New Point Addition Formulae for ECC Applications

Nicolas Méloni

New Point Addition Formulae for ECC Applications

Nicolas Meloni[1,2]

[1] Institut de Mathématiques et de Modélisation de Montpellier,
Univ. Montpellier 2, France
[2] Laboratoire d'Informatique,
de Robotique et de Microélectronique de Montpellier,
CNRS, Univ. Montpellier 2, France
nicolas.meloni@lirmm.fr

Abstract. In this paper we propose a new approach to point scalar multiplication on elliptic curves defined over fields of characteristic greater than 3. It is based on new point addition formulae that suit very well to exponentiation algorithms based on Euclidean addition chains. However finding small chains remains a very difficult problem, so we also develop a specific exponentiation algorithm, based on Zeckendorf representation (i.e. representing the scalar k using Fibonacci numbers instead of powers of 2), which takes advantage of our formulae.

Keywords: elliptic curve, scalar multiplication, exponentiation, Fibonacci, addition chains.

1 Introduction

Since its introduction by Miller and Koblitz [11,9], elliptic curve cryptography (ECC) has received a lot of attention and has subsequently become one of the main standards in public key cryptography. The main operation (in terms of computations) of such systems is the point scalar multiplication, i.e. the computation of the point $[k]P = P + \cdots + P$, where k is an integer and P a point on a curve. It involves hundreds of multiplications on the underlying field which means that some efforts are to be made on optimizing this computation. This is precisely what this paper deals with.

A point scalar multiplication is just a sequence of point additions, being themselves made of several multiplications, squarings and inversions on a finite field. So improvements can be done at the finite field level by developing faster modular multiplication algorithms, at the curve level by improving the point addition and finally at the algorithmic level by proposing exponentiation algorithms adapted to the context of elliptic curves.

In this paper we will contribute to the two last levels. First we will propose new point addition formulae in a specific case. More precisely, if one computes $P_3 = P_1 + P_2$ on a curve then computing $P_3 + P_1$ or $P_3 + P_2$ can be done at a very low computational cost. Then we will compare this approach to existing works

C. Carlet and B. Sunar (Eds.): WAIFI 2007, LNCS 4547, pp. 189–201, 2007.

done by Montgomery [13] and generalized by Brier and Joye on one hand [1] and by Lopez and Dahab [10] on the other hand. Those formulae suit very well to Euclidean addition chains which will lead to a very efficient point multiplication algorithm as long as one is able to find a small chain to compute a given integer. This problem being still difficult, we will propose next to represent the scalar k using Fibonacci numbers instead of powers of 2. That is to say writing $k = \sum_{i=1}^{n} F_i$, where F_i is the ith Fibonacci number. This will allow us to propose a "Fibonacci-and-add" algorithm taking advantage of our formula.

2 Elliptic Curve Arithmetic

Definition 1. *An elliptic curve E over a field K denoted by E/K is given by the equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, a_2, a_3, a_4, a_6 \in K$ are such that, for each point (x, y) on E, the partial derivatives do not vanish simultaneously.

In practice, the equation can be simplified into

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

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$, over field of characteristic greater than 3.

The set of points of E/K is an abelian group. There exist explicit formulae to compute the sum of two points, and several coordinate systems have been proposed to speed up this computation. For a complete overview of those coordinates, one can refer to [3,6]. As an example, in Jacobian coordinates, the curve E (over a field of characteristic greater than 3) is given by $Y^2 = X^3 + a_4XZ^4 + a_6Z^6$, the point (X, Y, Z) on E corresponds to the affine point $(\frac{X}{Z^2}, \frac{Y}{Z^3})$ and the formulae are:

Addition:
$P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ and $P + Q = (X_3, Y_3, Z_3)$

$$A = X_1Z_1^2,\ B = X_2Z_1^2,\ C = Y_1Z_2^3,\ D = Y_2Z_1^3,\ E = B - A,\ F = D - C$$

and

$$X_3 = -E^3 - 2AE^2 + F,\ Y_3 = -CE^3 + F(AE^2 - X_3),\ Z_3 = Z_1Z_2E$$

Doubling:
$[2]P = (X_3, Y_3, Z_3)$

$$A = 4X_1Y_1^2,\ B = 3X_1^2 + a_4Z_1^4$$

and

$$X_3 = -2A + B^2,\ Y_3 = -8Y_1^4 + B(A - X_3),\ Z_3 = 2Y_1Z_1.$$

The computation cost is 12 multiplications (M) and 4 squarings (S) (8M and 3S if one of the point is given in the form (X,Y,1)) for the addition and 4M and 6S for the doubling.

