JOURNAL OF THE
CHUNGCHEONG MATHEMATICAL SOCIETY
Volume 15, No. 2, December 2002

AN ALGORITHM FOR MULTIPLICATIONS IN F_{2^m}

SeYoung Oh* and ChungSup Yoon**

Abstract. An efficient algorithm for the multiplication in a binary finite filed using a normal basis representation of F_{2^m} is discussed and proposed for software implementation of elliptic curve cryptography. The algorithm is developed by using the storage scheme of sparse matrices.

1. Introduction

For implementing elliptic curve public-key cryptographic system over binary finite fields of characteristic two, the calculation of $Q = nP$, for P a base point on the elliptic curve and n an integer, is the core operation, which mostly involves multiplications in the binary finite field. Therefore, reducing words operations and memory required to perform the field multiplication is crucial for efficient software implementation of the scalar multiplication nP in elliptic curve cryptographic systems.

There are many different basis of the binary field F_{2^m} over F_2. The multiplication of F_{2^m} can be quite different depending on the choice of the basis of F_{2^m}. ANSI X9.62 permits only polynomial basis and normal basis of the binary field F_{2^m} for the secure implementation of elliptic curve cryptography. The polynomial basis representation , however, seems to offer better performance in software implementation than the normal basis.

Received by the editors on January 16, 2003.
2000 *Mathematics Subject Classifications*: Primary 94A60.
Key words and phrases: Elliptic Curve Cryptography, Binary finite field, Normal basis.

In this paper we discuss the multiplication using a normal basis representation of F_{2^m} and propose an efficient algorithm for the multiplication to implement in software. A software implementation of field multiplication using a polynomial basis representation has been proposed in [7], while a normal basis representation was used to implement in software in [11]. The proposed algorithm in this paper is similar to the algorithm in the latter article. Our algorithm is , however, to speed up the computation by reducing the memory storage and using a storage scheme for a sparse matrix.

In section 2, we discuss some mathematical backgrounds for finite field and its normal basis representation. Multiplication matrix is introduced to get a mathematical formula of field multiplication, and a modified one of the algorithm proposed in [11] is described in section 3. The proposed algorithm can be applied to the both of type I and II optimal normal basis representation.

2. Mathematical Backgrounds

- Binary finite field : F_{2^m}

A finite field F_q(or Galois field) of order q(finite) is a set with q elements in which the algebraic operations of addition, substraction, multiplication and division by nonzero elements are possible and the algebraic laws of commutativity and distributivity hold. If $q > 1$ is an integer, then a finite field of order q exists if q is a prime power and not otherwise. When $q = 2^m$ for some m, the field F_{2^m} is called a binary finite field. The field F_{2^m} can be viewed as a vector space of dimension m over F_2, or the 2^m possible bit strings of length m. For example there exists a basis $\{\alpha_0, \alpha_1, \cdots, \alpha_{m-1}\}$ in F_{2^m} such that each $\alpha \in F_{2^m}$ can be written uniquely in the form $\alpha = \sum_{i=0}^{m-1} a_i\alpha_i, a_i \in F_2$. Now we can represent α as the $0-1$ vector $(a_0a_1 \cdots a_{m-1})$. There are many common representations for binary finite fields. Some representations may lead to more efficient implementations of field arithmetic in hardware or in software. The normal basis representation for F_{2^m} is focused in this article.

If a subset $B = \{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^{m-1}}\}$ of F_{2^m} for $\beta \in F_{2^m}$ is linearly independent over F_2 , then the subset is called a normal basis of F_{2^m} over F_2. For every positive integer m there exists a normal basis of F_{2^m} (see[1]). For all $\alpha \in F_{2^m}$, we can write $\alpha = \sum_{i=0}^{m-1} a_i \beta^{2^i}, a_i \in \{0,1\}$ and represent α by interpreting $0-1$ vector or n-bit string $(a_0 a_1 a_2 \cdots a_{m-1})$. Note that all of the elements of a normal basis are distinct roots of the same irreducible binary polynomial f of degree m over F_2, which is called normal polynomial. Such normal polynomials exist for every degree m.

