Delaying and Merging Operations in Scalar Multiplication: Applications to Curve-Based Cryptosystems

Roberto Maria Avanzi[1,⋆]

Faculty of Mathematics and Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany
`roberto.avanzi@ruhr-uni-bochum.de`

Abstract. In this paper we introduce scalar multiplication algorithms for several classes of elliptic and hyperelliptic curves. The methods are variations on Yao's scalar multiplication algorithm where independent group operations are shown in an explicit way. We can thus merge several group operations and reduce the number of field operations by means of Montgomery's trick. The results are that scalar multiplication on elliptic curves in even characteristic based on point halving can be improved by at least 10% and the performance of Koblitz curves by 25% to 32%.

1 Introduction

This paper describes efficient methods for scalar multiplication on elliptic curves (EC) and hyperelliptic curves (HEC). We provide replacements for techniques such as: double–and–add, halve–and–add [18,25], and methods based on the Frobenius operation [19,26,27].

Our scenario is the following: the base point is *variable*, and memory usage is *not* an issue (such as on personal computers). We are *not* concerned with minimizing memory usage: we consider the cases where we do not significantly change it, and where we can use as much memory as we want. The techniques developed here are applicable to a wide range of groups, that satisfy the following assumptions:

1. The group $(G, +, 0)$ is an algebraic variety, with explicit formulae for an addition, which uses field inversion, and for a second operation which is a group automorphism.
2. The speed of field inversion relative to field multiplication is in a quite common range for software implementations, i.e. from 6 to 30.
3. The group automorphism ϕ satisfies the following properties:
   (a) ϕ is identified with an element (also denoted by ϕ) of a number ring R, identified with a subring of the automorphism ring of G.

⋆ Supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

(b) We can represent each rational integer n to the the "base of ϕ" using a suitable digit set $\mathcal{D} \subset \mathbb{Z}[\phi] \subset R$ in the form

$$n \equiv \sum_{i=0}^{\ell} n_i \phi^i \bmod \Im \ , \tag{1}$$

where $n_i \in \mathcal{D}$ and $\Im$ is a (possibly trivial) ideal of R of elements acting like the identity on the group. Relation (1) implies that

$$n \cdot P = \sum_{i=0}^{\ell} n_i \phi^i(P) \ . \tag{2}$$

Important examples of group–automorphism pairs (G, ϕ) are

1. ϕ is the doubling of an element of G, i.e. $\phi(P) = 2 \cdot P$ for all $P \in G$. We identify ϕ with the numer 2. Typical examples are EC and HEC [16].
2. Halving in a subgroup G of an EC in even characteristic, i.e. for a given $P \in G$ computing $Q \in G$ such that $2Q = P$. The group G must have odd order N and ϕ is identified with $1/2$ modulo N.
3. The Frobenius operation on a Koblitz curve [19] E over a degree m extension of $\mathbb{F}_2$. The map ϕ is the Frobenius endomorphism (usually denoted by τ), ϕ^m operates trivially on the points of E, and $\Im$ is the ideal generated by $(\phi^m - 1)/(\phi - 1)$. Here ϕ is identified with $\tau := \frac{\mu + \sqrt{-7}}{2}$ where $\mu = \pm 1$ depends on the curve.
4. ϕ is the multiplication by a small constant p, where G is the rational point group of an EC over $\mathbb{F}_{p^m}$. Such curves have been proposed for pairing based systems [6,13,15].

Our starting point is the observation that Yao's scalar multiplication algorithm [28] contains group operations that are independent from each other. If the implementation of these operations makes use of field inversions, these can be merged using a trick by Montgomery. This trick has been already used to merge group operations in the context of scalar multiplication. These improvements have been restricted to: simultaneous scalar products; improving the precomputation stage [12]; devising formulae to compute expressions of the form $3P$, $4P$, $2P + Q$ etc. on elliptic curves using only one inversion [7] and thus reduce the number of inversions required in a slightly modified Horner scheme; and when the base point is fixed [22]. *The present paper is the first application of Montgomery's trick to the main loop of Yao's algorithm for a single scalar multiplication with a variable base point.*

Our techniques can improve the performance of scalar multiplication on EC using point halving, as well as on Koblitz curves. The improvements in these cases can often reach and exceed 10% and 30% respectively. On the other hand, the methods do not seem to bring savings to EC and HEC scalar multiplication based on point doubling.

The paper is organised as follows. In § 2 we describe the classical Φ-and-add and Yao's scalar multiplication methods. Our techniques are introduced in § 3, and analysed in § 4. § 5 is devoted to further refinements and practical usage of our methods, including operation counts in some settings of cryptographic relevance. Finally, in § 6 we conclude.

2 Scalar Multiplication Techniques

As in the introduction let $(G, +, 0)$ be a group, with a composition operation $+$ called addition and neutral element 0, endowed with a group automorphism ϕ enjoying the property that scalars can be written "to the base of ϕ" as in (1). Examples of such groups have already been mentioned. Some examples of recodings for the scalar n are: the binary and radix-r representation of integers; the Non Adjacent Form (NAF) [24]; the GNAF [8]; sliding window expansions [11,2] and their τ-adic counterparts [26,27]; and fractional windows [23].

2.1 Φ-and-Add Scalar Multiplication

Algorithm 1 describes several scalar multiplication techniques commonly known also as Horner schemes.

Let (G, ϕ) satify the requirements set in the introduction. Before proceeding we need some notation.

Notation 1. *The digit set $\mathcal{D}$ can be of the form $\mathcal{D} = \{0\} \cup \mathcal{D}^+$ with $0 \notin \mathcal{D}^+$ or $\mathcal{D} = \{0\} \cup \mathcal{D}^+ \cup (-\mathcal{D}^+)$ with $\mathcal{D}^+ \cap (-\mathcal{D}^+) = \emptyset$.*

The set $\mathcal{D}^+$ is called set of positive *digits, the digits in $-\mathcal{D}^+$ are the* negative *digits. For $d \in \mathcal{D}$ define the* sign *of a digit as follows:* $\mathrm{Sign}(d) = 0$ *if $d = 0$,* $\mathrm{Sign}(d) = 1$ *if $d \in \mathcal{D}^+$, and* $\mathrm{Sign}(d) = -1$ *if $d \in -\mathcal{D}^+$. If $d \in \mathcal{D}^+$ or $d = 0$ then put $|d| = d$, otherwise (if $d \in -\mathcal{D}^+$) put $|d| = -d$. This is the* absolute value *of a digit.*

Note that we consider two cases according to the fact whether computing $-P$ from P is a computationally expensive or negligible operation. The situation where $\mathcal{D} = \{0\} \cup \mathcal{D}^+ \cup (-\mathcal{D}^+)$ occurs when in the group G the computation of the inverse takes negligible time. In what follows we shall only consider this situation as it is the most common one in our context. The other case is just a simple adaptation.