The computation of $[k]P$ is usually done using Algorithm 1. It requires about $\log_2(k)$ doublings and $w(k)$ additions, where $w(k)$ is the Hamming weight of k. Several methods have been developed to reduce both the cost of a doubling and the number of additions. This can be achieved by using, for example, modified Jacobian coordinates and windowing methods [4]. Other methods include the use of a different number system to represent the scalar k, as the double base number system [5]. Finally over binary fields doubling can be replaced by other endomorphisms such as the Frobenius endomorphism [14] or point halving [7].

Algorithm 1. Double-and-add

Data: $P \in E$ and $k = (k_{l-1}, \ldots, k_0)_2 \in \mathbb{N}$.
Result: $[k]P \in E$.
begin
  $Q \leftarrow P$
  for $i = l-2 \ldots 0$ **do**
    $Q \leftarrow [2]Q$
    if $k_i = 1$ **then**
      $Q \leftarrow Q + P$
    end
  end
end
return Q

3 New Point Addition Formulae

Let K be a field of characteristic greater than 3, E/K an elliptic curve, $P_1 = (X_1, Y_1, Z)$ and $P_2 = (X_2, Y_2, Z)$ two points (in Jacobian coordinates) on E sharing the same z-coordinate. Then if we note $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3)$ we have:

$$\begin{aligned}
X_3 &= (Y_2Z^3 - Y_1Z^3)^2 - (X_2Z^2 - X_1Z^2)^3 - 2X_1Z^2(X_2Z^2 - X_1Z^2)^2 \\
&= ((Y_2 - Y_1)^2 - (X_2 - X_1)^3 - 2X_1(X_2 - X_1)^2)Z^6 \\
&= ((Y_2 - Y_1)^2 - (X_1 + X_2)(X_2 - X_1)^2)Z^6 \\
&= X_3'Z^6 \\
Y_3 &= -Y_1Z^3(X_2Z^2 - X_1Z^2)^3 \\
&\quad +(Y_2Z^3 - Y_1Z^3)(X_1Z^2(X_2Z^2 - X_1Z^2)^2 - X_3) \\
&= (-Y_1(X_2 - X_1)^3 + (Y_2 - Y_1)(X_1(X_2 - X_1)^2 - X_3'))Z^9 \\
&= Y_3'Z^9 \\
Z_3 &= Z^2(X_2Z^2 - X_1Z^2) \\
&= Z(X_2 - X_1)Z^3 \\
&= Z_3'Z^3
\end{aligned}$$

Thus we have $(X_3, Y_3, Z_3) = (X_3'Z^6, Y_3'Z^9, Z_3'Z^3) \sim (X_3', Y_3', Z_3')$.

So when P_1 and P_2 have the same z-coordinate, $P_1 + P_2$ can be obtained using the following formulae:

Addition:
$P_1 = (X_1, Y_1, Z), P_2 = (X_2, Y_2, Z)$ and $P_1 + P_2 = (X_3', Y_3', Z_3')$

$$A = (X_2 - X_1)^2,\ B = X_1 A,\ C = X_2 A,\ D = (Y_2 - Y_1)^2$$

and

$$\begin{aligned} X_3' &= D - B - C\ , \\ Y_3' &= (Y_2 - Y_1)(B - X_3) - Y_1(C - B)\ , \\ Z_3' &= Z(X_2 - X_1)\ . \end{aligned}$$

This addition involves 5M and 2S.

