Elliptic Curve Scalar Multiplication Combining Yao's Algorithm and Double Bases

Nicolas Méloni and M. Anwar Hasan

Department of Electrical and Computer Engineering
University of Waterloo

Abstract. In this paper we propose to take one step back in the use of double base number systems for elliptic curve point scalar multiplication. Using a modified version of Yao's algorithm, we go back from the popular double base chain representation to a more general double base system. Instead of representing an integer k as $\sum_{i=1}^{n} 2^{b_i}3^{t_i}$ where (b_i) and (t_i) are two decreasing sequences, we only set a maximum value for both of them. Then, we analyze the efficiency of our new method using different bases and optimal parameters. In particular, we propose for the first time a binary/Zeckendorf representation for integers, providing interesting results. Finally, we provide a comprehensive comparison to state-of-the-art methods, including a large variety of curve shapes and latest point addition formulae speed-ups.

Keywords: Double-base number system, Zeckendorf representation, elliptic curve, point scalar multiplication, Yao's algorithm.

1 Introduction

In order to compute elliptic curve point multiplication, that is to say kP where P is a point on an elliptic curve, defined over a prime field, and k is an integer, a lot of effort has been made to adapt and optimize generic exponentiation methods (such as Non-adjacent form (NAF), window NAF and fractional window NAF). In 1995, Dimitrov and Cooklev [8] have introduced the use the double base number system (DBNS) to improve modular exponentiation speed. The idea is to represent k as a sum of terms of the form $c_i 2^{b_i}3^{t_i}$ with $c_i = 1$ or -1. The main advantage of this representation is the fewer number of terms it requires. A very interesting case is when the base element x is fixed, so that one can precompute all the $x^{2^{b_i}3^{t_i}} \mod p$. The DBNS seems to be not that efficient in the case of a randomly chosen element. In order to overcome this problem and adapt the DBNS to elliptic curve point multiplication, Dimitrov, Imbert and Mishra have introduced the concept of double base chains, where the integer k is still represented as a sum of $c_i 2^{b_i}3^{t_i}$ but with the restriction that (b_i) and (t_i) must be two decreasing sequences [9]. The restriction causes the number of terms to increase, but allows to perform the scalar multiplication using a Horner like scheme. Allowing c_i to belong to a larger set than $\{-1, 1\}$ as well as choosing optimal parameters based on the ratio of the number of doublings to that of triplings also helped to achieve better results.

The original double base representation has probably not been utilized as much as it should have been for developing improved exponentiation algorithms. To the end, our contribution is to show that the use of a modified version of Yao's algorithm allows to partly overcome the drawbacks of the DBNS. By imposing a maximum bound on b_i's and t_i's, that is clearly less restrictive than the double base chain condition, we show that our method provides significant improvement even when compared to the most recently optimized double base methods. Moreover, we introduce a binary/Zeckendorf method which, on the classical Weierstrass curve, provides similar results.

2 Background

In this section, we give a brief review of the materials used in the paper.

2.1 Elliptic curves

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 this paper, we only deal with curves defined over a prime finite field ($K = \mathbb{F}_p$) of characteristic greater than 3. In this case, the equation can be simplified to

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

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$. Points are affine points (x, y) satisfying the curve equation and a point at infinity. The set of points $E(K)$ defined over K forms an abelian group. There exist explicit formulae to compute the sum of two points that involves field inversions. When the field inversion operation is considerably costlier than a field multiplication, one usually uses a projective version of the above equation. In this case, a point is represented by three, or more, coordinates, and many such projective coordinate systems have been proposed to speed up elliptic curve group operations. For a complete overview of those coordinates, one can refer to [7, 14].

Another feature of the elliptic curve group law is that it allows fast composite operations as well as different type of additions. To take full advantage of our point scalar multiplication method, and in addition to the classical addition (**ADD**) and doubling (**DBL**) operations, we consider the following operations:

- **tripling (TPL)**: point tripling
- **readdition (reADD)**: addition of a point that has been added before to another point
- **mixed addition (mADD)**: addition of a point in affine coordinate (i.e. $Z = 1$) to another point

In addition to those coordinate systems and composite operations, many curve shapes have been proposed to improve group operation formulae. In this paper, we will consider a variety of curve shapes including:

- tripling oriented Doche-Icart-Kohel curves (3DIK) [10]
- Edwards curves (Edwards) [13, 3] with inverted coordinates [4]
- Hessian curves [6, 15, 16]
- Extended Jacobi Quartics (ExtJQuartic) [6, 12, 15]
- Jacobi intersections (JacIntersect) [6, 17]
- Jacobian coordinates (Jacobian) with the special case $a_4 = -3$ (Jacobian-3).