In Steps 5 and 6, $n_i \cdot P$ is taken from the table computed in Step 1.

2.2 Yao's Method

Algorithm 2 is a straigthforward modification of Yao's [28] algorithm obtained upon replacing doublings with ϕ and distinguishing between "positive" and "negative" digits.

Observe that we are adding to registers corresponding to the various digits, instead of adding from a table indexed by them, and at the end we have to recombine the result.

Algorithm 2 is less used than Algorithm 1 as it is considered slower: First of all, a careful observation of the steps required, including replacing the first addition to registers initially set to zero with assignments, shows that (for example in

Algorithm 1. ϕ-and-add method for scalar multiplication

INPUT: An element P of a group G endowed with an addition $+$ and an automorphism ϕ, and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), and the n_i from a digit set $\mathcal{D}$ as per Notation 1.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Precompute $d \cdot P$ for all the digits $d \in \mathcal{D}^+$ and store them in a table
2.  $Q \leftarrow 0 \in G$
3.  for $i = \ell$ downto $0$ do
4.      $Q \leftarrow \phi(Q)$
5.      if $n_i \in \mathcal{D}^+$ then $Q \leftarrow Q + n_i \cdot P$
6.      if $n_i \in -\mathcal{D}^+$ then $Q \leftarrow Q - (-n_i) \cdot P$
7.  return $y$
```

the case $\Phi = 2$) Algorithms 1 and 2 take the same number of group operations. Furthermore, the choice of the representation for the group elements plays a big role in determining the performance: For EC [12] and genus 2 HEC [20] one can represent the elements using different coordinate systems, and there are formulae that add elements given in different coordinate systems with output in another system: In Algorithm 1 we choose a coordinate system with fast and inversion-free doubling for Q, and keep the elements $d \cdot P$ in affine coordinates to add them to Q faster. A similar strategy has not been investigated for Algorithm 1 yet. Hence, Yao's method can even be *less* efficient than the Horner scheme.

We call the computation of $\sum_{d \in \mathcal{D}^+} dX_d$ the *final accumulation step*. If $\mathcal{D}^+ = \{1, 3, 5, \ldots, t\}$ with t odd, then we can compute $\sum_{d \in \mathcal{D}^+} dX_d$ as two time the sum of the terms X_t, $X_t + X_{t-2}$, $X_t + X_{t-2} + X_{t-4}$, ..., $X_t + X_{t-2} + \cdots + X_3$, to which we add $X_t + X_{t-2} + \cdots + X_3 + X_1$. Each of the partial summands is computed by a successive group addition. The total number of operations is $t-1$ group additions and one doubling. If $t = 2^{w-1} - 1$ this amounts to $2^{w-1} - 2$ group additions and one doubling.

Algorithm 2. Yao's scalar multiplication algorithm

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Create a table of accumulator registers $X_d \leftarrow 0 \in G$ for all $d \in \mathcal{D}^+$
2.  $Q \leftarrow P$
3.  for $i = 0$ to $\ell$ do
4.      if $n_i \in \mathcal{D}^+$ then $X_{n_i} \leftarrow X_{n_i} + Q$
5.      if $n_i \in -\mathcal{D}^+$ then $X_{-n_i} \leftarrow X_{-n_i} - Q$
6.      $Q \leftarrow \phi(Q)$                                   [if $(i \neq \ell)$]
7.  return $\sum_{d \in \mathcal{D}^+} d \cdot X_d$
```

3 Delaying Group Additions to Collect Field Inversions

3.1 First Approach: Conservative Memory Usage

In Steps 4 and 5 of Yao's algorithm at successive iterations of the main loop, the group additions are in fact independent from each other until a digit (or its opposite) is encountered again. In fact, the value of Q depends only on the previous value of Q and thus only on P, and not on the other computations of the algorithms; hence, as long as we add independently computed points to different registers (i.e. these additions are associated to digits with absolute values different from each other), these additions can be performed in any order. We want to perform these group additions simultaneously using Montgomery's trick of simultaneous inversions. A realization of this idea is given as Algorithm 3.

Instead of performing the addition right away as in Algorithm 2, we first store information which is sufficient to keep track of the operations to be done, i.e. the value of Q to be added and the register that it has to be added to. To do this we keep a table $(A_j, d_j) \in G \times \mathcal{D}^+$, called the *queue*, containing up to q pairs, where A_j is the element (a value taken by the variable Q) to be added to the d_j-th element of the sequence of accumulator registers $\{X_d\}$. We also implement a routine called `FLUSH` that finally performs the delayed, independent, group additions $X_{d_j} \leftarrow X_{d_j} + A_j$ for $j = 1, 2, ..., k$ in parallel, where k is the number of pairs currently in the queue. Such a routine is invoked in the following cases: when a digit equal to a digit already in the queue (or its opposite) is encountered, when the queue itself contains q different elements and cannot accept any further pairs (A_j, d_j), or at the end of the algorithm, before performing the final accumulation step. As a consequence, there is no need for a value of q larger than the size of the digit set, i.e. $q \leq \#\mathcal{D}$.

Note that the first addition to an accumulator register is in fact just an assignment. This is explicitly reflected in the algorithm itself (Step 7).

3.2 Second Approach: Using More Memory

Algorithm 4 first computes all the $\phi^i(P)$ with $n_i \neq 0$, i.e. those that get added to or subtracted from the accumulator registers. Then it tries to collect them in groups as large as possible where each element corresponds to a different digit value. In fact, the absolute values of non zero digit which are *consecutive* in a given integer recoding usually do not form a *complete* set of positive digits. But, if we first compute all the $\phi^i(P)$, we do not need to group the additions in the same order the corresponding digits appear in the recoding. The hope here is to be able to collect groups of as many as $\#\mathcal{D}^+$ additions together. As §§ 4.2 and 5.3 will show, this will result in a gain over the previous approach. As in Algorithm 3 the first additions to accumulator registers are replaced by assignments.

