효율적인 $GF(p^m)$ 멱승 연산을 이용한 타원곡선 기저점의 고속 생성

(Fast Generation of Elliptic Curve Base Points Using Efficient Exponentiation over $GF(p^m)$)

이 문 규 †

(Mun-Kyu Lee)

요 약 Koblitz와 Miller가 암호시스템에 타원곡선을 사용할 것을 제안한 이래, 타원곡선 암호에 관한 다양한 연구가 진행되어 왔다. 타원곡선 암호는 타원곡선 상의 점들이 덧셈 연산에 대한 군을 형성한다는 관찰에 기반하고 있는데, 안전한 암호를 실현하기 위해서는 군의 위수에 큰 소수를 인자로 포함하는 적절한 타원곡선을 찾고 이 큰 소수를 위수로 갖는 기저점을 찾는 작업이 매우 중요하다. 현재까지 타원 곡선을 찾거나 해당 군의 위수를 계산하는 방법에 관해서는 많은 연구가 있어 왔으나, 곡선이 주어질 때 기저점을 찾는 문제에 대한 연구 결과는 많지 않다. 이에 본 논문에서는 $GF(p^m)$ 상에서 정의된 타원곡선 상에서 임의의 기저점을 찾는 효율적인 방안을 제시한다. 먼저 우리는 기저점을 찾는 데 있어 가장 중요한 연산이 멱승 연산임을 밝히고, 다음에 $GF(p^m)$ 상에서의 멱승을 빠르게 하기 위한 효율적인 알고리즘들을 제시한다. 마지막으로 이 알고리즘들을 구현하여 다양한 실제 타원 곡선 상에서 실험한 결과들을 제시하는데, 이에 따르면 본 논문에서 제안하는 알고리즘은 이진 멱승에 기반한 기저점 탐색 알고리즘에 비해 탐색 속도를 1.62-6.55 배 향상시킴을 확인할 수 있다.

키워드 : 타원 곡선, 파라미터 생성, 기저점, 최적확장체, 멱승

Abstract Since Koblitz and Miller suggested the use of elliptic curves in cryptography, there has been an extensive literature on elliptic curve cryptosystem (ECC). The use of ECC is based on the observation that the points on an elliptic curve form an additive group under point addition operation. To realize secure cryptosystems using these groups, it is very important to find an elliptic curve whose group order is divisible by a large prime, and also to find a base point whose order equals this prime. While there have been many dramatic improvements on finding an elliptic curve and computing its group order efficiently, there are not many results on finding an adequate base point for a given curve. In this paper, we propose an efficient method to find a random base point on an elliptic curve defined over $GF(p^m)$. We first show that the critical operation in finding a base point is exponentiation. Then we present efficient algorithms to accelerate exponentiation in $GF(p^m)$. Finally, we implement our algorithms and give experimental results on various practical elliptic curves, which show that the new algorithms make the process of searching for a base point 1.62-6.55 times faster, compared to the searching algorithm based on the binary exponentiation.

Key words : Elliptic Curve, Parameter Generation, Base Point, Optimal Extension Field, Exponentiation

1. Introduction

Since the use of elliptic curves in cryptography was first suggested by Koblitz [1] and Miller [2], an extensive research on elliptic curve cryptosystem (ECC) has been done. Especially, there have been many works to improve the security and the efficiency of ECC.

ECC uses the fact that the points on an elliptic curve form an additive group under 'point addition' operation. From the viewpoint of security, an elliptic curve should be chosen so that its group order may

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10957-0) 지원으로 수행되었음. 본 논문은 ICCSA 2006 (LNCS 3983, pp.584-593) 논문의 개정본임

† 종신회원 : 인하대학교 컴퓨터공학부 교수
mklee@inha.ac.kr

논문접수 : 2005년 9월 22일
심사완료 : 2006년 8월 1일

be divisible by a sufficiently large prime, i.e., order $=hr$ for a large prime r and a small integer h [3, 4]. Then all cryptographic protocols are performed over a subgroup generated by a point G of order r, which is called a base point. Actually, the most difficult part of generating elliptic curve parameters is finding such an elliptic curve and an adequate base point of prime order [3].

