto \{1, \ldots, r\}, \quad u(Q) = \lfloor u^*(Q) \cdot r \rfloor + 1$$

이고, S_i에 속하는 점 Q의 집합은 다음과 같이 구할 수 있다.

$$S_i = \{Q \in \mathrm{E}(GF(q)) : u(Q) = i\}$$

5.2 구분점을 이용한 병렬화 및 시퀀스 길이 제한

Pollard-ρ를 구현하는 방법으로 저장용량을 필요로 하지 않는 Floyd의 cycle-finding 알고리즘 [7]이 있지만 이는 공격시간을 볼 때 그 효율성에서 떨어지고 반대로 시퀀스에서 약간의 점들을 저장하면($O(\sqrt{n})$보다는 작게) 공격시간을 줄일 수 있다고 알려져 있다 [25, 26]. Quisquater와 Delescaille는 DES 공격에서 어떤 구별되는 성질을 갖는 점들(distinguished points, 구분점)만 저장하여 효과적인 공격을 가능케 하는 방법을 제시했다 [17]. 예를 들어 $\mathrm{E}(GF(2^{23}))$의 공격에서 시퀀스의 점들 $Z_i = (x,y)$중 $y \in GF(2^{23})$ 좌표의 MSB부터 8 bit가 0으로 채워진 점을 구분점으로 정한다. 만일 x를 구분점으로 이용 $Z_i = (x,y)$, $Z_j = -Z_i = (x, x+y)$ 2개 모두 구분점으로 저장되었을 때 Z_i, Z_j 두 점 사이는 물론 다른 점들과도 충돌이 발생치 않는다. 따라서 x를 y대신 구분점으로 채택하면 공격시간 및 저장용량 면에서 좋지 않다.

Pollard-ρ의 병렬화는 구분점을 이용해 효과적으로 구현될 수 있다 [26]. 단순히 m개 프로세서 각각에서 동시에 충돌쌍을 찾을 때를 가정하면 각 프로세서에서의 계산량은 단지 $\sqrt{\frac{\pi n}{2}} \rightarrow \sqrt{\frac{\pi n}{2m}}$으로 줄어들뿐 선형적인 속도증가($\frac{1}{m}$만큼 시간의 감소)를 가져오지 못한다. 반면, 구분점을 이용해 m개 프로세서 각각에서 구분점을 찾아 중앙으로 모아 공격하면 결과적으로 하나의 프로세서가 iteration을 통해 검색해야 할 구분점 개수와 계산량을 전체 m개에서 $\frac{1}{m}$씩 일을 나누어 한 것과 동일하다. 따라서 각 프로세서에서의 계산량은 $\sqrt{\frac{\pi n}{2}} \rightarrow \sqrt{\frac{\pi n}{2}}/m$ 로 줄어 m만큼의 선형적인 속도증가 효과를 볼 수 있다.

구분점을 이용한 Pollard-ρ 공격에서 구분점을 찾게 되면 식 (3)의 a_0, b_0를 다시 생성해 또 다시 구분점을 찾을 때까지 시퀀스를 만들어 나간다. 하지만 이러한 시퀀스가 구분점을 만들어 내지 못한채 $Z_1 \rightarrow Z_2 \rightarrow \cdots \rightarrow Z_1$과 같이 다시 시작점 자신(똑같은 a_i, b_i를 갖는)으로 반복하는 경우(trivial cycle)가 발생할 수도 있다. 따라서 이 문제를 해결하려면 구분점이 발생할 때 까지의

field	a, b	order (#E)	largest prime factor (n)
$GF(2^{19})$	1, 1	0x8031E($\simeq 2^{19}$)	0x4018F($\simeq 2^{18}$)
$GF(2^{23})$	1, 1	0x80144E($\simeq 2^{23}$)	0x400A27($\simeq 2^{22}$)
$GF(2^{37})$	0, 1	0x200008262C($\simeq 2^{37}$)	0xDBEB99F($\simeq 2^{28}$)
$GF(2^{41})$	0, 1	0x2000023387C($\simeq 2^{41}$)	0x800008CE1F($\simeq 2^{39}$)