Table 1 summarize the cost of those operations on all the considered curves.

Curve shape	DBL	TPL	ADD	reADD	mADD
3DIK	2M+7S	6M+6S	11M+6S	10M+6S	7M+4S
Edwards	3M+4S	9M+4S	10M+1S	10M+1S	9M+1S
ExtJQuartic	2M+5S	8M+4S	7M+4S	7M+3S	6M+3S
Hessian	3M+6S	8M+6S	6M+6S	6M+6S	5M+6S
InvEdwards	3M+4S	9M+4S	9M+1S	9M+1S	8M+1S
JacIntersect	2M+5S	6M+10S	11M+1S	11M+1S	10M+1S
Jacobian	1M+8S	5M+10S	11M+5S	10M+4S	7M+4S
Jacobian-3	3M+5S	7M+7S	11M+5S	10M+4S	7M+4S

(1) proposed in this work

Table 1. Elliptic curve operations cost.

Finally, some more optimizations can be found in [21, 19] for the quintupling formulae. In section 4, we use the specific formulae from [20] using the z-coordinate trick to compute Fibonacci number point multiples. One can also refer to [2] for an extensive overview of different formulae, coordinates systems, curve shapes and their latest updates.

2.2 Double base number system

Let k be an integer. As mentioned earlier, one can represent k as the sum of terms of the form $c_i 2^{b_i} 3^{t_i}$, where $c_i \in \{-1, 1\}$. Such a representation always exists. In fact, this number system is quite redundant. One of the most interesting properties is that, among all the possible representations for a given integer, some of them are really sparse, that is to say that the number of non-zero terms is quite low.

To compute DBNS representation of an integer, one usually use a greedy algorithm. It consists of the following: find the closest integer of the form $2^{b_i} 3^{t_i}$ to k, subtract it from k and repeat the process with $k' = k - 2^{b_i} 3^{t_i}$ until it is equal to zero.

Performing a point scalar multiplication using this number system is relatively easy. Letting k be equal to $\sum_{i=1}^{n} c_i 2^{b_i} 3^{t_i}$, one just needs to compute $[c_i 2^{b_i} 3^{t_i}]P$ for $i = 1$ to

n and then add all the points. If the number of additions is indeed quite low, in practice such a method requires too many doublings and triplings. That is why the general DBNS representation has been considered to be not suitable for point scalar multiplication.

To overcome this problem, Dimitrov, Imbert, and Mishra [9] have introduced the concept of double-base chains. In this system, k is still represented as $\sum_{i=1}^{n} c_i 2^{b_i} 3^{t_i}$, but with the restriction that (b_i) and (t_i) must be two decreasing sequences, allowing a Horner-like evaluation of kP using only b_1 doublings and t_1 triplings. Computing such a representation can be done using Algorithm 1. The main drawback of this method is that it significantly increases the number of point additions.

Algorithm 1 Computing a double-base chain computing k

Input: $k \geq 0$
Output: $k = \sum_{i=1}^{n} s_i 2^{b_i} 3^{t_i}$ with $(b_i, t_i) \searrow$

while $k \neq 0$ **do**
  $s = 1$
  Find the best default approximation of k of the form $z = 2^b 3^t$ with $b \leq b_{max}$ and $t \leq t_{max}$
  Print(s, b, t)
  $b_{max} = b; t_{max} = t$
  if $k < z$ then $s = -s$
  $k = |k - z|$
end while

Some improvements have been proposed by applying various modifications including the possibility for c_i to be chosen in a larger set than $\{-1, 1\}$ [11], the use of multiple bases [21], etc. One can finally refer to [1] for a view of the latest optimizations.