While there have been many practical improvements on finding an elliptic curve and computing its order efficiently, e.g. [5], [6] and [7], there are not many results on how to find an adequate base point on a given curve.

In this paper, we propose an efficient method to find a random base point on a given elliptic curve. As the underlying field for elliptic curve, we choose Optimal Extension Field (OEF) $GF(p^m)$ [8, 9], which is known as the best choice for software implementation of ECC [10]. Specifically, our contributions are as follows.

- First, we show that the critical operation in finding a base point on an elliptic curve over $GF(p^m)$ is exponentiation.
- Then we present efficient exponentiation algorithms over OEF. Our algorithms are based on the observation that p-th powering is very fast in OEF. Hence we can use several techniques used for exponentiation using a normal basis representation of $GF(p^m)$ [11-13], where p-th powering is almost free.
- We implement our algorithms and give experimental results on various practical elliptic curves, which show that the new algorithms accelerate the searching process for a base point by factors of 1.62-6.55, compared to the searching algorithm based on the square-and-multiply exponentiation algorithm.

2. Optimal Extension Field

2.1 Optimal Extension Field (OEF)

An OEF [8, 9] is a finite field $GF(p^m)$ that satisfies the following:

1. p is a prime less than but close to the word size of the target processor.
2. $p=2^n-c$, where $\log_2 c \le n/2$, and
3. An irreducible binomial $f(x)=x^m-w$ exists.

For a cryptographic application, an odd prime m is used [4]. An element A in an OEF is represented as $A=\sum_{i=0}^{m-1} a_i x^i$ using the polynomial basis representation, where $a_i \in GF(p)$. An OEF enables us to exploit the full computing power of a general purpose processor in software implementation of elliptic curve cryptosystems (ECCs).

2.2 Multiplication and squaring in OEF [8, 9]

A multiplication in OEF is composed of two stages. The first one is an ordinary polynomial multiplication of two OEF elements A and B, producing an intermediate product C' of degree less than or equal to $2m-2$. The schoolbook method to calculate the coefficients of C' requires m^2 multiplications and $(m-1)^2$ additions in the subfield $GF(p)$.

The second stage is the reduction stage where $C' \bmod f(x)$ is calculated to get $C=A \cdot B \bmod f(x) \in GF(p^m)$. By $x^m \equiv w \bmod f(x)$, one can do this using $m-1$ multiplications and $m-1$ additions in $GF(p)$.

Therefore, one multiplication in $GF(p^m)$ requires m^2+m-1 multiplications and m^2-m additions in $GF(p)$.

In the case of squaring, the only difference is that the number of coefficient multiplications in the first stage is reduced to $m(m+1)/2$. Hence, the required number of $GF(p)$ operations is $m^2/2+3m/2-1$ multiplications and m^2-m additions in total.

Note that a multiplication in $GF(p)$ is much more expensive than an addition in $GF(p)$ on most of the current general purpose CPUs. Hence we will ignore the cost for additions in $GF(p)$ throughout this paper.

2.3 p-th powering in OEF [14]

For $A=\sum_{i=0}^{m-1} a_i x^i \in GF(p^m)$, the p-th power is represented as $a_0+\sum_{i=1}^{m-1} a_i x^{ip}$ since $a_i^p=a_i$ in $GF(p)$. By $x^m \equiv w \bmod f(x)$, we get

$$x^{ip} \equiv x^{(ip \bmod m)} w^{\lfloor ip/m \rfloor} \bmod f(x)$$

for $1 \le i \le m-1$, and

$$A^p = a_0 + \sum_{i=1}^{m-1} a_i w^{\lfloor ip/m \rfloor} x^{(ip \bmod m)}.$$

After reordering terms, we obtain a polynomial

basis representation of A^p. Note that we can precompute $w^{\lfloor ip/m \rfloor}$, since p, m, and w are independent of A. Hence, a p-th powering operation can be done by only $m-1$ on-line multiplications in $GF(p)$.

