Elliptic Curve Cryptography on a Palm OS Device

André Weimerskirch[1], Christof Paar[2], and Sheueling Chang Shantz[3]

[1] CS Department, Worcester Polytechnic Institute, USA
weika@wpi.edu
[2] ECE and CS Department, Worcester Polytechnic Institute, USA
christof@ece.wpi.edu
[3] Networking and Security Research, Sun Microsystems Laboratories, USA
Sheueling.Chang@eng.sun.com

Abstract. The market for Personal Digital Assistants (PDA) is growing rapidly and PDAs are becoming increasingly interesting for commercial transactions. One requirement for further growing of eCommerce with mobile devices is the provision of security. We implemented elliptic curves over binary fields on a Palm OS device. We chose the NIST recommended random and Koblitz curves over $GF(2^{163})$ that are providing a sufficient level of security for most commercial applications. Using Koblitz curves a typical security protocol like Diffie-Hellman key exchange or ECDSA signature verification requires less than 2.4 seconds, while ECDSA signature generation can be done in less than 0.9 seconds. This should be tolerated by most users.

Keywords: Elliptic Curves, Koblitz Curves, Binary Fields, Palm OS

1 Introduction

The market for Personal Digital Assistants (PDA) is growing rapidly and PDAs are becoming increasingly interesting for commercial transactions. For eCommerce, provision of security is a must. Since elliptic curve cryptosystems are a promising match for embedded systems because of their short operand lengths and efficient arithmetic, we implemented elliptic curves over binary fields on a Palm OS device to investigate if today's PDAs are sufficient for secure transactions. The most popular PDAs use the Palm Operating System (Palm OS). We used a Handspring Visor model with 2 MB of memory. This device has a Motorola Dragonball CPU that provides eight data registers and seven address registers, all of them 32-bit in size [15]. The processor offers 16-bit and 32-bit operations and runs at 16 MHz. To the author's knowledge there were only two elliptic curve implementations on a Palm Pilot reported yet. In [2] PGP was ported to wireless devices and [3] analyzes electronic commerce applications on a Palm Pilot. However, the first implementation was not optimized for the Palm Pilot while the second one uses a commercial library. Both papers also point out that the popular RSA system is very slow on a Palm Pilot. For most electronic commerce applications the security provided by elliptic curves over $GF(2^{163})$ should be sufficient as long as there are no substantial improvements in solving the elliptic curve discrete logarithm problem (DLP). This bit length is often considered to be security-equivalent to RSA with 1024-bit key length [1]. We chose the NIST recommended random and Koblitz curves over $GF(2^{163})$ and selected binary curves since the integer multiplication unit of the Dragonball processor is very slow. Koblitz curves allow shorter run times while they provide nearly the same level of security according to current knowledge about attacks. Our implementation is mostly based on the algorithms used in a comprehensive software implementation for a PC [4] and Solinas' work about Koblitz curves [19].

2 Arithmetic in $GF(2^m)$

2.1 Field Representation

For our implementation we used a polynomial basis representation. Let $f(x) = x^m + r(x)$ be an irreducible binary polynomial of degree m with small weight, that is a trinomial or pentanomial. The elements of $GF(2^m)$ are represented by the binary polynomials of degree at most $m-1$. Addition and multiplication in $GF(2^m)$ are performed as polynomial operations modulo $f(x)$. An element $a \in GF(2^m)$ is written as the polynomial $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and is stored as binary vector $a = (a_{m-1}, \ldots, a_0)$. We store a in an array A of 16-bit words of size $s = \lceil m/16 \rceil$ and write $A = (A[s-1], \ldots, A[0])$. The rightmost bit of $A[0]$ is a_0 and a_{m-1} is part of $A[s-1]$. The bits left of a_{m-1} are set to zero. For our implementation we used an array of twelve 16-bit words to store an element of $GF(2^{163})$ such that we can also consider A to be an array of $s' = 6$ 32-bit words.

