Faster Explicit Formulas for Computing Pairings over Ordinary Curves

Diego F. Aranha[1,⋆], Koray Karabina[2,⋆], Patrick Longa[3], Catherine H. Gebotys[3], and Julio López[1]

[1] University of Campinas
{dfaranha,jlopez}@ic.unicamp.br
[2] Certicom Research
kkarabina@rim.com
[3] University of Waterloo
{plonga,cgebotys}@uwaterloo.ca

Abstract. We describe efficient formulas for computing pairings on ordinary elliptic curves over prime fields. First, we generalize lazy reduction techniques, previously considered only for arithmetic in quadratic extensions, to the whole pairing computation, including towering and curve arithmetic. Second, we introduce a new compressed squaring formula for cyclotomic subgroups and a new technique to avoid performing an inversion in the final exponentiation when the curve is parameterized by a negative integer. The techniques are illustrated in the context of pairing computation over Barreto-Naehrig curves, where they have a particularly efficient realization, and are also combined with other important developments in the recent literature. The resulting formulas reduce the number of required operations and, consequently, execution time, improving on the state-of-the-art performance of cryptographic pairings by 28%-34% on several popular 64-bit computing platforms. In particular, our techniques allow to compute a pairing under 2 million cycles for the first time on such architectures.

Keywords: Efficient software implementation, explicit formulas, bilinear pairings.

1 Introduction

The performance of pairing computation has received increasing interest in the research community, mainly because Pairing-Based Cryptography enables efficient and elegant solutions to several longstanding problems in cryptography such as Identity-Based Encryption [1,2], powerful non-interactive zero-knowledge proof systems [3] and communication-efficient multi-party key agreements [4]. Recently, dramatic improvements over the figure of 10 million cycles presented in [5] made possible to compute a pairing at the 128-bit security level in 4.38 million cycles [6] when using high-speed vector floating-point operations, and

⋆ This work was completed while these authors were at the University of Waterloo.

K.G. Paterson (Ed.): Eurocrypt 2011, LNCS 6632, pp. 48–68, 2011.

2.33 million cycles [7] when the fastest integer multiplier available in Intel 64-bit architectures is employed.

This work revisits the problem of efficiently computing pairings over large-characteristic fields and improves the state-of-the-art performance of cryptographic pairings by a significant margin. First of all, it builds on the latest advancements proposed by several authors:

- The Optimal Ate pairing [8] computed entirely on twists [9] with simplified final line evaluations [6] over a recently-introduced subclass [10] of the Barreto-Naehrig (BN) family of pairing-friendly elliptic curves [11].
- The implementation techniques described by [7] for accelerating quadratic extension field arithmetic, showing how to reduce expensive carry handling and function call overheads.

On the other hand, the following new techniques are introduced:

- The notion of lazy reduction, usually applied for arithmetic in quadratic extensions in the context of pairings, as discussed in [12], is generalized to the towering and curve arithmetic performed in the pairing computation. In a sense, this follows a direction opposite to the one taken by other authors. Instead of trying to encode arithmetic so that modular reductions are faster [13,6], we insist on Montgomery reduction and focus our efforts on reducing *the need* of computing reductions. Moreover, for dealing with costly higher-precision additions inserted by lazy reduction, we develop a flexible methodology that keeps intermediate values under Montgomery reduction boundaries and maximizes the use of operations without carry checks. The traditional operation count model is also augmented to take into account modular reductions individually.
- Formulas for point doubling and point addition in Jacobian and homogeneous coordinates are carefully optimized by eliminating several commonly neglected operations that are not inexpensive on modern 64-bit platforms.
- The computation of the final exponentiation is improved with a new set of formulas for compressed squaring and efficient decompression in cyclotomic subgroups, and an arithmetic trick to remove a significant penalty incurred when computing pairings over curves parameterized by negative integers.

The described techniques produce significant savings, allowing our illustrative software implementation to compute a pairing under 2 million cycles and improve the state-of-the-art timings by 28%-34% on several different 64-bit computing platforms. Even though the techniques are applied on pairings over BN curves at the 128-bit security level, they can be easily extended to other settings using different curves and higher security levels [14].

This paper is organized as follows. Section 2 gives an overview of Miller's Algorithm when employed for computing the Optimal Ate pairing over Barreto-Naehrig curves. Section 3 presents the generalized lazy reduction technique and its application to the improvement of towering arithmetic performance. Different optimizations to curve arithmetic, including the application of lazy reduction,

are discussed in Section 4. Section 5 describes our improvements on the final exponentiation. Section 6 summarizes operation counts and Section 7 describes our high-speed software implementation and comparison of results with the previously fastest implementation in the literature. Section 8 concludes the paper.

2 Preliminaries

An *admissible bilinear pairing* is a non-degenerate efficiently-computable map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups of points in an elliptic curve E and $\mathbb{G}_T$ is a subgroup of the multiplicative group of a finite field. The core property of map e is linearity in both arguments, allowing the construction of novel cryptographic schemes with security relying on the hardness of the Discrete Logarithm Problem in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$.

Barreto and Naehrig [11] described a parameterized family of elliptic curves $E_b : y^2 = x^3 + b, b \neq 0$ over a prime field $\mathbb{F}_p$, $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$, with prime order $n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$, where $u \in \mathbb{Z}$ is an arbitrary integer. This family is rather large and easy to generate [10], providing a multitude of parameter choices; and, having embedding degree $k = 12$, is well-suited for computing asymmetric pairings at the 128-bit security level [12]. It admits several optimal derivations [8] of different variants of the Ate pairing [15] such as R-ate [16], Optimal Ate [8] and χ-ate [17].