2.4 Square-and-multiply exponentiation in OEF

Exponentiation is to compute A^e for a field element A and a non-negative integer e, and there has been an extensive research on fast exponentiation. Since an exponentiation is composed of many multiplications and squarings, one of the promising approach for fast exponentiation is to reduce the number of underlying field multiplications (and squarings) [15]. The most well-known techniques of this kind include m-ary method [16], window method [16], addition chain method [17], redundant number systems [16], and common multiplicand methods [18, 19]. However, in most cases, the performance gains obtained by these general methods are under 50% compared to the cost of basic square-and-multiply algorithm (a.k.a. binary method) [16], if precomputation [20, 21] is not used.[1] Therefore, in this paper, we set the simplest square-and-multiply algorithm as a standard to estimate the performance of our new exponentiation algorithm, which enables us to do more precise analysis.

Now we begin by giving an analysis on the number of $GF(p)$ operations when we do an exponentiation over its extension field $GF(p^m)$. For an element $A \in GF(p^m)$ and an integer $e = \sum_{i=0}^{l-1} e_i 2^i$, where $l = \lceil \log_2(e+1) \rceil$ and $e_i \in \{0, 1\}$, Algorithm 1 computes A^e using the left-to-right square-and-multiply method.

Algorithm 1. Square-and-multiply exponentiation

1. $B \leftarrow 1$.
2. for i from $l-1$ to 0 do
3. $B \leftarrow B^2$.
4. if $(e_i = 1)$ then $B \leftarrow B \cdot A$.
5. od
6. output B.

If we assume that line 4 is executed $l/2$ times on the average, the number of multiplications in $GF(p)$ in Algorithm 1 is

$$l(m^2/2+3m/2-1)+l(m^2+m-1)/2 = l(m^2+2m-3/2), \quad (1)$$

according to the estimation given in Section 2.2.

3. Base Point Generation and Exponentiation

In this section, we examine the relation between base point generation and exponentiation. An elliptic curve over $GF(p^m)$ is given by

$$E: Y^2 = X^3 + AX + B,$$

where $A, B \in GF(p^m)$ and $4A^3+27B^2 \neq 0$. It is well known that E forms an additive group under point addition operation.

If we select A and B from $GF(p)$ and we still consider X and Y over $GF(p^m)$, then elliptic curve operations, especially scalar multiplications, can be accelerated. We will call this kind of curve a Koblitz-type curve which is a general case of Koblitz curve with $p=2$. For distinction, we will call a general curve with $A, B \in GF(p^m)$ a random curve.

For cryptographic applications, an elliptic curve is chosen so that its group order may be divisible by a sufficiently large prime, i.e., order $= hr$ for a large prime r and a small integer h [3]. (We call h the cofactor.) Then all cryptographic protocols are performed over a subgroup generated by a point G of order r, which is called a base point. Hence, it is crucial to find an adequate base point in cryptographic applications.

Algorithm 2 shows an algorithm that finds a random base point G [3, 4]. In this algorithm, the most time-consuming part is the computation of a square root in line 1.4. (Note that this part may be performed several times.) The cost to compute hP (line 2) is negligible for a typical random curve, since a curve is selected so that $h=1$ in this case. For a Koblitz-type curve, h is selected so that

1) For example, if a 512-bit exponent e is used, the cost for square-and-multiply exponentiation is 768 multiplications on the average. On the other hand, all of the general methods use more than 512 multiplications, since they cannot elliminate the squaring costs completely. If we can use precomputation methods [20, 21] or special kinds of finite field representations such as normal bases [11], then the number of multiplications will drop to under 200.

$h \approx p$. Hence the cost for line 2 is comparable to that of line 1.4 in this case.

Algorithm 2. Finding a random point G of order r

1. Generate a random point F (not O) on E as follows:
   1.1 Choose random $X \in GF(p^m)$.
   1.2 Set $Z \leftarrow X^3 + AX + B$.
   1.3 If $Z = 0$, then go to step 1.1
   1.4 If a square root of Z exists, then set it as Y; otherwise, go to step 1.1.
   1.5 Set $P \leftarrow (X, Y)$.
2. Set $G \leftarrow hP$.
3. If $G = O$, then go to step 1.
4. Output G.