2.2 Addition

Addition over binary fields is performed by a bitwise XOR. Since the Motorola Dragonball CPU performs one 32-bit XOR faster than two 16-bit XORs [15] we add two binary vectors A and B by performing five 32-bit XORs and one 16-bit XOR. We denote this operation by $\oplus$.

2.3 Multiplication

To compute $c = a \cdot b$ we first compute the polynomial $c'(x) = a(x) \cdot b(x)$ and then reduce it to $c(x) \equiv c'(x) \bmod f(x)$.

Polynomial Multiplication Algorithm 1 computes $c' = a \cdot b$ by using a window method [12]. First polynomials $B_u = u \cdot b(x)$ are precomputed for $0 \leq u < 2^w$ where w is the window size. In each step of the loop w bits of a are considered. We unrolled the two nested FOR loops completely which resulted in a slight performance gain. By $C\{j\}$ we denote the bit vector $(C[s-1], \ldots, C[j])$. In step

Algorithm 1 Comb method with window size $w = 4$

```
INPUT: Binary polynomials $a(x)$ and $b(x)$ of degree at most $m-1$.
OUTPUT: The binary polynomial $c'(x) = a(x) \cdot b(x)$.
1: Compute $B_u(x) = u(x) \cdot b(x)$ for all polynomials $u(x)$ of degree at most 3.
2: $C' \leftarrow 0$
3: for $i = 3$ downto 0 do
4:   for $j = 0$ to $s-1$ do
5:     Let $u = (u_3, u_2, u_1, u_0)$, where $u_k$ is bit $(4i+k)$ of $A[j]$.
6:     $C'\{j\} = C'\{j\} \oplus B_u$
7:   end for
8:   if $i \neq 0$ then
9:     $C' \leftarrow C'x^4$
10:  end if
11: end for
12: Return $c'(x)$
```

6 the m-bit vector B_u is added to C' where the rightmost bit of B_u is added to the rightmost bit of $C'\{j\}$.

We also experimented with the Karatsuba Algorithm [7] as described in Algorithm 2. However, our results were always slower than the above described comb method. We implemented the Karatsuba Algorithm three times recursively and applied the comb method with windows size $w = 3$ to the resulting degree-20 polynomials.

Algorithm 2 Karatsuba Algorithm

```
INPUT: Binary polynomials $a(x)$ and $b(x)$ of degree at most $m-1$.
OUTPUT: The binary polynomial $c'(x) = a(x) \cdot b(x)$.
1: Write $a(x) = a_1(x)x^{m/2} + a_0(x)$ and $b(x) = b_1(x)x^{m/2} + b_0(x)$
2: $D_0(x) \leftarrow a_0(x)b_0(x)$
3: $D_1(x) \leftarrow a_1(x)b_1(x)$
4: $D_2(x) \leftarrow (a_0(x) \oplus a_1(x))(b_0(x) \oplus b_1(x))$
5: $c'(x) \leftarrow D_1(x)x^m \oplus (D_2(x) \oplus D_0(x) \oplus D_1(x))x^{m/2} \oplus D_0(x)$
6: Return $c'(x)$
```

Polynomial Reduction If $f(x)$ is a trinomial or a pentanomial with middle terms close to each other, reduction of $c'(x)$ modulo $f(x)$ can be efficiently performed one word at a time. Algorithm 3 performs the modulo reduction by $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. It is based on the fact that

$$\begin{aligned} x^{163} &\equiv x^7 + x^6 + x^3 + 1 \bmod f(x) \\ &\vdots \\ x^{324} &\equiv x^{168} + x^{167} + x^{164} + x^{161} \bmod f(x) \end{aligned}$$

A word $C[i]$ is now reduced by adding $C[i]$ four times to C, with the rightmost bit of $C[i]$ added to the bits as desribed on the right side of the above congruences. For example, reduction of $C[9]$ is performed by adding $C[9]$ four times to C, with the rightmost bit of $C[9]$ added to the bits 132, 131, 128 and 125 of C. Note that we used 32-bit arithmetic and 32-bit words since XOR and shift operations for 32-bit words have lower runtime than for two 16-bit words. Therefore the array value $C[0]$ describes the bits 0 to 31 of the value c.