Let $E[n]$ be the subgroup of n-torsion points of E and $E' : y^2 = x^3 + b/\xi$ be a sextic twist of E with ξ not a cube nor a square in $\mathbb{F}_{p^2}$. For the clear benefit of direct benchmarking, but also pointing that performance among variants is roughly the same, we restrict the discussion to computing the Optimal Ate pairing defined as in [6]:

$$a_{opt} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$$
$$(Q, P) \to (f_{r,Q}(P) \cdot l_{[r]Q,\pi_p(Q)}(P) \cdot l_{[r]Q+\pi_p(Q),-\pi_p^2(Q)}(P))^{\frac{p^{12}-1}{n}},$$

where $r = 6u + 2 \in \mathbb{Z}$; the map $\pi_p : E \to E$ is the Frobenius endomorphism $\pi_p(x, y) = (x^p, y^p)$; groups $\mathbb{G}_1, \mathbb{G}_2$ are determined by the eigenspaces of π_p as $\mathbb{G}_1 = E[n] \cap \mathrm{Ker}(\pi_p - [1]) = E(\mathbb{F}_p)[n]$ and $\mathbb{G}_2$ as the preimage $E'(\mathbb{F}_{p^2})[n]$ of $E[n] \cap \mathrm{Ker}(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^{12}})[n]$ under the twisting isomorphism $\psi : E' \to E$; the group $\mathbb{G}_T$ is the subgroup of n-th roots of unity $\mu_n \subset \mathbb{F}^*_{p^{12}}$; $f_{r,Q}(P)$ is a normalized function with divisor $(f_{r,Q}) = r(Q) - ([r]Q) - (r-1)(\mathcal{O})$ and $l_{Q_1,Q_2}(P)$ is the line arising in the addition of Q_1 and Q_2 evaluated at point P.

Miller [18,19] proposed an algorithm that constructs $f_{r,P}$ in stages by using a double-and-add method. When generalizing the denominator-free version [20] of Miller's Algorithm for computing the pairing a_{opt} with the set of implementation-friendly parameters suggested by [10] at the 128-bit security level, we obtain Algorithm 1. For the BN curve we have $E : y^2 = x^3 + 2, u = -(2^{62} + 2^{55} + 1) < 0$. In order to accommodate the negative r (line 9 in Algorithm 1), it is required to compute a cheap negation in $\mathbb{G}_2$ to make the final accumulator T the result of $[-|r|]Q$, and an expensive inversion in the big field $\mathbb{G}_T$ to obtain the correct

pairing value $f_{-|r|,Q}(P) = (f_{|r|,Q}(P))^{-1}$, instead of the value $f_{|r|,Q}(P)$ produced at the end of the algorithm. The expensive inversion will be handled later at Section 5 with the help of the final exponentiation.

Algorithm 1. Optimal Ate pairing on BN curves (generalized for $u < 0$)

Input: $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, r = |6u+2| = \sum_{i=0}^{\log_2(r)} r_i 2^i$
Output: $a_{opt}(Q,P)$

1. $T \leftarrow Q$, $f \leftarrow 1$
2. **for** $i = \lfloor \log_2(r) \rfloor - 1$ **downto** 0 **do**
3. $\quad f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow 2T$
4. $\quad$ **if** $r_i = 1$ **then**
5. $\quad\quad f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$
6. **end for**
7. $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p^2(Q)$
8. **if** $u < 0$ **then**
9. $\quad T \leftarrow -T, f \leftarrow f^{-1}$
10. **end if**
11. $f \leftarrow f \cdot l_{T,Q_1}(P), T \leftarrow T + Q_1$
12. $f \leftarrow f \cdot l_{T,-Q_2}(P), T \leftarrow T - Q_2$
13. $f \leftarrow f^{(p^{12}-1)/n}$
14. **return** f

3 Tower Extension Field Arithmetic

Miller's Algorithm [18,19] employs arithmetic in $\mathbb{F}_{p^{12}}$ during the accumulation steps (lines 3,5,11-12 in Algorithm 1) and at the final exponentiation (line 13 in the same algorithm). Hence, to achieve a high-performance implementation of pairings it is crucial to perform arithmetic over extension fields efficiently. In particular, it has been recommended in [21] to represent $\mathbb{F}_{p^k}$ with a tower of extensions using irreducible binomials. Accordingly, in our targeted setting we represent $\mathbb{F}_{p^{12}}$ using the flexible towering scheme used in [22,5,7,10] combined with the parameters suggested by [10]:

- $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta)$, where $\beta = -1$.
- $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}[s]/(s^2 - \xi)$, where $\xi = 1 + i$.
- $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$, where $\xi = 1 + i$.
- $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^4}[t]/(t^3 - s)$ or $\mathbb{F}_{p^6}[w]/(w^2 - v)$.

It is possible to convert from one towering $\mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^6} \rightarrow \mathbb{F}_{p^{12}}$ to the other $\mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^4} \rightarrow \mathbb{F}_{p^{12}}$ by simply permuting the order of coefficients. The choice $p \equiv 3 \pmod 4$ accelerates arithmetic in $\mathbb{F}_{p^2}$, since multiplications by $\beta = -1$ can be computed as simple subtractions [10].