3 Modified Yao's Algorithm

3.1 Yao's algorithm

Published in 1976 [22], Yao's algorithm can be seen as the right-to-left counterpart of the classical Brauer algorithm. Let $k = k_{l-1}2^{l-1} + \cdots + k_1 2 + k_0$ with $k_i \in \{0, 1, \ldots, 2^w - 1\}$, for some w. The algorithm first computes $2^i P$ for all i lower than $l - 1$ by successive doublings. Then it computes $d(1)P, \ldots, d(2^w - 1)P$, where $d(j)$ is the sum of the 2^i such that $k_i = j$. Said differently, it mainly consists in considering the integer k as

$$1 \times \underbrace{\sum_{k_i=1} 2^i}_{d(1)} + 2 \times \underbrace{\sum_{k_i=2} 2^i}_{d(2)} + \cdots + (2^w - 1) \times \underbrace{\sum_{k_i=2^w-1} 2^i}_{d(2^w-1)} .$$

We can see that $d(1)$ is the sum of all the powers of 2 associated to digit 1, $d(2)$ is the sum of all the powers of 2 associated to digit 2 etc. Finally kP is obtained as $d(1)P +$

$2d(2)P+\cdots+(2^w-1)d(2^w-1)P$. In order to save some group operations, it is usually computed as $d(2^w-1)P+(d(2^w-1)P+d(2^w-2)P)+\cdots+(d(2^w-1)P+\cdots+d(1)P)$.

Example 1. Let $k = 314159$. We have $\mathrm{NAF}_3(\mathrm{k}) = 100\,0300\,1003\,0000\,5007$, $l = 19$ and $2^w - 1 = 7$. One can compute kP in the following way:

- consider k as $1 \times (2^{18} + 2^{11}) + 3 \times (2^{14} + 2^8) + 5 \times 2^3 + 7 \times 2^0$
- compute $P, 2P, 4P, \ldots 2^{18}P$
- $d(1)P = 2^{18}P + 2^{11}P$, $d(3)P = 2^{14}P + 2^8P$, $d(5)P = 2^3P$, $d(7)P = P$
- $kP = 2(d(7)P) + 2(d(7)P + d(5)P) + 2(d(7)P + d(5)P + d(3)P) + d(7)P + d(5)P + d(3)P + d(1)P = 7d(7)P + 5d(5)P + 3d(3)P + d(1)P$

In this example, we have:

$$\begin{aligned} d(1) &= 100\,0000\,1000\,0000\,0000 \\ d(3) &= 000\,0100\,0001\,0000\,0000 \\ d(5) &= 000\,0000\,0000\,0000\,1000 \\ d(7) &= 000\,0000\,0000\,0000\,0001 \\ k &= 100\,0300\,1003\,0000\,5007 \\ &= 7d(7) + 5d(5) + 3d(3) + d(1) \end{aligned}$$

3.2 Modified Yao's algorithm

Now, we adapt the preceding algorithm in order to take advantage of the DBNS representation. To do so, let us consider k in (one of) its DBNS form: $k = 2^{b_n}3^{t_n} + \cdots + 2^{b_1}3^{t_1}$. As in Yao's original algorithm, we first compute 2^iP for all i lower than $\mathtt{max}(b_j)$. Then, for all j lower than $\mathtt{max}(t_i)$, we define $d(j)P$ as the sum of all the $2^{b_i}P$ such that $t_i = j$. Finally we have $kP = d(0)P + 3d(1)P + \ldots 3^{\mathtt{max}(t_i)}d(\mathtt{max}(t_i))P$.

Example 2. Let $k = 314159$. One of the representations of k in the DBNS is

$$2^{10}3^5 + 2^83^5 + 2^{10}3 + 2^23^2 + 3^2 + 2,$$

$\mathtt{max}(a_i) = 10$ and $\mathtt{max}(b_i) = 5$. One can compute kP in the following way:

- compute $P, 2P, 2^2P, \ldots, 2^{10}P$
- $d(0)P = 2P$, $d(1)P = 2^{10}P$, $d(2)P = 2^2P + P$, $d(5) = 2^{10}P + 2^8P$
- $kP = 3(3(3^3d(5)P+d(2)P)+d(1)P)+d(0)P = 3^5d(5)P+3^2d(2)P+3d(1)P+d(0)P$

We can see that the number of operations is $\mathtt{max}(b_i)$ doublings, $\mathtt{max}(t_i)$ triplings and $n-1$ additions. With our modified algorithm, we obtain the same complexity as the double-base chain method. However, in our case, the numbers of doublings and triplings are independent, which means that $2^{\mathtt{max}(b_i)}3^{\mathtt{max}(t_i)}$ can be quite larger than k. It can be seen as a waste of operations, as we could expect it to just as large as k . In order to reduce this additional cost, we simply propose to use a maximum bound for both the b_i's and the t_i's so that $2^{max(b_i)}3^{max(t_i)} \sim k$.

