An Improved Algorithm for Arithmetic on a Family of Elliptic Curves*

Jerome A. Solinas

National Security Agency, Ft. Meade, MD 20755, USA

Abstract. It has become increasingly common to implement discrete-logarithm based public-key protocols on elliptic curves over finite fields. The basic operation is *scalar multiplication:* taking a given integer multiple of a given point on the curve. The cost of the protocols depends on that of the elliptic scalar multiplication operation.

Koblitz introduced a family of curves which admit especially fast elliptic scalar multiplication. His algorithm was later modified by Meier and Staffelbach. We give an improved version of the algorithm which runs 50% faster than any previous version. It is based on a new kind of representation of an integer, analogous to certain kinds of binary expansions. We also outline further speedups using precomputation and storage.

Keywords: elliptic curves, exponentiation, public-key cryptography.

1 Introduction

It has become increasingly common to implement discrete-logarithm based public-key protocols on elliptic curves over finite fields. More precisely, one works with the points on the curve, which can be added and subtracted. If we add the point P to itself n times, we denote the result by nP. The operation of computing nP from P is called *scalar multiplication* by n. Elliptic public-key protocols are based on scalar multiplication, and the cost of executing such protocols depends mostly on the complexity of the scalar multiplication operation.

Scalar multiplication on an elliptic curve is analogous to exponentiation in the multiplicative group of integers modulo a fixed integer m. Various techniques have been developed [1] to speed modular exponentiation using memory and precomputations. Such methods, for the most part, carry over to elliptic scalar multiplication.

There are also efficiency improvements available in the elliptic case that have no analogue in modular exponentiation. There are three kinds of these:

1. One can choose the curve, and the base field over which it is defined, so as to optimize the efficiency of elliptic scalar multiplication. Thus, for example, one might choose the field of integers modulo a Mersenne prime, since modular

* This paper presents the results of cryptographic research conducted at NSA and does not necessarily represent the policies of the NSA or U.S. Government.

reduction is particularly efficient [2] in that case. This option is not available for, say, RSA systems, since the secret primes are chosen randomly in order to maintain the security of the system.

2. One can use the fact that subtraction of points on an elliptic curve is just as efficient as addition. (The analogous statement for integers (mod m) is false, since modular division is more expensive than modular multiplication.) The efficient methods for modular exponentiation all involve a sequence of squarings and multiplications that is based on the binary expansion of the exponent. The analogous procedure for elliptic scalar multiplication uses a sequence of doublings and additions of points. If we allow subtractions of points as well, we can replace [3] the binary expansion of the coefficient n by a more efficient *signed binary expansion* (i.e., an expansion in powers of two with coefficients 0 and ± 1).
3. One can use *complex multiplication*. Every elliptic curve over a finite field[2] comes equipped with a set of operations which can be viewed as multiplication by complex algebraic integers (as opposed to ordinary integers). These operations can be carried out efficiently for certain families of elliptic curves. In these cases, they can be utilized in various ways [5] to increase the efficiency of elliptic scalar multiplication.

It is the purpose of this paper to present a new technique for elliptic scalar multiplication. This new algorithm incorporates elements from all three of the above categories. The new method is 50% faster than any method previously known for operating on a non-supersingular elliptic curve.

2 Field and Elliptic Operations in $\mathbb{F}_{2^m}$

We begin with a brief survey of the various operations we will need in the field $\mathbb{F}_{2^m}$ and on elliptic curves over this field.

Squaring. We will assume that the field $\mathbb{F}_{2^m}$ is represented in terms of a *normal basis:* a basis over $\mathbb{F}_2$ of the form

$$\left\{\theta, \theta^2, \theta^{2^2}, \ldots, \theta^{2^{m-1}}\right\} .$$

The advantage of this representation is that squaring a field element can be accomplished by a one-bit cyclic shift of the bit string representing the element. This property will be crucial in what follows. If m is not divisible by 8, then one can use Gaussian cyclotomic periods to construct easily [6] an efficient normal basis for $\mathbb{F}_{2^m}$. (Since our application will require m to be prime, we can always use the Gaussian method.)