Algorithm 3 Modular reduction by $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$

INPUT: A binary polynomial $c'(x)$ of degree at most 324.
OUTPUT: $c'(x) \bmod f(x)$.

```
1: for $i = 10$ downto 6 do
2:     $T \leftarrow C'[i]$
3:     $C[i-6] \leftarrow C[i-6] \oplus (T << 29)$
4:     $C[i-5] \leftarrow C[i-5] \oplus (T << 4) \oplus (T << 3) \oplus T \oplus (T >> 3)$
5:     $C[i-4] \leftarrow C[i-4] \oplus (T >> 28) \oplus (T >> 29)$
6: end for
7: $T \leftarrow C[5]$ AND 0xFFFFFFF8
8: $C[0] \leftarrow C[0] \oplus (T << 4) \oplus (T << 3) \oplus T \oplus (T >> 3)$
9: $C[1] \leftarrow C[1] \oplus (T >> 28) \oplus (T >> 29)$
10: $C[5] \leftarrow C[5]$ AND 0x00000007
11: Return $(C[5], \ldots, C[0])$
```

2.4 Squaring

Squaring in $GF(2^m)$ is a linear operation and much faster than multiplying two arbitrary elements [17]. To square $a(x) = \sum_{i=0}^{m-1} a_i x^i$ we compute $a(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i}$ which is obtained by inserting a 0-bit between consecutive bits of the binary representation of a. The result is reduced modulo $f(x)$. Algorithm 4 describes how this can be done using a precomputed table. As before we used 32-bit words and 32-bit operations.

2.5 Inversion

Instead of using the Extended Euclidean Algorithm to compute an inversion, we used Algorithm 5 to compute a division $\frac{a}{b}$ directly [18]. It has roughly the same running time as the Extended Euclidian Algorithm and therefore saves one multiplication to compute a field division. Note that division by x is accomplished by a right-shift operation. The comparison between two elements $a(x)$ and $b(x)$ is done by considering the bit vectors a and b as integers.

Algorithm 4 Squaring in $GF(2^m)$

```
INPUT: $a \in GF(2^m)$
OUTPUT: $c = a^2 \in GF(2^m)$.
1: Precompute for each byte $b = (b_7, \ldots, b_0)$ the 16-bit vector $T(b) = (0, b_7, \ldots, 0, b_0)$.
2: for $i = 0$ to $s' - 1$ do
3:   Let $A[i] = (A_3[i], A_2[i], A_1[i], A_0[i])$ where $A_j[i]$ are bytes.
4:   $C'[2i] \leftarrow (T(A_1[i]), T(A_0[i]))$
5:   $C'[2i+1] \leftarrow (T(A_3[i]), T(A_2[i]))$
6: end for
7: $c(x) = c'(x) \bmod f(x)$
8: Return $c(x)$
```

