Efficient Algorithms for Elliptic Curve Cryptosystems*

Jorge Guajardo
(guajardo@ece.wpi.edu)

Christof Paar
(christof@ece.wpi.edu)

ECE Department
Worcester Polytechnic Institute
Worcester, MA 01609, USA

Abstract. This contribution describes three algorithms for efficient implementations of elliptic curve cryptosystems. The first algorithm is an entirely new approach which accelerates the multiplications of points which is the core operation in elliptic curve public-key systems. The algorithm works in conjunction with the k-ary or sliding window method. The algorithm explores computational advantages by computing repeated point doublings directly through closed formulae rather than from individual point doublings. This approach reduces the number of inversions in the underlying finite field at the cost of extra multiplications. For many practical implementations, where field inversion is at least four times as costly as field multiplication, the new approach proofs to be faster than traditional point multiplication methods. The second algorithm deals with efficient inversion in composite Galois fields of the form $GF((2^n)^m)$. Based on an idea of Itoh and Tsujii, we optimize the algorithm for software implementation of elliptic curves. The algorithm reduced inversion in the composite field to inversion in the subfield $GF(2^n)$. The third algorithm describes the application of the Karatsuba-Ofman Algorithm to multiplication in $GF((2^n)^m)$. We provide a detailed complexity analysis of the algorithm for the case that subfield arithmetic is performed through table look-up. We apply all three algorithms to an implementation of an elliptic curve system over $GF((2^{16})^{11})$. We provide absolute performance measures for the field operations and for an entire point multiplication.

1 Introduction

Elliptic curve (EC) cryptosystems were first suggested by Miller [16] and Koblitz [8]. A main feature that makes EC attractive is the relatively short operand length relative to RSA and systems based on the discrete logarithm (DL) in finite fields. Cryptosystems which explore the DL problem over EC can be built with an operand length of 150–200 bits [15]. IEEE [11] and other standard bodies such as ANSI and ISO are in the process of standardizing EC cryptosystems. EC can provide various security services such as key exchange, privacy through

* This work was sponsored in part by GTE Corporation.

encryption, and sender authentication and message integrity through digital signatures. For these reasons it is expected that EC will become very popular for many information security applications in the near future. It is thus very attractive to provide algorithms which allow for efficient implementations of EC cryptosystems. Our contribution will deal with such algorithms.

Efficient algorithms for EC can be classified into high-level algorithms, which operate with the group operation, and into low-level algorithms, which deal with arithmetic in the underlying finite field. For efficient implementations it is obviously the best to optimize both types of algorithms. Our contribution will introduce three algorithms, one high-level algorithm for point multiplication and two low-level for finite field inversion and multiplication, respectively.

In Sect. 3 we introduce an entirely new approach for accelerating the multiplication of points on an EC. The approach works in conjunction with the k-ary and the sliding window methods. The method is applicable to EC over any field, but we provide worked-out formulae for EC over fields of characteristic two.

Although EC cryptosystems can be based on finite fields of any characteristic, practical systems have only been implemented over prime fields or Galois fields of characteristic two. Section 4 shows an algorithm for efficiently computing the inverse of an element in the composite Galois field $GF((2^n)^m) \cong GF(2^k)$. The algorithm is based on an idea by Itoh and Tsujii [6], but our approach is optimized for a standard basis representation and for binary field polynomials. The algorithm reduces inversion in the composite field to inversion in the subfield $GF(2^n)$. Unlike the inversion algorithms in [21, 22], the inversion algorithm is not based on Euclid's algorithm.

Section 5 provides a detailed treatment of the Karatsuba-Ofman algorithm (KOA) applied to field multiplication in $GF((2^n)^m)$. We provide a complexity analysis of the KOA for software implementations where arithmetic in the subfield $GF(2^n)$ is based on table look-up.

Section 6 shows the actual performance of all three algorithms in an implementation of an EC cryptosystem over $GF(2^{176}) \cong GF((2^{16})^{11})$. We provide absolute timing measurements for an entire EC multiplication as well as timings for individual operations.

2 Previous Work

As stated above, this contribution describes a new algorithm for point multiplication and optimized inversion and multiplication algorithms for arithmetic in composite Galois fields $GF((2^n)^m)$. In the sequel we will summarize some of the previous work in each of those areas.

The problem of multiplying a point P of an EC by a (large) integer n is analogous to exponentiation of an element in a multiplicative group to the nth power. The standard algorithm for this problem is the binary exponentiation method (or square-and-multiply algorithm) which is studied in detail in [7]. A generalization of the binary method is the k-ary method [2] which processes k exponent bits in one iteration. Further improvements of the k-ary method

include the sliding window method [10]. In [21] a version of the k-ary method is applied to point multiplication for an EC. The method that we propose in Sect. 3 explores arithmetic advantages that occur if several point doublings are computed from closed expressions rather than from computing several individual doublings. This approach is unique to EC systems and, to our knowledge, has not been reported anywhere else. In [9] and [14] direct formulae to compute $Q = nP$ are introduced, where n is an integer. However, these formulae are based on the computation of division polynomials and they do not appear to be computationally efficient if used for the fast calculation of nP.