Our emphasis in this paper will be the case in which the field arithmetic is be implemented in hardware. Although the algorithms that follow will be efficient

[2] We restrict our attention to elliptic curves that are not *supersingular*, since such curves are cryptographically weak. (See [4].)

in software as well, the full advantage of our method occurs in hardware, where the bit shifts (and therefore field squarings) are virtually free.

Addition and Multiplication. We may neglect the cost of additions in $\mathbb{F}_{2^m}$ since they involve only bitwise XORs. A multiplication (of distinct elements) takes about m times as long, just as in the case of integer arithmetic. The cost of an elliptic operation depends mostly on the number of field multiplications it uses.

Inversion. Multiplicative inversion in $\mathbb{F}_{2^m}$ can be performed in

$$L(m-1)+W(m-1)-2$$

field multiplications using the method of [7]. Here $L(k)$ represents the length of the binary expansion of k, and $W(k)$ the number of ones in the expansion. This fact may be a consideration when choosing the degree m. (Alternatively, one can use the Euclidean algorithm [8], but one must first convert from the normal basis representation to the more familiar polynomial basis form, and then back again after the inversion.)

Elliptic Addition. The standard equation for an elliptic curve over $\mathbb{F}_{2^m}$ is the *Weierstrass equation*

$$E: \quad y^2+xy=x^3+ax^2+b \tag{1}$$

where $b \neq 0$. Public key protocols based on this curve work on the group consisting of the points (x,y) on this curve, along with the group identity $\mathcal{O}$. (The element $\mathcal{O}$ is called the *point at infinity*, but it is most convenient to represent it[3] by $(0,0)$.) The following algorithm inputs the points $P_0=(x_0,y_0)$ and $P_1=(x_1,y_1)$ on E and returns their sum $P_2=(x_2,y_2)$.

Algorithm 1. *(Elliptic Group Operation)*

If $P_0=\mathcal{O}$ then output $P_2 \leftarrow P_1$ and stop
If $P_1=\mathcal{O}$ then output $P_2 \leftarrow P_0$ and stop
If $x_0=x_1$
    then
        if $y_0+y_1=x_1$ then output $\mathcal{O}$ and stop
        else
            $\lambda \leftarrow x_1+y_1/x_1$
            $x_2 \leftarrow \lambda^2+\lambda+a$
            $y_2 \leftarrow x_1^2+(\lambda+1)\,x_2$
    else
        $\lambda \leftarrow (y_0+y_1)/(x_0+x_1)$
        $x_2 \leftarrow \lambda^2+\lambda+x_0+x_1+a$

[3] This does not cause confusion, because the origin is never on E.

$$y_2 \leftarrow (x_1 + x_2)\lambda + x_2 + y_1$$
Output $P_2 \leftarrow (x_2, y_2)$

To *subtract* the point $P = (x, y)$, one adds the point $-P = (x, x + y)$.

Except for the special cases involving $\mathcal{O}$, the above addition and subtraction operations each require 1 multiplicative inversion and 2 multiplications.[4] (As always, we disregard the cost of adding and squaring field elements.)

Elliptic Scalar Multiplication. The basic technique for elliptic scalar multiplication is the *addition-subtraction method.* This begins with the *nonadjacent form* (NAF) of the coefficient n: a signed binary expansion with the property that no two consecutive coefficients are nonzero. For example,

$$\text{NAF}(29) = \langle 1, 0, 0, -1, 0, 1 \rangle \tag{2}$$

since $29 = 32 - 4 + 1$.

Just as every positive integer has a unique binary expansion, it also has a unique NAF. Moreover, NAF(n) has the fewest nonzero coefficients of any signed binary expansion of n (see [1]). There are several ways to construct the NAF of n from its binary expansion. We present the one that most resembles the new algorithm we will present in §3.

The idea is to divide repeatedly by 2. Recall that one can derive the binary expansion of an integer by dividing by 2, storing off the remainder (0 or 1), and repeating the process with the quotient. To derive a NAF, one allows remainders of 0 or ± 1. If the remainder is to be ± 1, one chooses whichever makes the quotient even.