As they require special conditions, our formulae are logically more efficient than any general or mixed addition formulae. What is more striking is the fact that they are more efficient than any doubling formulae (the best doubling is obtained using modified Jacobian coordinates and requires 4M and 4S).

The comparison with Montgomery's elliptic curves arithmetic is a lot more interesting. At a first sight the approaches look very similar. Indeed on Montgomery's curves the arithmetic is based on the fact that it is easy to compute the x and z-coordinates of $P_1 + P_2$ from the x and z-coordinates of P_1, P_2 and $P_1 - P_2$. The computational cost of this addition is 4M and 2S, which is lower than with our formula, but requires additional computations to recover the y-coordinate. Besides, recovering the y-coordinate requires to perform the point scalar multiplication using the Montgomery ladder algorithm. In the case of Euclidean addition chains exponentiation (treated in the next section) one cannot recover the y-coordinate from Montgomery's formulae. On the other hand we will show that it is possible not to compute the y-coordinate with our formulae. In this case the computational cost of our formulae is reduced to 4M+2S.

Finally notice that not every elliptic curves are Montgomery's curves (Brier and Joye generalized this approach to general curves [1] but in this case the computational cost rises to 9M and 2S) whereas our formulae work on any curve (as long as the characteristic of the underlying field is greater than 3).

It seems unlikely for both P_1 and P_2 to have the same z-coordinate. Fortunately the quantities $X_1 A = X_1(X_2 - X_1)^2$ and $Y_1(C - B) = Y_1(X_2 - X_1)^3$ computed during the addition can be seen as the x and y-coordinates of the point $(X_1(X_2 - X_1)^2, Y_1(X_2 - X_1)^3, Z(X_2 - X_1)) \sim (X_1, Y_1, Z)$. Thus it is possible to add P_1 and $P_1 + P_2$ with our new formulae.

Remark 1. *The same observation can be made from the doubling formulae, indeed the quantities* $A = X_1(2Y_1)^2$ *and* $8Y_1^4 = Y_1(2Y_1)^3$ *are the* x *and* y*-coordinates of the point* $(X_1(2Y_1)^2, Y_1(2Y_1)^3, 2Y_1 Z_1) \sim (X_1, Y_1, Z_1)$ *allowing us to compute* $P + [2]P$ *without additional computation.*

So we now have at our disposal an operator `NewADD` working the following way: let P_1 and P_2 be two points sharing the same z-coordinate then `NewADD`(P_1, P_2) returns two points, $P_1 + P_2$ and P_1, sharing the same z-coordinate.

Example 1. *One can compute* $[25]P$ *in the following way:*

- *NewADD*$([2]P, P) = ([3]P, [2]P)$
- *NewADD*$([2]P, [3]P) = ([5]P, [2]P)$
- *NewADD*$([2]P, [5]P) = ([7]P, [2]P)$

- *NewADD*$([7]P, [2]P)$=$([9]P, [7]P)$
- *NewADD*$([9]P, [7]P)$=$([16]P, [9]P)$
- *NewADD*$([16]P, [9]P)$=$([25]P, [16]P)$

Remark 2. *The same kind of formulae can be developed in characteristic two. However we do not deal with this case in the remainder of the paper. Indeed Lopez and Dahab showed [10] that all curve can be turned into Montgomery's. Moreover many other methods, as fast doublings, point halving etc, lead to very efficient exponentiation algorithms so that our approach is no longer relevant.*

4 Point Scalar Multiplication

From the previous section we have seen that our formulae are quite efficient in terms of computational cost (more than a doubling) but cannot be used with classical double-and-add algorithms and require specific exponentiation schemes, as the one shown on example 1.

4.1 Euclidean Addition Chains

In this section we first show that the `NewADD` operator suits very well to Euclidean addition chains. We will then explain why finding such chains that are small is difficult.