Software implementations of EC over composite Galois field $GF((2^n)^m)$ were first described in [5] for the field $GF((2^8)^{13})$. More recently, EC systems for the field $GF((2^{16})^{11})$ were described independently in [22] and [1]. In all cases, field multiplication is accomplished through table look-up in the subfield $GF(2^n)$. Neither reference explores advanced convolution algorithms, such as the Karatsuba-Ofman algorithm (KOA), as it is proposed here. The KOA has been studied for general polynomial multiplication in [3] and [7]. An application of the KOA to polynomials over finite fields in the context of computer hardware is described in [19]. None of the previous contributions provide a detailed complexity analysis of the algorithm as it is done in Sect. 5 of this work. Composite Galois fields are applied to a hardware architecture for public-key algorithms in [20].

Previously, there were two principle approaches to inversion in finite fields. One is based on the extended Euclidean algorithm, the other one is based on Fermat's Little Theorem. [21] optimizes a version of the Euclidean algorithm, named "almost inverse algorithm," for an EC implementation over $GF(2^{155})$. [22] applies the same algorithm for EC over the composite field $GF((2^{16})^{11})$. In [6] an efficient method for inversion in $GF(2^k)$ is discussed. In Sect. 6 of the reference the method is extended to composite Galois fields $GF((2^n)^m)$. The reference only considers a normal basis representation, whereas we will optimize the inversion algorithm for composite fields in standard basis representation and for binary field polynomials.

3 Point Multiplication with a Reduced Number of Inversions

3.1 Elliptic Curves over $GF(2^k)$

In this paper, we will only be concerned with non-supersingular elliptic curves. Thus, an elliptic curve E will be defined to be the set of points (x, y) with coordinates x and y lying in the Galois field $GF(2^k)$ and satisfying the cubic equation $y^2 + xy = x^3 + ax^2 + c$, where $a, c \in GF(2^k)$, $c \neq 0$, together with the point at infinity $\mathcal{O}$. The points (x, y) form an abelian group under "addition" where the group operation is defined as in [15]. In what follows, we will only be concerned with the doubling of a point P, $2P = (x_1, y_1)$. This is achieved by

$$x_1 = \left(x + \frac{y}{x}\right)^2 + \left(x + \frac{y}{x}\right) + a \tag{1}$$

$$y_1 = x^2 + \left(x + \frac{y}{x}\right) x_1 + x_1 \tag{2}$$

From (1) and (2) it can be seen that the doubling of two points in E will require one inverse, two multiplications, five additions, and two squarings in the underlying field $GF(2^k)$. Notice also that in most practical applications, inversion is by far the most expensive operation to perform. In the following we will develop a scheme which reduces the number of field inversions.

3.2 A New Approach to Point Doubling

Public-key schemes which use EC are based on the DL problem in the point group of the EC. Such schemes include analogies to the Diffie-Hellman key establishment protocol, the ElGamal encryption, and various digital signature schemes. The basic operation for the DL problem for EC is "multiplication" of a point $P \in E$ with an integer n, which is of the order of $\#E$. One way of performing this operation is analogous to the square and multiply algorithm for exponentiation [7] and it is known as "repeated double and add" [11]. A generalization of this method is the k-ary method [2, 10] which reduces the number of additions needed in the regular double and add. Theorem 1 was adapted from [13] and it describes the algorithm as it applies to EC.

Theorem 1 *Let $P \in E$ and $n = (e_t e_{t-1} \cdots e_1 e_0)_b$ be the radix representation of the multiplier n in base b where $b = 2^k$ for $k \geq 1$. Then, $Q = nP$ can be computed using the following algorithm.*

Algorithm (Input: $P = (x, y)$; Output: $Q = nP$)
1. Precomputation
   1.1 $P_0 \leftarrow \mathcal{O}$ (Point at infinity)
   1.2 For $i = 1$ to $2^k - 1$
      $P_i = P_{i-1} + P$ (i.e., $P_i = i \cdot P$)
2. $Q \leftarrow \mathcal{O}$
3. For $i = t$ to 0
   3.1 $Q \leftarrow 2^k Q$
   3.2 $Q \leftarrow Q + P_{e_i}$
4. Return(Q)

Notice that Step 1.2 requires $2^k - 3$ additions and one doubling, Step 3.1 involves the doubling of point Q, k times, and Step 3.2 requires one point addition. The complexity of the k-ary method with t iterations is thus $kt + 1$ point doublings, $2^k - 3 + t$ point additions. Since point doublings are the most costly operations, it is extremely attractive to find ways of accelerating the doubling operation. In the following, we will introduce an entirely new approach to compute repeated point doublings over an EC.