Remark 1. Counters for the number of digits that still do not have the `used` bit set, as well as $\#\mathcal{D}^+$ many similar counters for the digits equal to each $d \in \mathcal{D}^+$, can be kept to speed up implementation of the tests in Steps 11 and 14. The last values found for the minimal i's in Step 15 can also be kept for each positive

Algorithm 3. Scalar multiplication with delayed group operations

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1, a parameter q.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Create a table of $accumulator\ registers\ X_d \leftarrow 0_G$ for all $d \in \mathcal{D}^+$
2.  Create a table (called the $queue$) $(A_k, d_k)$ for $1 \le k \le q$
3.  $Q \leftarrow P$
4.  for $i = 0$ to $\ell$ do
5.      if $n_i \neq 0$ do
6.          if $X_{|n_i|} = 0_G$ then                          [First addition to register]
7.              $X_{|n_i|} \leftarrow \mathrm{Sign}(n_i) Q$
8.          else                         [Otherwise, accumulate additions to be merged]
9.              if $|n_i| = d_j$ for some $j$ with $1 \le j \le q$ then FLUSH, $k \leftarrow 0$
10.             if $n_i \in \mathcal{D}^+$ then $k \leftarrow k+1$, $(A_k, d_k) \leftarrow (Q, n_i)$
11.             if $n_i \in -\mathcal{D}^+$ then $k \leftarrow k+1$, $(A_k, d_k) \leftarrow (-Q, -n_i)$
12.             if $k = q$ then FLUSH, $k \leftarrow 0$
13.     $Q \leftarrow \phi(Q)$                                         [if $(i \neq \ell)$]
14. if $k \neq 0$ then FLUSH
15. return $\sum_{d \in \mathcal{D}^+} d \cdot X_d$
```

digit, to avoid scanning the whole recoding each time. These counters permit to keep to total time spent scanning the recoding and building the queues linear in the recoding length, and negligible with respect to the group operations.

4 Performance Analysis

4.1 Analysis of the Conservative Memory Method (Algorithm 3)

Our analysis is based on the analysis of the birthday paradox.

Let $\mathcal{E}s$ be the expected number of elements in the queue of length q before we find a colliding digit, i.e. a digit equal, up to sign, to one already met after the last time the queue was flushed.

We adopt a urn-and-balls model: In an urn there are r balls, each inscribed with one of the positive digits. We repeatedly draw one ball at random from the urn, remember the result of the draw and replace the ball in the urn. As soon as when we draw a ball which we have already drawn, or after we have drawn q different balls, we stop, and record the number s of draws. We want to find the expected value $\mathcal{E}s$ of s.

We assume that in our representations all nonzero digits occur with equal probability. This is in fact the case in sliding or fixed windows methods (non fractional), both binary and τ-adic, or generic radix p representations.

The expected value $\mathcal{E}s$ is equal to $\sum_{j=0}^{q} p_j$ where p_j is the probability there is no match between balls after j draws. It is also clear that $p_1 = 1$, $p_2 = \left(1 - \frac{1}{r}\right)$

Algorithm 4. Scalar multiplication with delayed group operations, second method

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1, a parameter q.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Create a table of accumulator registers $X_d \leftarrow 0 \in G$ for all $d \in \mathcal{D}^+$
2.  Create a table (called the queue) $(A_k, d_k)$ for $1 \le k \le \#\mathcal{D}^+$
3.  Create a usage boolean table used$[i]$ for $0 \le i \le \ell$
4.  $Q \leftarrow P$
5.  for $i = 0$ to $\ell$ do
6.      if $n_i \neq 0$ then
7.          store $Q^{(i)} := Q\ (= \phi^i(P)\ )$, used$[i]$ $\leftarrow$ false
8.      else
9.          used$[i]$ $\leftarrow$ true
10.     $Q \leftarrow \phi(Q)$
11. while $(\exists\, i :$ used$[i] =$ false$)$ do
12.     $k \leftarrow 0$
13.     for all $d \in \mathcal{D}^+$ do
14.         if $(\exists\, i : ($used$[i] =$ false and $|n_i| = d))$ then
15.             Let $i$ be minimal among those with used$[i] =$ false and $|n_i| = d$
16.             if $X_{|n_i|} = 0_G$ then
17.                 $X_{|n_i|} \leftarrow \mathrm{Sign}(n_i)\, Q^{(i)}\ \left( = \mathrm{Sign}(n_i)\phi^i(P)\right)$
18.             else
19.                 if $n_i \in \mathcal{D}^+$ then
20.                     $k \leftarrow k+1,\ (A_k, d_k) \leftarrow (\ Q^{(i)}, d)\ \left( = (\ \phi^i(P), d)\right)$
21.                 if $n_i \in -\mathcal{D}^+$ then
22.                     $k \leftarrow k+1,\ (A_k, d_k) \leftarrow (-Q^{(i)}, d)\ \left( = (-\phi^i(P), d)\right)$
23.             used$[i]$ $\leftarrow$ true
24.     FLUSH, $k \leftarrow 0$
25. return $\sum_{d \in \mathcal{D}^+} d \cdot X_d$
```

and in general $p_j = \left(1 - \frac{1}{r}\right)\left(1 - \frac{2}{r}\right) \cdots \left(1 - \frac{j-1}{r}\right)$. Therefore $\mathcal{E}s = s(r,q) = 1 + \sum_{j=2}^{q} \prod_{k=1}^{j-1} \left(1 - \frac{k}{r}\right)$ where $r = 2^{w-2}$ in the sliding window case.

The following table collects the expected values of s for different positive digit set sizes in sliding window methods. The last column contains the maximal value of $\mathcal{E}s$, achieved for $q \ge r$.

$r \setminus q$	2	3	4	5	6	8	asym.
2	1.500						1.500
4	1.750	2.125	2.219				2.219
8	1.875	2.531	2.941	3.146	3.223	3.245	3.245
16	1.938	2.758	3.424	3.924	4.268	4.603	4.704
32	1.969	2.877	3.700	4.420	5.028	5.907	6.774

4.2 Analysis of the Large Memory Method (Algorithm 4)

This case is more complicated than the previous one.