3.1 Lazy Reduction for Tower Fields

The concept of lazy reduction goes back to at least [23] and has been advantageously exploited by many works in different scenarios [24,25,12]. Lim and Hwang [24] showed that multiplication in $\mathbb{F}_{p^k}$, when $\mathbb{F}_{p^k} = \mathbb{F}_p[x]/(x^k - w)$ is seen as a direct extension over $\mathbb{F}_p$ via the irreducible binomial $(x^k - w)$ with $w \in \mathbb{F}_p$, can be performed with k reductions modulo p. In contrast, it would normally require either k^2 reductions using conventional multiplication, or $k(k+1)/2$ reductions using Karatsuba multiplication. Lazy reduction was first employed in the context of pairing computation by [12] to eliminate reductions in $\mathbb{F}_{p^2}$ multiplication. If one considers the tower $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^6} \to \mathbb{F}_{p^{12}}$, then this approach requires $2 \cdot 6 \cdot 3 = 36$ reductions modulo p, and $3 \cdot 6 \cdot 3 = 54$ integer multiplications for performing one multiplication in $\mathbb{F}_{p^{12}}$; see [12,5,7].

In this section, we generalize the lazy reduction technique to towering-friendly fields $\mathbb{F}_{p^k}$, $k = 2^i 3^j$, $i \geq 1, j \geq 0$, conveniently built with irreducible binomials [26]. We show that multiplication (and squaring) in a tower extension $\mathbb{F}_{p^k}$ only requires k reductions and still benefits from different arithmetic optimizations available in the literature to reduce the number of subfield multiplications or squarings. For instance, with our approach one now requires $2 \cdot 3 \cdot 2 = 12$ reductions modulo p and 54 integer multiplications using the tower $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^6} \to \mathbb{F}_{p^{12}}$ to compute one multiplication in $\mathbb{F}_{p^{12}}$; or 12 reductions modulo p and 36 integer multiplications to compute one squaring in $\mathbb{F}_{p^{12}}$. Although wider in generality, these techniques are analyzed in detail in the context of Montgomery multiplication and Montgomery reduction [27], which are commonly used in the context of pairings over ordinary curves. We explicitly state our formulas for the towering construction $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^6} \to \mathbb{F}_{p^{12}}$ in Section 3.3. To remove ambiguity, the term *reduction modulo p* always refers to modular reduction of double-precision integers.

Theorem 1. *Let $k = 2^i 3^j$, $i, j \in \mathbb{Z}$ and $i \geq 1, j \geq 0$. Let*

$$\mathbb{F}_p = \mathbb{F}_{p^{k_0}} \to \mathbb{F}_{p^{k_1}} = \mathbb{F}_{p^2} \to \cdots \to \mathbb{F}_{p^{k_{i+j-2}}} \to \mathbb{F}_{p^{k_{i+j-1}}} \to \mathbb{F}_{p^{k_{i+j}}} = \mathbb{F}_{p^k}$$

be a tower extension, where each extension $\mathbb{F}_{p^{k_{\ell+1}}}/\mathbb{F}_{p^{k_\ell}}$ is of degree either 2 or 3, which can be constructed using a second degree irreducible binomial $x^2 - \beta_\ell$, $\beta_\ell \in \mathbb{F}_{p^{k_\ell}}$, or a third degree irreducible binomial $x^3 - \beta_\ell$, $\beta_\ell \in \mathbb{F}_{p^{k_\ell}}$, respectively. Suppose that β_ℓ can be chosen such that, for all $a \in \mathbb{F}_{p^{k_\ell}}$, $a \cdot \beta_\ell$ can be computed without any reduction modulo p. Then multiplication in $\mathbb{F}_{p^k}$ can be computed with $3^i 6^j$ integer multiplications and $k = 2^i 3^j$ reductions modulo p for any k.

Proof. We prove this by induction on $i + j$. The base case is $i + j = 1$ ($i = 1$ and $j = 0$). That is, $k = 2$, and we have a tower $\mathbb{F}_p \to \mathbb{F}_{p^2}$ with $\mathbb{F}_{p^2} = \mathbb{F}_p[x]/(x^2 - \beta)$. For any $a = a_0 + a_1 x, b = b_0 + b_1 x \in \mathbb{F}_{p^2}$, $a_i, b_i \in \mathbb{F}_p$, we can write

$$a \cdot b = (a_0 b_0 + a_1 b_1 \beta) + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1)x,$$

which can be computed with 3 integer multiplications and 2 reductions modulo p (note that we ignore multiplication by β, by our assumption).

Next, consider

$$\mathbb{F}_p \rightarrow \mathbb{F}_{p^2} \rightarrow \cdots \rightarrow \mathbb{F}_{p^{k_{i+j}}} \rightarrow \mathbb{F}_{p^{k_{i+j+1}}},$$

where $k_{i+j+1} = 2^{i+1}3^j$, or $k_{i+j+1} = 2^i 3^{j+1}$. In the former case, let $\mathbb{F}_{p^{k_{i+j+1}}} = \mathbb{F}_{p^{k_{i+j}}}[x]/(x^2-\beta)$ and $a = a_0+a_1x$, $b = b_0+b_1x \in \mathbb{F}_{p^{k_{i+j+1}}}$, $a_i, b_i \in \mathbb{F}_{p^{k_{i+j}}}$. Then

$$a \cdot b = (a_0b_0 + a_1b_1\beta) + [(a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1]\, x, \tag{1}$$