Our new approach is based on the following principle. First, observe that the k-ary method relies on k *repeated* doublings. The new approach allows computation of $2^k P = (x_k, y_k)$ directly from $P = (x, y)$ without computing the intermediate points $2P$, $2^2 P$, $\cdots$,$2^{k-1} P$. Such direct formulae are obtained by

inserting (1) and (2) into one another. For instance, for $4P = 2^2P = (x_2, y_2)$, we obtain

$$x_2 = \frac{\zeta^2 + (\delta\gamma)\zeta}{(\delta\gamma)^2} + a \tag{3}$$

$$y_2 = \frac{\zeta(\delta\gamma)x_2 + (\delta^2)^2}{(\delta\gamma)^2} + x_2, \tag{4}$$

where $\gamma = x^2$,$\eta = \gamma + y$, $\delta = \eta^2 + \eta x + a\gamma$, $\xi = \eta x + \gamma$, and $\zeta = \delta(\delta + \xi) + \gamma^2\gamma$. Notice that (3) and (4) imply that $2^2P = (x_2, y_2)$ can be computed with one inverse, nine multiplications, ten additions, and six squarings. The advantage of (3) and (4) is that they only require one inversion as opposed to the two inversions that two separate double operations would require for computing $4P$. The "price" that must be paid is $9 - 4 = 5$ extra multiplications if squarings and additions are ignored. For $k = 2$, the direct formulae (3) and (4) trade thus one inversion at the cost of 5 multiplications. It is easy to see that the formulae are an advantage in situations where multiplication is at least five times as costly as inversion. However, this "break-even point" decreases if the method is extended to the computation of 2^kP for $k > 2$ as described below.

We continued in a similar manner and found expressions for $2^3P = (x_3, y_3)$ and $2^4P = (x_4, y_4)$. Again, these expressions, shown in the appendix, only require one inversion as opposed to the three or four inversions that the regular double and add algorithm would require in each one of these cases. It is important to point out that the point P has to be an element with an order larger than 2^k. This last requirement ensures that $4P$, $8P$, or $16P$ will never equal $\mathcal{O}$. Notice that this is compliant with [11].

3.3 Comparison

For application in practice it is highly relevant to compare the complexity of our newly derived formulae with that of the double and add algorithm. If we note that our method reduces inversions at the cost of multiplications, the performance of the new method depends on the cost factor of one inversion relatively to the cost of one multiplication. For this purpose we introduce the notion of a "break-even point." Since it is possible to express the time that it takes to perform one inversion in terms of the equivalent number of multiplication times, we define the break even point as the number of multiplication times needed per inversion so that our formulae outperform the regular double and add algorithm. The results are summarized in Table 1.

3.4 Applications

Our new method for point doubling can directly be applied to all EC defined over fields of characteristic two, regardless of the specific field representation. For instance, the formulae in the appendix are applicable to a composite field representation $GF((2^n)^m)$ as well as to binary field representations $GF(2^l)$. In

Table 1. Complexity comparison: Individual doublings vs. direct computation of several doublings

Calculation	Method	Complexity				Break-Even Point
		Sq.	Add.	Mult.	Inv.	
$4P$	Direct Doublings	6	10	9	1	1 inv. > 5 mult.
	Individual Doublings	4	10	4	2	
$8P$	Direct Doublings	7	17	14	1	1 inv. > 4 mult.
	Individual Doublings	6	15	6	3	
$16P$	Direct Doublings	15	20	19	1	1 inv. > 3.7 mult.
	Individual Doublings	8	20	8	4	

addition, the choice of of the basis does not matter. Standard, dual, and normal basis representations are all possible. The latter observation is of special interest since efficient inversion methods for a normal basis representation appear not to be known, and inversion based on Fermat's Theorem using addition chains requires at least 7 multiplications if $l > 128$. We expect our method yields a considerable acceleration in such situations.

In situations where the ratio between an inversion time and a multiplication is three or smaller [22], our method may not give an advantage. However, we would like to point out that as shown in Sect. 6, the repeated point multiplication behaves better than predicted in our implementation, so that the break-even points might in practice be even lower than shown in Table 1.

4 Efficient Inversion in Composite Fields $GF((2^n)^m)$

Composite Galois fields for EC systems were explored earlier in [5, 1, 22]. The following notation will be used here. We consider arithmetic in an extension field of $GF(2^n)$. The extension degree is denoted by m, so that the field can be denoted by $GF((2^n)^m)$. This field is isomorphic to $GF(2^n)/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i, p_i \in GF(2^n)$, is a monic irreducible polynomial of degree m over $GF(2^n)$. In the following, a residue class will be identified with the polynomial of least degree in this class. We consider a standard (or polynomial or canonical) basis representation of a field element A:

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \tag{5}$$

where $a_i \in GF(2^n)$. Note that it is possible to choose $P(x)$ with binary coefficients if $\gcd(n, m) = 1$ [12], a fact that will be explored for the inversion algorithm below.