Algorithm 2. *(NAF)*

Input n
Set $k \leftarrow n$
Set $\mathcal{S} \leftarrow \langle\,\rangle$
While $k > 0$
    If k **odd**
        then set $u \leftarrow 2 - (k \pmod 4)$
        else set $u \leftarrow 0$
    Set $k \leftarrow k - u$
    Prepend u **to** $\mathcal{S}$
    Set $k \leftarrow k/2$
EndWhile
Output $\mathcal{S}$

For example, to derive (2), one applies Alg. 2 with $n = 29$. The results are shown in Fig. 1.

[4] There does exist a faster algorithm for doubling a point, but we relegate it to the Appendix since it does not fit well with the best hardware implementations of normal bases.

Fig. 1. Computing a NAF.

k	u	S
29		$\langle\rangle$
28	1	
14		$\langle 1\rangle$
14	0	
7		$\langle 0,1\rangle$
8	-1	
4		$\langle -1,0,1\rangle$
4	0	
2		$\langle 0,-1,0,1\rangle$
2	0	
1		$\langle 0,0,-1,0,1\rangle$
0	1	
0		$\langle 1,0,0,-1,0,1\rangle$

Note that, although we have phrased the algorithm in terms of integer arithmetic, it can be implemented in terms of bit operations on the binary expansion of n. No arithmetic operations are needed beyond integer addition by 1.

In the derivation of the ordinary binary expansion, the sequence k is decreasing, but that is not true in general in Alg. 2. As a result, the NAF of a number may be longer than its binary expansion. Fortunately, it can be at most one bit longer, because

$$2^{\ell} < 3n < 2^{\ell+1}$$

where ℓ is the bit length of NAF(n). (See [3].)

We now implement elliptic scalar multiplication using the NAF. Given the NAF

$$n = \sum_{i=0}^{\ell-1} e_i 2^i \ ,$$

the elliptic scalar multiplication $Q = nP$ is performed as follows.

Algorithm 3. *(Addition-Subtraction Method)*

Input P
Set $Q \leftarrow P$
For $i = \ell - 2$ **downto** 1 **do**
  Set $Q \leftarrow 2Q$
  If $e_i = 1$ **then set** $Q \leftarrow Q + P$
  If $e_i = -1$ **then set** $Q \leftarrow Q - P$
Output Q

The cost of Alg. 3 depends on the bit length ℓ of NAF(n), which we now estimate. It follows from the Hasse theorem that the order of an elliptic curve over $\mathbb{F}_{2^m}$ is

$$\#E(\mathbb{F}_{2^m}) = 2^m + O(2^{m/2}) \ . \tag{3}$$

Most public-key protocols on elliptic curves use a base point of prime order p. Since all of the curves (1) have even order, then p must be at most $2^{m-1} + O(2^{m/2})$. We can assume that $n < p$; thus[5] $\ell \leq m$.

It follows that Alg. 3 requires about m doubles at most. The number of additions is (about) the number of nonzero coefficients in NAF(n). The average density of nonzero coefficients among NAF's is 1/3 (see [3]). Therefore the average cost of Alg. 3 is $\sim m$ doubles and $\sim m/3$ additions, for a total of $\sim 4m/3$ elliptic operations. This is about one-eighth faster than the *binary method*, which uses the ordinary binary expansion in place of the NAF and therefore requires an average of $\sim m/2$ elliptic additions rather than $\sim m/3$.

3 Anamolous Binary Curves

Two extremely convenient families of curves [5] are the *anamolous binary curves* (or ABC's). These are the curves E_0 and E_1 defined over $\mathbb{F}_2$ by

$$E_a : y^2 + xy = x^3 + a\,x^2 + 1 \ .$$

We denote by $E_a(\mathbb{F}_{2^m})$ the group of $\mathbb{F}_{2^m}$-rational points on E_a. This is the group on which the public-key protocols are performed. The group should be chosen so that it is computationally difficult to compute discrete logarithms of its elements. Thus, for example, the order $\#E_a(\mathbb{F}_{2^m})$ should be divisible by a large prime (see [9]). Ideally, $\#E_a(\mathbb{F}_{2^m})$ should be a prime or the product of a prime and small integer. This can only happen when m is itself prime, for otherwise there are large divisors arising from subgroups $E_a(\mathbb{F}_{2^d})$ where d divides m.