표 1: 실험에 사용된 유한체와 타원곡선 위수

시퀀스 길이의 적정한 제한값이 필요하다. 일반적으로 $\mathsf{E}(GF(2^m))$ 상의 한 점을 임의로 선택했을 때 y 좌표 상위 l bit가 0으로 채워져 있는 구분점의 확률은 $\theta = \frac{2^{m-l}}{2^m} = \frac{1}{2^l}$ 이 된다. 또한 θ를 이용해서 구분점 발견 까지의 평균 시퀀스 길이를 구하면 기하 분포(geometric distribution)를 따르므로 $\frac{1}{\theta} = 2^l$이 된다. 본 실험에서는 시퀀스 길이의 제한을 $\frac{10}{\theta} = 10 \cdot 2^l$로 두어 이 제한을 넘는 확률을 $(1-\theta)^{\frac{10}{\theta}} \simeq e^{-10} = 0.0045\%$ 정도로 맞추어 사용하였다.

5.3 유한체 및 타원곡선 선택

실험에 사용한 타원곡선은 § 4.5에서 소개된 Frobenius endomorphism을 통한 공격이 가장 효과적으로 나타날 수 있는 *Koblitz* 곡선[10, 24]을 선택했다. 이 곡선의 특징은 $GF(2^m)$상의 타원곡선 식 $y^2+xy = x^3+ax^2+b$에서 $a = 0$ 또는 1 이고 $b = 1$인 형태이며 $m = ed$에서 $e = 1$이므로 φ계산이 $\varphi : (x, y) \mapsto (x^2, y^2)$로 매우 간단하다. 또한 이 곡선은 $a, b \in GF(2)$이므로 $\#\mathsf{E}(GF(2^m))$은 *Weil conjecture*를 이용해 쉽게 구할 수 있다 [10]. 선택된 타원곡선 각각에 대해서 MOV attack [14]에 대한 안전성을 체크하였다.

실험에 사용된 타원곡선들은 표 1과 같다. C++로 구현된 타원곡선 연산 라이브러리, LiDIA [11]를 이용 SPARC Ultra-2 200MHz에서 실험했다. 실험결과는 G, Q를 달리하며 각각의 타원곡선에서 100번씩 공격하여 평균값을 얻었다.

6 실험 결과

6.1 제안된 방법들의 비교 분석

본 절에서는 제안된 방법들의 실험결과를 토대로 성능을 비교 한다. 실험 방법은 제안 방법들의 특징을 살리기 위해 시퀀스의 모든 점들을 저장($O(\sqrt{n})$) 비교하는 방식을 채택하며 시퀀스 길이 제한이나 구분점 등의 방법은 사용하지 않았다. 구현시 점들 저장시 메모리에 구현한 해쉬테이블[1]을 이용하였다. 또한 $GF(2^{41})$에서는 저장해야할 점들의 양이 사용 컴퓨터의 용량을 넘어 제외 시켰다. 실험 대상은 다음과 같다.

- P_3 : Pollard-ρ, iterating 함수 $f_P(Z)$
- T_{20} : Teske에 의해서 § 4.2에서 제안된 방법, iterating 함수 $f_T(Z)$
- W_N : Wiener에 의해서 § 4.4에서 제안된 방법, iterating 함수 $f_T(Z)$

[1] 검색하는데 걸리는 복잡도를 $O(1)$로 만든다.

field ($GF(2^m)$)	P_3	T_{20}	W_N	W_{FR}	$\sqrt{2m}$	$\frac{I_{T_{20}}}{I_{W_{FR}}}$	$\frac{t_{T_{20}}}{t_{W_{FR}}}$
$GF(2^{19})$ iteration(I)	518	474	354	108	6.16	4.15	3.73
시간(t)	1.32s	1.19s	0.91s	0.32s			
$GF(2^{23})$ iteration(I)	1937	1896	1440	401	6.78	4.72	3.41
시간(t)	5.61s	5.08s	3.95s	1.49s			
$GF(2^{37})$ iteration(I)	17355	14213	10723	1985	8.60	7.16	6.10
시간(t)	114.00s	86.91s	64.53s	14.24s			