As stated above, inversion is the most costly arithmetic operation in EC systems. In the following an inversion method based on Fermat's Little Theorem will be developed which is entirely different from the approach in [22, 21]. The

basic property of the algorithm developed in this section is that inversion in $GF((2^n)^m)$ is reduced to inversion in the subfield $GF(2^n)$. It is important to point out that subfield inversion can be done extremely fast through table look-up provided n is moderate, say $n \leq 16$ [22]. We extend and optimize the idea in [6] for the case of a standard basis representation as suggested in [18] and we show a major computational advantage for the case that the field polynomial has only coefficients from $GF(2)$.

We want to determine the inverse of $A \in GF((2^n)^m)$, $A \neq 0$, and A is given as in (5). By applying Fermat's Theorem, we can readily obtain that $A^{2^{nm}-1} = AA^{2^{nm}-2} = 1 \bmod P(x)$, from which it follows that

$$A^{-1} = A^{2^{nm}-2}. \tag{6}$$

Equation (6) shows that the inverse of an element $A \in GF((2^n)^m)$ can be computed by raising it to the power of $2^{nm} - 2 = 2 + 2^2 + 2^3 + \cdots + 2^{nm-1}$ using addition chains [7]. However, by noticing that the inversion in the composite field $GF((2^n)^m)$ can be reduced to inversion in the ground field $GF(2^n)$, one obtains a better method to calculate the inverse of an element A.

Theorem 2 *[17] The multiplicative inverse of an element A of the composite field $GF((2^n)^m)$, $A \neq 0$, can be computed by*

$$A^{-1} = (A^r)^{-1} A^{r-1} \bmod P(x),$$

where $A^r \in GF(2^n)$ and $r = (2^{nm} - 1)/(2^n - 1)$.

A central observation is that A^r is an element of the subfield. Computing the inverse through Theorem 2 requires four steps: exponentiation in $GF((2^n)^m)$ (A^{r-1}), multiplication in $GF((2^n)^m)$ with $AA^{r-1} \in GF(2^n)$, inversion in $GF(2^n)$, and multiplication of $(A^r)^{-1}A^{r-1}$. Each of the steps will be analyze below.

4.1 Exponentiation in $GF((2^n)^m)$

The first step in the algorithm above is the computation of A^{r-1} where $A \in GF((2^n)^m)$. Notice that r can be expressed as a sum of powers as follows:

$$r - 1 = \frac{2^{nm} - 1}{2^n - 1} - 1 = 2^n + 2^{2n} + 2^{3n} + \cdots + 2^{(m-1)n}$$

A^{r-1} can now be computed using addition chains. The method requires

$$\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1 \tag{7}$$

general multiplications and at most $m - 1$ exponentiations to the power of 2^n [6], with both types of operations performed in $GF((2^n)^m)$($H_w()$ denotes the Hamming weight of the binary representation of the operand). Notice that the number of multiplications in $GF((2^n)^m)$ as given by (7) determines in essence the overall complexity of the inversion algorithms. Exponentiation is realized

as explained below. Let B and C be elements of $GF((2^n)^m)$. We want to find $C(x) = B(x)^{2^n}$, where $B(x) = \sum_{i=0}^{m-1} b_i x^i$. This is done as follows [12]:

$$C(x) = \sum_{i=0}^{m-1} c_i x^i = \left(\sum_{i=0}^{m-1} b_i x^i \right)^{2^n} = \sum_{i=0}^{m-1} b_i x^{i2^n}, \quad b_i \in GF(2^n). \tag{8}$$

Assuming $2^n > m-1$, there are $m-1$ powers of x which must be reduced modulo the field polynomial $P(x)$, namely the powers x^{i2^n}, $i = 1, 2, \ldots, m-1$. We use the following notation for the representation of these powers in the residue classes modulo $P(x)$:

$$x^{i2^n} = s_{0,i} + s_{1,i}x + \cdots + s_{m-1,i}x^{m-1} \bmod P(x), \quad i = 1, 2, \ldots, m-1.$$

Using the coefficients $s_{j,i}$, the exponentiations in (8) can be expressed in matrix form as

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & s_{0,1} & s_{0,2} & \cdots & s_{0,m-1} \\ 0 & s_{1,1} & s_{1,2} & \cdots & s_{1,m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & s_{m-1,1} & s_{m-1,2} & \cdots & s_{m-1,m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{pmatrix}. \tag{9}$$

A main computational advantage occurs if $P(x)$ is chosen to have only binary coefficients, as suggested in [22, 1]. In this case, all powers $x^a \bmod P(x)$ belong to a subfield whose elements are represented by binary polynomials. In particular, all coefficients $s_{i,j}$ in (9) are binary, i.e., elements from $GF(2)$. Since both n and $P(x)$ are known ahead of time, and thus the entire exponentiation matrix in (9), one exponentiation is reduced to only $(m^2 - 3m + 2)/2$ additions in $GF(2^n)$ on average. Moreover, exponentiations of the form $B(x)^{2ln}$, $l > 1$, which occur in the algorithm, can be computed with one *single* matrix multiplication which is analogous to (9).