Actually, the orders $\#E_a(\mathbb{F}_{2^m})$ are never prime, because they always contain the point $(0, 1)$, which is easily seen to have order 2. The best result to be hoped for, then, is that the order is twice a prime. This happens relatively frequently for E_1. The values of $m \leq 512$ for which $\#E_1(\mathbb{F}_{2^m})$ is twice a prime are

$$m = 3, 5, 7, 11, 17, 19, 23, 101, 107, 109, 113, 163, 283, 311, 331, 347, 359 \ .$$

The curves E_0 contain the points $(1, 0)$ and $(1, 1)$, which are easily seen to have order 4. The best result to be hoped for among the curves E_0, then, is that the order is 4 times a prime. The values of $m \leq 512$ for which this happens are

$$m = 5, 7, 13, 19, 23, 41, 83, 97, 103, 107, 131, 233, 239, 277, 283, 349, 409 \ .$$

[5] A further one-bit improvement on this bound is possible if we use the identity

$$n(x, y) = (p - n)(x, x + y)$$

whenver $n > p/2$. Moreover, if a has trace 0 over $\mathbb{F}_2$, we save yet another bit since the order of E must be divisible by 4.

Since the ABC's are defined over $\mathbb{F}_2$, they have the property that, if $P = (x, y)$ is a point on E_a, then so is the point (x^2, y^2). Moreover, one can verify from Alg. 1 that

$$(x^4, y^4) + 2(x, y) = (-1)^{1-a}(x^2, y^2) \tag{4}$$

for every (x, y) on E_a. This relation can be written more easily in terms of the Frobenius (squaring) map over $\mathbb{F}_2$:

$$\tau(x, y) = (x^2, y^2) \ .$$

Using this notation, (4) becomes

$$\tau(\tau P) + 2P = (-1)^{1-a}\tau P$$

for all $P \in E$. Symbolically, this can be written

$$(\tau^2 + 2)P = (-1)^{1-a}\tau P \ .$$

This means that the squaring map can be regarded as implementing multiplication by the complex number τ satisfying

$$\tau^2 + 2 = (-1)^{1-a}\tau \ .$$

Explicitly, this number is

$$\tau = \frac{(-1)^{1-a} + \sqrt{-7}}{2} \ .$$

By combining the squaring map with ordinary scalar multiplication, we can multiply points on E_a by any element of the ring $\mathbb{Z}[\tau]$. We say that E_a has *complex multiplication* by τ. (See [5].)

The reason why this property is useful for elliptic scalar multiplication is that multiplication by τ, being implemented by squaring, is essentially free when $\mathbb{F}_{2^m}$ is represented in terms of a normal basis. Thus it is worthwhile, when computing nP, to regard n as an element of $\mathbb{Z}[\tau]$ rather than as "just" an integer. More precisely, one replaces the (signed) binary expansion of the coefficient with the (signed) *τ-adic expansion*. That is, one represents n as a sum and difference of distinct powers of τ.

For example, with $a = 0$ we have

$$9 = \tau^5 - \tau^3 + 1 \ . \tag{5}$$

Thus, if $P = (x, y)$ is a point on E_0, then

$$9P = (x^{32}, y^{32}) - (x^8, y^8) + (x, y) \ .$$

The above example gives 9 as what we call a *τ-adic NAF*, since no two consecutive terms are nonzero. (Both [5] and [10] use signed τ-adic expansions, but neither kind has the nonadjacency property.) As we shall see, the use of the τ-adic NAF gives a significant reduction in the number of terms, just as the

NAF gives a significant improvement over the binary expansion in the case of integers.

The τ-adic NAF has a property analogous to the NAF for integers, namely that *every element of the ring $\mathbb{Z}[\tau]$ has a unique τ-adic NAF.* We shall prove the existence by providing the construction. (The proof of uniqueness is similar to that of the NAF for integers.)

