# Fast Algorithms for Elliptic Curve Cryptosystems over Binary Finite Field

[Published in K. Y. Lam and E. Okamoto, Eds., *Advances in Cryptology – ASIACRYPT '99*, vol. 1716 of *Lecture Notes in Computer Science*, pp. 75–85, Springer-Verlag, 1999.]

Yongfei Han[1], Peng-Chor Leong[2], Peng-Chong Tan[2], and Jiang Zhang[1]

[1] Gemplus Corporate R&D, APDC Security & Crypto Dept
89, Science Park Drive #04-01/05, Singapore 118261
yfh69@hotmail.com
[2] Centre for Advanced Information Systems, School of Applied Science,
Nanyang Technological University, Singapore 639798

**Abstract.** In the underlying finite field arithmetic of an elliptic curve cryptosystem, field multiplication is the next computational costly operation other than field inversion. We present two novel algorithms for efficient implementation of field multiplication and modular reduction used frequently in an elliptic curve cryptosystem defined over $GF(2^n)$. We provide a complexity study of the two algorithms and present an implementation performance of the algorithms over $GF(2^{167})$.

**Keywords.** Galois field arithmetic, elliptic curve cryptosystems, field multiplication, modular reduction.

## 1 Introduction

In 1985, Neil Koblitz and Victor Miller independently proposed the elliptic curve cryptosystem, whose security rests on the discrete logarithm problem over points on an elliptic curve. Elliptic curve cryptography can be used to provide both a digital signature scheme and an encryption scheme. With the apparent advantage of high cryptographic strength relative to key size, elliptic curve cryptosystems [9, 14] have gained much popularity in the implementation of discrete logarithm based public key protocols. The shorter key size generally leads to improved computational efficiencies and smaller storage and bandwidth requirements. Although elliptic curve cryptosystem can be based on finite field of any characteristic, it is generally practical to implement within the prime or binary finite field [9, 14].

Certain classes of elliptic curves such as the subfield curves, supersingular and anomalous binary curves have been proposed which provide improved efficiencies in implementation. However the extra structure provided by these curves

are subjected to attack and reviewed recently [17, 4]. We consider only non-supersingular and non-anomalous elliptic curves over non-composite field in the paper. The algorithms presented are specifically for binary finite field with standard basis representation.

Efficient implementation of elliptic curve cryptography can be focused on 2 levels. At elliptic curve group level, fast algorithms for multiplying a base point $P$ of an elliptic curve may be applied [5–7]. The computation of multiplying a point $P$ of an elliptic curve group by a large integer $d$ is analogous to exponentiation of an element in a multiplicative group to the $d^{th}$ power. The generally accepted algorithm for the computation is the "square-and-multiply" algorithm. Signed digit representation, $k$-SR representation, addition chains and sliding window methods are applied to the computation of scalar multiplication, as the are employed to exponentiation [1, 2].

For the underlying finite field arithmetic, more efficient algorithms speeding up computation of field multiplication and inversion may be introduced. Field multiplication is the next costly operation other than field inversion. Various algorithms, such as the transformation to projective coordinates trade field inversion for field multiplication. Hence, it is desirable to provide fast and effective field multiplication and modular reduction.

The purpose of this paper is to present new approaches for field multiplication and a modular reduction commonly performed in elliptic curve cryptosystems defined over binary finite field.

First, we review previous works in section 2. In section 3, we present a method to speed up computation of field multiplication. This algorithm is applicable to standard basis representation of elements in Galois field $GF(2^n)$. The algorithm is based on modified classical "shift-and-add" method. Through elimination of extensive shiftings, our algorithm is suited for microprocessors that have small word size, and only instruction that can shift only one bit at a time. Such microprocessors are common in 8 or 16 bit microcontrollers and smartcards. While there exist fast binary finite field multiplication with the use of table look-up [10], such algorithms are generally not suitable for computing multiplication of field elements with degree $> 5$ using low end microprocessor with limited memory.

In section 4, we present an efficient modular reduction based on optimization of Schroeppel's modular reduction technique [16], and our method is more efficient than Schroeppe;'s approach. Detailed analysis of the complexity and performances of our algorithms will also be presented.