4.2 Multiplication in $GF((2^n)^m)$, where the Product is an Element of $GF(2^n)$

The second step performs the operation

$$A^r = A^{r-1}A \bmod P(x),$$

where $A^r \in GF(2^n)$, and the two operands are elements in $GF((2^n)^m)$. We will show that this operation can be considerably less costly than general multiplication in $GF((2^n)^m)$ if $P(x)$ is chosen carefully. We consider the multiplication $H(x) = F(x)G(x) \bmod P(x)$ where $F, G \in GF((2^n)^m)$ and $H \in GF(2^n)$. First, we consider the pure polynomial multiplication of F and G:

$$H'(x) = F(x)G(x) = \left(\sum_{i=0}^{m-1} f_i x^i \right) \left(\sum_{i=0}^{m-1} g_i x^i \right) = \left(\sum_{i=0}^{2m-2} h'_i x^i \right).$$

We know that $H'(x) \equiv H(x) = h_0 \bmod P(x)$, i.e., that all but the zero coefficient of $H'(x)$ vanish after reduction modulo $P(x)$. Hence we only have to compute those coefficients h'_i , $i = 0, 1, \cdots, m-2$, which influence h_0. For instance, for $m = 11$ and $P(x) = x^{11} + x^2 + 1$, it follows that $H(x) = h_0 = h'_0 + h'_{11} + h'_{20}$ which requires only 12 multiplications and 11 additions in $GF(2^n)$ as opposed to 121 multiplications and 100 additions for a general multiplication.

4.3 Inversion in $GF(2^n)$ and Multiplication of an Element from $GF(2^n)$ with an Element from $GF((2^n)^m)$

The third and fourth steps carry small complexities since both involve operations with elements of the subfield. First, we calculate the inverse of A^r with two table look-ups [22] since A^r is an element of the ground field. Finally, we compute $A^{-1} = (A^r)^{-1}A^{r-1}$ by multiplying $(A^r)^{-1}$, which is also an element of $GF(2^n)$, with A^{r-1}, an element of $GF((2^n)^m)$. This last operation requires m multiplications in $GF(2^n)$. Notice that there is no reduction modulo $P(x)$ since all arithmetic is done in $GF(2^n)$.

4.4 Inversion in $GF((2^{16})^{11})$

As a way of summarizing the previous sections, we consider the special case for which $n = 16$ and $m = 11$. In this case, we chose as field polynomial the trinomial $P(x) = x^{11} + x^2 + 1$. Using this polynomial, one finds A^{r-1} with 4 multiplications in $GF((2^{16})^{11})$ and 5 exponentiations to the 2^n, 2^{2n}, and 2^{4n} powers. $A^r = A^{r-1}A$ can be computed using 12 multiplications and 10 additions in $GF(2^{16})$, $(A^r)^{-1}$ requires one inversion in the subfield $GF(2^{16})$, and $(A^r)^{-1}A^{r-1}$ involves 11 subfield multiplications in $GF(2^{16})$. We would like to mention that the extended Euclidean algorithm in [22] allows an inversion in almost 3 multiplication times (assuming an optimized multiplication routine) for the same composite field. Although the algorithm is faster, we believe that our method can be of advantage in situations where *predictable* timings for the inversion are desired.

5 Fast Multiplication in Composite Galois Fields $GF((2^n)^m)$

With respect to complexity, field multiplication is the next costly operation in EC systems. Since the new point multiplication algorithm from Sect. 3 trades field inversions for field multiplications, it is especially attractive to provide efficient multiplication algorithms. In this section we apply the Karatsuba-Ofman Algorithm (KOA) to polynomials over Galois fields $GF(2^n)$ of degree $m-1$ which represent a field element in $GF((2^n)^m)$. First, we consider the general KOA as it is applied to two polynomials $A(x)$ and $B(x)$ with maximum degree $m-1$ over the field $\mathcal{F}$. We derive new formulae for the multiplicative and additive complexity for the cases $m = 2^t l$ and $m = 2^t l - 1$ based on the analysis by [19]. Finally, we

define two new operations, table look-up (TLU) and exponent addition (EXPA), and derive their complexities for the two cases. These two operations are of central importance for an exact complexity analysis of a software implementation of the KOA in composite fields.