- Producing an irreducible polynomial of degree m over F_2

To compute the reduction of polynomials modulo $f(x)$ efficiently, $f(x)$ has a small number of terms. The irreducible polynomials with the least number of terms are the trinomials $x^m + x^k + 1$. Provided that one exists, it is the best choice for the field polynomial. If a trinomial of degree m does not exist, then the next best polynomials are the pentanomials $x^m + x^a + x^b + x^c + 1$. For every m up to 1000, there exists either an irreducible trinomial or pentanomial of degree m. [2] provides a table with an example for each m up to 1000. An irreducible polynomial of degree m over F_2 also can be constructed by the following algorithm:

Algorithm 2.1 : Irreducible polynomials over F_2

```
0. choose $f(x)$ with an odd number of nonzero terms, at least one odd-degree term and 1 as constant term.
1. d $\leftarrow$ deg($f(x)$)
2. $u(x) \leftarrow x$
```

```
3. for $i = 1$ to $\lfloor \frac{d}{2} \rfloor$ do
   $3-1.$ $u(x) \leftarrow u(x)^2$ mod $f(x)$
   $3-2.$ $g(x) \leftarrow u(x) + x, b(x) \leftarrow f(x))$
   $3-3.$ while $b(x) \neq 0$
         $i)$ compute $g(x) = b(x)q(x) + r(x)$.
         $ii)$ $g(x) \leftarrow b(x), b(x) \leftarrow r(x)$.
   $3-4.$ If $g(x) \neq 1$ then go to 0.
4. print $f(x)$ .
```

From the statistical point of view, it is known that the probability that $f(x)$ is irreducible is about $4/m$. Thus one can find an irreducible polynomial of degree m out of $m/4$ choices.

- Finding a normal basis for F_{2^m}

A normal basis is specified by its field polynomial $f(x)$. After producing an irreducible polynomial of degree m over F_2, we want to check if it is normal by the following algorithm 2.2 [2].

Algorithm 2.2 Checking the normality of $f(x)$

input : $f(x)$: irreducible polynomial of degree m over F_2.

output : normal polynomial $f(x)$ over F_2

```
0. Choose an irreducible polynomial of degree $m$ over $F_2$ by
   algorithm 2.1 and Algorithm 2.2
1. Compute $m \times m$ matrix $A$ such that
```

$$\begin{pmatrix} x \\ x^2 \\ x^{2^2} \\ \vdots \\ x^{2^{m-1}} \end{pmatrix} = \begin{pmatrix} \\ A \\ \\ \end{pmatrix} \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{m-1} \end{pmatrix} (mod\, f(x))$$

```
2. Determine the inverse matrix $A^{-1}$ of $A$ over $F_2$.
```

2.1 If A^{-1} exists, then $f(x)$ is normal.

2.2 Otherwise, $f(x)$ is not normal, and go to 0.

It is known that the probability of a randomly chosen irreducible polynomial of degree m over F_2 being normal is at least 0.2 if $m \leq 2000$, and is usually much higher. That is, one of five chosen irreducible polynomial is normal. In algorithm 2.2, the step 1 provides a normal basis for F_{2^m}. If we take some element $\beta \in F_{2^m}$, the polynomial representation is :

$$\beta = a_{00} + a_{01}x + a_{02}x^2 + \cdots + a_{0m-1}x^{m-1}\,.$$

By repeating squaring modulo 2, we can compute the elements of A. If A^{-1} exists, a normal basis can be formed using the set $\{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^2}\}$.

3. Multiplication over normal basis of F_{2^m}

In elliptic curve cryptography over F_{2^m} , we need two kinds of operations, addition and doubling of points, on a commutative abelian group. Those operations involve just a few arithmetic operations , squaring , multiplication, and inversion in the binary finite field F_{2^m}. There are many different basis of F_{2^m} over F_2. Some basis lead to more efficient software or hardware implementations of the arithmetic in F_{2^m} than other basis. ANSI 9.62 permits two kinds of basis : polynomial basis and normal basis. Fast implementation techniques for F_{2^m} arithmetic have been studied intensively in the past twenty years. Especially the multiplication technique is the most important since it is a major part for implementing elliptic curve cryptographic systems. It is known that normal basis representation performs in hardware [3-5], while polynomial basis representation is more efficient in software

[6-8]. In this section we propose an algorithm of multiplication in F_{2^m} using normal basis.