We begin with the observation that $x + y\tau$ is divisible by τ if and only if x is even. One direction of this statement follows from the identity

$$(u + v\tau)\,\tau = -2v + (u + (-1)^{1-a}v)\,\tau \ ,$$

and the other from the fact that, if $x = 2v$, then

$$x + y\tau = (y + (-1)^{1-a}v - \tau v)\,\tau \ .$$

We now present the algorithm [11] for computing the τ-adic NAF. It is completely analogous to Alg. 2, but here we are dividing by τ rather than by 2. The ring $\mathbb{Z}[\tau]$ is Euclidean with norm function

$$N(x + y\tau) = x^2 + (-1)^{1-a}\,xy + 2y^2 \ .$$

Since τ has norm 2, the possible remainders upon division by τ are ± 1. Earlier algorithms chose the remainder that minimized the norm of the quotient; this is analogous to the basic division algorithm for generating the binary expansion of an integer. What we shall do instead is to choose the remainder that makes the quotient divisible by τ (i.e., having real part even). This is analogous to the computation of the NAF for integers.

Algorithm 4. *(τ-adic NAF)*

```
Input x_0, y_0
Set x ← x_0, y ← y_0
Set S ← ⟨⟩
While x ≠ 0 or y ≠ 0,
        If x odd,
            then set u ← 2 − (x − 2y (mod 4))
            else set u ← 0
        Set x ← x − u
        Prepend u to S
        Set (x, y) ← (y + (−1)^a x/2, −x/2)
EndWhile
Output S
```

For example, to derive (5), one applies Alg. 4 with $a = 0$, $x = 9$, and $y = 0$. The results are shown in Fig. 2.

Note that the implementation of Alg. 4 involves nothing more complicated than integer addition. (This is slightly more than is required by Alg. 2, which only adds 1 to an integer.)

Fig. 2. Computing a generalized NAF.

x	y	u	S
9	0		$\langle\rangle$
8	0	1	
4	−4		$\langle 1\rangle$
4	−4	0	
−2	−2		$\langle 0,1\rangle$
−2	−2	0	
−3	1		$\langle 0,0,1\rangle$
−2	1	−1	
0	1		$\langle -1,0,0,1\rangle$
0	1	0	
1	0		$\langle 0,-1,0,0,1\rangle$
0	0	1	
0	0		$\langle 1,0,-1,0,0,1\rangle$

An argument similar to the one [3] in the NAF case proves that the average density of nonzero terms among τ-adic NAF's is 1/3. There is a drawback to this representation, however: the τ-adic NAF of an integer n is about twice as long as its ordinary NAF. This is because Alg. 4 begins with n, which is an element of $\mathbb{Z}[\tau]$ with norm n^2, and repeatedly divides by τ, which has norm 2.

The solution is to adopt the following modification from [10]. Recall that multiplication by τ is implemented by a one-bit circular shift of each of the m-long bit strings representing the coordinates of P. Multiplication by τ^m, then, involves m such shifts, returning each coordinate to its original state. In other words, $\tau^m P = P$ for all $P \in E_a(\mathbb{F}_{2^m})$. It follows that, if α and β are elements of $\mathbb{Z}[\tau]$ with $\alpha \equiv \beta \pmod{\tau^m - 1}$, then $\alpha P = \beta P$ for all P.

This means that, to multiply by n, one need not work with n itself, but rather the remainder obtained from dividing n by $\tau^m - 1$. Since $\mathbb{Z}[\tau]$ is Euclidean, this remainder will have norm smaller than that of $\tau^m - 1$. The norm of $\tau^m - 1$ is precisely the order of $E_a(\mathbb{F}_{2^m})$, and this is roughly 2^m by (3). Thus the τ-adic NAF of the remainder will have length $\sim m$, only half as long as the τ-adic NAF of n itself. Moreover, the average density is still only 1/3. To see this, one must examine the distribution of the residues (mod $\tau^m - 1$) of the integers; see [1].

To implement this improvement, one needs a division algorithm in $\mathbb{Z}[\tau]$. The following algorithm inputs the dividend $u + v\tau$ and divisor $r + s\tau$ and outputs a quotient $w + z\tau$ and remainder $x + y\tau$, the latter having smaller norm than the divisor.