5.1 Complexity of the KOA for Polynomials of Degree $2^t l - 1$

In this section we generalize Theorem 1 from [19]. We consider the product of two polynomials $A(x)$ and $B(x)$ with maximum degree of $2^t l - 1$ over a field $\mathcal{F}$. In particular, we want to find $C(x) = A(x)B(x)$ such that $\deg(C(x)) \leq 2m - 2$ with $m = 2^t l$, t an integer. Since the algorithm can only run for t iterations (as many powers of 2 as there are in m), we obtain in the final step polynomials with maximum degree of $l - 1$ which are then multiplied using the school book method.

Theorem 3 *Consider two arbitrary polynomials in one variable of degree less than or equal to $m - 1$ where $m = 2^t l$, with coefficients in a field $\mathcal{F}$ of characteristic 2. Then, by using the Karatsuba-Ofman algorithm the polynomials can be multiplied with:*

$$\#MUL = l^2 \left(\frac{m}{l}\right)^{\log_2 3} = l^{2-\log_2 3} m^{\log_2 3}$$

$$\#ADD = (l-1)^2 \left(\frac{m}{l}\right)^{\log_2 3} + (8l-2)\left(\frac{m}{l}\right)^{\log_2 3} - 8m + 2$$

Notice that for $l = 1$ the expressions in Theorem 3 reduce to those given in Theorem 1 of [19].

5.2 Complexity of the KOA for Polynomials of Degree $2^t l - 2$

In the previous section, we covered the case $m = 2^t l$. As stated earlier we often choose $\gcd(n, m) = 1$. Since it is often desired to have n even there is a need to consider the case for $m = 2^t l - 1$. In particular, we want to find $C(x) = A(x)B(x)$ such that $\deg(C(x)) \leq 2m - 2$ with $m = 2^t l - 1$, t an integer. Then, we can represent $A(x)$ and $B(x)$ by adding an extra term with coefficient $a_m = 0$. For convenience, we will introduce the parameter $r = m + 1$ and express $A(x)$ and $B(x)$ as follows:

$$\begin{aligned} A(x) &= x^{\frac{r}{2}}(0x^{\frac{r}{2}-1} + a_{r-2}x^{\frac{r}{2}-2} + \cdots + a_{\frac{r}{2}}) + (a_{\frac{r}{2}-1}x^{\frac{r}{2}-1} + \cdots + a_0) \\ B(x) &= x^{\frac{r}{2}}(0x^{\frac{r}{2}-1} + b_{r-2}x^{\frac{r}{2}-2} + \cdots + b_{\frac{r}{2}}) + (b_{\frac{r}{2}-1}x^{\frac{r}{2}-1} + \cdots + b_0) \end{aligned} \quad (10)$$

Notice that now the polynomials $A(x)$ and $B(x)$ have an even number of coefficients ($r = m + 1 = 2^t l$), allowing us to split them in half and to apply the general KOA to (10) t times. This reduces this problem to the case for $m = 2^t l$, permitting us to apply the same equations. However, since we have one less coefficient the final multiplicative and additive complexities are reduced. Theorem 4 summarizes the results.

Theorem 4 *Consider two arbitrary polynomials in one variable of degree less than or equal to $m-1$ where $m = 2^t l - 1$, with coefficients in a field $\mathcal{F}$ of characteristic 2. Then, by using the Karatsuba-Ofman algorithm the polynomials can be multiplied with:*

$$\#MUL = l^2 \left(\frac{m+1}{l}\right)^{\log_2 3} - 2l + 1$$

$$\#ADD \leq (l-1)^2 \left(\frac{m+1}{l}\right)^{\log_2 3} + (8l-2)\left(\frac{m+1}{l}\right)^{\log_2 3} - 8(m+1) + 2$$

5.3 Complexity Analysis for Software Implementations

It was shown in [22, 1] that multiplication and inversion in $GF(2^n)$ can be done through table look-up. Since all non-zero elements of $GF(2^n)$ form a cyclic group we can express all elements $a_i \in GF(2^n)$ as a multiple of a primitive element α: $a_i = \alpha^i$. Then, we store all the pairs (a_i, i) in two tables, log and antilog, sorted by the first component (a_i) and second component (i), respectively. Thus, the product of two elements $a_j, a_k \in GF(2^n)$ can be obtained as follows:

$$a_j a_k = \text{antilog}(\log(a_j) + \log(a_k)) \pmod{2^n - 1} \tag{11}$$

Notice that (11) implies that two elements of the ground field $GF(2^n)$ can be multiplied using three table look-up operations and one addition modulo the order of the multiplicative group (exponent addition). It is important to point out that depending on the hardware platform (e.g., microprocessor, RISC, etc.) the relative speed for the two types of operations can differ dramatically. For instance, in our implementation where $n = 16$ it was found that access to the large look-up tables took 6 clock cycles on a DEC Alpha workstation, whereas element addition and exponent addition took about 2 clock cycles on average. Thus, in order to obtain valid performance predictions one needs exact counts of the number of operations. Based on these two new operations table look-up (TLU) and exponent addition (EXPA) we have derived new formulae and re-written Theorems 3 and 4 as follows. For more details on the derivation of the theorems and corollaries, refer to [4].