We conclude this article with the comparison of implementation results for field multiplication and reduction over $GF(2^{167})$. Relative to the the classical "shift-and-add" method of multiplication, our implementation result shows approximately 12 percent reduction in computation time for a general purpose 32 bits microprocessor. As for the field modular reduction, 14 percent reduction of the computation cost can be realised.

## 2 Previous Works

An elliptic curve, defined on a field $K$ =GF($2^n$) where $n$ is a prime, is the set of solution points$(x, y)$ to an equation of the form:

$$y^2 + xy = x^3 + ax + b$$

with $a, b \in K$.

The set of points on an elliptic curve, together with a special point called the *point of infinity* can be equipped with an Abelian group structure by the following point addition operation:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

$$x = a + \lambda^2 + \lambda + x_1 + x_2\,, \qquad y = (x_1 + x)\lambda + x + y_1$$

And by following point doubling operation:

$$\lambda = x_1 + \frac{y_1}{x_1}$$

$$x_2 = a + \lambda^2 + \lambda\,, \qquad y_2 = x_1^2 + \lambda x_2 + x_2$$

The simplest technique to compute multiplication in $GF(2)$ is to use the "shift-and-add" method. As no arithmetic carry over is involved, the "shift-and-add" method is a neat and easy method for implementation. Addition in $GF(2)$ is simply the bitwise exclusive-or operation.

Selection of an elliptic curve is a critical step before the implementation. The curve selected should not be a supersingular curve or anomalous curve.

For computation of field multiplication over $GF(2^n)$, we noted that word level multiplication in $GF(2)$ is usually not supported in general microprocessors. There are 2 common software implementation techniques to achieve the $GF(2)$ multiplication.

- Table look-up method
- Emulation using "shift-and-add" technique

In Table look-up method [10], the field multiplication result are first pre-computed. A simple method is to use 2 tables, to store the higher order and lower order of the multiplication result. The tables are addressed using the bits of the multiplier and multiplicand. Therefore a 8-bits word for $GF(2)$ multiplication would require $2 \times 2^8 \times 2^8 \times 8$ bits $= 128$ Kbyte of storage space for the look up tables. For 16 bit operands, $2 \times 2^{16} \times 2^{16} \times 16$ bits $= 16$ GByte of look up table would be required. Although there exists techniques [10] to handle 16 bit $GF(2)$ multiplication using 8-bits look up table, we noted that the overheads is not favourable for microprocessors without special shift instruction and not practical with devices with extremely limited memory.

The simplest technique to compute multiplication in $GF(2)$ is to use the "shift-and-add" method. Addition in $GF(2)$ is simply the bitwise exclusive-or operation. As no arithmetic carry over is involved, the "shift-and-add" method is a neat and simple method for implementation.

## 3 A New Approach for Multiplication

To compute the multiplication of two field elements $A$ and $B$ in standard basis over $GF(2^n)$, the classical "shift-and-add" algorithm as described in [16] is commonly used. The classical method typically incurs computational cost of shifting $2s(n-1)$ bits of the intermediate results; where $n$ is the number of bits of the field.

Our algorithm attempts to eliminate the extensive number of shift operations which inherently contributes to a large part of the computational cost. Our method requires shifting $2s(w-1)$ bits of the intermediate results; where $w$ is the wordsize of the microprocessor, and $s = \lceil n/w \rceil$. This contributes to greater performance improvement, particularly for microprocessors with small word size. It is noted that the number of field additions remains the same as classical method since it is dependent on the hamming weight of the multiplier. As 'addition' in $GF(2)$ operation does not involve 'carry', with the addition operator defined as exclusive-or operation, the saving in shift operations is possible with our new algorithm.