표 2: iteration 회수 및 공격시간

- W_{FR} : Wiener에 의해서 § 4.5에서 제안된 방법, iterating 함수 $f_T(Z)$

실험 결과는 표 2와 같다. 표에서 iteration 회수는 I, 공격시간은 t로 표기하였다. 공격시간은 $W_{FR} < W_N < T_{20} < P_3$ 순서대로 W_{FR}이 가장 성능이 좋다. E($GF(2^m)$)에서 W_{FR}은 이론상 T_{20}에 비해 $\sqrt{2d} = \sqrt{2m}$ 만큼 I, t가 줄어들어야 한다. 결과를 살펴보면 W_{FR}이 T_{20}에 비해서 I가 감소하는 정도를 보이는 값 $\frac{I_{T_{20}}}{I_{W_{FR}}}$이 $\sqrt{2m}$ 보다 약 20%정도 작다. 또한 t 감소 비 $\frac{t_{T_{20}}}{t_{W_{FR}}}$는 $\frac{I_{T_{20}}}{I_{W_{FR}}}$ 보다 값이 작다. 그 이유로는 W_{FR}에서 매 iteration 마다 Frobenius endomorphism 계산($\varphi(Z)$)에 시간이 걸리기 때문이다.

W_N은 T_{20}보다 이론상 $\sqrt{2}$정도 공격시간 감축이 일어나야 한다. 결과는 $\frac{I_{T_{20}}}{I_{W_N}}$이 $\sqrt{2}$보다 약 7% 정도 작다. 대신 $Q \to -Q$계산에 부담이 없어 $\frac{I_{T_{20}}}{I_{W_N}} \simeq \frac{t_{T_{20}}}{t_{W_N}}$ 이 된다.

Teske는 $GF(p)$상의 타원곡선에서 T_{20}이 P_3에 비해서 성능이 향상되었다고 밝혔다 [25]. 실험 결과를 통해 이 사실이 본 실험에서도 적용됨을 확인 할 수 있다. 마지막으로 W_{FR}을 제외한 P_3, T_{20}, W_N의 $f(Z)$의 특성으로 미루어볼 때 타원곡선이나 그 선택된 유한체와 관계없이 동일한 결과를 나타낼 것이라고 예측할 수 있다.

6.2 구분점을 이용한 최적의 수행결과

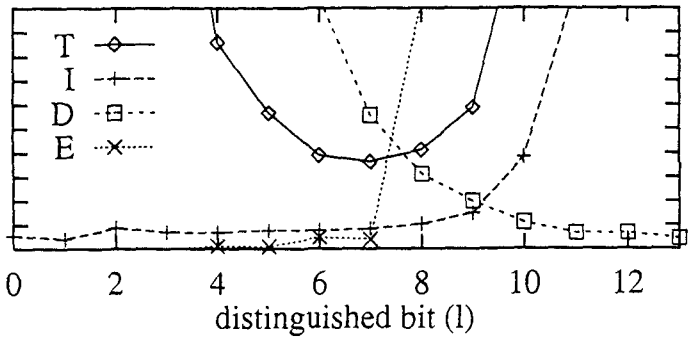

그림 1: P_3에서의 구분점에 따른 영향

field	P_3	T_{20}	W_N	W_{FR}	W_{FRm}
$GF(2^{19})$ 시간	0.26s	0.26s	0.25s	0.32s	0.28s
구분 bit l	5	5	5	-	5
iteration	822	729	720	108	684
$\frac{\text{iteration}}{\sqrt{n}}$	1.604	1.423	1.405	0.211	1.335
저장된 구분점 개수	23	22	21	108	19
$GF(2^{23})$ 시간	0.90s	0.76s	0.73s	1.49s	0.83s
구분 bit l	7	7	7	-	7
iteration	3804	2836	2796	401	3010
$\frac{\text{iteration}}{\sqrt{n}}$	1.856	1.384	1.365	0.196	1.469
저장된 구분점 개수	28	21	22	401	22
$GF(2^{37})$ 시간	6.64s	5.50s	4.94s	14.24s	5.64s
구분 bit l	9	10	9	-	10
iteration	24865	18879	17328	2367	20176
$\frac{\text{iteration}}{\sqrt{n}}$	1.637	1.243	1.141	0.156	1.329
저장된 구분점 개수	48	38	31	2367	18
$GF(2^{41})$ 시간	296s	234s	227s	680s	249s
구분 bit l	12	12	12	-	12
iteration	1187753	915982	821539	101865	937245
$\frac{\text{iteration}}{\sqrt{n}}$	1.602	1.235	1.108	0.137	1.264
저장된 구분점 개수	286	226	199	101865	230
$\frac{\text{평균 iteration}}{\sqrt{n}}$	1.675	1.321	1.254	0.175	1.349