Corollary 1 *Consider two arbitrary polynomials in one variable of degree less than or equal to $m-1$ where $m = 2^t l$, with coefficients in a field $\mathcal{F}$ of characteristic 2. Then, by using the Karatsuba-Ofman algorithm the polynomials can be multiplied with:*

$$\#ADD = (l-1)^2 \left(\frac{m}{l}\right)^{\log_2 3} + (8l-2)\left(\frac{m}{l}\right)^{\log_2 3} - 8m + 2$$

$$\#TLU = l(l+2)\left(\frac{m}{l}\right)^{\log_2 3}$$

$$\#EXPA = l^2 \left(\frac{m}{l}\right)^{\log_2 3}$$

Corollary 2 *Consider two arbitrary polynomials in one variable of degree less than or equal to $m-1$ where $m = 2^t l - 1$, with coefficients in a field $\mathcal{F}$ of characteristic 2. Then, by using the Karatsuba-Ofman algorithm the polynomials can be multiplied with:*

$$\#ADD \leq (l-1)^2 \left(\frac{m+1}{l}\right)^{\log_2 3} + (8l-2)\left(\frac{m+1}{l}\right)^{\log_2 3} - 8(m+1) + 2$$

$$\#TLU \leq \left(\frac{m+1}{l}\right)^{\log_2 3} l(l+2) - 2l - 1$$

$$\#EXPA = l^2 \left(\frac{m+1}{l}\right)^{\log_2 3} - 2l + 1$$

5.4 Multiplication in $GF((2^{16})^{11})$

We summarize this section by considering the complexity of a multiplication in $GF((2^n)^m)$ for $n = 16$ and $m = 11$. We can apply Corollary 2 and let $m = 11$, $l = 3$, and $t = 2$. From there we obtain that one needs 140 additions and 76 multiplications in $GF(2^{16})$ or equivalently, at most 140 additions, 124 table look-ups, and 76 exponent additions. On the other hand, when using the school book method for multiplication one would require 121 multiplications and 100 coefficient additions or, in terms of table look-ups, 100 coefficient additions, $121+22 = 143$ table look-ups, and 121 exponent additions. If one compares both complexities and ignores exponent and coefficient additions, one can readily see that the theoretical improvement in the timing for the multiplication operation when using the Karatsuba-Ofman algorithm would be about 12.5 percent.

6 Implementation and Timings

This section describes the application of the various algorithms to an actual EC system over the field $GF(2^{16})^{11}) \cong GF(2^{176})$. A DEC Alpha 3000, a 175 MHz RISC architecture, was used to perform all measurements. Table 2 shows the timings for several arithmetic operations in $GF((2^{16})^{11})$. It was found that by applying the KOA to field multiplication one obtains a 10% improvement over the school book method. Using the timings in Table 2, we computed estimates for the timings for repeated point doublings using individual doublings and direct doublings for the computation of $8P$ and $16P$. We compared these predictions with the actual timings as shown in Table 3. Interestingly, the actual timings are considerably better than expected. We attribute this observation to the reduced overhead in the software implementation (e.g., fewer function calls and variable initializations). Table 4 compares timings for point multiplication for different parameters k in the k-ary method and the improved k-ary method including the formulae for direct point doubling. It was found that the optimum value for the window size in the k-ary method was $k = 4$. We achieved a speed-up of almost 19% using the new formulae. Table 5 presents the timings for several algorithms

Table 2. Timings for various field operations in $GF((2^{16})^{11})$.

Type of Operation	Average Timing (μsec)
176 bit addition	1.19
176 bit squaring	4.23
176 x 176 bit multiplication	38.56
176 bit inverse	158.73

Table 3. Timing comparison: Individual doublings vs. direct computation of several doublings in $GF((2^{16})^{11})$.

Calculation	Method	Predicted Timing	Measured Timing	% Improvement	
				Predicted	Measured
$8P$	Direct Doublings	748.41 μsec	904.812 μsec	0.30	12.5
	Individual Doublings	750.78 μsec	1.035 msec		
$16P$	Direct Doublings	978.62 μsec	1.141 msec	2.24	17.85
	Individual Doublings	1.001 msec	1.389 msec		

used to compute nP. Notice that the last entry of Table 5 corresponds to the nP calculation where the point P is known ahead of time, thus it is possible to pre-compute a table of multiples of P. This implies that the formulae derived in Sect. 3 were not used in this algorithm.

