Bailey-Paar 최적확장체 연산의 개선
(Improvement on Bailey-Paar's Optimal Extension Field Arithmetic)

이 문 규 †
(Mun-Kyu Lee)

요 약 최적확장체(Optimal Extension Field: OEF)는 유한체의 일종으로서, 타원곡선 암호시스템의 소프트웨어 구현에 있어 매우 유용하다. Bailey 및 Paar는 p^i 거듭제곱 연산을 비롯하여 다수의 효율적인 OEF 연산 알고리즘을 제안하였으며, 또한 암호 응용에 적합한 OEF를 생성하기 위한 효과적인 알고리즘을 제안하였다. 본 논문에서는 Bailey-Paar의 p^i 거듭제곱 알고리즘이 적용되지 않는 반례를 제시하며, 또한 그들의 OEF 생성 알고리즘은 실제로 OEF가 아닌 유한체를 OEF로 출력하는 오류가 있음을 보인다. 본 논문에서는 이러한 문제들을 해결한 개선된 알고리즘들을 제시하고, OEF의 개수에 관한 수정된 통계치를 제시한다.

키워드 : 암호, 타원곡선, 최적확장체(OEF), 역원, 프로베니우스 사상

Abstract Optimal Extension Fields (OEFs) are finite fields of a special form which are very useful for software implementation of elliptic curve cryptosystems. Bailey and Paar introduced efficient OEF arithmetic algorithms including the p^ith powering operation, and an efficient algorithm to construct OEFs for cryptographic use. In this paper, we give a counterexample where their p^ith powering algorithm does not work, and show that their OEF construction algorithm is faulty, i.e., it may produce some non-OEFs as output. We present improved algorithms which correct these problems, and give improved statistics for the number of OEFs .

Key words : Cryptography, Elliptic Curve, Optimal Extension Field (OEF), Inversion, Frobenius Map

1. Introduction

Since Koblitz [1] and Miller [2] proposed to use elliptic curves in cryptography, an extensive research has been done on elliptic curve cryptosystems (ECCs). For the efficient realization of ECCs, it is important to implement underlying finite field operations efficiently.

· 본 연구는 한국과학재단 목정지초연구(R01-2006-000-10957-0) 지원으로 수행되었음

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

논문접수 : 2005년 4월 15일
심사완료 : 2008년 4월 30일

정보과학회논문지: 시스템 및 이론 제35권 제7호(2008.8)

Finite fields are denoted by $GF(p^m)$, where p is a prime and m is a positive integer. Many finite fields have been suggested for ECCs, for example, binary fields (i.e. the case with $p = 2$ and a large m) and prime fields (i.e. the case with m=1 and a large p). However, it is known that for a software implementation of ECC, *Optimal Extension Fields* (OEFs) [3] are the best choice [4].

An OEF [3] is a finite field $GF(p^m)$ such that:

- p is a pseudo-Mersenne prime, i.e. a prime of the form $p = 2^n \pm c$, where $\log_2 c \leq \lfloor n/2 \rfloor$.
- An irreducible binomial $P(x) = x^m - w$ exists over $GF(p)$.

Theorem 1 in [3] describes the cases where a finite field is an OEF:

Theorem 1 [3]. Let $m \geq 2$ be an integer and $w \in GF(p)^*$. Then the binomial $x^m - w$ is irreducible in $GF(p)[x]$ if and only if the following two conditions are satisfied:

(i) each prime factor of m divides the order e of w

over $GF(p)$, but not $(p-1)/e$;
(ii) $p \equiv 1 \bmod 4$ if $m \equiv 0 \bmod 4$.

To represent an element in an OEF, a standard (or polynomial) basis representation is used, i.e. an element $A(x)$ in an OEF $GF(p^m)$ is written as

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0,$$

where $a_i \in GF(p)$.