### 3.1 Efficient Field Multiplication Algorithm

**Definition 1.** *Let $A, B = (b_{sw-1}...b_1b_0) \in GF(2^n)$ be the bit-string representation of the multiplier $B$. $B$ can be partitioned into $s$ blocks and each block is of length $w$ bits, and $s = \lceil n/w \rceil$. Denote $t_i = A \cdot b_i \cdot x^i, i \in \{0..n-1\}$*

The field multiplication result of $C = A \times B = A \sum_{i=0}^{w-1} x^i \left( \sum_{j=0}^{s-1} \left( b_{jw+i} x^{jw} \right) \right)$ where $B = (b_{sw-1}...b_1b_0) \in GF(2^n)$ can be re-expressed as:

$$
\begin{aligned}
C &= (t_0 + t_w x^w + t_{2w} x^{2w} + \cdots + t_{(s-1)w}) x^{(s-1)w} + (t_1 x + t_{w+1} x^{w+1} \\
&\quad + t_{2w+1} x^{2w+1} + \cdots + t_{(s-1)w+1} x^{(s-1)w+1}) \\
&\quad + \cdots + (t_{w-1} x^{w-1} + t_{w+(w-1)} x^{w+(w-1)} + t_{2w+(w-1)} x^{2w+(w-1)} \\
&\quad + \cdots + t_{(s-1)w+(w-1)} x^{(s-1)w+(w-1)}) \\
&= A(b_0 + b_w.x^w + b_{2w}.x^{2w} + ... + b_{(s-1)w}.x^{(s-1)w}) \\
&\quad + A(b_1.x^1 + b_{w+1}.x^{w+1} + b_{2w+1}.x^{2w+1} + \cdots + b_{(s-1)w+1}.x^{(s-1)w+1}) \\
&\quad + \cdots + A(b_{w-1}.x^{w-1} + b_{2w-1}.x^{2w-1} + b_{3w-1}.x^{3w-1} + \cdots + b_{sw-1}.x^{sw-1}) \\
&= A(b_0 + b_w.x^w + b_{2w}.x^{2w} + ... + b_{(s-1)w}.x^{(s-1)w}) \\
&\quad + x(A(b_1 + b_{w+1}.x^w + b_{2w+1}.x^{2w} + \cdots + b_{(s-1)w+1}.x^{(s-1)w} \\
&\quad + \cdots + x^{w-1}(A(b_{w-1} + b_{w+(w-1)}.x^w + b_{2w+(w-1)}.x^{2w} \\
&\quad + b_{(s-1)w+(w-1)}.x^{(s-1)w}) \ldots)
\end{aligned}
$$

The efficient field multiplication is based on the following model of computing the intermediate results of $(t_0, t_w, ..., t_{(s-1)w})$ first and progressively on the next

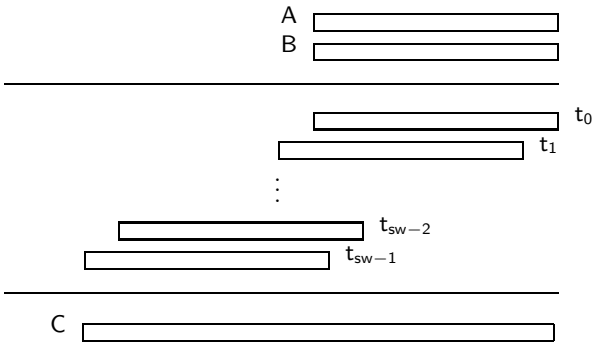


**Fig. 1.** Model of improved multiplication algorithm

tuple $(t_1, t_{w+1}, ..., t_{(s-1)w+1})$, until all the intermediate results are computed in which the last tuple is $(t_{w-1}, t_{w+(w-1)}, ..., t_{(s-1)w+(w-1)})$.

**Algorithm 1** *Denote LeftShift(X) as left shifting the coefficient representation of the polynomial X by 1 bit and C[j] denotes the $j^{th}$ word of the coefficient representation of polynomial C where $j \in \{0..(s-1)\}$.*