Algorithm 5. *(Division in the Ring $\mathbb{Z}[\tau]$)*

Input u, v, r, s
Set $k \leftarrow ru + su + 2sv$,

$\ell \leftarrow rv - su$
Set $h \leftarrow r^2 + (-1)^{1-a} rs + 2s^2$
Set $w \leftarrow \lfloor k/h \rceil$,
$z \leftarrow \lfloor \ell/h \rceil$
Set $x \leftarrow u - rw + 2sz$,
$y \leftarrow v - sw - rz - sz$
Output w, z, x, y

To apply Alg. 5, one needs to express $\tau^m - 1$ as an expression of the form $r + s\tau$. This is done via Lucas sequences. Let $U_0 = 0$, $U_1 = 1$, and

$$U_k = (-1)^{1-a} U_{k-1} - 2U_{k-2}$$

for $k \geq 2$. It is easy to prove that

$$\tau^m = U_m \tau - 2U_{m-1} \ .$$

Thus we have the following procedure for computing nP in $E_a(\mathbb{F}_{2^m})$.

Algorithm 6. *(Scalar Multiplication on ABC's)*

1. Divide n by $U_m \tau - (2U_{m-1} + 1)$ via Alg. 5.
2. Compute the τ-adic NAF

$$\langle e_\ell, e_{\ell-1}, \ldots, e_1, e_0 \rangle$$

of the remainder via Alg. 4.

3. **Set** $Q \leftarrow e_\ell P$
4. **For** i **from** $\ell - 1$ **downto** 1 **do**
Set $Q \leftarrow \tau Q (=$ **Shift**$[Q])$
If $e_i = 1$ **then set** $Q \leftarrow Q + P$
If $e_i = -1$ **then set** $Q \leftarrow Q - P$
5. **Output** Q.

Except for Step 1 (i.e. Alg. 5), the only arithmetic required by Alg. 6 is binary field arithmetic and integer addition. Alg. 5, on the other hand, requires several multiplications and divisions involving m-bit numbers. Thus it is less well suited to hardware, and more expensive in software, than the other steps.[6] The running time of Step 1, however, is negligible compared to the actual elliptic scalar multiplication (see [10]).

Since $\ell \approx m$, then Alg. 6 requires $\sim m/3$ additions and no doubles. This is at least 50% faster than any of the earlier versions, as is shown in Table 1.

[6] On the other hand, Alg. 5 can be replaced by simpler and more efficient algorithms that do much the same thing. For example, one might use a "double-and-add" method of "building up" to the integer n via its binary expansion, reducing when needed by suitable multiples of $\tau^m - 1$. Such reductions would involve additions rather than multiplications and divisions. Details are not available as of this writing, but it seems that an efficient implementation could be developed which would yield a τ-adic NAF of only a few bits over the output of Alg. 5.

Table 1. Comparison of Elliptic Scalar Multiplication Techniques.

Type of Curve	Method	Length of Expansion	Avg. Density	Avg. # of Elliptic Operations
General	Binary Method	m	1/2	$3m/2$
"	Addition-Subtraction (1989)	m	1/3	$4m/3$
ABC	Koblitz, Balanced (1991)	$2m$	3/8	$3m/4$
"	Meier-Staffelbach (1992)	m	1/2	$m/2$
"	τ-adic NAF (1997)	m	1/3	$m/3$

The "length" and "density" columns give the approximate length of the relevant representation of the number and the average density of nonzero terms. The density figure of 3/8 for Koblitz' "balanced" expansions is from experimental observation and may be only an approximation.

4 Precomputation and Memory Speedups

We can obtain still more dramatic savings by precomputing and storing some "small" τ-adic multiples of P. By this we mean the points αP for which $\alpha \in \mathbb{Z}[\tau]$ has a short τ-adic NAF. These precomputed values can then be used as needed when going through the τ-adic expansion of n. This is essentially a "τ-adic window method." We illustrate with a simple example: that of using windows of a fixed width w.

This method is very similar to the fixed-width version of the window method for ordinary NAF's of integers. Consider the following example. We let the width $w = 4$ and $n = 22310$. Then NAF(n) is given by

$$\langle 1, 0, -1, 0, -1, 0, 0, -1, 0, 0, 1, 0, 1, 0, -1, 0 \rangle \ . \tag{6}$$

We now rewrite (6) by allowing nonzero coefficients to take on the values ± 3, ± 5, ± 7, ± 9 as well as ± 1. (This choice reflects the fact that the odd numbers 1 through 9 are the ones with NAF of length 4 or less.) We go right to left, as in Fig. 3.