To compute a square root, we use Algorithm 3 or 4 according to the form of p^m [3, 4]. (We omit the case $p^m \equiv 1 \bmod 8$ since our exponentiation algorithm does not apply to this case.) Note that exponentiation over $GF(p^m)$ (line 2 of Algorithms 3 and 4) is the only significant operation in the square root computation from the viewpoint of execution time.

Algorithm 3. Computing a square root of $Z \in GF(p^m)$ $(p^m \equiv 3 \bmod 4)$

1. Set $u \leftarrow (p^m - 3)/4$.
2. Compute $Y \leftarrow Z^{u+1} \in GF(p^m)$.
3. If $Y^2 = Z$, then output Y; otherwise no square root exists.

Algorithm 4. Computing a square root of $Z \in GF(p^m)$ $(p^m \equiv 5 \bmod 8)$

1. Set $u \leftarrow (p^m - 5)/8$.
2. Compute $B \leftarrow (2Z)^u \in GF(p^m)$.
3. Compute $C \leftarrow 2ZB^2$.
4. Compute $Y \leftarrow ZB(C-1)$.
5. If $Y^2 = Z$, then output Y; otherwise no square root exists.

By the above discussion, we see that an efficient exponentiation over $GF(p^m)$ is very important for efficient generation of a base point.

4. New Exponentiation Algorithm in OEF

In this section, we give a new algorithm to compute $A^e \in GF(p^m)$. In our algorithm,

- the exponent e is regarded as a p-ary number, i.e., $e = \sum_{i=0}^{s-1} e'_i p^i$, where $s = \lceil \log_p (e+1) \rceil$ and $0 \le e'_i \le p-1$, and
- each coefficient e'_i is regarded as a binary number, i.e., $e'_i = \sum_{j=0}^{t-1} e'_{ij} 2^j$, where $t = \lceil \log_2 (p+1) \rceil$ and $e'_{ij} \in \{0,1\}$.

Thus we can see A^e as

$$\begin{aligned} A^e &= \left(A^{p^{s-1}}\right)^{e'_{s-1}} \times \left(A^{p^{s-2}}\right)^{e'_{s-2}} \times \cdots \times A^{e'_0} \\ &= \left(A^{p^{s-1}}\right)^{e'_{s-1,t-1}2^{t-1} + e'_{s-1,t-2}2^{t-2} + \cdots + e'_{s-1,0}} \\ &\quad \times \left(A^{p^{s-2}}\right)^{e'_{s-2,t-1}2^{t-1} + e'_{s-2,t-2}2^{t-2} + \cdots + e'_{s-2,0}} \\ &\quad \vdots \\ &\quad \times A^{e'_{0,t-1}2^{t-1} + e'_{0,t-2}2^{t-2} + \cdots + e'_{0,0}}. \end{aligned}$$

Now we can compute A^e in two stages. In the first stage, we construct a p^i-th power table T, i.e., $T_i = A^{p^i}$ for $0 \le i \le s-1$. Here the p-th powering operations are used $s-1$ times. The second stage is a simultaneous square-and-multiply exponentiation of $(T_{s-1})^{e'_{s-1}}, (T_{s-2})^{e'_{s-2}}, \ldots, (T_0)^{e'_0}$. Algorithm 5 shows the complete procedure.

Algorithm 5. Exponentiation in $GF(p^m)$ using efficient p-th powering

1. $T_i \leftarrow A^{p^i}$ for $0 \le i \le s-1$.
2. $B \leftarrow 1$.
3. for j from $t-1$ to 0 do
4. $\quad B \leftarrow B^2$.
5. $\quad$ for i from $s-1$ to 0 do
6. $\quad\quad$ if $(e'_{ij} = 1)$ then $B \leftarrow B \cdot T_i$.
7. $\quad$ od
8. od
9. output B.

The most time-consuming parts of Algorithm 5 are lines 1, 4 and 6. (The execution time for other parts can be ignored.) We estimate the amount of required computation for these lines as follows:

- Since a p-th powering operation is done by $m-1$ multiplications in $GF(p)$, the total amount of computation in line 1 is

$$(s-1)(m-1) \quad (2)$$

multiplications in $GF(p)$.