Definition 2. *An addition chain computing an integer k is given by a sequence $v = (v_1, \ldots, v_s)$ where $v_1 = 1,\ v_s = k$ and $\forall\ 1 \leq i \leq s,\ v_i = v_{i_1} + v_{i_2}$ for some i_1 and i_2 lower than i.*

Definition 3. *An Euclidean addition chain (EAC) computing an integer k is an addition chain which satisfies $v_1 = 1, v_2 = 2, v_3 = v_2 + v_1$ and $\forall\ 3 \leq i \leq s-1$, if $v_i = v_{i-1} + v_j$ for some $j < i-1$, then $v_{i+1} = v_i + v_{i-1}$ (case 1) or $v_{i+1} = v_i + v_j$ (case 2).*

Case 1 will be called big step (we add the biggest of the two possible numbers to v_i) and case 2 small step (we add the smallest one).

As an example, $(1, 2, 3, 4, 7, 11, 15, 19, 34)$ is an Euclidean addition chain computing 34. For instance, in step 4 we have computed 4=3+1, thus in step 5 we must add 3 or 1 to 4, in other words from step 4 we can only compute 5=4+1 or 7=4+3. In this example we have chosen to compute 7=4+3 so, at step 6, we can compute 10=7+3 or 11=7+4 etc. Another classical example of EAC is the Fibonacci sequence $(1, 2, 3, 5, 8, 13, 21, 34)$ (which is only made of big steps).

Finding such chains is quite simple, it suffices to choose an integer g coprime with k and apply the subtractive form of Euclid's algorithm.

Example 2. *Let $k = 34$ and $g = 19$ and let apply them the subtractive form of Euclid's algorithm:*

$$\begin{aligned} 34 - 19 &= 15 \quad \textit{(big step)} \\ 19 - 15 &= 4 \quad \textit{(small step)} \\ 15 - 4 &= 11 \quad \textit{(small step)} \end{aligned}$$

$$\begin{aligned} 11-4 &= 7 \quad \textit{(big step)} \\ 7-4 &= 3 \quad \textit{(big step)} \\ 4-3 &= 1 \quad \textit{(small step)} \\ 3-1 &= 2 \\ 2-1 &= 1 \\ 1-1 &= 0 \end{aligned}$$

Reading the first number of each line gives the EAC $(1, 2, 3, 4, 7, 11, 15, 19, 34)$.

Finally, in order to simplify the writing of the algorithm, we will use the following notation : if $v = (1, 2, 3, v_4, \ldots, v_s)$ is an EAC then we only consider the chain from v_4 and we replace all the v_i's by 0 if it has been computed using a big step and by 1 for a small step.

For instance the sequence: $(1,\ 2,\ 3,\ 4,\ 7, 11, 15, 19, 34)$
will be written: $(1, 0,\ 0,\ \ 1,\ \ 1,\ 0)$.

Finally we note the chain $c = (c_4, \ldots, c_s)$ instead of v in order to prevent confusion between both representations.

We can now propose an algorithm performing a point scalar multiplication and using only the `NewADD` operator.

Algorithm 2. Euclid-Exp(c, P)

Data: P, $[2]P$ with $Z_P = Z_{[2]P}$ and an EAC $c = (c_4, \ldots, c_s)$ computing k ;
Result: $[k]P \in E$;
begin
  $(U_1, U_2) \leftarrow ([2]P, P)$
  for $i = 4 \ldots s$ **do**
    if $c_i = 0$ **then**
      $(U_1, U_2) \leftarrow \texttt{NewADD}(U_1, U_2)$;
    else
      $(U_1, U_2) \leftarrow \texttt{NewADD}(U_2, U_1)$;
    end
  end
  $(U_1, U_2) \leftarrow \texttt{NewADD}(U_1, U_2)$;
  return U_1
end

Example 3. *Let us see what happens with the chain* $c = (1, 0, 0, 1, 1, 0)$ *computing 34:*

$$\begin{aligned} & \textit{first we compute} && ([2]P, P) \\ c_4 = 1 \quad & \textit{so we compute} && \texttt{NewADD}(P, [2]P) = ([3]P, P) \\ c_5 = 0 \quad & \textit{so we compute} && \texttt{NewADD}([3]P, P) = ([4]P, [3]P) \\ c_6 = 0 \quad & \textit{so we compute} && \texttt{NewADD}([4]P, [3]P) = ([7]P, [4]P) \end{aligned}$$