W_{FR}은 구분점 개념을 도입하지 않는다.

표 3: 구분점을 채택한 각 유한체에서의 최적의 공격결과

§ 5.2에서 소개한 구분점만 저장 비교하는 방식으로 § 6.1에서 필요한 $O(\sqrt{n})$ 정도의 저장용량을 줄일 수 있다. 또한 점들을 비교하는데 걸리는 시간을 줄여 iteration 회수는 근소하게 늘더라도 전체적으로 공격시간을 줄일 수 있다.

그림 1은 E($GF(2^{23})$)에서 구분 bit l을 증가시킴에 따라 iteration 수(I), 저장되는 구분점 개수(D), 시퀀스 제한을 넘는 비율(E), 공격시간(T)을 각각의 최고 값으로 normalize 시킨 결과이다. I, D, E의 변화에 따라 T가 U자 형태로 최저점(t_{min})이 존재한다. 공격시간 T는 $\propto I, D, E$ 관계에 있으며 그 중 I, D가 가장 영향을 많이 미친다. 따라서 I, D가 만나는 $l = 9$에서 t_{min}을 예측할 수 있다. 하지만 $l = 9$일 때 E가 현격히 증가해 a_0, b_0값을 다시 생성하는 시간이 걸린다. 또한 D증가에 의한 공격시간 증가는 실험결과 6.4.2를 참고하면 $D = O(n^{\alpha}), \alpha < \frac{1}{2}$ 정도로 $I = O(\sqrt{n})$보다 작다[2]. 따라서 이 2가지 이유로 l이 $9 \rightarrow 7$로 옮긴 지점에서 t_{min}이 존재한다.

[2]구분점 비교에 걸리는 시간 역시 iteration당 계산 시간보다 작다.

클라이언트 개수	클라이언트당 구분점 개수	speed-up ratio
1	212	1
2	123	1.72
4	61	3.47
6	37	5.72
8	28	7.57
10	23	9.2

표 4: E($GF(2^{41})$)에서 m 개의 클라이언트에 의한 병렬화

표 3은 § 6.1에서 비교했던 P_3, T_{20}, W_N, W_{FR} 4가지 경우와 W_{FR}을 변형시킨 W_{FRm}, 5개를 비교하였다. 가장 빠른 공격시간을 보이는 것은 W_N으로서 평균 iteration 회수가 $1.254\sqrt{n}$으로 Lemma 1의 이론치 $1.253\sqrt{n}$과 거의 근접하게 나오는 좋은 성능을 보인다. 반면 표 2에서 구분점을 사용하지 않았을 때 가장 빨랐던 W_{FR}은 성능이 가장 좋지 않다. 그 이유로는 W_{FR}은 iteration 회수가 다른 공격법의 17% 밖에 되지 않는 이점에도 불구하고 다항식기저(polynomial basis)에서의 $\varphi(Z)$의 유한체 제곱 계산의 부담이 존재하기 때문이다. 따라서 W_{FR}에 정규기저(normal basis)를 이용해 $\varphi(Z)$의 제곱 계산을 굉장히 효율적으로 할 수 있으나 그 경우 $f(Z)$에서의 타원곡선 연산이 느려지는 것을 염두에 두어야 한다.