Algorithm 5 Modular Division in $GF(2^m)$

```
INPUT: $a, b \neq 0 \in GF(2^m)$
OUTPUT: $c = \frac{a}{b} \bmod f(x) \in GF(2^m)$.
1: $u \leftarrow b, v \leftarrow f(x), c \leftarrow a, d \leftarrow 0$.
2: while $u \neq d$ do
3:   if $u \bmod 2 = 0$ then
4:     $u \leftarrow \frac{u}{x}$
5:     if $c \bmod 2 = 0$ then
6:       $c \leftarrow \frac{c}{x}$
7:     else
8:       $c \leftarrow \frac{c \oplus f(x)}{x}$
9:     end if
10:  else if $v \bmod 2 = 0$ then
11:    $v \leftarrow \frac{v}{x}$
12:    if $d \bmod 2 = 0$ then
13:      $d \leftarrow \frac{d}{x}$
14:    else
15:      $d \leftarrow \frac{d \oplus f(x)}{x}$
16:    end if
17:  else if $u > v$ then
18:    $u \leftarrow \frac{u \oplus v}{x}$, $c \leftarrow c \oplus d$
19:    if $c \bmod 2 = 0$ then
20:      $c \leftarrow \frac{c}{x}$
21:    else
22:      $c \leftarrow \frac{c \oplus f(x)}{x}$
23:    end if
24:  else
25:    $v \leftarrow \frac{u \oplus v}{x}$, $d \leftarrow c \oplus d$
26:    if $d \bmod 2 = 0$ then
27:      $d \leftarrow \frac{d}{x}$
28:    else
29:      $d \leftarrow \frac{d \oplus f(x)}{x}$
30:    end if
31:  end if
32: end while
33: Return $c(x)$
```

2.6 Timings

Table 1 displays the timings for one field operation. We spent most time implementing the comb method since the field multiplication is the crucial operation. Reduction and squaring can be implemented efficiently. Division is very expensive and will be avoided where possible.

Table 1. Timings in ms. for one field operation

		time
Multiplication	Comb Method	2.35
	Karatsuba	4.41
Reduction		0.24
Squaring		0.49
Division		38.01

3 Elliptic Curve Basics

3.1 Arithmetic

An elliptic curve over $GF(2^m)$ is defined by the (affine) curve equation

$$E : Y^2 + XY = X^3 + aX + b \tag{1}$$

where $a, b \in GF(2^m)$ and $b \neq 0$. If $a, b \in GF(2)$, i.e., $b = 1$ and $a = 0$ or 1 the curve has special properties that can be used for efficient arithmetic and it is called Koblitz curve. All points $P = (x, y)$ that satisfy (1) and an additional point at infinity $\mathcal{O}$ form a group $E(GF(2^{163}))$. Assume $P_1 = (x_1, y_1) \neq \mathcal{O}, P_2 = (x_2, y_2) \neq \mathcal{O}$ and $P_1 \neq -P_2$. Then $P_3 = (x_3, y_3) = P_1 + P_2$ is computed as follows.

If $P_1 \neq P_2$

$$\begin{aligned} \lambda &= \frac{y_1 + y_2}{x_1 + x_2} \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1 \end{aligned}$$

If $P_1 = P_2$

$$\begin{aligned} \lambda &= \frac{y_1}{x_1} + x_1 \\ x_3 &= \lambda^2 + \lambda + a \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1 \end{aligned}$$

The group operation requires one division and one multplication in either case. By using projective coordinates as described later the division can be avoided. A scalar or point multiplication is defined as repeated addition via

$$k \cdot P = \underbrace{P + \ldots + P}_{k \text{ times}}$$

There are no efficient attacks known on elliptic curves. The DLP for random curves $E(GF(2^m))$ can be solved on average in $2^{m/2}$ steps, e.g., by using Pollard's Rho method [13]. Therefore a curve over $GF(2^{163})$ is considered appropriate to obtain a secret key for a symmetric cipher with a key length of around 80 bits. An attack on Koblitz curves using the special structure shortens the running time by a factor of $\sqrt{m}$ [20].

3.2 Point Representation

If inversion in $GF(2^m)$ is expensive relative to multiplications it may be more efficient to represent points in projective coordinates. Since a field division is more expensive than 10 multiplications we use projective coordinates as proposed in [10] where the projective point (X, Y, Z) corresponds to the affine point $(X/Z, Y/Z^2)$. The doubling formula $(X_2, Y_2, Z_2) = 2(X_1, Y_1, Z_1)$ for projective coordinates is given by

$$\begin{aligned} Z_2 &= Z_1^2 X_1^2 \\ X_2 &= X_1^4 + bX_1^4 \\ Y_2 &= bZ_1^4 Z_2 + X_2(aZ_2 + Y_1^2 + bZ_1^4) \end{aligned}$$