which can be computed with 3 multiplications in $\mathbb{F}_{p^{k_{i+j}}}$, namely a_0b_0, $a_1b_1\beta$ and $(a_0 + a_1)(b_0 + b_1)$ (again, we ignore multiplication by β). By the induction hypothesis, each multiplication in $\mathbb{F}_{p^{k_{i+j}}}$ requires 3^i6^j integer multiplications, and 2^i3^j reductions modulo p. Also, three reductions modulo p, when computing a_0b_0, $a_1b_1\beta$ and $(a_0+a_1)(b_0+b_1)$, can be minimized to two reductions modulo p (see (1)). Hence, multiplication in $\mathbb{F}_{p^{k_{i+j+1}}}$ can be computed with $3{\cdot}3^i6^j = 3^{i+1}6^j$ integer multiplications and $2 \cdot 2^i3^j = 2^{i+1}3^j$ reductions modulo p.

The latter case, $k_{i+j+1} = 2^i3^{j+1}$, can be proved similarly, by considering $\mathbb{F}_{p^{k_{i+j+1}}} = \mathbb{F}_{p^{k_{i+j}}}[x]/(x^3 - \beta)$, and the Karatsuba multiplication formula for degree 3 extensions instead of (1). □

It is also straightforward to generalize the procedure above to any formula other than Karatsuba which also involves only sums (or subtractions) of products of the form $\sum \pm a_ib_j$, with $a_i, b_j \in \mathbb{F}_{p^{k_l}}$, such as complex squaring or the Chung-Hasan asymmetric squaring formulas [28].

For efficiency purposes, we suggest a different treatment for the highest layer in the tower arithmetic. Theorem 1 implies that reductions can be completely delayed to the end of the last layer by applying lazy reduction, but in some cases (when the optimal k is already reached and no reductions can be saved) it will be more efficient to perform reductions immediately after multiplications or squarings. This will be illustrated with the computation of squaring in $\mathbb{F}_{p^{12}}$ in Section 3.3.

In the Miller Loop, reductions can also be delayed from the underlying $\mathbb{F}_{p^2}$ field during multiplication and squaring to the arithmetic layer immediately above (i.e., the point arithmetic and line evaluation). Similarly to the tower extension, on this upper layer reductions should only be delayed in the cases where this technique leads to fewer reductions. For details, see Section 4.

There are some penalties when delaying reductions. In particular, *single-precision* operations (with operands occupying $n = \lceil \lceil \log_2 p \rceil / w \rceil$ words, where w is the computer word-size) are replaced by *double-precision* operations (with operands occupying $2n$ words). However, this disadvantage can be minimized in terms of speed by selecting a field size smaller than the word-size boundary because this technique can be exploited more extensively for optimizing double-precision arithmetic.

3.2 Selecting a Field Size Smaller Than the Word-Size Boundary

If the modulus p is selected so that $l = \lceil \log_2 p \rceil < N$, where $N = n \cdot w$, n is the exact number of words required to represent p, i.e., $n = \lceil l/w \rceil$, and w is

the computer word-size, then several consecutive additions without carry-out in the most significant word (MSW) can be performed before a multiplication of the form $c = a \cdot b$, where $a, b \in [0, 2^N - 1]$ such that $c < 2^{2N}$. In the case of Montgomery reduction, the restriction is given by the upper bound $c < 2^N \cdot p$. Similarly, when delaying reductions the result of a multiplication without reduction has maximum value $(p-1)^2 < 2^{2N}$ (assuming that $a, b \in [0, p]$) and several consecutive double-precision additions without carry-outs in the MSW (and, in some cases, subtractions without borrow-outs in the MSW) can be performed before reduction. When using Montgomery reduction up to $\sim \lfloor 2^N/p \rfloor$ additions can be performed without carry checks.

Furthermore, cheaper single- and double-precision operations exploiting this "extra room" can be combined for maximal performance. The challenge is to optimally balance their use in the tower arithmetic since both may interfere with each other. For instance, if intermediate values are allowed to grow up to $2p$ before multiplication (instead of p) then the maximum result would be $4p^2$. This strategy makes use of cheaper single-precision additions without carry checks but limits the number of double-precision additions that can be executed without carry checks after multiplication with delayed reduction. As it will be evident later, to maximize the gain obtained with the proposed methodology one should take into account relative costs of operations and maximum bounds.

In the case of double-precision arithmetic, different optimizing alternatives are available. Let us analyze them in the context of Montgomery arithmetic. First, as pointed out by [7], if $c > 2^N \cdot p$, where c is the result of a double-precision addition, then c can be restored with a cheaper single-precision subtraction by $2^N \cdot p$ (note that the first half of this value consists of zeroes only). Second, different options are available to convert negative numbers to positive after double-precision subtraction. In particular, let us consider the computation $c = a + l \cdot b$, where $a, b \in [0, mp^2]$, $m \in \mathbb{Z}^+$ and $l < 0 \in \mathbb{Z}$ s.t. $|lmp| < 2^N$, which is a recurrent operation (for instance, when $l = \beta$). For this operation, we have explored the following alternatives, which can be integrated in the tower arithmetic with different advantages:

Option 1: $r = c + (2^N \cdot p/2^h),\ r \in [0, mp^2 + 2^N \cdot p/2^h],\ h$ is a small integer s.t. $\left|lmp^2\right| < 2^N \cdot p/2^h < 2^N \cdot p - mp^2$.
Option 2: if $c < 0$ then $r = c + 2^N \cdot p,\ r \in [0, 2^N \cdot p]$.
Option 3: $r = c - lmp^2,\ r \in [0, (|l| + 1)mp^2]$, s.t. $(|l| + 1)mp < 2^N$.
Option 4: if $c < 0$ then $r = c - lmp^2, r \in [0, |lmp^2|]$.