$c_7 = 1$ *so we compute* $\texttt{NewADD}([4]P, [7]P) = ([11]P, [4]P)$
$c_8 = 1$ *so we compute* $\texttt{NewADD}([4]P, [11]P) = ([15]P, [4]P)$
$c_9 = 0$ *so we compute* $\texttt{NewADD}([15]P, [4]P) = ([19]P, [15]P)$
and finally we compute $\texttt{NewADD}([19]P, [15]P)$ *which gives* $[34]P$

If we consider that the point P is given in affine coordinate (that is $Z = 1$) then the doubling step can be performed using 3M and 3S and so, the total computational cost of our algorithm is $(5s - 7)$M and $(2s - 1)$S.

Remark 3. *Some cryptographic protocols only require the x-coordinate of the point $[k]P$. In this case it is possible to save one multiplication by step of Algorithm 2 by noticing that Z does not appear during the computation of X_3' and Y_3', thus it is not necessary to compute Z_3' during the process. Appendix A shows how to recover the x-coordinate in the end.*

4.2 About Euclid's Addition Chains Length

At this point we know that Euclidean addition chains are easy to compute, however finding small chains is a lot more complicated.

We begin by a theorem proved by D. Knuth and A. Yao in 1975 [8].

Theorem 1. *Let $S(k)$ denote the average number of steps to compute $gcd(k, g)$ using the subtractive Euclid's algorithm when g is uniformly distributed in the range $1 \leq g \leq k$. Then*

$$S(k) = 6\pi^{-2}(\ln k)^2 + O(\log k(\log\log k)^2)$$

This theorem shows that if, in order to find an EAC for an integer k, we choose an integer g at random, it will return a chain of length about $(\ln k)^2$, which is too long to be used with ECC. Indeed, for a 160-bit exponent, we will see in the last section that to be efficient, Algorithm 2 requires chains of length at most 320, whereas the previous theorem tells us that, theoretically, random chains have a length of 7000 on average (it is rather 2500 in practice).

A classic way to limit the length of EAC is to choose g close to $\frac{k}{\phi}$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden section. This guarantees that the last steps of the EAC will be big steps. In practice this method allows to find EAC of an average length of 1100.

A second obvious way to find shorter chains is to try many g around $\frac{k}{\phi}$ and to keep the shortest chain. A more precise study can be found in [12].

Considering 160-bit integers, finding EAC of length 320 can be done by checking (on average) about 30 $g's$. Finding shorter chains is a lot more difficult, as an example finding chains of length 270 requires testing more than 45 000 $g's$. Such a computation can not be integrated into any exponentiation algorithm so, if some offline computations cannot be performed, one should not expect to use EAC whose length is shorter than 320.

5 Using Zeckendorf Representation

We have seen that finding a small Euclidean addition chain that compute a large integer is quite difficult. However if the integer k is a Fibonacci number then an optimal chain is quite easy to compute. Indeed the Fibonacci sequence is an optimal chain. The idea proposed in this section is to switch from binary to the Zeckendorf representation in order to replace doublings by Fibonacci numbers computations.

5.1 A Fibonacci-and-Add Algorithm

Theorem 2. *Let k be an integer and $(F_i)_{i\geq 0}$ the Fibonacci sequence, then k can be uniquely written in the form:*

$$k = \sum_{i=2}^{l} d_i F_i,$$

with $d_i \in \{0,1\}$ and $d_i d_{i+1} = 0$

An integer k written in this form is said to be in Zeckendorf representation and will be denoted as $k = (d_{l-1}, \ldots, d_2)_{\mathcal{Z}}$. Such a representation is easy to compute as it can be obtained using a greedy algorithm. An equivalent of the double-and-add algorithm is proposed next.