The projective form of the addition formula is

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

For the special case $Z_1 = 1$, i.e., (X_1, Y_1) are affine coordinates this can be computed as follows:

$$\begin{aligned} &A = Y_1 Z_0^2 + Y_0,\ B = X_1 Z_0 + X_0,\ C = Z_0 B \\ &D = B^2(C + aZ_0^2),\ Z_2 = C^2,\ E = AC,\ X_2 = A^2 + D + E \\ &F = X_2 + X_1 Z_2,\ G = X_2 + Y_1 Z_2,\ Y_2 = EF + Z_2 G \end{aligned}$$

In case that $a = 0$ or 1 point doubling requires 4 field multiplications. Mixed Point Addition requires 9 multiplication. We based all point multiplication methods on these projective point doubling and mixed point addition. We only used affine point operations for point precomputations.

4 Random Curves

4.1 Curve Parameters

For our implementation we used a NIST recommended random curve [16] with the parameters

$$a = \mathtt{1}$$

$$b = \texttt{0x 2 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD}$$

and group order

$$\#E(GF(2^{163})) = 2 \cdot 5846006549323611672814742442876390689256843201587$$

NIST also recommends the randomly chosen base point $G = (G_x, G_y)$ where

$$G_x = \texttt{0x 3 F0EBA162 86A2D57E A0991168 D4994637 E8343E36}$$
$$G_y = \texttt{0x 0 D51FBC6C 71A0094F A2CDD545 B11C5C0C 797324F1}$$

4.2 Point Multiplication

There are several methods known to compute $kP \in E(GF(2^m))$ where $k \approx 2^m$. The binary double-and-add method [13] requires m doublings and $m/2$ additions on average. The addition-subtraction method requires only $m/3$ additions on average [14]. It is based on the nonadjacent form (NAF) of the coefficient k. NAF(k) is a unique signed binary expansion with the property that no two consecutive coefficients are nonzero. It has the fewest nonzero coefficients of any signed binary expansion of k, on average $m/3$. Window methods precompute some values and operate on more than one bit of the coefficient k at the same time. Window methods also reduce the number of additions. The sliding window method uses a variable window size. It has an effect equivalent to using fixed windows one bit larger [1]. On average this method requires $m+1$ doublings and $2^{w-1} - 1 + \frac{m}{w+1}$ additions where w is the largest window size. A slight improvement can be gained by using a windowed addition-subtraction method [8]. This is accomplished by using a windowed NAF. A width-w NAF of k is a unique expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where each nonzero k_j is odd and less than 2^{w-1} in absolute value, and among any w consecutive coefficients at most one is nonzero. This method requires $m+1$ doublings and $2^{w-2} - 1\frac{m}{w+1}$ additions on average. A different approach for point multiplication based on Montgomery's idea was proposed in [11]. It requires $6m$ field multiplications and squarings and does not need any extra memory storage.

If a fixed base point is used we can use precomputed points as done by the fixed base comb method [9]. Using 2^w precomputed points this method requires $d-1$ doublings and $(d-1)(2^w-1)/2^w$ additions where $d = \lceil m/w \rceil$. We implemented this method with a window size of $w = 4$ and $w = 8$. This requires $2^4 = 16$ precomputed points and $16 \cdot 2 \cdot 22 = 704$ bytes, and $2^8 = 256$ precomputed points and $256 \cdot 2 \cdot 22 = 11264$ bytes, respectively. The precomputed points can easily be stored on the Palm device.

4.3 Timings

Table 2 displays the timings for one point multiplication on a Handspring Visor with 2 MB of memory. The implementation was done in C using the Code Warrior IDE. One can see that the differences are relatively small. While the

Montgomery method has always the same running time the other methods depend on the coefficient k. The timings were obtained by taking the average time of multiply test runs with random coefficients. When precomputed points can be used the running time is small. A typical key-exchange protocol like Diffie-Hellman or the ECDSA signature verification require one point multiplication by a random point and one point multiplication by a fixed point. This can be done in 3.5 seconds using the Montgomery method and fixed base comb method with $w = 8$. ECDSA signature generation requires a point multiplication by a fixed base point which can be done in 0.8 seconds.