- Line 4 is executed t times. Hence, according to the estimation given in Section 2.2, the required number of multiplications in $GF(p)$ is

$$t(m^2/2+3m/2-1). \quad (3)$$

- Assuming the half of e'_{ij}'s are one as in the square-and-multiply algorithm, we see that line 6 is executed $st/2$ times. Hence, the required number of multiplications in $GF(p)$ is

$$st(m^2+m-1)/2. \quad (4)$$

5. Optimized Exponentiation for Elliptic Curve Point Generation

In this section we show that our new exponentiation algorithm can be optimized further to produce a base point over an elliptic curve. We will use the fact that the exponents u in Algorithms 3 and 4 are fixed and they have special structures.

First, we consider the case that $p^m \equiv 3 \bmod 4$. Note that p^m-3 can be represented as an m-digit p-ary number, i.e.,

$$(p-1,p-1,\ldots,p-1,p-3),$$

in a vector representation. This can be rewritten as

$$(p-3,3p-1,\ldots,p-3,3p-1,p-3), \quad (5)$$

since $(p-1)p+(p-1)=(p-3)p+(3p-1)$ and m is odd. Because $p^m \equiv 3 \bmod 4$ implies $p \equiv 3 \bmod 4$ and $3p \equiv 1 \bmod 4$, (5) is divisible by 4. Therefore we obtain

$$u=(p^m-3)/4$$
$$=((p-3)/4,(3p-1)/4,\ldots,(p-3)/4,(3p-1)/4,(p-3)/4).$$

Hence we can compute A^u as

$$A^u=\left(A\times A^{p^2}\times\cdots\times A^{p^{m-1}}\right)^{(p-3)/4} \times\left(A^p\times A^{p^3}\times\cdots\times A^{p^{m-2}}\right)^{(3p-1)/4}.$$

Similarly, for the case that $p^m \equiv 5 \bmod 8$, we can compute A^u as

$$A^u=\left(A\times A^{p^2}\times\cdots\times A^{p^{m-1}}\right)^{(p-5)/8} \times\left(A^p\times A^{p^3}\times\cdots\times A^{p^{m-2}}\right)^{(5p-1)/8}.$$

Algorithm 6 shows the complete procedure for these two cases, where $t=\lceil \log_2(p+1) \rceil$.

Algorithm 6. Optimized computation of A^u ($p^m \equiv 3 \bmod 4$ or $p^m \equiv 5 \bmod 8$)

1. if ($p \equiv 3 \bmod 4$) then $e_0 \leftarrow (p-3)/4$; $e_1 \leftarrow (3p-1)/4$.
   else $e_0 \leftarrow (p-5)/8$; $e_1 \leftarrow (5p-1)/8$.
2. $T_0 \leftarrow A\times A^{p^2}\times\cdots\times A^{p^{m-1}}$; $T_1 \leftarrow A^p\times A^{p^3}\times\cdots\times A^{p^{m-2}}$.
3. $B \leftarrow 1$.
4. for j from $t-1$ to 0 do
5. $\quad B \leftarrow B^2$.
6. $\quad$ if ($e_{0j}=1$) then $B \leftarrow B\cdot T_0$.
7. $\quad$ if ($e_{1j}=1$) then $B \leftarrow B\cdot T_1$.
8. od
9. output B.

Now we count the number of required operations.

- To compute T_1 in line 2, we need $(m-2)$ p-th powering operations and $(m-3)/2$ multiplications over $GF(p^m)$. T_0 can be computed using one p-th powering operation and one multiplication, since $T_0=A\times T_1^p$.
- Line 5 is executed t times. Hence we need t squarings.
- Lines 6 and 7 are executed t times in the worst case. We can't use the estimation that the half of e_{0j}'s and e_{1j}'s are ones on the average, since e_0 and e_1 are fixed for a specific p and they have much more ones than the average value. (This is because p has been selected so that it may contain many ones in the binary representation to satisfy the OEF property 2 given in Section 2.1.)