In particular, Options 2 and 4 require conditional checks that make the corresponding operations more expensive. Nevertheless, these options may be valuable when negative values cannot be corrected with other options without violating the upper bound. Also note that Option 2 can make use of a cheaper single-precision subtraction for converting negative results to positive. Options 1 and 3 are particularly efficient because no conditional checks are required. Moreover, if l is small enough (and h maximized for Option 1) several following operations can avoid carry checks. Between both, Option 1 is generally more efficient

because adding $2^N \cdot p/2^h$ requires less than double-precision if $h \leq w$, where w is the computer word-size.

Next, we demonstrate how the different design options discussed in this section can be exploited with a clever selection of parameters and applied to different operations combining single- and double-precision arithmetic to speed up the extension field arithmetic.

3.3 Analysis for Selected Parameters

For our illustrative analysis, we use the tower $\mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^6} \rightarrow \mathbb{F}_{p^{12}}$ constructed with the irreducible binomials described at the beginning of this section. When targeting the 128-bit security level, single- and double-precision operations are defined by operands with sizes $N = 256$ and $2N = 512$, respectively. For our selected prime, $\lceil \log_2 p \rceil = 254$ and $2^N \cdot p \approx 6.8p^2$. Notation is fixed as following: (i) $+, -, \times$ are operators not involving carry handling or modular reduction for boundary keeping; (ii) $\oplus, \ominus, \otimes$ are operators producing reduced results through carry handling or modular reduction; (iii) a superscript in an operator is used to denote the extension degree involved in the operation; (iv) notation $a_{i,j}$ is used to address j-th subfield element in extension field element a_i; (v) lower case t and upper case T variables represent single- and double-precision integers or extension field elements composed of single and double-precision integers, respectively. The precision of the operators is determined by the precision of the operands and result. Note that, as stated before, if $c > 2^N \cdot p$ after adding $c = a + b$ in double-precision, we correct the result by computing $c - 2^N \cdot p$. Similar to subtraction, we refer to the latter as "Option 2".

The following notation is used for the cost of operations: (i) m, s, a denote the cost of multiplication, squaring and addition in $\mathbb{F}_p$, respectively; (ii) $\tilde{m}, \tilde{s}, \tilde{a}, \tilde{i}$ denote the cost of multiplication, squaring, addition and inversion in $\mathbb{F}_{p^2}$, respectively; (iii) m_u, s_u, r denote the cost of unreduced multiplication and squaring producing double-precision results, and modular reduction of double-precision integers, respectively; (iv) $\tilde{m}_u, \tilde{s}_u, \tilde{r}$ denote the cost of unreduced multiplication and squaring, and modular reduction of double-precision elements in $\mathbb{F}_{p^2}$, respectively. For the remainder of the paper, and unless explicitly stated otherwise, we assume that double-precision addition has the cost of $2a$ and $2\tilde{a}$ in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$, respectively, which approximately follows what we observe in practice.

We will now illustrate a selection of operations for efficient multiplication in $\mathbb{F}_{p^{12}}$, beginning with multiplication in $\mathbb{F}_{p^2}$. Let $a, b, c \in \mathbb{F}_{p^2}$ such that $a = a_0 + a_1 i, b = b_0 + b_1 i, c = a \cdot b = c_0 + c_1 i$. The required operations for computing $\mathbb{F}_{p^2}$ multiplication are detailed in Algorithm 2. As explained in Beuchat *et al.* [7, Section 5.2], when using the Karatsuba method and $a_i, b_i \in \mathbb{F}_p$, $c_1 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1 = a_0 b_1 + a_1 b_0 < 2p^2 < 2^N \cdot p$, additions are single-precision, reduction after multiplication can be delayed and hence subtractions are double-precision (steps 1-3 in Algorithm 2). Obviously, these operations do not require carry checks. For $c_0 = a_0 b_0 - a_1 b_1$, c_0 is in interval $[-p^2, p^2]$ and a negative result can be converted to positive using **Option 1** with $h = 2$ or **Option 2**, for which the final c_0 is in the range $[0, (2^N \cdot p/4) + p^2] \subset [0, 2^N \cdot p]$ or

$[0, 2^N \cdot p]$, respectively (step 4 in Algorithm 2). Following Theorem 1, all reductions can be completely delayed to the next arithmetic layer (higher extension or curve arithmetic).