NEW-METHOD-FOR-FIELD MULTIPLICATION$(A, B)$

```
1   C ← 0
2   for j = w − 1 downto 0
3       do p ← 0
4           for k ← 1 downto s
5               do if B_{kw−1} = 1
6                   then for i ← (s − 1) + p downto p
7                       do C[i] ← C[i] ⊕ A[i − p]
8               p ← p + 1
9           LeftShift(C)
10          LeftShift(B)
11  return C
```

| | No. of shift operations | |
|---|---|---|
| Shift operation cost | "Shift-and-add" method | Our method |
| Number of shift operation on C | $2s(n-1)$ | $2s(w-1)$ |
| Number of shift operation on B | $s(w-1)$ | $s(w-1)$ |
| Total shift operations | $2s(n-1)+s(w-1)$ | $2s(w-1)+s(w-1)$ |

**Table 1.** Computational time cost comparison for field multiplication

Using the classical "shift-and-add" method, $2s(n-1)$ shift operations are required to compute the field multiplication in $GF(2^n)$. With the new ap-

proach, only $2s(w-1)$ shift operations are incurred without any need for pre-computation. The relation between the number of bits $n$ of the underlying field and the word size, $w$ would determine if the new algorithm would be more efficient compared to the binary method. The new algorithm will perform even more efficiently than the binary method when

$$2s(w-1) < 2s(n-1) \text{ or equivalently } w < n$$

The word size of general microprocessor are usually 8, 16, 32 and 64 bits and for elliptic curve over $GF(2^n)$, $n$ is usually chosen to be about 160 bits. It is noted that when the field size of the elliptic curve is increased, the new algorithm will perform more efficiently compare to the classical "shift-and-add" field multiplication method. This is because the number of shift operation performed on element C would remain unchanged for the new approach, whereas the number of shift operations using the "shift-and-add" method would depend on the field size $n$.

Table 2 compares the two methods based on a typical 167 bits field for elliptic curve cryptosystem, with $s$ defined as the number of words, and $w$ as the wordsize of the microprocessor.

| 167 bits field | | No. of shift operations | | |
|---|---|---|---|---|
| $w$ | $s$ | "Shift-and-add" method | Our method | Percentage Savings |
| 8 | 21 | 7119 | 441 | 94% |
| 16 | 11 | 3817 | 495 | 87% |
| 32 | 6 | 2178 | 558 | 74% |
| 64 | 3 | 1185 | 567 | 52% |

**Table 2.** Computational time cost comparison for $GF(2^{167})$ field multiplication

## 4 Modular Reduction

The result of field multiplication requires storage length of $2 \times sw$. Modular reduction can be done very efficiently with an irreducible polynomial, such as trinomial and pentanomial using shifts and additions. The idea is to zero out the upper bits and add the representation of each original term right shifted by some quantity. Schroeppel et al. describes a practical approach of working on one computer word at a time to systematically perform the polynomial modular reduction [16].

We consider a trinomial modulus of the expression $x^n+x^k+1$, where $n = 167$, $k = 6$. After each field multiplication or squaring, the result must be reduced modulo $F(x) = x^{167} + x^6 + 1$.

The product of two polynomials of degree 166 produces a polynomial of degree 332. Assume the polynomial to be reduced is:

$$P(x) = a_{332}x^{332} + ... + a_1x + a_0$$

Then the reduction modulo $x^{167}+x^6+1$ proceeds by reducing each term modulo the trinomial and subtracting it from the result. We noted that:

$$x^{167} \equiv x^6 + 1$$
$$x^n \equiv x^{n-161} + x^{n-167} \pmod{F(x)}$$

Instead of working on one computer word at a time, and lowering the degree of the polynomial by a word, proceeding from the high order terms to the low, our approach is to work on $\frac{s}{2}$ words at a time and lowering the degree by $\frac{s}{2}$. For our approach to be effective, it is therefore desirable to choose a trinomial with low $k$ degree.