4 Extending the modified Yao's Algorithm

We have seen how Yao's algorithm can be adapted to the double-base number system. In this section, we generalize our approach to different number systems via an extended version of Yao's algorithm.

4.1 Generalization of Yao's algorithm

We have seen that Yao's algorithm can be efficiently adapted to the double-base number system. We can now derive a general form of Yao's algorithm based on any number system using two sets of integers.

Let $\mathcal{A} = \{a_1, \ldots, a_r\}$ and $\mathcal{B} = \{b_1, \ldots, b_t\}$ be two sets of integers. Let k be an integer that can be written as $\sum_{i=1}^{n} a_{f(i)} b_{g(i)}$ with $f : \{1, \ldots n\} \rightarrow \{1, \ldots r\}$ and $g : \{1, \ldots n\} \rightarrow \{1, \ldots t\}$. It is possible to use a generalized version of Yao's algorithm to compute kP. To do so, we first compute the $b_i P$'s, for $i = 1 \ldots t$. Then, for $j = 1 \ldots r$, we compute $d(j)P$ as the sum of all the $b_{g(i)}P$ such that $f(i) = j$. In other terms, $d(1)P$ will be the sum of all the $b_{g(i)}P$ associated to a_1, $d(2)P$ will be the sum of all the $b_{g(i)}P$ associated to a_2 etc. Finally, $kP = a_1 d(1)P + a_2 d(2)P + \cdots + a_n d(n)P$.

It is easy to see that with a proper choice of sets, we find again the previous forms of the algorithm. The original version is associated to the sets $\mathcal{A} = \{1, 2, \ldots, 2^n\}$ and $\mathcal{B} = \{1, 3, 5, \ldots, 2^w - 1\}$ and the double-base version to $\mathcal{A} = \{1, 2, \ldots, 2^{b_{max}}\}$ $\mathcal{B} = \{1, 3, \ldots, 3^{t_{max}}\}$. We also remark that both sets can contain negative integers. As the operation $P \rightarrow -P$ is almost free on elliptic curves, we always consider signed representation in our experiments.

The aim of the following subsections is to present a different set of integers to improve the efficiency of our method.

4.2 Double-base system using Zeckendorf representation

Let $(F_n)_{n \geq 0}$ be the Fibonacci sequence defined as $F_0 = 0, F_1 = 1, \forall n \geq 0, F_{n+2} = F_{n+1} + F_n$. Any integer can be represented as a finite sum of Fibonacci numbers [23]. Just like in the case of the classical double-base system, we introduce a mixed binary-Zeckendorf number system (BZNS). It simply consists of representing an integer k as $2^{b_n} F_{Z_n} + \cdots + 2^{b_1} F_{Z_1}$. Computing such a representation can be done using the same kind of greedy algorithm as with the classical DBNS.

Remark 1. The choice of such a representation is not arbitrary. It is based on the fact that on elliptic curves in Weierstraßform, the sequence $F_2P, \ldots F_nP$ can be efficiently computed thanks to the formulae proposed in [20]. In that case, each point addition is performed faster than a doubling.

We now apply our generalized Yao's algorithm to the sets $\{F_2, \ldots, F_{Z_{max}}\}$ and $\{1, 2, \ldots, 2^{b_{max}}\}$. In this case, we first compute F_iP for all i lower than Z_{max}, by consecutive additions. Then, for all j lower than $\mathtt{max}(b_i)$, we define $d(j)P$ as the sum of all the $F_{Z_i}P$ such that $b_i = j$. Finally we have $kP = d(0)P + 2d(1)P + \ldots 2^{\mathtt{max}(b_i)} d(\mathtt{max}(b_i))P$.

Example 3. Let $k = 314159$. One of the representations of k in the BZNS is

$$2^8 F_{16} + 2^8 F_{13} + 2^5 F_{10} + 2F_9 + 2F_5 + F_2$$

$\mathtt{max}(b_i) = 8$ and $\mathtt{max}(Z_i) = 16$. One can compute kP in the following way:

- consider k as $2^8(F_{16} + F_{13}) + 2^5 F_{10} + 2(F_9 + F_5) + F_2$
- compute $P, 2P, 3P, \dots, F_{16}P$
- $d(0)P = F_2P$, $d(1)P = F_9P + F_5P$, $d(5)P = F_{10}P$, $d(8) = F_{16}P + F_{13}P$
- $kP = 2(2^4(2^3 d(8)P + d(5)P) + d(1)P) + d(0)P = 2^8 d(8)P + 2^5 d(5)P + 2d(1)P + d(0)P$

5 ECC implementation and comparisons

In this section, we provide a comprehensive comparison between our different versions of Yao's algorithm and the most recent double-base chain methods.