Table 2. Timings in sec. for one point multiplication on random curves

	time
Addition-subtraction	3.31
Sliding windows ($w = 4$)	3.07
Width-w addition-subtraction ($w = 4$)	2.96
Montgomery	2.73
Precomputation (Fixed base comb, $w = 4$)	1.43
Precomputation (Fixed base comb, $w = 8$)	0.79

5 Koblitz Curves

Koblitz curves were first introduced in [6]. All described facts and methods are due to Solinas [19]. The advantage of Koblitz curves is that point multiplication methods can be changed in such a way that point doublings is replaced by the Frobenius map. There are two Koblitz curves that use $a = 0$ or $a = 1$. Let $\mu = (-1)^{1-a}$ and τ be the Frobenius map. The Frobenius $\tau : E(GF(p^m)) \rightarrow E(GF(p^m))$ is defined as $\tau(x, y) = (x^p, y^p)$. Since $p = 2$ this can be done efficiently. It is known that $(\tau^2+2)P = \mu\tau P$ for all $P \in E(GF(2^m))$. Therefore τ can be expressed as the complex number $\tau = (\mu + \sqrt{-7})/2$. Since $\tau^2 + 2 = \mu\tau$ every integer k can be expressed as $r_1\tau + r_0$ where $r_0, r_1 \in \mathbb{Z}$. The main idea is to replace a coefficient k by a τ-adic number k' with $k = k'$ and to compute $k'P$. Since $k'P$ can be written as $k'_{l-1}\tau^{l-1}(P) + \ldots + k'_0 P$ point multplication does not require any point doublings. The τ-adic representation of k has to be computed in such a way that it has short bit length. This is done using modulo reduction in $\mathbb{Z}[\tau]$. Note that this reduction requires multi-precision integer arithmetic.

5.1 Curve Parameters

Again we used a NIST recommended curve [16] with the parameters

$$a = 1, b = 1$$

and group order

$$\#E(GF(2^{163})) = 2 \cdot 5846006549323611672814741753598448348329118574063$$

NIST also provides the base point $G = (G_x, G_y)$ where

$$G_x = \texttt{0x2 FE13C053 7BBC11AC AA07D793 DE4E6D5E 5C94EEE8}$$
$$G_y = \texttt{0x2 89070FB0 5D38FF58 321F2E80 0536D538 CCDAA3D9}$$

5.2 Point Multiplication

Similar to the addition-subtraction method for random curves we implemented a signed binary τ-adic method that uses a reduced τ-adic NAF of k. The τ-adic NAF of k is the unique expression $k = \sum_{i=0}^{l-1} k_i \tau^i$ where $k_i \in \{-1, 0, 1\}$ and no two consecutive coefficients k_i are nonzero. Since the Frobenius map can be computed very efficiently the expected running time is $m/3$ point additions. The cost to compute the NAF of k is much more expensive than for random curves though. The method can be improved by using a window technique. This is called the τ-adic width-w window method. It is based on the τ-adic width-w NAF that is defined very similar as the binary width-w NAF for random curves. This method requires $2^{w-2} - 1 + \frac{m}{w+1}$ additions on average. As before we used mixed projective point addition.

If a fixed base point is used precomputation reduces the time for one point multplication. This is easily achieved by applying the τ-adic width-w window method. Instead of precomputing points for each multiplication the points are only precomputed once such that the window size can be choosen larger.

5.3 Timings

Table 3 displays the values for one point multiplication. Note that the windows τ-adic version with $w = 4$ is faster than with $w = 5$ although the later one has a slightly lower complexity. Also, precomputation using more points is not as efficient as for random curves since the computational overhead to compute the τ-adic representation of the scalar k' increases. The usual time for a key exchange or signature verification is around 2.4 seconds while a signature generation can be done in 0.9 seconds. This is significiantly faster than for random curves.