**Algorithm 2** *Let $A$ be the result of field multiplication or squaring prior to modular reduction. $A$ has degree of at most $2n-2$. $A$ can be partitioned into $2s$ blocks and each block is of length $w$ bits. Let $A_i$ denotes the $i^{th}$ block of the partition of field element $A$. CarryRightShift(Q,d,T) denotes right shifting the memory location range in $Q$ by $d$ bits making use of the word shift with carry instruction available in general microprocessor, and that the carry bits are stored in $T$. $Temp1$ and $Temp2$ are registers of wordsize $w$. The following algorithm performs the modular reduction on $A$ using the trinomial modulus, $x^n \equiv x^k + 1 \pmod{x^n + x^k + 1}$, when $k < \lfloor \frac{n}{2} \rfloor$ and $n - k > w$.*

NEW-METHOD-FOR-TRINOMIAL-MODULAR-REDUCTION($A$)

1 $p \leftarrow ((n-k) \bmod w)$
2 $q \leftarrow \lfloor \frac{n-k}{w} \rfloor$
3 $u \leftarrow (n \bmod w)$
4 $t \leftarrow \lfloor \frac{n}{w} \rfloor$
5 $Temp1 \leftarrow 0$
6 $Temp2 \leftarrow 0$
7 CarryShiftRight($A_{2s-1}...A_{2s-\lceil \frac{s}{2} \rceil}, p, Temp1$)
8 **for** $j = 2s-1$ **downto** $2s - \lceil \frac{s}{2} \rceil$
9 **do** $A_{j-t} \leftarrow A_{j-t} \oplus A_j$
10 $A_{2s-\lceil \frac{s}{2} \rceil - t - 1} \leftarrow A_{2s-\lceil \frac{s}{2} \rceil - t - 1} \oplus Temp1$
11 CarryShiftRight($A_{2s-1}...A_{2s-\lceil \frac{s}{2} \rceil}, u-p, Temp1$)
12 **for** $j = 2s-1$ **downto** $2s - \lceil \frac{s}{2} \rceil$
13 **do** $A_{j-q} \leftarrow A_{j-q} \oplus A_j$
14 $A_{2s-\lceil \frac{s}{2} \rceil - q - 1} \leftarrow A_{2s-\lceil \frac{s}{2} \rceil - q - 1} \oplus Temp1$
15 CarryShiftRight($A_{s-1+\lfloor \frac{s}{2} \rfloor}...A_s, p, Temp1$)
16 **for** $j = s - 1 + \lfloor \frac{s}{2} \rfloor$ **downto** $s$
17 **do** $A_{j-t} \leftarrow A_{j-t} \oplus A_j$

18 $A_{s-t-1} \leftarrow A_{s-t-1} \oplus Temp1$
19 CarryShiftRight$(A_{s-1+\lfloor \frac{s}{2} \rfloor} ... A_s, u-p, Temp1)$
20 **for** $j = s-1+\lfloor \frac{s}{2} \rfloor$ **downto** $s$
21 **do** $A_{j-q} \leftarrow A_{j-q} \oplus A_j$
22 $A_{s-q-1} \leftarrow A_{s-q-1} \oplus Temp1$
23 $mask \leftarrow (2^w - 1) \ll (n \bmod w)$
24 $Temp1 \leftarrow A_{s-1} \wedge mask$
25 CarryShiftRight$(Temp1, p, Temp2)$
26 $A_{(s-1)-q} \leftarrow A_{(s-1)-q} \oplus Temp1$
27 $A_{(s-1)-q-1} \leftarrow A_{(s-1)-q-1} \oplus Temp2$
28 **return** $C \leftarrow (A_{(s-1)} ... A_0)$

Table 3 compares the computational cost on the number of shift and addition operations required for Schroeppel's method and our improved method.

| | No. of operations | |
|---|---|---|
| Operation cost | Schroeppel's method | Improved method |
| No. of shift operations on $A$ | $s(p+(w-p))$ | $s(max(p,u))$ |
| No. of shift operations on temporary variables | $p+(w-p)$ | $3\times$ max$(p,u)$ |
| Total shift operations | $(s+1)\times w$ | $(s+3)\times$(max$(p,u)$) |
| Total no. of Field Additions | $4s+2$ | $2s+6$ |

**Table 3.** Computational cost comparison for field modular reduction

A careful choice of reduction trinomial that has small value of $p = ((n-k) \bmod w)$ and $u = (n \bmod w)$ will boost efficiencies in our new algorithm. Further to the elimination of extensive shifting, the number of field additions is also reduced by a factor of 2 in our approach, this is achieved with the alignment on the degrees of the congruent terms with the microprocessor word size.