All arithmetic operations over an OEF are performed modulo the field polynomial $P(x)$. Therefore, addition and subtraction of two field elements is implemented in a straightforward manner by modular addition or modular subtraction of the coefficients in their polynomial representations. Field multiplication is done in two stages: polynomial multiplication and reduction modulo $P(x)$.

On the other hand, the inversion operation is more complicated than other arithmetic operations. Bailey and Paar [3] have proposed an efficient inversion algorithm, which uses the fact that for any element $A(x) \in GF(p^m)$, $A^r(x)$ is in its subfield $GF(p)$, where $r = (p^m - 1)/(p - 1)$. See Algorithm 1.

Algorithm 1. Optimal Extension Field Inversion [3]
Input: nonzero $A(x) \in GF(p^m)$.
Output: $B(x) \in GF(p^m)$ such that $A(x)B(x) \equiv 1 \bmod P(x)$.

1. $B(x) \leftarrow A^{r-1}(x)$.
2. $c_0 \leftarrow A(x)B(x)$. $\triangleright c_0 = A^r(x) \in GF(p^m)$.
3. $c \leftarrow c_0^{-1} \bmod p$. $\triangleright c = A^{-r}(x) \in GF(p^m)$.
4. $B(x) \leftarrow B(x)c$.

The most time-consuming part of this algorithm is Step 1. Since $r = p^{m-1} + p^{m-2} + \cdots + p + 1$, we have the p-adic representation $r - 1 = (11\ldots10)_p$, and $A^{r-1}(x)$ can be computed using a p-adic addition chain method. For example, for $m = 6$, $A^{r-1}(x)$ is computed as follows:

$$B \leftarrow A^p = A^{(10)}; C \leftarrow BA = A^{(11)}; B \leftarrow C^{p^2} = A^{(1100)}; B \leftarrow BC = A^{(1111)};$$
$$B \leftarrow B^p = A^{(11110)}; B \leftarrow BA = A^{(11111)}; B \leftarrow B^p = A^{(111110)}.$$

Hence, we require an efficient p^i th powering algorithm as well as a field multiplication algorithm that we have already discussed above.

In [3], Bailey and Paar presented an efficient p^i th powering algorithm, i.e., an iterative Frobenius map algorithm that adopts a pre-computation table. This algorithm was used for faster software implementation of OEF arithmetic. They also introduced an algorithm to construct OEFs suitable for various platforms, and provided statistics for the number of OEFs that exist for various choices of n and tables of OEFs.

In this paper, we give a counterexample to the p^i th powering algorithm given in [3], which is consistent with Baktir and Sunar's observation [5]. Then we show that the OEF construction algorithm (Algorithm 3 of [3]) is faulty, i.e., it may generate some non-OEFs as output, since it omits a filtering procedure. We present an improved construction algorithm and also give improved statistics for the number of OEFs.

2. Computation of the p^i th Power on OEFs

In this section, we examine the p^i th powering operation, i.e., *the ith iterate of Frobenius map*. First, note that for an element

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$$

in $GF(p^m)$, the ith iterate of the Frobenius map is written as

$$A^{p^i}(x) = a_{m-1}x^{(m-1)p^i} + \cdots + a_1 x^{p^i} + a_0 \bmod P(x), \quad (1)$$

since $a_j^{\,p} \equiv a_j \bmod p$ by Fermat's Little Theorem.

Corollary 2 in [3] claims that each variable term $(x^j)^{p^i}$ in (1) satisfies

$$(x^j)^{p^i} \equiv w^{q_{ij}} x^j \bmod P(x), \quad (2)$$

where $q_{ij} = \lfloor jp^i / m \rfloor$. Thus (1) corresponds to

$$A^{p^i}(x) = a_{m-1}w^{q_{i,m-1}}x^{m-1} + \cdots + a_1 w^{q_{i,1}} x + a_0 \bmod P(x). \quad (3)$$

Since all $w^{q_{ij}}$, $1 \le i, j \le m-1$, in (3) can be pre-computed, the computation of (3) requires only $m - 1$ subfield multiplications.