Suppose that a given recording $n \equiv \sum_{i=0}^{\ell} n_i \phi^i$ has k non zero digits, and that there are r nonnegative integers $a_1 \geq a_2 \geq \ldots \geq a_r$, $r = \mathcal{D}^+$, adding up to k, with the following property: we can write $\mathcal{D}^+ = \{d_1, d_2, \ldots, d_r\}$ such that for $1 \leq i \leq r$ the digit d_i appears exactly a_i times in the recoding.

Algorithm 4 will then build a_r queues of r elements (therefore merging r inversions in one, with $3(r-1)$ additional multiplications, a_r times – if the addition formula requires only one inversion), $a_{r-1} - a_r$ queues of $r-1$ elements (therefore merging $r-1$ inversions $a_{r-1} - a_r$ times), ..., will perform $a_2 - a_3$ pairs of additions simultaneously (merging each time two inversions in one inversion and 3 multiplications), and at the end perform $a_1 - a_2$ single additions. The final number of blocks will then be $a_r + (a_{r-1} - a_r) + (a_{r-2} - a_{r-1}) + \cdots + (a_2 - a_3) + (a_1 - a_2) = a_1$. If there is just one field inversion per group addition, the final number of inversions and multiplications replacing the $k-1$ inversions coming from group additions in the original scalar multiplication is a_1 inversions and $3(k - a_1)$ multiplications[1]. This quantity depends only on a_1, which is the maximum of the a_i (and on k). We are thus interested in the expected value of a_1. The computation of this value is not easy and there are asymptotic estimates for this kind of "occupancy of boxes" problem, see for example [17, Ch. II, 6]. However, this does not solve our question for small values of the parameters r and k. We now describe approaches more suitable to direct computation.

First, we drop the assumption that $a_1 \geq a_2 \geq \ldots \geq a_r$, but keep that the nonnegative integers a_i add up to k. Write $\mathbf{a} := [a_1, a_2, ..., a_r]$ and $|\mathbf{a}| = a_1 + a_2 + \ldots + a_r$. Instead of a_1 we are now interested in $\max(\mathbf{a}) := \max(a_1, ..., a_r)$. The expected value of $\max(\mathbf{a})$ is

$$\frac{1}{r^k} \sum_{\mathbf{a} \in \mathbb{N}_0^r \,:\, |\mathbf{a}|=k} \binom{k}{\mathbf{a}} \max(\mathbf{a}) \tag{3}$$

where $\binom{k}{\mathbf{a}}$ is the multinomial coefficient $\binom{k}{a_1\ a_2\ \cdots\ a_r} = \frac{k!}{a_1! a_2! \cdots a_r!}$. Expression (3) can be easily evaluated by a simple computer program, but the number of summands increases very quickly and soon gets out of control. The best way to estimate the expected value of $\max(\mathbf{a})$ is then by averaging the value obtained on a few million random recodings, i.e. by a Monte Carlo method. This can be done by a second, short computer program. We first considered "small" values of r and k, that could be handled by our computer using (3), then and we chose the number of experiments for the second computer program so that the two outcomes agreed to at least 4 decimal places; then, we computed other expected values for $\max(\mathbf{a})$ using only the second program.

In the next table we display the results for some values of ℓ and $r = 2^{w-2}$, where k is given as the closest integer to $\frac{\ell}{w+1} - r$ – in other words we are using

[1] If there are t inversions per groups addition, we replace $t(k-1)$ inversions by ta_1 inversions and $3t(k - a_1)$ multiplications. This case may occur in some circumstances if a formula with more inversions is less expensive than a formula with less inversions but a much larger number of multiplications.

width-w signed sliding window methods (binary or τ-adic) and take into account the fact that the first addition to each accumulator register is just an assignment. We provide the expected queue length $\mathcal{E}q = k/\mathcal{E}\max(\mathbf{a})$, too.

ℓ	$w=3$ $(r=2)$			$w=4$ $(r=4)$			$w=5$ $(r=8)$			$w=6$ $(r=16)$			$w=7$ $(r=32)$		
	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$
160	38	21.443	1.772	28	9.806	2.856	19	4.764	3.989	7	1.905	3.675	4	1.181	3.386
192	46	25.691	1.791	35	11.877	2.947	24	5.662	4.239	12	2.575	4.660	8	1.665	4.806
240	58	32.025	1.811	44	14.497	3.035	32	7.047	4.541	18	3.320	5.422	14	2.247	6.231
256	62	34.129	1.817	48	15.649	3.067	35	7.554	4.663	21	3.657	5.742	16	2.392	6.688
320	78	42.512	1.835	60	19.071	3.146	45	9.207	4.888	30	4.625	6.486	24	2.990	8.026
512	126	67.469	1.868	98	29.681	3.302	75	13.944	5.378	57	7.244	7.869	48	4.515	10.632

Note that when using the *minimum* of $\mathbf{a}$ in place of the maximum in all the computations we just described we get the expected number of "full" queues, i.e. those of length r. The computations show that with overwhelming probability there is always at least one such queue.

5 Practical Aspects

5.1 Optimizing the Precomputation and Accumulation Steps

One potential performance problem with scalar multiplication methods derived from Yao's is the final accumulation step, i.e. the computation of $\sum_{d\in\mathcal{D}^+} X_d$, especially for larger window sizes.

5.1.1 Precomputations

For the precomputation phase for ϕ-and-add methods there is a very efficient method by Cohen (see [12]). In general we have a total of $2^{w-2}-1$ group additions and 1 group doubling fused in $w-1$ groups, in other words requiring just $w-1$ inversions. It has been described originally for EC with doubling, but can be used also with halving, and HEC.

Other coordinate systems for the precomputations can be chosen, but only if inversion is extremely expensive; for example using affine coordinates in the precomputation phase with Cohen's approach is the best choice if $\mathtt{I} < 33.9\mathtt{M}$ with a key length of 192 bits.

Another classical precomputation strategy requires just 2 inversions. We first compute $2P$ in affine coordinates and then get $3P = P + 2P, \ldots, (2n+1)P = (2n-1)P+2P$ each as a mixed coordinate addition (Jacobian coordinates for odd characteristic EC and Lopez-Dahab coordinates for even characteristic EC). To get back the precomputations in affine coordinates, all Z-coordinates are inverted together by Montgomery's trick.

For our values of the I/M ratio Cohen's method is almost consistently the best precomputation strategy.