As a result, we have the expression

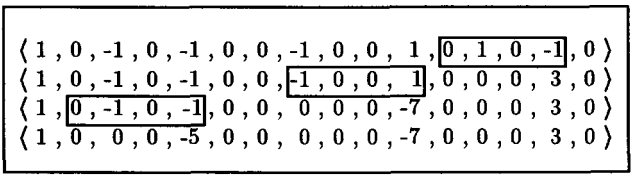
$$22310 = 2^{15} - 5 \cdot 2^{11} - 7 \cdot 2^5 + 3 \cdot 2 \ .$$

Therefore, we can multiply the point P by 22310 by precomputing $3P$, $5P$, $7P$, $9P$ and calculating

$$22310P = 2^{15}P - 2^{11}(5P) - 2^5(7P) + 2(3P)$$

via the suitable generalization of Alg. 3.

Fig. 3. Widening a NAF.



Applying the method for the general width-w case requires

$$C(w) = (2^w - (-1)^w)/3$$

values to be precomputed and stored. The resulting width-w NAF has the property that any w consecutive coefficients include at most one nonzero entry. The average density of nonzero coefficients among width-w NAF's is $(w+1)^{-1}$.

The same width-w NAF calculations can be used in the τ-adic case. The example analogous to the above is multiplication by

$$\alpha = \tau^{15} - \tau^{13} - \tau^{11} - \tau^8 + \tau^5 + \tau^3 - \tau \ ,$$

since the τ-adic NAF of α is given by (6). To devise a width-w τ-adic NAF of α, we allow nonzero coefficients to take on the values $\pm\beta_3$, $\pm\beta_5$, $\pm\beta_7$, $\pm\beta_9$ as well as ± 1, where β_k is the element of $\mathbb{Z}[\tau]$ whose τ-adic NAF is the same as the ordinary NAF of k. (Explicit values are given in Fig. 4.)

Fig. 4. Analogues of the Small Odd Integers.

$\mathrm{NAF}(3) = \langle 1, 0, -1 \rangle$	$\beta_3 = \tau^2 - 1$	$P_3 = (\tau^2 - 1)P$
$\mathrm{NAF}(5) = \langle 1, 0, 1 \rangle$	$\beta_5 = \tau^2 + 1$	$P_5 = (\tau^2 + 1)P$
$\mathrm{NAF}(7) = \langle 1, 0, 0, -1 \rangle$	$\beta_7 = \tau^3 - 1$	$P_7 = (\tau^3 - 1)P$
$\mathrm{NAF}(9) = \langle 1, 0, 0, 1 \rangle$	$\beta_9 = \tau^3 + 1$	$P_9 = (\tau^3 + 1)P$

The calculation shown in Fig. 3 shows that the width-4 τ-adic NAF of α is

$$\alpha = \tau^{15} - \beta_5 \cdot \tau^{11} - \beta_7 \cdot \tau^5 + \beta_3 \cdot \tau \ .$$

Thus one computes

$$\alpha P = \tau^{15} P - \tau^{11} P_5 - \tau^5 P_7 + \tau P_3$$

by precomputing and storing the points P_i given in Fig. 4.

To perform this procedure in general requires enough memory to store $C(w)$ points, including P itself. The precomputation requires $C(w) - 1$ elliptic additions, and no memory other than that used to store the $C(w)$ points. The main

computation is the analogue of Alg. 6, performed on a length-m, width-w, τ-adic NAF with average density $(w+1)^{-1}$. The total work, then, is

$$\sim \frac{2^w}{3} + \frac{m}{w+1} \text{ elliptic additions.}$$

Table 2 gives the performance of this algorithm on the curve $E_1(\mathbb{F}_{2^{163}})$ for various widths. (Entries are rounded to the nearest integer.) The case $w = 2$ is the ordinary method of §3. By choosing $w = 4$ or 5, one saves roughly one-third the work. For larger w, the precomputation costs overshadow any savings on the real-time computation.

Table 2. Performance at Various Widths.

Width	Number of Elliptic Operations		
	Precomp-utation	Real Time (avg)	Total (avg)
2	0	52	52
3	2	39	41
4	4	31	35
5	9	26	35
6	20	23	43
7	42	19	61

It is remarkable that one can perform a general elliptic scalar multiplication on $E_1(\mathbb{F}_{2^{163}})$ using only about 35 multiplicative inversions and 70 field multiplications.