The squaring in a normal basis representation can also be easily done by a cyclic right shift of the bit string representation, while multiplication is more complicated depending on the basis chosen. Multiplication for normal basis can be described as follows([9] and [10]). Let $U_0 = (a_0 a_1 \cdots a_{m-1})$ and $V_0 = (b_0 b_1 \cdots b_{m-1})$ be two elements in F_{2^m}, and

$$U_0 \cdot V_0 = C = (c_0 c_1 \cdots c_{m-1}) = \sum_{k=0}^{m-1} c_k \beta^{2^k}$$

$$U_0 \cdot V_0 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \beta^{2^i} \beta^{2^j}.$$

Once each cross-product term $\beta^{2^i}\beta^{2^j} = \sum_{k=0}^{m-1} \lambda_{ij}^{(k)} \beta^{2^k}$ is mapped to a sum over the basis terms, we get the following equations by comparing the coefficients of β^{2^k}

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \lambda_{ij}^{(k)} \qquad k = 0, 1, \cdots, m-1. \tag{1}$$

Field addition and substraction are implemented as componentwise exclusive OR(XOR).

Theorem 3.1 $c_k = U_k \Lambda V_k^T \quad k = 0, 1, \cdots, m-1,$ where $U_k = (a_k a_{k+1} \cdots a_{m-1+k})$, $V_k = (b_k b_{k+1} \cdots b_{m-1+k})$ with subscripts modulo m and $\Lambda = (\lambda_{ij}^{(0)})$ is $m \times m$ sparse 0-1 matrix independent of k.

Proof : From $(\beta^{2^i}\beta^{2^j})^{2^{-l}} = \beta^{2^{i-l}}\beta^{2^{j-l}} \in F_{2^m}$ for $0 \le l \le m-1$, it can be represented by the normal basis terms :

$$\beta^{2^{i-l}}\beta^{2^{j-l}} = \sum_{k=o}^{m-1} \lambda_{i-l j-l}^{(k)} \beta^{2^k}. \tag{2}$$

On the other hand,

$$\begin{aligned}
(\beta^{2^i}\beta^{2^j})^{2^{-l}} =& (\sum_{k=0}^{m-1} \lambda_{ij}^{(k)} \beta^{2^k})^{2^{-l}} \\
=& \lambda_{ij}^{(l)}\beta^{2^0} + \lambda_{ij}^{(l+1)}\beta^{2^1} + \cdots + \lambda_{ij}^{(m-1)}\beta^{2^{m-l-1}} + \lambda_{ij}^{(0)}\beta^{2^{m-l}} + \\
& \lambda_{ij}^{(1)}\beta^{2^{m-l+1}} + \cdots + \lambda_{ij}^{(l-1)}\beta^{2^{m-1}} \\
=& \lambda_{ij}^{(0)}\beta^{2^{m-l}} + \lambda_{ij}^{(1)}\beta^{2^{m-l+1}} + \cdots + \lambda_{ij}^{(l-1)}\beta^{2^{m-1}} + \lambda_{ij}^{(l)}\beta^{2^0} + \\
& \cdots + \lambda_{ij}^{(m-1)}\beta^{2^{m-l-1}} \\
=& \lambda_{ij}^{(0)}\beta^{2^{-l}} + \lambda_{ij}^{(1)}\beta^{2^{-l+1}} + \cdots + \lambda_{ij}^{(l-1)}\beta^{2^{-1}} + \lambda_{ij}^{(l)}\beta^{2^0} + \cdots + \\
& \lambda_{ij}^{(m-1)}\beta^{2^{m-l-1}} \\
=& \sum_{k=0}^{m-1} \lambda_{ij}^{(k)} \beta^{2^{k-1}} . \qquad (3)
\end{aligned}$$

Note that $\beta^{2^{m-l}} = \beta^{2^{-l}}$ since $\beta^{2^m} = 1$. Comparing the coefficients of β^{2^0} yields

$$\lambda_{ij}^{(l)} = \lambda_{i-lj-l}^{(0)} \qquad i, j = 0, 1, \cdots, m-1, \quad 0 \le l \le m-1 .$$

The equation (1) can be rewritten as

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \lambda_{i-kj-k}^{(0)} \quad . \qquad (4)$$

Now we change the indices by transformation $i - k = i', j - k = j'$ and (4) leads to

$$\begin{aligned} c_k &= \sum_{i\prime=-k}^{m-1-k} \sum_{j\prime=-k}^{m-1-k} a_{i\prime+k} b_{j\prime+k} \lambda_{i\prime j\prime}^{(0)} \\ &= \sum_{i\prime=0}^{m-1} \sum_{j\prime=0}^{m-1} a_{i\prime+k} b_{j\prime+k} \lambda_{i\prime j\prime}^{(0)} \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij}^{(0)} \\ &= U_k \Lambda V_k^T, \end{aligned}$$

where $\Lambda = (\lambda_{ij}^{(0)}), U_k = (a_k a_{k+1} \cdots a_{k+m-1}),$ and $V_k = (b_k b_{k+1} \cdots b_{k+m-1})$.