As Baktir and Sunar [5] have pointed out, however, there is a flaw in the proof of Corollary 2 in [3], and the above simplification does not apply to all OEFs. Note that the proof of (2) is based on the claim that $jp^i \bmod m = j$. We give a counterexample to this claim. Consider an OEF with $p = 2^8 - 15 = 241$, $m = 27$ and $w = 2$, which is given in Table 9 in [3]. For $i = 1$, we see that $j \cdot 241^1 \bmod 27 \neq j$ for $1 \le j \le 26$ except $j = 9, 18$.

This problem can be solved by eliminating the

simplified part related to this faulty claim and correcting (2). Since $x^m \equiv w \bmod P(x)$, $(x^j)^{p^i}$ can be written as:

$$(x^j)^{p^i} \equiv x^{\lfloor jp^i/m \rfloor m+(jp^i \bmod m)} \equiv w^{q_{ij}} x^{jp^i \bmod m} \bmod P(x). \quad (4)$$

Hence we can construct an improved p^i th powering method as a simple extension of the algorithm given in [6]. First, using the fact that $\gcd(m,p) = 1$ for practical OEFs with $m \ll p$, we can easily show that $j_1 p^i \bmod m = j_2 p^i \bmod m$ is equivalent to $j_1 = j_2$. Therefore, the map $\pi : j \mapsto jp^i \bmod m$ is bijective. The flaw in Corollary 2 in [3] is that an identity map has been used instead of this map. Then what we have to do is to perform the same pre-computation and the same on-line computation as [3], and to reorder the terms according to this bijective map. Note that the problem of Bailey and Paar's algorithm is that this reordering is missing due to the use of an incorrect map. The following algorithm shows the corrected algorithm for the ith iterate of Frobenius map, where $w^{q_{ij}}$'s $(1 \le j \le m-1)$ are available in a pre-computation table.

Algorithm 2. Computation of the p^i th power

Input: $A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$.

Output: $A'(x) = a'_{m-1}x^{m-1} + \cdots + a'_1 x + a'_0 = A^{p^i}(x)$.

1. $b_j \leftarrow a_j w^{q_{ij}}$ for $1 \le j \le m-1$. ▷ subfield multiplications
2. $a'_{\pi(j)} \leftarrow b_j$ for $0 \le j \le m-1$. ▷ reordering

This algorithm requires $m-1$ subfield multiplications and some additional operation for π map computation. Note that, however, the costs for π maps are negligible.

3. Improved Algorithm for OEF Construction

Although every OEF offers considerable computational advantages in software implementation of ECCs, there are two special cases of OEF which yield additional advantages as follows:

- A Type I OEF has $p = 2^n \pm 1$, i.e. $c = 1$.
- A Type II OEF has an irreducible binomial $P(x) = x^m - 2$, i.e. $w = 2$.

Note that a Type I OEF provides a very efficient subfield modular reduction, and a Type II OEF provides a very efficient extension field reduction modulo $P(x)$.

Section 8 of [3] introduces an algorithm to construct Type II OEFs, and various tables are given including a table of OEFs generated by this algorithm. But in fact this table, i.e. Table 9 in [3], contains several incorrect elements which are not OEFs as follows:

- For $p = 251$ and $m = 25$, the order of 2 over $GF(251)$ is $e = 50$. On the other hand, m has a single prime factor 5. Since 5 divides both of e and $(p-1)/e = 5$, $GF(251^{25})$ is not an OEF. Similarly, $GF(257^{32})$ is not an OEF either.
- For $p = 241$ and $m = 25$, the order of 2 over $GF(241)$ is $e = 24$. On the other hand, m has a single prime factor 5. Since 5 does not divide $e = 24$, $GF(241^{25})$ is not an OEF.