6 Conclusion

We implemented a NIST recommended random and Koblitz curve over $GF(2^{163})$ on a Palm OS device. A normal transaction such as a key exchange or signature verification can be done in less than 2.4 seconds while signature generation can be done in less than 0.9 seconds. Koblitz curves are particular suitable for these devices since they allow running times that will probably be tolerated by most users.

Table 3. Timings in sec. for one point multiplication on Koblitz curves

	time
τ-adic	1.67
τ-adic width-w ($w = 4$)	1.51
τ-adic width-w ($w = 5$)	1.68
Precomputation ($w = 6$)	1.08
Precomputation ($w = 10$)	0.87

References

1. I. Blake, G. Seroussi and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, Cambridge, 1999.
2. M. Brown, D. Cheung, D. Hankerson, J. L. Hernandez, M. Kirkup, and A. Menezes. PGP in Constrained Wireless Devices. *Proceedings of the 9th USENIX Security Symposium*, 2000.
3. N. Daswani and D. Boneh. Experimenting with Electronic Commerce on the Palm Pilot. *Financial Cryptography '99*, LNCS 1648, Springer-Verlag, 1-16, 1999.
4. D. Hankerson, J. L. Hernandez and A. Menezes. Software Implementation of Elliptic Curve Cryptography Over Binary Fields. *Cryptographic Hardware and Embedded Systems, CHES 2000*, LNCS 1965, Springer-Verlag, 1-24, 2000.
5. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, 2000.
6. N. Koblitz. CM-curves with good cryptographic properties. *Advances in Cryptology – CRYPTO '91*, LNCS 576, Springer-Verlag, 279-287, 1992.
7. D.E. Knuth. *Seminumerical Algorithms.* Addision-Wesley, 1981.
8. K. Koyama and Y. Tsuruoka. Speeding up elliptic curve cryptosystems by using a signed binary window method. *Advances in Cryptology – Crypto '92*, LNCS 740, Springer-Verlag, 345-357, 1993.
9. C. Lim and P. Lee. More flexible exponentiation with precomputation. *Advances in Cryptology - Crypto '94*, LNCS 839, Springer-Verlag, 95-107, 1994.
10. J. López and R. Dahab. Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. *Selected Areas in Crytography - SAC '98*, LNCS 1556, Springer-Verlag, 201-212, 1999.
11. J. López and R. Dahab. Fast multiplication on Elliptic Curves over $GF(2^n)$ without Precomputation, *Cryptographic Hardware and Embedded Systems-CHES '99*, LNCS 1717, Springer-Verlag, 316-327, 1999.
12. J. López and R. Dahab. High-Speed Software Multiplication in $\mathbb{F}_{2^m}$, IC Technical Reports, IC-00-09, Institute of Computing, University of Campinas, May 2000, Available from http://www.dcc.unicamp.br/ic-main/publications-e.html.
13. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
14. F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique théorique et Applications*, 24, 531-544, 1990.
15. Motorola. M68000 8-/16-/32-Bit Microprocessors User's Manual, Ninth Edition.
16. National Institute of Standards and Technolgy. *Recommended Elliptic Curves for Federal Government Use*, May 1999, available from http://csrc.nist.gov/encryption.

17. R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck. Fast Key Exchange with Elliptic Curve Systems. *Advances in Cryptology - Crypto '95*, LNCS 963, Springer-Verlag, 43-56, 1995.
18. S. Shantz. From Euclid's GCD to Montgomery Multiplication to the Great Divide, preprint, 2000.
19. J. A. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19(2/3), 195-249, 2000.
20. M. Wiener and R. Zuccherato. Faster attacks on elliptic curve cryptosystems. *Selected Areas in Cryptography*, LNCS 1556, Springer-Verlag, 190-200, 1999.