Acknowledgements

We would like to thank Dan Beauregard for providing the EC implementation.

7 Appendix - Formulae for $8P$ and $16P$

In this section we present the improved formulae to find $8P = 2^3P = (x_3, y_3)$ and $16P = 2^4P = (x_4, y_4)$ where $P = (x, y)$ is an element of prime order belonging to the cyclic subgroup corresponding to the largest prime factor in the order of E.

1. Formulae for $8P = 2^3P = (x_3, y_3)$

$$x_3 = \frac{\omega^2 + \omega\rho}{\rho^2} + a$$

$$y_3 = \frac{(v^2)^2 + \omega\rho x_3}{\rho^2} + x_3$$

Table 4. Comparison of average time required to perform the nP calculation using the regular k-ary method and the improved k-ary with four direct doublings

Method	Window Size k	Average Timing (in msec)
k-ary	3	87
	4	84
	5	88
k-ary with formulae	4	68

Table 5. Timings for elliptic curve operations

Operations	Method/Type of Operation	Average Timing
Multiply new elliptic curve point ($n \rightarrow 176bits$)	Double and Add Algorithm	95 msec
	k-ary method ($k = 4$)	84 msec
	k-ary method with formulae ($k = 4$)	68 msec
Multiply known elliptic curve point ($n \rightarrow 176bits$)	Brickell's Algorithm ($base = 2^4 = 16$)	20 msec

2. Formulae for $16P = 2^4P = (x_4, y_4)$

$$x_4 = \frac{\theta^2 + \theta\mu\rho^2}{(\mu\rho^2)^2} + a$$

$$y_4 = \frac{(\mu^2)^2 + (\theta\mu\rho^2)x_4}{(\mu\rho^2)^2} + x_4$$

where $\gamma = x^2$,$\eta = \gamma + y$, $\delta = \eta^2 + \eta x + a\gamma$, $\xi = \eta x + \gamma$, $\zeta = \delta(\delta + \xi) + \gamma^2\gamma$, $\tau = \delta\gamma$, $\upsilon = \zeta^2 + \tau\zeta + \tau^2 a$, $\rho = \upsilon\tau^2$, $\omega = \upsilon(\upsilon + \zeta\tau) + (\tau\delta^2)^2 + \rho$, $\mu = \omega^2 + \omega\rho + a\rho^2$, and $\theta = \mu^2 + \mu(\omega\rho) + \mu\rho^2 + (\upsilon^2\rho)^2$.

References

1. D. Beauregard. Efficient algorithms for implementing elliptic curve public-key schemes. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, MA, May 1996.
2. H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, Berlin, 1993.
3. R.J. Fateman. Polynomial multiplication, powers and asymptotic analysis: Some comments. *SIAM J. Comput.*, 7(3):196–21, September 1974.
4. J. Guajardo. Efficient algorithms for elliptic curve cryptosystems. Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, MA, May 1997.

5. G. Harper, A. Menezes, and S. Vanstone. Public-key cryptosystems with very small key lengths. In *Advances in Cryptology — EUROCRYPT '92*, pages 163–173, May 1992.
6. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.
7. D.E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.
8. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
9. N. Koblitz. Constructing elliptic curve cryptosystems in characteristic 2. In *Advances in Cryptology — CRYPTO '90*, pages 156–167. Springer-Verlag, Berlin, 1991.
10. C. K. Koc. Analysis of sliding window techniques for exponentiation. *Computers and Mathematics with Applications*, 30(10):17–24, November 1995.
11. J. Koeller, A. Menezes, M. Qu, and S. Vanstone. Elliptic Curve Systems. Draft 8, IEEE P1363 Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography, May 1996. working document.
12. R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications.* Addison-Wesley, Reading, Massachusetts, 1983.
13. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, Florida, 1997.
14. A. J. Menezes, S. A. Vanstone, and R. J. Zuccherato. Counting points on elliptic curves over F_{2^m}. *Mathematics of Computation*, 60(201):407–420, January 1993.
15. A.J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.
16. V. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO '85*, pages 417–426. Springer-Verlag, Berlin, 1986.
17. C. Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields.* PhD thesis, (Engl. transl.), Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1994.
18. C. Paar. Some remarks on efficient inversion in finite fields. In *1995 IEEE International Symposium on Information Theory*, page 58, Whistler, B.C. Canada, September 17–22 1995.
19. C. Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–861, July 1996.
20. C. Paar and P. Soria-Rodriguez. Fast arithmetic architectures for public-key algorithms over galois fields $GF((2^n)^m)$. In *Advances in Cryptology — EUROCRYPT '97*, pages 363–378, 1997.
21. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. *Advances in Cryptology, Crypto 95*, pages 43–56, 1995.
22. E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In *Asiacrypt '96.* Springer Lecture Notes in Computer Science, 1996.