Algorithm 3. Fibonacci-and-add(k, P)

```
Data: P ∈ E(K), k = (d_l, ..., d_2)_Z;
Result: [k]P ∈ E;
begin
    (U, V) ← (P, P)
    for i = l − 1 ... 2 do
        if d_i = 1 then
            U ← U + P          (add step);
        end
        (U, V) ← (U + V, U)    (Fibonacci step) ;
    end
    return U
end
```

Example 4. *Computation of [25]P with 25 = 21 + 3 + 1=* $(1000101)_{\mathcal{Z}}$*:*

- *initialization:* $(U, V) \leftarrow (P, P)$
- $d_7 = 0$ *:* $(U, V) \leftarrow ([2]P, P)$
- $d_6 = 0$ *:* $(U, V) \leftarrow ([3]P, [2]P)$
- $d_5 = 0$ *:* $(U, V) \leftarrow ([5]P, [3]P)$
- $d_4 = 1$ *:* $U \leftarrow [6]P$ *then* $(U, V) \leftarrow ([9]P, [6]P)$
- $d_3 = 0$ *:* $(U, V) \leftarrow ([15]P, [9]P)$
- $d_2 = 1$ *:* $U \leftarrow [16]P$ *then* $(U, V) \leftarrow ([25]P, [16]P)$
- *return* $U = [25]P$

The Zeckendorf representation needs 44% more digits in comparison with the binary method. For instance a 160-bit integer will require around 230 Fibonacci digits. However, the density of 1's in this representation is lower. From [2] we know that the density of 1's is about 0.2764. This means that representing a 160-bits integer requires, on average, 80 powers of 2 but only 64 Fibonacci numbers ($\simeq 230 \times 0.2764$).

More generally, for a n-bit integer, the classical double-and-add algorithm requires on average $1.5 \times n$ operations (n doublings and $\frac{n}{2}$ additions) and the Fibonacci-and-add requires $1.83 \times n$ operations ($1.44 \times n$ "Fibonacci" and $0.398 \times n$ additions). In other words the Fibonacci-and-add algorithms requires about 23% more operations.

5.2 Using `NewADD`

We want to adapt Algorithm 5.1 to elliptic curves using the `NewADD` operator. It is clear that, as long as U and V are two points sharing the same z-coordinate, the Fibonacci step just consists of one use of `NewADD`. This means that a sequence of 0's in the Zeckendorf representation of the k can be performed by a sequence of `NewADD`.

We need now to compute $U + P$ return $U + P$ and V with the same z-coordinate. Let us suppose that $U = (X_U, Y_U, Z)$, $V = (X_V, Y_V, Z)$ and $P = (x, y, 1)$. First we compute the point $P' = (xZ^2, yZ^3, Z) \sim P$ (3M+S) so that one can compute $U + P = (X_{U+P}, Y_{U+P}, Z_{U+P})$ using `NewADD` (5M+2S). Then on one hand we have $Z_{U+P} = (X_U - xZ^2)Z$. On the other hand $(X_U - xZ^2)^2$ and $(X_U - xZ^2)^3$ have been computed during the computation of $U + P$ (see the quantities A and $C - B$ in our formulae in section 3) so that updating the point V to $(X_V(X_U - xZ^2)^2, Y_V(X_U - xZ^2)^3, Z(X_U - xZ^2))$ requires only 2M.

As a conclusion the final computational cost of an add step is 10M+3S.

All this is summarized in the following algorithm:

Algorithm 4. Fibonacci-and-add(k, P)

```
Data: P ∈ E(K), k = (d_l, ..., d_2)_Z;
Result: [k]P ∈ E;
begin
    (U, V) ← (P, P)
    for i = l − 1 ... 2 do
        if d_i = 1 then
            update P;
            (U, .) ← NewADD(U, P) ;
            update V;
        end
        (U, V) ←NewADD (U, V) ;
    end
    return U
end
```

We have seen that this algorithm is expected to perform $1.44 \times n$ Fibonacci steps and $0.398 \times n$ add steps (where n is the bit length of k). Then the average complexity of this algorithm is $(11.18 \times n)$M + $(4.07 \times n)$S.