5.1.2 Accumulation

We present a technique that applies to the final accumulation step of Yao's algorithm. Since the digit set for doubling and halving based methods for EC and HEC is the same, we give a unitary treatment. Before proceeding we introduce some notation.

Notation 2. $\mathtt{M}$, *resp.* $\mathtt{I}$, $\mathtt{S}$ *denote field multiplication, resp. inversion, squaring.* $\mathtt{A}$ *and* $\mathtt{D}$ *mean group addition and doubling.* $\mathtt{A}[i]$, *resp.* $\mathtt{D}[i]$, *denote* $i \geq 1$ *simultaneous group additions, resp. doublings.*

For any two integers a, b *we can write* $\mathbf{X}_{a,b} = X_a + X_b$. *By* $\mathbf{X}_{[a\,..\,b]}$ *we understand the sum* $X_a + X_{a+2} + \ldots + X_b$ *for* $a < b$ *and* a, b *odd. The length of a sum of the form* $\mathbf{X}_{[a\,..\,b]}$ *is the number of summands* X_i *it contains.*

Let $N_{\mathtt{A}}$, resp $N_{\mathtt{D}}$, N_G be the number of $\mathtt{A}$s, $\mathtt{D}$s and groups of operations in an algorithm. Under the assumption that $\mathtt{A}$ and $\mathtt{D}$ both contain just one inversion, the operation count will be

$$N_{\mathtt{A}}(\mathtt{A} - \mathtt{I}) + N_{\mathtt{D}}(\mathtt{D} - \mathtt{I}) + N_G\mathtt{I} + 3(N_{\mathtt{A}} + N_{\mathtt{D}} - N_G)\mathtt{M} \ . \tag{4}$$

(The case where $\mathtt{A}$ and $\mathtt{D}$ contain more inversions is left to the reader.)

Here and in what follows $t = 2^{w-1}$. The standard approach consists in computing X_{t-1}, $\mathbf{X}_{t-3,t-1} = X_{t-3} + X_{t-1}$, $\mathbf{X}_{[t-5\,..\,t-1]} = X_{t-5} + X_{t-3} + X_{t-1}$,..., $\mathbf{X}_{[1\,..\,t-1]} = X_1 + X_3 + \ldots + X_{t-1}$ by successive additions, add all these sums but the last one together, double the result and add it to the last sum. This requires $2^{w-1} - 1$ group operations (one is a doubling).

If $w = 3$ then there are just two registers, X_1 and X_3 and the computation of $X_1 + 3X_3$ is done by first computing $X_1 + X_3$ and $2X_3$ at the same time, merging one addition and one doubling, followed by another addition. For EC this takes $2\mathtt{I} + 12\mathtt{M} + 3\mathtt{S}$ (and one more $\mathtt{S}$ if the characteristic is odd). Using different coordinate systems we can first compute $X_1 + X_3$ and $2X_3$ in Jacobian coordinates then add the results and convert them into affine coordinates. This takes $1\mathtt{I} + 21\mathtt{M} + 11\mathtt{S}$ in odd characteristic and $1\mathtt{I} + 23\mathtt{M} + 5\mathtt{S}$ in characteristic 2. The first method is faster for EC in odd characteristic if $\mathtt{I} < 11\mathtt{M}$ and $\mathtt{I} < 12\mathtt{M}$ in characteristic 2.

For all $w \geq 4$ we present a simple strategy by which we can compute the final expression with $3 \cdot 2^{w-2} - w$ group operations (one is a doubling), by fusing them in $3w - 5$ groups. On the other hand, the classic approach requires $2^{w-1} - 1$ group additions (one is a doubling).

We first compute $\mathbf{X}_{1,3}$, $\mathbf{X}_{5,7}$, ..., $\mathbf{X}_{t-3,t-1}$ (additions of adjacent elements in the list $X_1, X_3, \ldots, X_{t-1}$) by means of one $\mathtt{A}[2^{w-3}]$, then $\mathbf{X}_{[1\,..\,7]}$, $\mathbf{X}_{[9\,..\,15]}$, ..., $\mathbf{X}_{[(t-7)\,..\,(t-1)]}$ (quadruplets of adjacent elements) by one $\mathtt{A}[2^{w-4}]$, and so on until we get the length 2^{w-2} sum $\mathbf{X}_{[1\,..\,(t-1)]}$ by one $\mathtt{A}$.

The second phase of our strategy consists in calculating the sums $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \leq i \leq t - 3$.

The last sum which we just computed involves 2^{w-2} summands X_i. All the partial sums which we still require and whose length is a multiple of 2^{w-3} (namely, $\mathbf{X}_{[1\,..\,(t-1)]}$ and $\mathbf{X}_{[(t/2+1)\,..\,(t-1)]}$) are already known.

We first get the sums of a number of terms which is multiple of 2^{w-4}: the first one, the last and in general all those whose length is a power of two are already known. One sum is to be computed (cost: one $\mathtt{A}$).

Then we compute the $3 = 2^2 - 1$ sums of length multiple of 2^{w-5} with one $\mathtt{A}[3]$. The next step is to compute the $7 = 2^3 - 1$ sums of length multiple of 2^{w-6} with one $\mathtt{A}[7]$. We proceed this way until we compute the last $2^{w-3} - 1$ sums of an odd number of elements with an $\mathtt{A}[2^{w-3} - 1]$.

As an example, to get $\mathbf{X}_{[3\,..\,(t-1)]}$ we first need to add $\mathbf{X}_{[(t/4+1)\,..\,(t/2-1)]}$ to $\mathbf{X}_{[(t/2+1)\,..\,(t-1)]}$, to get $\mathbf{X}_{[(t/4+1)\,..\,(t-1)]}$, then we add $\mathbf{X}_{[(t/8+1)\,..\,(t/4-1)]}$ to this sum, $\mathbf{X}_{[(t/18+1)\,..\,(t/8-1)]}$, and so on, until we just add X_3 to $\mathbf{X}_{[5\,..\,(t-1)]}$. All partial summands $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \le i \le t-1$ are obtained this way, with the sums of addends $\mathbf{X}_{[a\,..\,b]}$ of the same length done in parallel.

The last phase consists in adding all the sums $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \le i \le t-1$ together. This can be done in a binary tree fashion, by $\mathtt{A}[2^{w-3} - 1]$ (note that $\mathbf{X}_{[1\,..\,(t-1)]}$ is not added), $\mathtt{A}[2^{w-4}]$, $\mathtt{A}[2^{w-5}]$, …, $\mathtt{A}[2]$, $\mathtt{A}$, this sum is then doubled and added to $\mathbf{X}_{[1\,..\,(t-1)]}$.