By the estimation given in Section 2.2 and Section 2.3, the worst-case analysis shows that the total number of multiplications in $GF(p)$ is

$$(m-1)((m-2)+1)+(m^2+m-1)((m-3)/2+1+2t)+(m^2/2+3m/2-1)t$$
$$=(m^3+2m^2-6m+3)/2+t(5m^2+7m-6)/2 \quad (6)$$

6. Comparison of Efficiency

In this section, we compare the computational costs of the square-and-multiply algorithm and the two new exponentiation algorithms. First, note that in the exponentiations in Algorithm 3 and Algorithm 4, the size of exponent is approximately the same as the group order, i.e., exponent $e \approx p^m$ in our context. Then l in (1) and s in (2) and (4) satisfies

$$l=\lceil \log_2(e+1) \rceil \approx m\log_2 p \quad \text{and}$$
$$s=\lceil \log_p(e+1) \rceil \approx m,$$

respectively. Also we can use an approximation

$t \approx \log_2 p$ in (3), (4) and (6). Then we can estimate the number of multiplications in $GF(p)$ for exponentiation of an element in $GF(p^m)$. See Table 1.

From Table 1, we can see that our general exponentiation algorithm given in Section 4 requires about a half of computation compared to the square-and-multiply algorithm, and the optimized algorithm given in Section 5 further reduces the amount of computation by eliminating the factor $\log_2 p$ in the leading term.

We remark that Algorithms 5 and 6 require only small amount of additional memory to store T_i's. (Although we should also store $w^{\lfloor ip/m \rfloor}$ values for p-th powering, it is not an overhead; the code for p-th powering is necessary for other operations such as a field inversion and elliptic curve point ope-

Table 1 Number of $GF(p)$ multiplications to compute $A^e \in GF(p^m)$

algorithms	number of multiplications in $GF(p)$
square-and-multiply (Alg.1)*	$\frac{\log_2 p}{2}(2m^3+4m^2-3m)$
new algorithm (Alg.5)*	$(m-1)^2+\frac{\log_2 p}{2}(m^3+2m^2+2m-2)$
optimized algorithm for $e=u$ (Alg.6)†	$\frac{m^3+2m^2-6m+3}{2}+\frac{\log_2 p}{2}(5m^2+7m-6)$

*average case analysis, †worst-case analysis

Table 2 Implemented OEFs and curves

OEF: $GF(p^m)$	curve: $Y^2=X^3+AX+B$
OEF 1†: $p=2^{16}-129$ $m=11$ $f(x)=x^{11}-3$	curve 1R: random curve with 176-bit order, cofactor $h=1$ [4] $A=$ 0X FF7C, $B=$ 0X 325Ax^{10} +5511x^9 +F0A7x^8 +B7FBx^7 +D906x^6 +1FBAx^5 +D032x^4 +CC2Dx^3 +EE25x^2 +C40Ax +ECAF
	curve 1K: Koblitz-type curve with 160-bit order, $h=65407$ [4] $A=$ 0X FF7C, $B=$ 0X 017F
OEF 2†: $p=2^{16}-17$ $m=17$ $f(x)=x^{17}-2$	curve 2R: random curve with 272-bit order, $h=1$ [4] $A=$ 0X FFEC, $B=$ 0X C3EDx^{16} +AB1Fx^{15} +5ED9x^{14} +2A01x^{13} +ACDEx^{12} +3D1Ex^{11} +A38Dx^{10} +5A95x^9 +9D10x^8 +1F9Ex^7 +5C63x^6 +86B7x^5 +7F7Ax^4 +66C1x^3 +6159x^2 +947Fx +4B36
OEF 3‡: $p=2^{15}-75$ $m=11$ $f(x)=x^{11}-2$	curve 3K: Koblitz-type curve with 150-bit order, $h=32420$ $A=$ 0X 0001, $B=$ 0X 0000
OEF 4†: $p=2^{31}-1$ $m=7$ $f(x)=x^7-3$	curve 4R: random curve with 217-bit order, $h=1$ [4] $A=$ 0X 7FFFFFFC, $B=$ 0X 039055B8x^6 +1A52D0E2x^5 +2EEE1471x^4 +07505B48x^3 +6A6BFE64x^2 +4C1292C9x +36BB468C
	curve 4K: Koblitz-type curve with 187-bit order, $h=2147444533$ $A=$ 0X 00000000, $B=$ 0X 00000005
OEF 5‡: $p=2^{29}-3$ $m=7$ $f(x)=x^7-2$	curve 5K:Koblitz-type curve with 162-bit order, $h=3563249795090$ $A=$ 0X 00000002, $B=$ 0X 00000000