5.1 Caching strategies

Caching intermediate results while computing an elliptic curve group operation is one very important optimization criteria. In this subsection, we show that the use of our generalized algorithm allows some savings that cannot be done with the traditional methods. To better clarify this point, we fully detail our caching strategy for curves in Weierstraßform using jacobian coordinates with parameter $a = 3$ (Jac-3). Similar methods are applicable to all the different curve types.

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

$$A = X_1 Z_1^2, \quad B = X_2 Z_1^2, \; C = Y_1 Z_2^3, \; D = Y_2 Z_1^3, \; E = B - A,$$
$$F = 2(D - C), G = (2E)^2, H = E \times G, I = A \times G,$$

and

$$X_3 = F^2 - H - 2I \; Y_3 = F(F - X_3) - 2CH, Z_3 = ((Z_1 + Z_2)^2 - Z_1^2 - Z_2^2)E$$

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

$$A = X_1 Y_1^2, B = 3(X_1 - Z_1)^2 (X_1 + Z_1)^2$$

and

$$X_3 = B^2 - 8A, Y_3 = -8Y_1^4 + B(4A - X_3), Z_3 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2.$$

One can verify that these two operations can be computed using 11M+5S and 3M+5S respectively. It has been shown that some of the intermediate results can be reused under particular circumstances. More precisely, if a point $P = (X_1, Y_1, Z_1)$ is added to any other point, it is possible to store the data Z_1^2 and Z_1^3. During the same scalar multiplication, if the point P is added again to another point, reusing those stored values saves 1M+1S. This is what is usually called a readdition and its cost is 10M+4S instead of 11M+5S. With mixed and, of course, the general addition (one of the added points has its z-coordinate equal to 1), this is the only kind of point additions that can occur in all the traditional scalar multiplication methods.

Our new method allows more variety in caching strategies and point addition situations. From the doubling formulae, we can see that if we store Z_1^2 after the doubling of P and if we have to add P to another point, reusing Z_1^2 saves 1S. Adding a point that has already been doubled will be called $dADD$.

We now apply this to our scalar multiplication algorithm. We first compute the sequence $P \to 2P \to \cdots \to 2^{b_{max}}P$. For each doubled point (i.e. $P \to 2P \to \cdots \to 2^{b_{max}-1}P$) , it is possible to store Z^2. Different situations can now occur:

- **addition after doubling (dADD)**: addition of a point that has already been doubled before
- **double addition after doubling (2dADD)**: addition of two points that have already been doubled before
- **addition after doubling + readdition (dreADD)**: addition of a point that has already been doubled before to a point that has been added before
- **double readition (2reADD)**: addition of two points that has been added before
- **addition after doubling + mixed addition dmADD**: addition of a point that has already been doubled before to a point in affine coordinate (i.e. $Z = 1$)
- **mixed readdition (mreADD)**: addition of a point in affine coordinate (i.e. $Z = 1$) to a point that has been added before

Remark 2. It is also possible to cache Z^2 after a tripling. Adding a point that has already been tripled has the same cost as that has been after a doubling. Thus, we will still call this operation $dADD$.

In Table 2 we summarize the costs of the different operations for each considered curve.

Curve shape	dADD	2dADD	dreADD	2reADD	dmADD	mreADD
3DIK	11M+6S	11M+6S	10M+6S	9M+6S	7M+4S	6M+4S
Edwards	10M+1S	10M+1S	10M+1S	10M+1S	9M+1S	9M+1S
ExtJQuartic	7M+3S	7M+2S	7M+2S	7M+2S	6M+2S	6M+2S
Hessian	6M+6S	6M+6S	6M+6M	6M+6S	5M+6S	5M+6S
InvEdwards	9M+1S	9M+1S	9M+1S	9M+1S	8M+1S	8M+1S
JacIntersect	11M+1S	11M+1S	11M+1S	11M+1S	10M+1S	10M+1S
Jacobian	11M+4S	10M+4S	10M+3S	9M+3S	7M+3S	6M+3S
Jacobian-3	11M+4S	10M+4S	10M+3S	9M+3S	7M+3S	6M+3S