W_{FRm}은 기존의 W_{FR}의 $Z_{i+1} = f_T(Z_i) \rightarrow \pm\lambda^j\varphi(Z_{i+1}) \rightarrow$ (비교 및 저장) 의 과정을 $Z_{i+1} = f_T(Z_i) \overset{(Z_{i+1}\text{ 구분점만족?})}{\longrightarrow} \pm\lambda^j\varphi(Z_{i+1}) \rightarrow$ (비교 및 저장) 으로 변형하여 구분점에 대해서만 Frobenius endomorphism을 이용해 W_{FR}에 비해 $\varphi(Z_i)$계산의 회수와 검색면적을 동시에 줄이려 하였다. 그러나 기존의 방법보다 iteration회수가 그리 크게 줄지 않아 별로 효과가 없음이 드러났다.

6.3 병렬화

§ 5.2에서 설명된 병렬화 방법을 구현하여 결과를 확인 하였다. 공격 방법은 W_N을 이용했으며 구현은 Sun Ultra-2(200MHz), Ultra-1(167MHz) 2가지 플랫폼에서 TCP/IP 프로토콜 레벨의 클라이언트/서버를 제작하였다. 클라이언트는 우선 서버에게 접속하여 G, Q와 기타 공통 파라미터들을 받고 '준비' 상태에 있다. 서버는 약속된 클라이언트가 모두 접속하면 '시작' 메시지를 모든 클라이언트에게 브로드캐스팅한다. 클라이언트들은 구분점들을 만들어 서버에게 전달하고 서버는 클라이언트들에게 받은 구분점들을 저장 비교하여 충돌쌍을 찾아 logarithm을 구한다.

표 4는 클라이언트를 1 ~ 10까지 증가시키면서 각 클라이언트당 발생한 구분점의 개수와 speed-up ratio(=클라이언트 1개 인 경우 만든 구분점의 개수를 m개의 클라이언트 각각에서 만든 구분점의 개수로 나눈 값)를 적은 것이다.

각 클라이언트에서 만든 구분점의 개수는 전체 공격시간과 정비례 관계에 있으며(§ 5.2참조) 네트웍 연결등에 걸리는 시간등은 고려치 않았다. 만일 대규모 클라이언트를 이용한 공격 시스템 설계라면 네트웍 부하를 조절하는 기능이 포함되어야 할 것이다. 구현시 클라이언트들간에 W_N의 $f_T(Z)$에 쓰이는 곱셈상수 $m_1, \ldots, m_{10}, n_1, \ldots, n_{10}$ 값이 동일하지 않으면 단일 프로세서에서 공

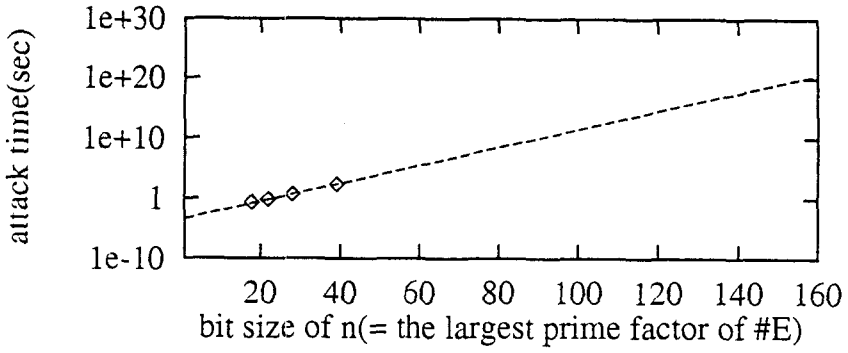

그림 2: W_N의 공격시간 예측(Ultra-2 200MHz)

격하는 것보다 성능이 떨어져 병렬공격이 전혀 효과를 볼 수 없음을 발견했다.

6.4 E($GF(2^{163})$)에 대한 공격결과 예측

소수 위수(prime order)를 갖는 그룹에서 행한 앞의 실험결과들은 적정한 범위의 위수에 대하여 실험이 행하여졌다면 큰 위수로 옮겨가도 그 실험결과의 성능은 유지된다 [25]. 이 가정하에 [1, 2]에서 권고한 안전한 ECC를 위한 최저치 E($GF(2^{163})$), $|n| = 160$ bit에 대하여 표 3의 W_N 결과 값으로 예상 공격시간 및 저장용량을 구할 수 있다.