The first one of the above two cases results from not checking if each prime factor of m divides $(p-1)/e$ or not. Actually this filtering code is missing in Algorithm 3 in [3].[1] We could not find the reason for the second case, since Algorithm in [3] already contains the filtering code for this case. Hence we conjecture that the second case occurred due to a certain implementation error.

Algorithm 3 below is our improved algorithm including the filtering procedure (lines 18-21). The algorithm proceeds by finding primes of the form $p = 2^n \pm c$ and then checking possible extension degrees m for the existence of a binomial. The bit length of the prime, i.e. n, is chosen based on the attributes of the target microprocessor, and the parameters *low*, *high*, and *mMax* are chosen so that the resulting OEFs may give a suitable level of security and efficiency for ECCs. Note that minor changes to this algorithm can produce Type I OEFs or general OEFs as in the original algorithm in [3].

Algorithm 3. Construction of Type II OEFs

Input: n, *mMax*,
and *low*, *high* bounds on bit length of field order.

Output: Type II OEFs with $low \le mn \le high$.

1) But it is not certain that Table 9 in [3] has been generated by the algorithm without the filtering process, since there are no other incorrect elements in Table 9.

```
1. cMax ← 2^⌊n/2⌋.
2. if cMax is even then
3.    cMax ← cMax - 1.
4. end if
5. c ← - cMax.
6. while c ≤ cMax do
7.    p ← 2^n + c.
8.    if p is prime then
9.       e ← (the order of 2 in GF(p)).
10.      for m ← 2 to mMax do
11.         if low ≤ mn ≤ high then
12.            BadMValue ← 0.
13.            for each prime factor d of m do
14.               if d does not divide e then
15.                  BadMValue ← 1.
16.                  break.
17.               end if
18.               if d divides (p - 1)/e then
19.                  BadMValue ← 1.
20.                  break.
21.               end if
22.            end for
23.            if BadMValue = 0 then
24.               if m ≡ 0 mod 4 then
25.                  if p ≡ 1 mod 4 then
26.                     output p,m.
27.                  end if
28.               else
29.                  output p,m.
30.               end if
31.            end if
32.         end if
33.      end for
34.   end if
35.   c ← c + 2.
36. end while
```

4. Improved Statistics for the Number of OEFs

Now we provide improved statistics for the number of OEFs, which are computed from the use of Algorithm 3. Table 1 shows the number of type II OEFs that exist for several choices of n. Note that we omit values for other n since they are the same as the original values given in Tables 5, 6 and 7 in [3].

5. Conclusions

We have given a counterexample where the p^i th powering algorithm in [3] does not work, and we have shown that the OEF construction algorithm (Algorithm 3 of [3]) is faulty. We have also shown that simple modifications of these algorithms can solve the problems, and we have presented correct statistics for the number of OEFs that exist for various choices of n.

References

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

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

[3] D.V. Bailey and C. Paar, "Efficient arithmetic in finite field extensions with application in elliptic curve cryptography," *Journal of Cryptology*, Vol. 14, 2001, pp.153-176.

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

[5] S. Baktir and B. Sunar, "Optimal Tower Fields," *IEEE Transactions on Computers*, Vol.53, No.10, 2004, pp. 1231-1243.

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

Table 1 Number of Type II OEFs

n	m													
	6	7	8	9	10	11	12	13	14	15	16	25	27	32
8												1	2	1
16				19	5	1	4	6	4	4	13			
19		9	21	25	4	7	6	4						
25	84	75	129	168	60									

이 문 규

1996년 서울대학교 컴퓨터공학과 학사
1998년 서울대학교 컴퓨터공학과 석사
2003년 서울대학교 전기컴퓨터공학부 박사. 2003년~2005년 한국전자통신연구원(선임연구원). 2005년 3월~현재 인하대학교 컴퓨터정보공학부(조교수). 관심분야는 정보보호, 암호학, 컴퓨터이론