†$p^m \equiv 3 \bmod 4$, ‡$p^m \equiv 5 \bmod 8$

Table 3 Timings for the computation of A^e with $e \approx p^m$ (μsec)

OEF	Algorithm 1	Algorithm 5 (random e)		Algorithm 6 (e fixed as u)	
	timings (A)	timings (B)	speedups (A)/(B)	timings (C)	speedups (A)/(C)
1	286	130	2.20	57	5.02
2	884	408	2.17	125	7.07
3	390	199	1.96	70	5.57
4	186	97	1.92	58	3.21
5	470	199	2.36	124	3.79

Table 4 Timings to produce a random base point (μsec)

curve	using Alg. 1	using Alg. 5		using Alg. 6	
	timings (A)	timings (B)	speedups (A)/(B)	timings (C)	speedups (A)/(C)
1R	570.1	417.2	1.37	129.6	4.40
1K	848.2	635.9	1.33	359.0	2.36
2R	1908.0	1294.2	1.47	291.5	6.55
3K	1168.5	813.4	1.44	525.9	2.22
4R	324.7	255.1	1.27	113.3	2.87
4K	580.3	516.5	1.12	396.7	1.46
5K	1521.4	1397.6	1.09	937.6	1.62

rations, regardless of the use of new exponentiation algorithm.)

7. Experimental Results

To verify the estimation given in the previous section, we implemented various OEFs and elliptic curves, and we measured the timings for exponentiation and point generation. First, Table 2 shows the OEFs and curves which we have implemented.

Table 3 shows the measured timings for an exponentiation A^e with $e \approx p^m$. We implemented the algorithms in C using djgpp-2.03 compiler on a Pentium 4 2.66GHz CPU. In Table 3, we first see that Algorithm 5 is about twice as fast as the square-and-multiply algorithm (Algorithm 1), which is consistent with the estimation given in the previous section.[2] We also see that computational speedups obtained using Algorithm 6 are greater than the estimation given in Table 1. Note that we have used a worst-case analysis for Algorithm 6 in Table 1.

Next, we measured the timings to produce a random base point using Algorithm 2. See Table 4. In this table, we see that if we use Algorithm 5 instead of Algorithm 1, then we obtain minor speedups. (The speedups are much smaller than those expected from Table 3, i.e., approximately two for random curves. Note that the real exponents u have special structures which are explained in Section 5 and they have larger Hamming weights than average.) The speedups obtained by using Algorithm 6, however, are significant: we can produce a random base point by 1.62 to 6.55 times faster. Note that the speedups on random curves are much greater than those on Koblitz-type curves, since the cost to compute hF in Algorithm 2 is not negligible in Koblitz-type curves, and it is an irreducible overhead.

8. Conclusions

We have presented efficient exponentiation algorithms over OEF, and we have used them to accelerate the process of searching for a base point on a given elliptic curve. According to our experimental results on various elliptic curves, the new algorithms make the searching process 1.62-6.55 times faster compared to the basic searching algorithm which uses square-and-multiply exponentiation.

Finally, we remark that our improvement does not apply to $GF(p^m)$ with $p^m \equiv 1 \bmod 8$. Therefore,

2) We mention that while we compared our algorithms' performance to that of the square-and-multiply algorithm, our algorithms are faster than other general algorithms such as the m-ary method, the window method, the addition chain methods, etc., since the speedups obtained by these methods are under 1.5 in most cases.

it would be an interesting research area to consider this case more completely.