6.4.1 공격시간

1 MIPS machine이 초당 4×10^4 타원곡선 덧셈을 한다는 가정 [1]을 통해 W_N의 $|n| = 160$ bit일 때의 이론적인 공격 시간은 표 3의 평균 iteration 회수 $1.254\sqrt{n}$을 이용해 다음과 같이 계산 할 수 있다.

$$\frac{1.254\sqrt{2^{160}}}{(4 \times 10^4) \cdot (60 \times 60 \times 24 \times 365)} \simeq 1.2 \times 10^{12} \text{ (MIPS years)}$$

Ultra-2 200MHz에서의 실제 공격시간을 extrapolation을 통해서 그림 2와 다음 예측식을 구할 수 있다.

$$\text{공격시간} = 0.193\, e^{\frac{|n|-18.650}{2.878}}$$

따라서 $|n| = 160$ bit일 때의 공격시간은 약 1.31×10^{13} (machine years) 정도 소요된다. P100MHz에서의 예측은 [3]에 나와 있다.

6.4.2 저장용량

먼저 표 3을 통해 l과 $|n|$의 평균 비율은 $\frac{l}{\log_2 n} \simeq 0.31$ 구할 수 있다. 따라서 임의의 E에서 구분 bit l은 $l \simeq 0.31 \log_2 n$에서 공격시간이 가장 짧다. 참고로 이 값은 구현 방법에 따라 약간의 차이가 있을 수는 있다.

l과 표 3에서 W_N의 평균 iteration 회수 및 § 5.2의 $\theta = \frac{1}{2^l}$을 이용해서 n에 대한 구분점 개수를

구하는 식을 유도할 수 있다.

$$\begin{aligned} \text{구분점 개수} &= \{\text{평균 iteration 회수}\} \cdot \{(\text{구분점이 될 확률}) = \theta\} \\ &\simeq 1.254\sqrt{n} \cdot \frac{1}{2^{0.31\log_2 n}} = 1.254\, n^{0.19} \end{aligned} \tag{6}$$

$|n|$이 160 bit일 때 필요한 저장용량을 식 (6)을 사용해 예측할 수 있다. 구분점 $(Z_i = (x, y), a_i, b_i)$-tuple 하나에 $GF(2^{163})$의 원소 4가지 총 81.5 바이트가 필요하므로 충돌시 까지 발생된 구분점 저장에 $80 \times 1.254 \times (2^{160})^{0.19} \simeq 135$ (GBytes) 정도 필요하다.

6.4.3 병렬화

위의 예측 데이타를 전 세계 1000억대(10^{10})의 컴퓨터로 병렬화 시켰을 때를 각각의 컴퓨터에서 소요되는 공격시간 및 저장용량을 계산 할 수 있다. 공격시간은 $\frac{1.31\times 10^{13}}{10^{10}} = 131$(machine years), 저장용량은 $135(\text{GB})/10^{10} = 14.50(\text{Byte})$ 정도 필요하다. 공격시간에 비해서 저장용량이 너무 작으므로 시스템을 더욱 최적화 시킨다면 어느 정도 공격시간을 줄이면서 저장용량을 늘릴 수 있을 것이다.

7 결 론

본 논문에서는 ECDLP 공격 방법인 Pollard-ρ와 그 변형들을 테스트하여 각 방법들 사이의 효율성에 대해서 논할 수 있었다. 또한 이 결과를 바탕으로 대강 E($GF(2^{163})$)일 때의 공격 소요시간과 필요한 저장용량을 예측할 수 있었다. 특히 네트웍을 통한 병렬화로 효과적으로 ECDLP를 풀 수 있음을 실험결과로 확인 할 수 있었다. 따라서 ECDLP에 기반한 암호시스템의 공격을 개인용 PC나 W/S등을 통한 병렬 및 분산처리 기법으로 공격시간을 크게 단축시킬 수 있다.