Table 2. New elliptic curve operations cost

5.2 Implementations and results

We have carried out experiments on 160-bit and 256-bit scalars over all the elliptic curves mentioned in section 2.1 and all values of b_{max}, t_{max} and Z_{max} such that $2^{b_{max}}3^{t_{max}}$ and $2^{b_{max}}F_{Z_{max}}$ are 160-bit or 256-bit integers. For each curve and each set of parameters, we have:

- generated 10000 pseudo random integers in $\{0, \ldots, 2^m - 1\} m = 160, 256$,
- converted each integer into the DBNS/BZNS systems using the corresponding parameters,
- counted all the operations involved in the point scalar multiplication process.

In Tables 3 and 4, we report the best results obtained for each case, with the best choice of parameters. Results are given in number of base field multiplications. To do so and in order to ease the comparison with previous works, we assume that $S = 0.8M$. However, different ratios could give slightly different results.

As the efficiency of any method is directly dependent on that of the curve operations, in appendix, we give in Tables 5, 6, 7 and 8 the curve operation count of our methods, in order to ease comparisons with future works that might use improved formulae. However, one has to be aware that those operation counts are only valid for the parameters they correspond to. A significant improvement of any of those curve operations may significantly change the optimal parameters for a given method.

As shown in Tables 3 and 4, our new method is very efficient compared to previously reported optimized double-base chains approaches [1] or optimized w-NAF methods [5], whatever the curve is. We obtain particularly good results on extended Jacobi Quartics, with which we improve the best results found in the literature, even taking into account the recent multi-base chains (or (2,3,5)NAF) [18]. However, one should note that part of those improvements are due to the fact that [1] and [5] uses older formulae for Extended JQuartic and Hessian curves. As an example, doubling is performed using 3M+4S instead of the actual 2M+5S. This saves 0.2M per doublings, that is to say around 32M (160×0.2M) for 160-bit scalars.

We can also see the interest of our new binary/Zeckendorf number system, for each curve where Euclidean additions are fast, it gives similar results as the classical double-base number system. It could be really interesting to generalize this number system to other curves and find more specific optimizations.

Finally, growing in size makes our algorithm even more advantageous, for every curves. Considering the (2,3,5)NAF method, no data are given for 256-bit scalars in the original paper. Due to lack of time, we have not been able to implement by ourselves this algorithm but we expect a similar behavior.

Curve shape	Method	b_{max}	t_{max}	Z_{max}	# multiplications
3DIK	4-NAF	-	-	-	1645.8
	DB chain	80	51	-	1502.4
	Yao-DBNS	44	74	-	1477.3
Edwards	4-NAF	-	-	-	1321.6
	DB chain	156	3	-	1322.9
	Yao-DBNS	140	13	-	1283.3
ExtJQuartic	4-NAF	-	-	-	1308.5
	DB chain	156	3	-	1311.0
	(2,3,5)NAF	131	12	-	1226.0
	Yao-DBNS	140	13	-	1210.9
Hessian	4-NAF	-	-	-	1601.9
	DB chain	100	38	-	1565.0
	Yao-DBNS	113	30	-	1501.8
InvEdwards	4-NAF	-	-	-	1287.8
	DB chain	156	3	-	1290.3
	(2,3,5)NAF	142	9	-	1273.8
	Yao-DBNS	140	13	-	1258.6
JacIntersect	4-NAF	-	-	-	1389.4
	DB chain	150	7	-	1438.8
	Yao-DBNS	143	11	-	1301.2
Jacobian	4-NAF	-	-	-	1573.8
	DB chain	100	38	-	1558.4
	Yao-DBNS	131	19	-	1534.9
	Yao-BZNS	142	-	28	1534.8
Jacobian-3	4-NAF	-	-	-	1511.9
	DB chain	100	38	-	1504.3
	(2,3,5)NAF	131	12	-	1426.8
	Yao-DBNS	131	19	-	1475.3
	Yao-BZNS	142	-	28	1476.9