5.3 Improvements

As with the binary case, it is possible to modify the Zeckendorf representation to reduce the number add step. As an example one can use a signed version of the Zeckendorf representation. In this case the density of 1's decreases to 0.2, which means that for an n-bit integer, the number of 1's is reduced to $0.29 \times n$.

If some extra memory is available (and with minor modifications of Algorithm 5.1) one can use some kind of window methods. For instance, one can modify the Zeckendorf representation using the following properties:

- $F_{n+3} + F_n = 2F_{n+2} \rightarrow 1001_{\mathcal{Z}} = 0200_{\mathcal{Z}}$
- $F_{n+3} - F_n = 2F_{n+1} \rightarrow 100\overline{1}_{\mathcal{Z}} = 0020_{\mathcal{Z}}$
- $F_{n+4} + F_n = 3F_{n+2} \rightarrow 10001_{\mathcal{Z}} = 00300_{\mathcal{Z}}$
- $F_{n+6} - F_n = 4F_{n+3} \rightarrow 1000001_{\mathcal{Z}} = 0004000_{\mathcal{Z}}$

Experiments seem to show that using these recoding rules allows to reduce the density of non zero digits to 0.135 so that the number of expected add steps in Algorithm 5.1 is reduced to $0.194 \times n$.

Remark 4. *Of course it is possible to find many more properties in the huge literature dedicated to Fibonacci numbers, however the four rules given previously are sufficient when dealing with 160-bit integers.*

6 Comparisons with Other Methods

In this section we give some practical results about the complexities of our point multiplication algorithms and compare them with other classical methods. More precisely in Table 1 we compare our formulae used with Euclidean addition chains to Montgomery's ladder and Euclidean chains on Montgomery's curves, and in Table 2 we compare our Fibonacci number based algorithm (and its improved version) to double-and-add, NAF and 4-NAF methods on general curves using mixed coordinates.

We assume that S=0.8M, that k is a 160-bit integer and refer to [4] for the complexity of the window method using mixed coordinate.

In Table 1 we can see that our new formulae allow to generalize the use of Euclidean chains without loss of efficiency. Moreover one can compute both the x and y-coordinates (with a little efficiency loss) which is not possible with Montgomery's formulae. However Montgomery's ladder still remains a lot more efficient than any methods.

Comparing similar algorithms in Table 2 shows that Fibonacci based algorithms are still slower than their binary equivalents. From 10 to 23 % slower for simple to window Fibonacci-and-add. However this has to be balanced by the fact that those algorithms naturally require a lot more operations than the

Table 1. Comparisons with algorithms on Montgomery curves

Algorithm	Curve type	recovery of y-coord.	Field Mult.
Mont. ladder	Montgomery	yes	1463
EAC-320	Montgomery	no	1792
EAC-270	Montgomery	no	1512
EAC-320	Weiestraß	yes	2112
EAC-270	Weiestraß	yes	1782
EAC-320	Weiestraß	no	1792
EAC-270	Weiestraß	no	1512

Table 2. Comparisons between binary and Fibonacci based algorithm

Algorithm	Coord.	Field Mult.
Double-and-add	Mixed	2104
NAF	Mixed	1780
4-NAF	Mixed	1600
Fibonacci-and-add	`NewADD`	2311
Signed Fib-and-add	`NewADD`	2088
Window Fib-and-add	`NewADD`	1960

binary ones. From 23 % more for the Fibonacci-and-add to 36% for the window version. So we can see that our formulae significantly reduce the additional computation cost of our Fibonacci based algorithms making then almost as efficient as the binary ones.

7 Summary

In this paper we have proposed new point addition formulae with a lower computational cost than the best known doubling. We have shown that these formulae are really well suited to a special type of addition chains: the Euclidean addition chains. Our formulae allow us to generalize the use of those chains to any elliptic curve without loss of efficiency, compared to Montgomery's formulae. In addition we have proposed a Fibonacci number based point scalar multiplication algorithm. In practice it requires a lot more operations than its binary counterpart, but coupled with our formulae the former becomes almost as efficient as the latter (the additional cost is reduced from 23 % to 10 %).