The matrix Λ is called multiplication sparse matrix of $f(x)$ for the normal basis of F_{2^m} and is depending only on $\lambda_{ij}^{(0)}$. Note that $\lambda_{ij}^{(0)}$ is the coefficient of β once $\beta^{2^i}\beta^{2^j}$ is computed. The computation of multiplication matrix Λ can be cumbersome in general. A different way to find Λ is provided in [2] as in the following theorem.

Theorem 3.2 If the normal polynomial $f(x) = x^m + \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \cdots + \alpha_1 x + \alpha_0$ and

$$D = AMA^{-1},$$

where A is a basis matrix from algorithm 2.2 and

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ & & & \ddots & & \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ \alpha_0 & \alpha_1 & \alpha_3 & \alpha_4 & \cdots & \alpha_{m-1} \end{pmatrix},$$

then $\Lambda = (\lambda_{ij}^{(0)}) = (D_{j-i,-i})$.

- An algorithm for the field multiplication

In order to implement the normal basis multiplication efficiently, we store the sparse matrix Λ more compactly using two vectors N_1 and N_2. For simplicity let $\Lambda = (\lambda_{ij}^{(0)}) = (\Lambda_{ij})$. Let T be the number $1's$ in Λ_{ij} and N_1 be a vector of length T where the entries are indices of the column, that is $j's$, with $1's$ in each row of Λ, and let $N_2(k)$ be an index of N_1, which indicates the first column with 1 in k^{th} row of Λ. For example, if

$$\wedge = (\lambda_{ij}{}^{(0)}) = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

then the number $1's$ in Λ is 7, that is, $T = 7$, and

$$N_1 = (2, 2, 3, 0, 1, 1, 3)$$
$$N_2 = (0, 1, 3, 5, 7)\,.$$

With these two vectors, we can figure out all of $1's$ in Λ. That is,

$$\begin{aligned}
\Lambda_{0\,N_1(N_2(0))} &= \Lambda_{02} = 1 \\
\Lambda_{1\,N_1(N_2(1))} &= \Lambda_{12} = 1 \\
\Lambda_{1\,N_1(N_2(1)+1)} &= \Lambda_{13} = 1 \\
\Lambda_{2\,N_1(N_2(2))} &= \Lambda_{20} = 1 \\
\Lambda_{2\,N_1(N_2(2)+1)} &= \Lambda_{21} = 1,
\end{aligned}$$

and so on. Let A(and B) be U_0(and V_0) in the previous section 3.1 respectively. To reduce the time and memory complexity, a new way of doing the pre-computation for A and B is provided in [11]. We use the scheme of pre-computation for an algorithm but only necessary pre-computations. Let w=32 bits and $t = \lfloor \frac{m}{32} \rfloor$ words with assumption of a 32-bit architecture platform for the implementation. The bits of a word are numbered from 0 to 31. The pre-computations are performed by storing m words for A,B, and C

$$\begin{aligned} A[i] &= (a_i, a_{i+1}, \cdots, a_{i+w-1}) \\ B[i] &= (b_i, b_{i+1}, \cdots, b_{i+w-1}) \\ C[i] &= (c_i, c_{i+1}, \cdots, c_{i+w-1}) \end{aligned} \tag{6}$$

for $i = 0, 1, \cdots, m-1$ in a wrap-around fashion. Then $C = A \cdot B$ can be represented as

$$C = (c[0], c[w], c[2w], \cdots, c[(t-1)w]), \quad c[iw] \in F_2^{32}, i = 0, 1, \cdots, t-1,$$

$$\begin{aligned} c[wk] &= \sum_{j=0}^{m-1} \sum_{i=0}^{m-1} A[(wk+i) \bmod m] \Lambda_{ij} B[(wk+j) \bmod m] \\ &= \sum_{i=0}^{m-1} A[(wk+i) \bmod m] \sum_{j=0}^{m-1} \Lambda_{ij} B[(wk+j) \bmod m], \end{aligned}$$