It is easy to see that the total operation count is $3 \cdot 2^{w-2} - w$ additions (of which one is in fact a doubling), fused in $3w - 5$ groups: the latter number, multiplied by the number of $\mathtt{M}$ per group operation (usually 1), is the number of $\mathtt{I}$ effectively computed. This number is usually much lower than $2^{w-1} - 1$ for the classical approach.

For $4 \le w \le 6$ it is easy to devise faster ad-hoc groupings of operations (possibly using mixed coordinates). For $3 \le w \le 8$ the groupings and the operation counts for EC are given in the following table. We also provide the ratio $\mathtt{I}/\mathtt{M}$, over which our method is faster than the classical approach.

w	Classic approach	Our method(s)		Example: Elliptic Curves			
				Field operation counts		*Thresholds*	
	Ops.	*Ops.*	*Groups*	Classic approach	Our method(s)	*odd char.*	*even char.*
3	3	3 –	2 –	$3\mathtt{I} + 6\mathtt{M} + 3\{+1\}\mathtt{S}$	$2\mathtt{I} + 9\mathtt{M} + 3\{+1\}\mathtt{S}$ $1\mathtt{I} + 21\{+2\}\mathtt{M} + 11\{+2\}\mathtt{S}$	3 9.50	3 9.50
4	7	9	6	$7\mathtt{I} + 14\mathtt{M} + 7\{+1\}\mathtt{S}$	$6\mathtt{I} + 27\mathtt{M} + 9\{+1\}\mathtt{S}$	13.50	13.40
5	15	20	8	$15\mathtt{I} + 30\mathtt{M} + 15\{+1\}\mathtt{S}$	$8\mathtt{I} + 76\mathtt{M} + 20\{+1\}\mathtt{S}$	6.93	6.71
6	31	40	12	$31\mathtt{I} + 62\mathtt{M} + 31\{+1\}\mathtt{S}$	$12\mathtt{I} + 164\mathtt{M} + 40\{+1\}\mathtt{S}$	5.60	5.46
7	63	89	16	$63\mathtt{I} + 126\mathtt{M} + 63\{+1\}\mathtt{S}$	$16\mathtt{I} + 397\mathtt{M} + 89\{+1\}\mathtt{S}$	6.04	5.88
8	127	184	19	$127\mathtt{I} + 254\mathtt{M} + 127\{+1\}\mathtt{S}$	$19\mathtt{I} + 863\mathtt{M} + 184\{+1\}\mathtt{S}$	5.90	5.74

Note: $\{+1\}$ *means "add 1" in odd characteristic.*

5.2 Operation Counts

We consider here the cost of different scalar multiplication algorithms on (hyper)elliptic curves. Our references are [10, §§ 13.2, 13.3, 15.1] and [5].

5.2.1 Binary Elliptic Curves with Point Doubling

Let us review the costs for a scalar multiplication (including precomputation) with the double-and-add method. Operation counts are from [10, §§ 13.2, 13.3].

Let $n = \sum_{i=0}^{\ell} n_i 2^i$ be the scalar, and w the window width. Put $\ell_1 = \ell - (w-1)/2$ and $K = 1/2 - 1/(w+1)$. On average $\ell_1 + K$ doublings and $v = (\ell_1 - K)/(w+1)$ additions are used. If only affine coordinates $\mathcal{A}$ are used, then an auxiliary system $\mathcal{A}'$ is introduced, v doublings use a special form ($2\mathcal{A}' = \mathcal{A}'$) that requires only $\mathtt{M} + 2\mathtt{S}$, whereas the remaining ones (of the form $2\mathcal{A}' = \mathcal{A}$) cost $\mathtt{I} + \mathtt{M} + \mathtt{S}$. The additions require the ability of adding two points in $\mathcal{A}$ with a result in $\mathcal{A}'$, and cost $2\mathtt{I} + 3\mathtt{M} + \mathtt{S}$. In the main loop we need about $(\ell_1 + K)(\mathtt{I}+\mathtt{M}+\mathtt{S}) + \frac{w+1}{\ell_1 - K}(3\mathtt{M}+2\mathtt{S})$ to compute $n \cdot P$. If we do not use inversions, we perform the computation using López-Dahab coordinates, and the cost is $(\ell_1 + K)(4\mathtt{M}+4\mathtt{S}+\mathtt{M}_2) + \frac{w+1}{\ell_1 - K}(8\mathtt{M}+5\mathtt{S}+\mathtt{M}_2) + (\mathtt{I}+2\mathtt{M})$ where $\mathtt{M}_2$ denotes multiplication by a fixed value. Note that there is a final conversion to affine coordinates. The precomputation phase is done as described in § 5.1.1.

With our algorithms we perform $\ell_1 + K$ doublings and $\widehat{v} := v - 2^{w-2}$ additions (recall that the first time a digit appears, a group addition is replaced by an assignment). The additions come in $\widehat{v}/\mathcal{E}s$ groups of merged additions, and the concrete values of $\mathcal{E}s$ are taken from the tables in the previous section.

Both in the Conservative Memory algorithm (§ 3.1) and in the Large Memory algorithm (§ 3.2) we perform the doublings in the coordinate system that has fastest doublings, i.e. López-Dahab's ($\mathcal{LD}$) and then convert the v points that get added (or written) to the accumulator registers to $\mathcal{A}$. The conversions are done in parallel (with some obvious modifications to the algorithms): in the Conservative Memory algorithm only $\mathcal{E}s$ points at a time are converted, and in the Large Memory algorithm all the v points are converted simultaneously. The cost of the main loop is about

$$(\ell_1{+}K)(4\mathtt{M}{+}4\mathtt{S}{+}\mathtt{M}_2){+}\Big(\frac{v}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s-1)\mathtt{M}) + 2v\,\mathtt{M}\Big){+}\Big(\frac{\widehat{v}}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s-1)\mathtt{M}) + \widehat{v}(2\mathtt{M}+\mathtt{S})\Big)$$

in the Conservative Memory algorithm and

$$(\ell_1 + K)(4\mathtt{M} + 4\mathtt{S} + \mathtt{M}_2) + (\mathtt{I} + 3(v-1)\mathtt{M} + 2v\,\mathtt{M}) + \left(\frac{\widehat{v}}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s-1)\mathtt{M}) + \widehat{v}(2\mathtt{M}+\mathtt{S})\right)$$

in the Large Memory algorithm. The cost of the final accumulation step is given in § 5.1.