References

1. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
2. Capocelli, R.M.: A generalization of fibonacci trees. In: Third In. Conf. on Fibonacci Numbers and their Applications (1988)

3. Cohen, H., Frey, G. (eds.): Handbook of Elliptic and Hyperelliptic Cryptography. Chapman & Hall, Sydney, Australia (2006)
4. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, Springer, Heidelberg (1998)
5. Doche, C., Imbert, L.: Extended double-base number system with applications to elliptic curve cryptography. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 335–348. Springer, Heidelberg (2006)
6. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2004)
7. Knudsen, E.W.: Elliptic scalar multiplication using point halving. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 135–149. Springer, Heidelberg (1999)
8. Knuth, D., Yao, A.: Analysis of the subtractive algorithm for greater common divisors. Proc. Nat. Acad. Sci. USA 72(12), 4720–4722 (1975)
9. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48, 203–209 (1987)
10. Lopez, J., Dahab, R.: Fast multiplication on elliptic curves over GF (2 m) without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
11. Miller, V.S.: Uses of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–428. Springer, Heidelberg (1986)
12. Montgomery, P.: Evaluating Recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via Lucas chains (1983), Available at `ftp.cwi.nl:/pub/pmontgom/Lucas.ps.gz`
13. Montgomery, P.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation 48, 243–264 (1987)
14. Solinas, J.A.: Improved algorithms for arithmetic on anomalous binary curves. Technical report, University of Waterloo (1999), http://www.cacr.math.uwaterloo.ca/techreports/1999/corr99-46.pdf

A Recovery of x-Coordinate

As said in section 3 the x-coordinate of the sum of two points P_1 and P_2 can be recovered without computing the z coordinate. Or in other word the value

$P_1 + P_2 = (X_{P_1+P_2}, Y_{P_1+P_2}, Z_{P_1+P_2})$ can be recovered thanks to the the following property:

Property 1. *Let $P_1 = (X_1, Y_1, Z)$, $P_2 = (X_2, Y_2, Z)$ and $P_1 + P_2 = (X_3, Y_3, Z_3)$ be points of an elliptic curve E given in Jacobian coordinates, then*

$$Z^2 = \frac{a}{2b}\left[\frac{(X_1 - X_2)(X_3 + 2Y_2Y_1 - X_1X_2(X_1 + X_2))}{Y_1^2 - Y_2^2 + X_2^3 - X_1^3} - (X_1 + X_2)\right]$$

Proof: P_1 and P_2 satisfy $Y^2 = X^3 + aXZ^4 + bZ^6$ so

$$Y_1^2 - Y_2^2 = X_1^3 - X_2^3 + aX_1Z^4 - aX_2Z^4 + bZ^6 - bZ^6$$

which gives

$$Z^4 = \frac{Y_1^2 - Y_2^2 + X_2^3 - X_1^3}{a(X_1 - X_2)}$$

Moreover

$$\begin{aligned} X_3 &= (Y_2 - Y_1)^2 - (X_1 + X_2)(X_2 - X_1)^2 \\ &= Y_2^2 - 2Y_2Y_1 + Y_1^2 - X_2^3 + X_2^2X_1 + X_1^2X_2 - X_1^3 \\ &= Y_2^2 - X_2^3 + Y_1^2 - X_1^3 - 2Y_2Y_1 + X_1X_2(X_1 + X_2) \\ &= aX_1Z^4 + bZ^6 + aX_2Z^4 + bZ^6 - 2Y_2Y_1 + X_1X_2(X_1 + X_2) \\ &= Z^4(a(X_1 + X_2) + 2bZ^2) - 2Y_2Y_1 + X_1X_2(X_1 + X_2) \end{aligned}$$

and so

$$Z^2 = \frac{a}{2b}\left[\frac{(X_1 - X_2)(X_3 + 2Y_2Y_1 - X_1X_2(X_1 + X_2))}{Y_1^2 - Y_2^2 + X_2^3 - X_1^3} - (X_1 + X_2)\right]$$

Recovering the final x-coordinate can be done in 8M, 4S and one inversion.