Table 3. Optimal parameters and operation count for 160-bit scalars

Curve shape	Method	b_{max}	t_{max}	Z_{max}	# multiplications
3DIK	4-NAF	-	-	-	2603.3
	DB chain	130	80	-	2393.2
	Yao-DBNS	63	122	-	2319.2
Edwards	4-NAF	-	-	-	2088.5
	DB chain	252	3	-	2089.7
	Yao-DBNS	220	23	-	2029.8
ExtJQuartic	4-NAF	-	-	-	2068.9
	DB chain	253	2	-	2071.2
	Yao-DBNS	215	26	-	1911.4
Hessian	4-NAF	-	-	-	2542.4
	DB chain	150	67	-	2470.6
	Yao-DBNS	185	45	-	2374.0
InvEdwards	4-NAF	-	-	-	2038.7
	DB chain	252	3	-	2041.2
	Yao-DBNS	220	23	-	1993.3
JacIntersect	4-NAF	-	-	-	2185.4
	DB chain	246	7	-	2266.1
	Yao-DBNS	236	13	-	2050.5
Jacobian	4-NAF	-	-	-	2492.1
	DB chain	160	61	-	2466.2
	Yao-DBNS	185	45	-	2416.2
	Yao-BZNS	227	-	44	2419.8
Jacobian-3	4-NAF	-	-	-	2391.8
	DB chain	160	61	-	2379.0
	Yao-DBNS	185	45	-	2316.2
	Yao-BZNS	22	-	44	2329.2

Table 4. Optimal parameters and operation count for 256-bit scalars

6 Conclusions

In this paper we have proposed an efficient generalized version of Yao's algorithm, less restrictive than the double-base chain method, to perform the point scalar multiplication on elliptic curves defined over prime fields. The main advantage of this representation is that it takes advantage of the natural sparseness of the double-base number system without any additional and unnecessary computations. In the end, our method performs faster than all the previous double-base chains methods, over all types of curves. On the extended Jacobi Quartics, it also provides the best result found literature, faster than the (2,3,5)NAF, recently claimed as the fastest scalar multiplication algorithm. Finally we have proposed a new number system, mixing binary and Zeckendorf representation. On curves providing fast Euclidean addition, the BZNS provides very good results.

Acknowledgment. This work was supported in part by NSERC grants awarded to Dr. Hasan. We also are very grateful to the anonymous referees of CHES 2009 for their invaluable comments.

References

1. D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. Optimizing double-base elliptic-curve single-scalar multiplication. In *Progress in Cryptology INDOCRYPT 2007*, volume 4859/2007 of *LNCS*, pages 167–182. Springer Berlin / Heidelberg, 2007.
2. D. J. Bernstein and T. Lange. Explicit-formulas database. Available at http://hyperelliptic.org/EFD.
3. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Asiacrypt*, volume 4833/2008 of *LNCS*, page 2950, 2007.
4. D. J. Bernstein and T. Lange. Inverted Edwards coordinates. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 4851 of *LNCS*, pages 20–27. Springer Berlin / Heidelberg, 2007.
5. D. J. Bernstein and T. Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In *Finite fields and applications: proceedings of Fq8*, pages 1–19, 2008.
6. D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Adv. Appl. Math.*, 7(4):385–434, 1986.
7. H. Cohen and G. Frey, editors. *Handbook of Elliptic and Hyperelliptic Cryptography*. Chapman & Hall, 2006.
8. V. Dimitrov and T. Cooklev. Two algorithms for modular exponentiation using nonstandard arithmetics. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 78(1):82–87, 1995.
9. V. Dimitrov, L. Imbert, and P. K. Mishra. Efficient and secure elliptic curve point multiplication using double-base chains. In *ASIACRYPT*, volume 3788/2005 of *LNCS*, pages 59–78, 2005.
10. C. Doche, T. Icart, and D. R. Kohel. Efficient scalar multiplication by isogeny decompositions. In *Public Key Cryptography*, volume 3958 of *LNCS*, pages 191–206. Springer Berlin / Heidelberg, 2006.
11. C. Doche and L. Imbert. Extended double-base number system with applications to elliptic curve cryptography. In *INDOCRYPT*, volume 4329 of *LNCS*, pages 335–348. Springer, 2006.
12. Sylvain Duquesne. Improving the arithmetic of elliptic curves in the Jacobi model. *Inf. Process. Lett.*, 104(3):101–105, 2007.
13. H. M. Edwards. A normal norm for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.
14. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
15. H. Hisil, G. Carter, and E. Dawson. New formulae for efficient elliptic curve arithmetic. In *INDOCRYPT*, volume 4859 of *LNCS*, pages 138–151, 2007.
16. H. Hisil, K. Koon-Ho Wong, G. Carter, and E. Dawson. An intersection form for jacobi-quartic curves. Personal communication, 2008.
17. P.-Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In *CHES*, pages 391–401. Springer-Verlag, 2001.
18. P. Longa and C. Gebotys. Setting speed records with the (fractional) multibase non-adjacent form method for efficient elliptic curve scalar multiplication. Technical report, Department of Electrical and Computer Engineering University of Waterloo, Canada, 2009.