Algorithm 2. Multiplication in $\mathbb{F}_{p^2}$ without reduction ($\times^2$, cost $\tilde{m}_u = 3m_u + 8a$)

Input: $a = (a_0 + a_1 i)$ and $b = (b_0 + b_1 i) \in \mathbb{F}_{p^2}$
Output: $c = a \cdot b = (c_0 + c_1 i) \in \mathbb{F}_{p^2}$

1. $T_0 \leftarrow a_0 \times b_0$, $T_1 \leftarrow a_1 \times b_1$, $t_0 \leftarrow a_0 + a_1$, $t_1 \leftarrow b_0 + b_1$
2. $T_2 \leftarrow t_0 \times t_1$, $T_3 \leftarrow T_0 + T_1$
3. $T_3 \leftarrow T_2 - T_3$
4. $T_4 \leftarrow T_0 \ominus T_1$ (Option 1 or 2)
5. **return** $c = (T_4 + T_3 i)$

Let us now define multiplication in $\mathbb{F}_{p^6}$. Let $a, b, c \in \mathbb{F}_{p^6}$ such that $a = (a_0 + a_1 v + a_2 v^2), b = (b_0 + b_1 v + b_2 v^2), c = a \cdot b = (c_0 + c_1 v + c_2 v^2)$. The required operations for computing $\mathbb{F}_{p^6}$ multiplication are detailed in Algorithm 3. In this case, $c_0 = v_0 + \xi[(a_1 + a_2)(b_1 + b_2) - v_1 - v_2], c_1 = (a_0 + a_1)(b_0 + b_1) - v_0 - v_1 + \xi v_2$ and $c_2 = (a_0 + a_2)(b_0 + b_2) - v_0 - v_2 + v_1$, where $v_0 = a_0 b_0, v_1 = a_1 b_1$ and $v_2 = a_2 b_2$. First, note that the pattern $s_x = (a_i + a_j)(b_i + b_j) - v_i - v_j$ repeats for each $c_x, 0 \leq x \leq 2$. After multiplications using Alg. 2 with **Option 1** ($h = 2$), we have $v_{i,0}, v_{j,0} \in [0, (2^N \cdot p/4) + p^2]$ and $v_{i,1}, v_{j,1} \in [0, 2p^2]$ (step 1 of Alg. 3). Outputs of single-precision additions of the forms $(a_i + a_j)$ and $(b_i + b_j)$ are in the range $[0, 2p]$ and hence do not produce carries (steps 2, 9 and 17 of Alg. 3). Corresponding $\mathbb{F}_{p^2}$ multiplications $r_x = (a_i + a_j)(b_i + b_j)$ using Alg. 2 with **Option 2** give results in the ranges $r_{x,0} \in [0, 2^N \cdot p]$ and $r_{x,1} \in [0, 8p^2]$ (steps 3, 10 and 18). Although $max(r_{x,1}) = 8p^2 > 2^N \cdot p$, note that $8p^2 < 2^{2N}$ and $s_{x,1} = a_{i,0} b_{j,1} + a_{i,1} b_{j,0} + a_{j,0} b_{i,1} + a_{j,1} b_{i,0} \in [0, 4p^2]$ since $s_x = a_i b_j + a_j b_i$. Hence, for $0 \leq x \leq 2$, double-precision subtractions for computing $s_{x,1}$ using Karatsuba do not require carry checks (steps 4 and 6, 11 and 13, 19 and 21). For computing $s_{x,0} = r_{x,0} - (v_{i,0} + v_{j,0})$, addition does not require carry check (output range $[0, 2(2^N \cdot p/4 + p^2)] \subset [0, 2^N \cdot p]$) and subtraction gives result in the range $[0, 2^N \cdot p]$ when using **Option 2** (steps 5, 12 and 20). For computing c_0, multiplication by ξ, i.e., $S_0 = \xi s_0$ involves the operations $S_{0,0} = s_{0,0} - s_{0,1}$ and $S_{0,1} = s_{0,0} + s_{0,1}$, which are computed in double-precision using **Option 2** to get the output range $[0, 2^N \cdot p]$ (step 7). Similarly, final additions with v_0 require **Option 2** to get again the output range $[0, 2^N \cdot p]$ (step 8). For computing c_1, $S_1 = \xi v_2$ is computed as $S_{1,0} = v_{2,0} - v_{2,1}$ and $S_{1,1} = v_{2,0} + v_{2,1}$, where the former requires a double-precision subtraction using **Option 1** ($h = 1$) to get a result in the range $[0, 2^N \cdot p/2 + 2^N \cdot p/4 + p^2] \subset [0, 2^N \cdot p]$ (step 14) and the latter requires a double-precision addition with no carry check to get a result in the range $[0, (2^N \cdot p/4) + 3p^2] \subset [0, 2^N \cdot p]$ (step 15). Then, $c_{1,0} = s_{1,0} + S_{1,0}$ and $c_{1,1} = s_{1,1} + S_{1,1}$ involve double-precision additions using **Option 2** to obtain results in the range $[0, 2^N \cdot p]$ (step 16). Results $c_{2,0} = s_{2,0} + v_{1,0}$ and $c_{2,1} = s_{2,1} + v_{1,1}$ require a double-precision addition using **Option 2** (final output range $[0, 2^N \cdot p]$, step 22) and a double-precision addition without carry

check (final output range $[0, 6p^2] \subset [0, 2^N \cdot p]$, step 23), respectively. Modular reductions have been delayed again to the last layer $\mathbb{F}_{p^{12}}$.