One could obtain still further speedups by using more general window methods. These would be straightforward adaptations of existing methods such as those found in [12]. On the other hand, such methods are less automatic than the above fixed-width-window technique, so that more complicated up-front calculations are needed.

Note added during review: the results of [10] have recently been generalized to curves defined over fields of 2^d elements for small d. For example, the curves with complex multiplication by $(\pm 1 + \sqrt{-15})/2$ are defined over $\mathbb{F}_{2^2}$. The results of this paper should also carry over to this more general situation.

References

1. D. Gordon, "A survey of fast exponentiation methods" *(to appear).*
2. D. E. Knuth, *Seminumerical Algorithms,* Addison-Wesley, 1981, p. 272.

3. F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains", *Inform. Theor. Appl.* **24** (1990), pp. 531–543.
4. A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *Proc. 23rd Annual ACM Symp. on Theory of Computing* (1991), pp. 80–89.
5. N. Koblitz, "CM curves with good cryptographic properties", *Proc. Crypto '91*, Springer-Verlag, 1992, pp. 279–287.
6. D. W. Ash, I. F. Blake, and S. Vanstone, "Low complexity normal bases", *Discrete Applied Math.* **25** (1989), pp. 191–210.
7. T. Itoh, O Teechai, and S. Trojii, "A fast algorithm for computing multiplicative inverses in $GF(2^t)$", *J. Soc. Electron. Comm.* (Japan) **44** (1986), pp. 31–36.
8. E. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, 1984, pp. 36-44.
9. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997, pp. 107–109.
10. W. Meier and O. Staffelbach, "Efficient multiplication on certain non-supersingular elliptic curves", *Proc. Crypto '92*, Springer-Verlag, 1993, pp. 333–344.
11. R. Reiter and J. Solinas, "Fast elliptic arithmetic on special curves", *NSA/R21 Informal Tech. Report*, 1997.
12. K. Koyama and Y. Tsuruoka, "Speeding up elliptic cryptosystems by using a signed binary window method", *Proc. Crypto '92*, Springer-Verlag, 1993, pp. 345–357.
13. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Proc. Crypto '95*, Springer-Verlag, 1995, pp. 43–56.
14. R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Univ. of Arizona Comp. Sci. Tech. Report* 95-03, 1995.
15. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Elsevier, 1977, pp. 277–279.

A Elliptic Doubling with Normal Bases

The following technique[7] carries out the doubling

$$(x_2, y_2) = 2\,(x_1, y_1)$$

of a point (for which $x_1 \neq 0$) on the curve

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

over $\mathbb{F}_{2^m}$, where the field is represented in terms of a normal basis. The usual algorithm requires 1 multiplicative inversion and 2 multiplications. The method given here replaces one of the general multiplications by a multiplication by a fixed constant (namely b). The operation of multiplying by a fixed constant is comparable in speed to field addition. Therefore the effective cost of this algorithm is 1 multiplicative inversion and 1 multiplication.

One begins by computing

$$x_2 = x_1^2 + \frac{b}{x_1^2}\ ,$$

[7] This method is of the kind alluded to in [13]. There it is credited to [14], which is not so easily available; hence its inclusion in this Appendix.

which is easily seen to equal the expression for x_2 appearing in Alg. 1. One then finds a root μ of the quadratic equation

$$\mu^2 + \mu = x_2 + a \ .$$

Since the field is being represented in terms of a normal basis, this process can be done without using anything more expensive than addition [15], so we can neglect its cost. The element μ will equal $\lambda + e$, where $e = 0$ or 1 and

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

Therefore

$$\mu x_1 + x_1^2 + y_1 = e x_1 \ .$$

This equation allows us to find e and therefore λ. Notice that it is not necessary to perform the multiplication μx_1 in full, but rather to compute one coordinate of the product. (We can choose any coordinate where the corresponding coordinate of x_1 is 1.) Computing one coordinate of a product costs the same as an addition, so the derivation of λ is virtually cost-free. To complete the doubling, one computes

$$y_2 = x_1^2 + (\lambda + 1) x_2 \ .$$