## 5 Performance of Implementation

The following table present the performance benchmark of the improved field multiplication and modular reduction in an elliptic curve cryptosystem defined over $GF(2^{167})$. The computation is based on C source codes compiled with Microsoft Visual C++ 5.0 without compiler's optimization. An Intel Pentium II 32 bit microprocessor running at 333 MHz was used to conduct the benchmarking.

Comparing our new approaches of field multiplication and modular reduction to the classical methods, the timing results shows about 12 percent improvement for the multiplication, and approximately 14 percent improvement is achieved for the modular reduction.

| Field Arithmetic | Classical method | Our method | Percentage Savings |
|---|---|---|---|
| Multiplication | 0.27ms | 0.20ms | 12% |
| Modular Reduction | 0.07ms | 0.06ms | 14% |

**Table 4.** Computational timing cost for field multiplication and modular reduction

## References


1. Yongfei Han, C. Mitchell, D. Gollmann, "Minimal Weight $k$-SR Representation." In Proceedings of fifth IMA Conference on Cryptography and Coding, LNCS1025, pages 34-43, Cirencester, U. K, 1995. Springer-Verlag, Berlin.
2. D. Gollmann, Yongfei Han, C. Mitchell, "Redundant integer representations and fast exponentiation." Designs, Codes and Cryptography, 7 (1996), pages 135-151.
3. Erik Dewin, "Fast software Implementation for Arithmetic Operations in $GF(2^n)$", Advances in Cryptology, Proceedings Asiacrypt '96, LNCS 1163, Springer-Verlag, 1996, p.p 65-76
4. Robert Gallant, Robert Lambert, Scott Vanstone, Improving the Parallelized Pollard Lambda Search on Binary Anomalous Curves, P1363 Standards Internet Web Site, 1998. Online. Available: http://grouper.ieee.org/groups/1363/contributions
5. Jorge Guajardo, Christof Paar, "Efficient Algorithms for Elliptic Curve Cryptosystem", CRYPTO '97, Springer-Verlag, LNCS 1294, pp. 342-356, 1997
6. Yongfei Han, Jiang Zhang, Peng-Chong Tan,"Efficient Elliptic Curve Cryptosystems ", International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99), Hong Kong, 1999.
7. Yongfei. Han, J. Zhang, P.-C. Tan," Direct Computation for Elliptic Curve Cryptosystem", Workshop on Cryptographic Hardware and Embedded Systems, Worcester Polytechnic Institute, Worcester, Massachusetts, 1999.
8. D.E Knuth, The Art of Computer Programming, Vol.2 : Seminumerical Algorithms, 2nd edition, Addison-Wesley, 1981.
9. N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation 48 (1987), 203-209
10. Cetin K. Koc, Tolga Acar. "Montgomery Multiplication in $GF(2^k)$", *Design, Codes and Cryptography, 1-14(1997)*, Kluwer Academic Publishers, Boston, 1997
11. R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge University Press, Cambridge, UK, 1994
12. A.J Menezes Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers, Boston, MA, 1993
13. R. J. McEliece. Finite Fields for Computer Scientist and Engineers. Kluwer Academic Publishers, 1987
14. V. Miller. Uses of elliptic curves in cryptography, In Advances in Cryptology - CRYPTO '85, pages 417-426. Springer-Verlag, Berlin 1986.
15. Francois Morain et J. Olivos, "Speeding up the Computation on an Elliptic Curve using Addition-Subtraction Chains", *RAIRO Informatique Theorique Et Applications - Theoretical Informatics and Applications*, 24,6, p. 531-543, 1990.
16. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. "Fast key exchange with elliptic curve systems". In D. Coppersmith, editor, Advances in Cryptology - CRYPTO 95, Lecture Notes in Computer Science, No. 973, pages 43-56, New York, NY, 1995. Springer-Verlag.

17. Michael J. Wiener, Robert J, Zuccherato, Faster Attacks on Elliptic Curve Cryptosystems, IEEE P1363 Standards Internet Web Site, April 1998. Online. Available: http://grouper.ieee.org/groups/1363/contributions