Algorithm 3. Multiplication in $\mathbb{F}_{p^6}$ without reduction ($\times^6$, cost of $6\tilde{m}_u + 28\tilde{a}$)

Input: $a = (a_0 + a_1 v + a_2 v^2)$ and $b = (b_0 + b_1 v + b_2 v^2) \in \mathbb{F}_{p^6}$
Output: $c = a \cdot b = (c_0 + c_1 v + c_2 v^2) \in \mathbb{F}_{p^6}$

1. $T_0 \leftarrow a_0 \times^2 b_0$, $T_1 \leftarrow a_1 \times^2 b_1$, $T_2 \leftarrow a_2 \times^2 b_2$ (Option 1, $h = 2$)
2. $t_0 \leftarrow a_1 +^2 a_2$, $t_1 \leftarrow b_1 +^2 b_2$
3. $T_3 \leftarrow t_0 \times^2 t_1$ (Option 2)
4. $T_4 \leftarrow T_1 +^2 T_2$
5. $T_{3,0} \leftarrow T_{3,0} \ominus T_{4,0}$ (Option 2)
6. $T_{3,1} \leftarrow T_{3,1} - T_{4,1}$
7. $T_{4,0} \leftarrow T_{3,0} \ominus T_{3,1}$, $T_{4,1} \leftarrow T_{3,0} \oplus T_{3,1}$ ($\equiv T_4 \leftarrow \xi \cdot T_3$) (Option 2)
8. $T_5 \leftarrow T_4 \oplus^2 T_0$ (Option 2)
9. $t_0 \leftarrow a_0 +^2 a_1$, $t_1 \leftarrow b_0 +^2 b_1$
10. $T_3 \leftarrow t_0 \times^2 t_1$ (Option 2)
11. $T_4 \leftarrow T_0 +^2 T_1$
12. $T_{3,0} \leftarrow T_{3,0} \ominus T_{4,0}$ (Option 2)
13. $T_{3,1} \leftarrow T_{3,1} - T_{4,1}$
14. $T_{4,0} \leftarrow T_{2,0} \ominus T_{2,1}$ (Option 1, $h = 1$)
15. $T_{4,1} \leftarrow T_{2,0} + T_{2,1}$ (steps 14-15 $\equiv T_4 \leftarrow \xi \cdot T_2$)
16. $T_6 \leftarrow T_3 \oplus^2 T_4$ (Option 2)
17. $t_0 \leftarrow a_0 +^2 a_2$, $t_1 \leftarrow b_0 +^2 b_2$
18. $T_3 \leftarrow t_0 \times^2 t_1$ (Option 2)
19. $T_4 \leftarrow T_0 +^2 T_2$
20. $T_{3,0} \leftarrow T_{3,0} \ominus T_{4,0}$ (Option 2)
21. $T_{3,1} \leftarrow T_{3,1} - T_{4,1}$
22. $T_{7,0} \leftarrow T_{3,0} \oplus T_{1,0}$ (Option 2)
23. $T_{7,1} \leftarrow T_{3,1} + T_{1,1}$
24. **return** $c = (T_5 + T_6 v + T_7 v^2)$

Finally, let $a, b, c \in \mathbb{F}_{p^{12}}$ such that $a = a_0 + a_1 w, b = b_0 + b_1 w, c = a \cdot b = c_0 + c_1 w$. Algorithm 4 details the required operations for computing multiplication. In this case, $c_1 = (a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0$. At step 1, $\mathbb{F}_{p^6}$ multiplications $a_0 b_0$ and $a_1 b_1$ give outputs in range $\subset [0, 2^N \cdot p]$ using Algorithm 3. Additions $a_0 + a_1$ and $b_0 + b_1$ are single-precision reduced modulo p so that multiplication $(a_0 + a_1)(b_0 + b_1)$ in step 2 gives output in range $\subset [0, 2^N \cdot p]$ using Algorithm 3. Then, subtractions by $a_1 b_1$ and $a_0 b_0$ use double-precision operations with **Option 2** to have an output range $[0, 2^N \cdot p]$ so that we can apply Montgomery reduction at step 5 to obtain the result modulo p. For $c_0 = a_0 b_0 + v a_1 b_1$, multiplication by v, i.e., $T = v \cdot v_1$, where $v_i = a_i b_i$, involves the double-precision operations $T_{0,0} = v_{2,0} - v_{2,1}, T_{0,1} = v_{2,0} + v_{2,1}, T_1 = v_0$ and $T_2 = v_1$, all performed with **Option 2** to obtain the output range $[0, 2^N \cdot p]$ (steps 6-7). Final addition with $a_0 b_0$ uses double-precision with **Option 2** again so that we can apply Montgomery reduction at step 9 to obtain the result modulo p. We remark that, by applying the lazy reduction technique using the operation sequence above, we

have reduced the number of reductions in $\mathbb{F}_{p^6}$ from 3 to only 2, or the number of total modular reductions in $\mathbb{F}_p$ from 54 (or 36 if lazy reduction is employed in $\mathbb{F}_{p^2}$) to only $k = 12$.

Algorithm 4. Multiplication in $\mathbb{F}_{p^{12}}$ ($\times^{12}$, cost of $18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}$)

Input: $a = (a_0 + a_1 w)$ and $b = (b_0 + b_1 w) \in \mathbb{F}_{p^{12}}$
Output: $c = a \cdot b = (c_0 + c_1 w) \in \mathbb{F}_{p^{12}}$
1. $T_0 \leftarrow a_0 \times^6 b_0$, $T_1 \leftarrow a_1 \times^6 b_1$, $t_0 \leftarrow a_0 \oplus^6 a_1$, $t_1 \leftarrow b_0 \oplus^6 b_1$
2. $T_2 \leftarrow t_0 \times^6 t_1$
3. $T_3 \leftarrow T_0 \oplus^6 T_1$ (Option 2)
4. $T_2 \leftarrow T_2 \ominus^6 T_3$ (Option 2)
5. $c_1 \leftarrow T_2 \bmod^6 p$
6. $T_{2,0,0} \leftarrow T_{1,2,0} \ominus T_{1,2,1}$, $T_{2,0,1} \leftarrow T_{1,2,0} \oplus T_{1,2,1}$ (Option 2)
7. $T_{2,1} \leftarrow T_{1,0}$, $T_{2,2} \leftarrow T_{1,1}$ (steps 6-7 $\equiv T_2 \leftarrow v \cdot T_1$)
8. $T_2 \leftarrow T_0 \oplus^6 T_2$ (Option 2)
9. $c_0 \leftarrow T_2 \bmod^6 p$
10. **return** $c = (c_0 + c_1 w)$