참 고 문 헌

[1] American National Standard X9.62-1998, Public Key Cryptography For The Financial Services Industry : The Ellipctic Curve Digital Signature Algorithm(ECDSA).

[2] American National Standard X9.63-1997, Public Key Cryptography For The Financial Services Industry : Elliptic Curve Key Agreement and Transport Protocols.

[3] Certicom ECC Challenge, 1997, Available from http://www.certicom.com/chal.

[4] DESCHAL. Internet-Linked Copmuters Challenge Data Encryption Standard. Press Release, 1997, Available from http://www.frii.com/ rcv/despr4.htm.

[5] Distributed.net. Secure Encryption Challenged by Internet-Linked Computers. Press Release, Oct. 1997. Available from http://www.distributed.net/pressroom/56-PR.html.

[6] P. Flajolet and A.M. Odlyzko, Random Mapping Statistics, *Advances in Cryptology-Eurocrypt '89*, Springer-Verlag, 1989, pp. 329-354.

[7] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

[8] D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and searching.* Addison-Wesley, Reading, Massachusetts, 1973.

[9] N. Koblitz, Elliptic curve cryptosystems, *Math. Comp.*, vol. 48, 1987, pp.203-209.

[10] N. Koblitz, CM-curves with good cryptographic properties, *Advances in Cryptology*, 1991, pp.187-199.

[11] LiDIA Group, Technische Universität Darmstadt. *LiDIA - A library for computational number theory*, Available from http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[12] A. Menezes, *Elliptic Curve Public key Cryptosystems*, Kluwer Academic Publishers, 1993.

[13] V. Miller, Uses of elliptic curves in cryptography, *Advances in cryptology - Crypto '85*, 218, 1986, pp.417-426.

[14] A. Menezes, T. Okamoto, and S. A. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, *Proceedings of the 23rd ACM Symp. Theory of Computing*, 1991. pp.80-89

[15] S. C. Pohlig and M. Hellman, An improved algorithm for computing logarithm over GF(p) and its cryptographic significance, *IEEE Trans. Inform. Theory*, IT-24, 1978, pp.106-110.

[16] J. M. Pollard, Monte Carlo methods for index computations mod p, *Math. Comp.*, vol. 32, 1978, pp.918-924.

[17] J.-J. Quisquater and J.-P. Delescaille, How easy is collisoin search? Application to DES, *Advances in Cryptology-Eurocrypt '89 Proceedings*, Springer-Verlag, 1989, pp. 429-434

[18] T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, preprint, 1997.

[19] O. Schirokauer, D. Weber, and Th. Denny, Discrete logarithms: *The effectiveness of the index calculus method*, Algorithmic Number Theory, (ANTS-II, Talence, France, 1996), LNCS, vol. 1122, Springer-Verlag, 1996, pp. 337-362.

[20] I. Semaev, Evaluation of discrete logarithms on some elliptic curves, to appear in *Math. Comp.*

[21] D. Shanks, Class mumber, a theory of factorization and genera. *Proc. Symp. Pure Math. 20*, AMS, Proivdence, R.I,. 1971, pp. 415-440

[22] J. H. Silverman, Elliptic Curve Discrete Logarithms and the Index Calculus, *Advances in Cryptology-AsiaCrypt '98*, Springer-Verlag, 1998, pp. 110-125.

[23] N. Smart, Announcement of an attack on the ECDLP, for anomalous elliptic curves, preprint, 1997.

[24] J. Solinas, An improved algorithm for arithmetic on a family of elliptic curves, *Advances in Cryptology-Crypto '97*, Springer-Verlag, 1997, pp. 367-371.

[25] E. Teske, Speeding up Pollard's rho method for computing discrete logarithms, *Technical Report No. TI-1/98*, Technische Hochschule Darmstadt, Darmstadt, Germany, 1998, Available from http://www.informatik.tu-darmstadt.de/TI.

[26] P. van Oorschot and M. J. Wiener, Parallel collision search with cryptanalytic applicatoins, to appear in Journal of Cryptology.

[27] M. J. Wiener and R. J. Zuccherato, Faster Attacks on Elliptic Curve Cryptosystems, contribution to P1363, Entrust Technologies, 1998.