5.2.2 Using Point Halving on Elliptic Curves

Let n, ℓ, w, ℓ_1, v as above. We use here point halving, which consists, given a point P, in finding a point Q such that $2Q = P$. It is denoted by $\mathtt{H}$. Operation counts are essentially as before, where halvings replace doublings in the main loops, but the precomputation and the final accumulation stages retain the doubling. Only affine coordinates are used, following [18,25]. According to [14], halving is about twice as fast as a doubling. The classic halve-and-add method costs then $(\ell_1 + K)\mathtt{H} + v\,\mathtt{A}$ in the main loop, the precomputations being as in § 5.2.1.

5.2.3 Elliptic Koblitz Curves

The reasoning is the same as in the previous Subsections, where doubling and halving are here replaced by Frobenius operations. As in § 5.2.1 we will determine operation counts when affine coordinates are used as well as with $\mathcal{LD}$

coordinates. The recoding used is a windowed τ-adic recoding using the digit set by Solinas [26,27]. Precomputation and final accumulation steps have to be adapted to the new context of τ-adic digit sets, essentially ad-hoc for each w: the difference in the operations counts are just a few Frobenius operations.

5.2.4 Elliptic Curves in Odd Characteristic

In odd characteristic the situation is entirely similar to that of binary Elliptic curves where point doubling is used. The details are similar and are thus omitted.

The coordinate systems used for the Horner scheme are: affine only, and a mixed coordinate approach involving precomputed points in $\mathcal{A}$ and intermediate computations done using Jacobian ($\mathcal{J}$) and Jacobian modified ($\mathcal{J}^m$) coordinates.

In our algorithms we perform the doublings in $\mathcal{J}^m$, since they have the fastest doubling, and then convert the required results to $\mathcal{A}$.

5.2.5 Higher Genus Curves

We consider here curves of genus 2 and 3 in even characteristic with the formulae of [21] in genus 2 and those of [5] for genus 3. For genus 2 a group addition, resp. doubling, costs $\mathtt{I} + 22\mathtt{M} + 3\mathtt{S}$, resp. $\mathtt{I} + 6\mathtt{M} + 5\mathtt{S}$. For genus 3 these operations cost $\mathtt{I} + 47.7\mathtt{M} + 6\mathtt{S}$, resp. $\mathtt{I} + 9.3\mathtt{M} + 11\mathtt{S}$. Apart from the operation counts for the group operations, the computations are performed as in the EC case with affine coordinates.

5.3 Comparisons and Results

Here we collect the operation counts relative to a field multiplication for several types of groups at different security levels. We compare the classical techniques (Algorithm 1), denoted by the label "Classic" with our algorithms, and for Koblitz curves also with the methods in [4]. CMA stands for *Conservative Memory Algorithm* (Algorithm 3) and LMA for *Large Memory Algorithm* (Algorithm 4). In some cases the "Classic" column contains two subcolumns: The one labeled $\mathcal{A}$ refers to computations entirely done in affine coordinates, whereas $\mathcal{LD}$, resp. $\mathcal{J}^m$ refers to the usage of López-Dahab, resp. Jacobian modified coordinates. The relative costs of the field operations in the even characteristic are taken from the implementations reported [3,5]. For the ratios in the odd characteristic case we refer to [1]: for example for fields of 160, 192 and 256 bits we have $\mathtt{I}/\mathtt{M}$ ratios around 30. The bit size of the groups in the table for HEC is approximative, and the field of definition is given, too.

The table for elliptic Koblitz Curves contains more comparisons to put the results of this paper in a broader perspective. Firstly, some curves whose size is outside the scope of this paper are considered: the sizes are those of the curves K-163, K-233, K-283, K-409 and K-571 as standardised by NIST [29]. Secondly, as already mentioned we compare also with two algorithms from [4] – for the details we refer the reader to the cited paper.

Clearly, we get mixed results. In some circumstances the new algorithms perform significantly better than the previous ones, in other cases the performance is similar, or worse. The Large Memory algorithm performs better than the Conservative Memory approach, but the differences are often quite small.

Elliptic Curves in Even Characteristic							
bits	Elliptic Curve, Doubling				Elliptic Curve, Halving		
	Classic		CMA	LMA	Classic	CMA	LMA
	$\mathcal{A}$	$\mathcal{LD}$					
163	1840.17	1144.70	1254.79	1213.14	853.57	773.61	768.22
199	2532.87	1383.57	1537.93	1474.85	989.05	870.29	859.46
233	3131.90	1583.06	1756.22	1699.34	1174.77	1022.08	1002.11

Elliptic Koblitz Curves						
bits	Classic		This paper		From [4]	
	$\mathcal{A}$	$\mathcal{LD}$	CMA	LMA	Alg. 3	Alg. 4
163	390.77	406.55	305.75	300.35	359.59	376.40
233	601.41	523.13	427.69	407.71	610.61	500.23
283	1040.88	605.06	639.28	568.96	928.49	585.94
409	3122.46	832.23	1266.51	1144.68	2719.01	801.94
571	7633.01	1311.20	2746.00	2428.36	6340.55	1058.10

Elliptic Curves in Odd Char.				
bits	Classic		CMA	LMA
	$\mathcal{A}$	$\mathcal{J}^m$		
160	6378.20	1422.42	1825.50	1528.52
192	7472.86	1670.41	2099.27	1787.72
256	9794.09	2166.41	2623.72	2321.92

Hyperelliptic Curves in Even Characteristic								
bits	Genus 2				Genus 3			
	Field	Classic	CMA	LMA	Field	Classic	CMA	LMA
160	$\mathbb{F}_{83}$	2877.14	2883.70	2882.04	$\mathbb{F}_{53}$	4394.68	4437.62	4435.47
192	$\mathbb{F}_{97}$	3300.71	3291.20	3287.44	$\mathbb{F}_{67}$	5219.82	5252.92	5248.99
256	$\mathbb{F}_{127}$	4146.97	4138.74	4133.97	$\mathbb{F}_{89}$	6531.11	6578.93	6575.54

For EC scalar multiplication with point doubling, a lot of research went into refined coordinate systems (that can dispense with inversions), and this explains why our approach, that is based for the most part on “good old” affine coordinates that require inversions, does not win. For HEC in even characteristic, that also profit from very refined doubling formulae, the situation is similar.