where $k = 0, 1, \cdots, t-1$. Now a new method for normal basis multiplication can be described. The proposed method is similar to the method provided in [11]. The method proposed in this article, however, uses the storage scheme for sparse matrix Λ whose memory required to be compatible for Λ in [11] is much less.

```
Algorithm 3.1 Multiplication of two elements in $F_{2^m}$
input : $A = (a_0 a_1 \cdots a_{m-1}), B = (b_0 b_1 \cdots b_{m-1}) \in F_{2^m}$
output : $C = (c_0 c_1 \cdots c_{m-1}) = (c[0], c[w], c[2w], \cdots, c[(t-1)w])$
  1. Compute $m$ vectors $A[i]$ and $B[i]$ for $i = 0, 1, \cdots, m-1$ as (6)
  2. for $k = 0$ to $\lfloor \frac{m}{w} \rfloor - 1$ do
      2.1. $c[kw] = 0 \in F_{2^{32}}$
```

```
2.2. for $i = 0$ to $m - 1$ do
    2.2.1. temp $\leftarrow 0$, temp $\in F_{2^{32}}$
    2.2.2. for $j = N_2(i)$ to $N_2(i+1) - 1$
        temp $\leftarrow$ temp $+ B[(N_1(j) + kw) \ \ mod \ \ m]$
    2.2.3. $c[kw] \leftarrow c[kw] + (temp \cdot A[(i + kw) \ \ mod \ \ m])$
2.3. print $c[kw]$
```

The number of word operations for step 2.2 is equal to T, which is the number of $1's$ in Λ, and $\lfloor \frac{m}{w} \rfloor$ word operations for step 2. Thus the total number of word operations for the multiplication is $O(T \cdot \frac{m}{w})$, which is the same as that in [11]. But the step 2.2 in our algorithm requires only T loops and T words storage, while the algorithm in [11] needs m^2 loop including "if" statements and the storage for the matrix Λ requires m^2 words. Thus the memory complexity is reduced to approximately $\frac{2T}{m^2}$ which is less than 4% when $T = 2m - 1$ and $m = 100$.

It has been proved in [10] that $2m - 1 \leq T \leq m^2$. When the lower bound $T = 2m - 1$ is attained, the normal basis is called optimal normal basis, for which multiplication is both simpler and more efficient. The existence of optimal normal basis (Type I and Type II) has been completely characterized in [10] and [12]. Since we use the sparse storing scheme for matrix Λ, the proposed algorithm here can handle efficiently the field multiplication on optimal normal basis representations of both Types I and II without modifying the algorithm.

References

1. R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge university press, 1987.
2. IEEE P 1363 - 2000, *Standard Specifications for Public Key Cryptography*, August, 1998.
3. J. L.Massey and J. K. Omura, *Computational method and apparatus for finite field arithmetic*, U. S. Patent 4, 587, 627, May 1986.
4. R. C. Mullin, *Multiple Bit Multiplier*, U. S. Patent 5, 787, 028, July 1998.

5. R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, *Computational Method and Apparatus for Finite Field Multiplication*, U. S. Patent 4, 745, 568, May 1988.
6. E. DeWin, A. Bosselaers, S. Vandenberghe, P. De Gersim and J. Vandewalle, *A Fast Software Implementation for Arithmetic Operations in F_{2^m}*, In proc. Asiacrypt '96 , 1996.
7. D. Hankerson, J. L. Hernandez, and A. Menezes, *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, In Proc. CHES' 2000, August 2000.
8. J. Lopez and R. Dahab, *High-speed Software Multiplication in F_{2^m}*, Technical report, IC-00-09, May 2000.
9. A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, Boston, 1993.
10. R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone and R. M. Wilson, *Optimal Normal Basis in $GF(p^n)$*, in Discrete App. Math. Vol.22, pp149-161, 1988.
11. P.Ning and Y. L. Yin, *Efficient Software Implementation for Finite Field Multiplication in Normal Basis*, preprint.
12. S. Gao and H. W. Lenstra, *Optimal Normal Basis*, Designs, Codes and Cryptography, vol.2, pp315-323, 1992.

*
SeYoung Oh
Department of Mathematics
Chungnam National University
Taejon 305-764, Korea
E-mail: soh@math.cnu.ac.kr

**
ChungSup Yoon
Department of Mathematics
Chungnam National University
Taejon 305-764, Korea
E-mail: csyoon@math.cnu.ac.kr