19. P. Longa and A. Miri. New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields. In *Public Key Cryptography*, volume 4939 of *LNCS*, pages 229–247. Springer Berlin / Heidelberg, 2008.
20. N. Meloni. New point addition formulae for ECC applications. In *Arithmetic of Finite Fields*, volume 4547 of *LNCS*, pages 189–201. Springer Berlin / Heidelberg, 2007.
21. P. K. Mishra and V. Dimitrov. Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation. In *Information Security*, volume 4779 of *LNCS*. Springer Berlin / Heidelberg, 2007.
22. A. C. Yao. On the evaluation of powers. *SIAM Journal on Computing*, 5(1):100–103, 1976.
23. E. Zeckendorf. Représentations des nombres naturels par une somme de nombre de Fibonacci ou de nombres de Lucas. *Bulletin de la Socit Royale des Sciences de Lige*, pages 179–182, 1972.

A Detailed operation counts

Curve shape	DBL	TPL	ADD	reADD	dADD	2dADD	2reADD	dreADD	mADD	dmADD	mreADD
3DIK	43.50	73.43	1.20	0.64	16.10	3.49	0.01	0.29	0.66	0.45	0.01
Edwards	139.12	12.84	1.68	1.55	18.48	0.97	0	0.01	1.59	0.22	0.01
ExtJQuartic	139.12	12.84	1.68	1.55	18.48	0.97	0	0.01	1.59	0.22	0.01
Hessian	112.22	29.73	1.26	1.07	17.40	1.63	0.01	0.17	1.07	0.28	0.03
InvEdwards	139.12	12.84	1.68	1.55	18.48	0.97	0	0.01	1.59	0.22	0.01
JacIntersect	142.19	10.94	2.40	1.64	17.71	0.81	0	0.13	2.22	0.29	0.03
Jacobian	130.10	18.71	1.43	1.09	18.36	1.11	0	0.14	1.31	0.25	0.03
Jacobian-3	130.10	18.71	1.43	1.09	18.36	1.11	0	0.14	1.31	0.25	0.03

Table 5. Detailed operation count for the Yao-DBNS scalar multiplication using 160-bit scalar

Curve shape	DBL	ZADD	ADD	reADD	mADD	2reADD	mreADD
Jacobian	141.27	25.53	19.55	0.48	1.72	0	0
Jacobian-3	141.27	25.53	19.55	0.48	1.72	0	0

Table 6. Detailed operation count for the Yao-BZNS scalar multiplication using 160-bit scalars

Curve shape	DBL	TPL	ADD	reADD	dADD	2dADD	2reADD	dreADD	mADD	dmADD	mreADD
3DIK	62.48	121.72	1.33	1.07	23.78	6.17	0.01	0.49	0.66	0.52	0.03
Edwards	219.17	22.89	1.87	2.37	28.46	1.61	0	0.23	1.48	0.35	0.06
ExtJQuartic	214.29	25.86	1.93	1.93	27.88	1.92	0.01	0.29	1.73	0.32	0.01
Hessian	184.33	44.73	1.38	1.47	26.58	2.57	0.01	0.21	1.18	0.34	0.03
InvEdwards	219.17	22.89	1.87	2.37	28.46	1.61	0	0.23	1.48	0.35	0.06
JacIntersect	235.26	12.94	2.37	3.04	29.31	1.39	0.02	0.24	2.25	0.33	0.05
Jacobian	184.33	44.78	1.38	1.47	26.58	2.57	0.01	0.21	1.18	0.34	0.03
Jacobian-3	184.33	44.78	1.38	1.47	26.58	2.57	0.01	0.21	1.18	0.34	0.03

Table 7. Detailed operation count for the Yao-DBNS scalar multiplication using 256-bit scalar

Curve shape	DBL	ZADD	ADD	reADD	mADD	2reADD	mreADD
Jacobian	226.3	41.4	30.01	0.74	1.20	0	0.02
Jacobian-3	226.3	41.4	30.01	0.74	1.20	0	0.02

Table 8. Detailed operation count for the Yao-BZNS scalar multiplication using 256-bit scalars