References

[1] N. Koblitz. "Elliptic curve cryptosystems," Mathematics of Computation, Vol. 48, pp.203-209, 1987.

[2] V. Miller. "Use of elliptic curves in cryptography," Advances in Cryptology- CRYPTO 85, LNCS, Vol. 218, pp.417-428, Springer-Verlag, 1986.

[3] IEEE P1363-2000, IEEE Standard Specifications for Public-Key Cryptography, 2000.

[4] TTAS.KO-12.0015, Digital Signature Mechanism with Appendix- Part 3: Korean Certificate-based Digital Signature Algorithm using Elliptic Curves, 2001.

[5] R. Schoof. "Elliptic curves over finite fields and the computation of square roots mod p," Mathematics of Computation, Vol.44, pp.483-494, 1985.

[6] R. Lercier and F. Morain, "Counting the number of points on elliptic curves over finite fields: strategies and performance," Advances in Cryptology-Eurocrypt 95, LNCS, Vol.921, pp.79-94. Springer, 1995.

[7] R. Lercier, "Finding good random elliptic curves for cryptosystems defined over F_{2^n}," Advances in Cryptology-Eurocrypt 97, LNCS, Vol.1233, pp.379-392. Springer, 1997.

[8] D. V. Bailey and C. Paar, "Optimal extension fields for fast arithmetic in public-key algorithms," Advances in Cryptology- CRYPTO 98, LNCS, Vol.1462, pp.472-485. Springer, 1998.

[9] D. V. Bailey and C. Paar, "Efficient arithmetic in finite field extensions with application in elliptic curve cryptography," Journal of Cryptology, Vol.14, No.3, pp.153-176, 2001.

[10] N. P. Smart, "A comparison of different finite fields for elliptic curve cryptosystems," Computers and Mathematics with Applications, Vol.42, pp.91-100, 2001.

[11] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "Fast exponentiation in $GF(2^n)$," Advances in Cryptology-EUROCRYPT 88, LNCS, Vol.330, pp.251-256, Springer, 1988.

[12] J. von zur Gathen, "Processor-efficient exponentiation in finite fields," Information Processing Letters, Vol.41, pp.81-86, 1992.

[13] M. K. Lee, Y. Kim, K. Park, and Y. Cho, "Efficient parallel exponentiation in $GF(q^n)$ using normal basis representations," Journal of Algorithms, Vol.54, pp.205-221, 2005.

[14] T. Kobayashi, "Base-ϕ method for elliptic curves of OEF," IEICE Trans. Fundamentals, Vol.E83-A, No.4, pp.679-686, 2000.

[15] D. M. Gordon, "A survey of fast exponentiation methods," Journal of Algorithms, Vol.27, pp.129-146, 1998.

[16] D. Knuth. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, Massachusetts, 3rd edition, 1998.

[17] J. Bos and M. Coster, "Addition chain heuristics," Advances in Cryptology- CRYPTO 89, LNCS, Vol.435, pp.400-407. Springer-Verlag, 1990.

[18] J.-C. Ha and S.-J. Moon, "Fast exponentiation with common-multiplicand modular multiplication," Journal of the Korea Information Science Society (C), Vol.3, No.5, pp.491-497, 1997.

[19] J.-C. Ha and S.-J. Moon, "A common-multiplicand method to the Montgomery algorithm for speeding up exponentiation," Information Processing Letters, Vol.66, pp.105-107, 1998.

[20] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, "Fast exponentiation with precomputation," Advances in Cryptology -Eurocrypt 92, LNCS, Vol.658, pp.200-207. Springer, 1993.

[21] C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation," Advances in Cryptology -CRYPTO 94, LNCS, Vol.839, pp.95-107. Springer, 1994.

이 문 규

1996년 2월 서울대학교 컴퓨터공학과 학사. 1998년 2월 서울대학교 컴퓨터공학과 석사. 2003년 8월 서울대학교 전기컴퓨터공학부 박사. 2003년 8월~2005년 2월 한국전자통신연구원(선임연구원). 2005년 2월~현재 인하대학교 컴퓨터공학부(조교수). 관심분야는 정보보호, 암호학, 컴퓨터이론