Halving and Frobenius methods profit the most because they use relatively many inversions that can be effectively reduced by our techniques. Savings of 10% to 15% (for halving methods) to 25-32% (for Frobenius methods) are not uncommon. In particular, for generic binary EC, halving brings noteworthy speedups, our methods being as much as 30 to 40% faster than the mixed coordinates implementation of the Horner scheme. We also confirm Schroeppel's claims that a scalar multiplication based on halving is usually (at least) twice as fast than one based on doubling in affine coordinates.

For Koblitz curves, the methods from [4] are already faster than classical methods for small curves, yet slower than ours for the sizes that are in the scope of the present paper. For larger sizes, the CMA and LMA methods lose ground and become slower than classic methods (starting from the 409 bit level). However, the inversion-free sublinear Algorithm 4 from [4], quickly reveals itself as the fastest scalar multiplication method for Koblitz curves of at least 409 bits, with speedups reaching 24% with respect to the mixed coordinates τ-adic Horner scheme for 571 bit curves.

6 Conclusions

In this paper we showed that Yao's algorithm presents an intrinsic parallelism that permits us to merge some of the group operations it performs. Two variants of this method are given: one adopts a strategy of augmenting memory usage in a conservative way, and is given as Algorithm 3; the second method, Algorithm 4, uses more memory but is also more efficient.

A simple analysis has been done. This is then used to estimate the costs of scalar multiplications for different groups used in cryptographic applications: EC over odd and even characteristic fields (in this case we also chose parameters that are presented in standards and have been considered by several authors for performance improvements), using various scalar multiplication techniques, as well as low genus HEC.

The techniques presented here can significantly improve the performance for EC cryptosystems in even characteristic when point halving is used, as well as Koblitz curve Cryptosystems. The gains in these cases can be estimated in the 10-15%, resp. 25-32% range with common timing ratios of field operations with respect to the field multiplication.

Acknowledgements. Part of this paper was written while visiting the Department of Mathematical Sciences, Stellenbosch University. I wish to express my gratitude to this institution for its hospitality. I am grateful to Johann Großschädl, Clemens Heuberger, Helmut Prodinger, Francesco Sica, Douglas Stabila and Nicolas Thériault for interesting discussions and remarks.

References

1. Avanzi, R.M.: Aspects of hyperelliptic curves over large prime fields in software implementations. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 148–162. Springer, Heidelberg (2004)
2. Avanzi, R.M.: A Note on the Signed Sliding Window Integer Recoding and its Left-to-Right Analogue. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 130–143. Springer, Heidelberg (2004)

3. Avanzi, R.M., Ciet, M., Sica, F.: Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 28–40. Springer, Heidelberg (2004)
4. Avanzi, R.M., Heuberger, C., Prodinger, H.: On Redundant τ-adic Expansions and Non-Adjacent Digit Sets. LNCS. vol. 4356, pp. 285–301, Springer, Heidelberg (to appear)
5. Avanzi, R.M., Thériault, N., Wang, Z.: Rethinking Low Genus Hyperelliptic Jacobian Arithmetic over Binary Fields: Interplay of Field Arithmetic and Explicit Formulae. CACR report 2006-07, Available at `http://www.cacr.math.uwaterloo.ca/tech_reports.html`
6. Barreto, P., Kim, H., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
7. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading Inversions for Multiplications in Elliptic Curve Cryptography. Designs Codes and Cryptography 39(2), 189–206 (2006)
8. Clark, W.E., Liang, J.J.: On arithmetic weight for a general radix representation of integers. IEEE Transactions on Information Theory IT- 19, 823–826 (1973)
9. Cohen, H.: A course in computational algebraic number theory. Graduate Texts in Math. 138, Springer, Heidleberg, 1993, Third corrected printing (1996)
10. Cohen, H., Frey, G. (eds.): The Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
11. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 282–290. Springer, Heidelberg (1997)
12. Cohen, H., Miyaji, A., Ono, T.: Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
13. Duursma, I., Lee, H-S.: Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 111–123. Springer, Heidelberg (2003)
14. Fong, K., Hankerson, D., López, J., Menezes, A.: Field Inversion and Point Halving Revisited. IEEE Trans. Computers 53(8), 1047–1059 (2004)
15. Galbraith, S., Harrison, K., Soldera, D.: Implementing the Tate pairing. In: Fieker, C., Kohel, D.R. (eds.) Algorithmic Number Theory. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
16. Koblitz, N.: Hyperelliptic cryptosystems. J. Cryptology 1, 139–150 (1989)
17. Kolchin, V.F., Sevast'yanov, B.A., Chistyakov, V.P.: Random Allocations. V.H. Winston and Sons, Washington DC (1978)
18. 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)
19. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
20. Lange, T.: Formulae for arithmetic on genus 2 hyperelliptic curves. Appl. Algebra Engrg. Comm. Comput. 15(5), 295–328 (2005)
21. Lange, T., Stevens, M.: Efficient doubling for genus two curves over binary fields. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 170–181. Springer, Berlin (2004)

22. Mishra, P.K., Sarkar, P.: Application of Montgomery's Trick to Scalar Multiplication for Elliptic and Hyperelliptic Curves Using a Fixed Base Point. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 41–54. Springer, Heidelberg (2004)
23. Möller, B.: Algorithms for Multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 165–180. Springer, Heidelberg (2001)
24. Reitwiesner, G.W.: Binary arithmetic. Advances in Computers 1, 231–308 (1960)
25. Schroeppel, R.: Elliptic curve point ambiguity resolution apparatus and method. International Application Number PCT/US00/31014, filed (November 9, 2000)
26. Solinas, J.A.: An improved algorithm for arithmetic on a family of elliptic curves. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 357–371. Springer, Heidelberg (1997)
27. Solinas, J.A.: Efficient Arithmetic on Koblitz Curves. Codes and Cryptography 19(2/3), 125–179 (2000)
28. Yao, A.C.: On the evaluation of powers. SIAM J. Comp. 5, 100–103 (1976)
29. National Institute of Standards and Technology. Digital Signature Standard. FIPS Publication 186-2 (February 2000)