As previously stated, there are situations when it is more efficient to perform reductions right after multiplications and squarings in the last arithmetic layer of the tower construction. We illustrate the latter with squaring in $\mathbb{F}_{p^{12}}$. As shown in Algorithm 5, a total of 2 reductions in $\mathbb{F}_{p^6}$ are required when performing $\mathbb{F}_{p^6}$ multiplications in step 4. If lazy reduction was applied, the number of reductions would stay at 2, and worse, the total cost would be increased because some operations would require double-precision. The reader should note that the approach suggested by [10], where the formulas in [28] are employed for computing squarings in internal cubic extensions of $\mathbb{F}_{p^{12}}$, saves $1\tilde{m}$ in comparison with Algorithm 5. However, we experimented such approach with several combinations of formulas and towering, and it remained consistently slower than Algorithm 5 due to an increase in the number of additions.

Algorithm 5. Squaring in $\mathbb{F}_{p^{12}}$ (cost of $12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}$)

Input: $a = (a_0 + a_1 w) \in \mathbb{F}_{p^{12}}$
Output: $c = a^2 = (c_0 + c_1 w) \in \mathbb{F}_{p^{12}}$
1. $t_0 \leftarrow a_0 \oplus^6 a_1, t_{1,0,0} \leftarrow a_{1,2,0} \ominus a_{1,2,1}$, $t_{1,0,1} \leftarrow a_{1,2,0} \oplus a_{1,2,1}$
2. $t_{1,1} \leftarrow a_{1,0}$, $t_{1,2} \leftarrow a_{1,1}$ (steps 2-3 $\equiv t_1 \leftarrow v \cdot a_1$)
3. $t_1 \leftarrow a_0 \oplus^6 t_1$
4. $c_1 \leftarrow (a_0 \times^6 a_1) \bmod^6 p$, $t_0 \leftarrow (t_0 \times^6 t_1) \bmod^6 p$
5. $t_{1,0,0} \leftarrow c_{1,2,0} \ominus c_{1,2,1}$, $t_{1,0,1} \leftarrow c_{1,2,0} \oplus c_{1,2,1}$
6. $t_{1,1} \leftarrow c_{1,0}$, $t_{1,2} \leftarrow c_{1,1}$ (steps 6-7 $\equiv t_1 \leftarrow v \cdot c_1$)
7. $t_1 \leftarrow t_1 \oplus^6 c_1$
8. $c_0 \leftarrow t_0 \ominus^6 t_1$, $c_1 \leftarrow c_1 \oplus^6 c_1$
9. **return** $c = (c_0 + c_1 w)$

4 Miller Loop

In this section, we present our optimizations to the curve arithmetic. To be consistent with other results in the literature, we do not distinguish between simple- and double-precision additions in the formulas below.

Recently, Costello *et al.* [9, Section 5] proposed the use of homogeneous coordinates to perform the curve arithmetic entirely on the twist. Their formula for computing a point doubling and line evaluation costs $2\tilde{m}+7\tilde{s}+23\tilde{a}+4m+1m_{b'}$. The twisting of point P, given in our case by $(x_P/w^2, y_P/w^3) = (\frac{x_P}{\xi}v^2, \frac{y_P}{\xi}vw)$, is eliminated by multiplying the whole line evaluation by ξ and relying on the final exponentiation to eliminate this extra factor [9]. Clearly, the main drawback of this formula is the high number of additions. We present the following revised formula:

$$X_3 = \frac{X_1Y_1}{2}\left(Y_1^2 - 9b'Z_1^2\right), \quad Y_3 = \left[\frac{1}{2}\left(Y_1^2 + 9b'Z_1^2\right)\right] - 27b'^2Z_1^4, \quad Z_3 = 2Y_1^3Z_1,$$
$$l = (-2Y_1Z_1y_P)vw + \left(3X_1^2x_P\right)v^2 + \xi\left(3b'Z_1^2 - Y_1^2\right). \tag{2}$$

This doubling formula gives the cost of $3\tilde{m}+6\tilde{s}+17\tilde{a}+4m+m_{b'}+m_{\xi}$. Moreover, if the parameter b' is cleverly selected as in [10], multiplication by b' can be performed with minimal number of additions and subtractions. For instance, if one fixes $b = 2$ then $b' = 2/(1+i) = 1-i$. Accordingly, the following execution has a cost of $3\tilde{m} + 6\tilde{s} + 19\tilde{a} + 4m$ (note that computations for E and $l_{0,0}$ are over $\mathbb{F}_p$ and $\overline{y_P} = -y_P$ is precomputed):

$$A = X_1 \cdot Y_1/2, \quad B = Y_1^2, \quad C = Z_1^2, \quad D = 3C, \quad E_0 = D_0 + D_1,$$
$$E_1 = D_1 - D_0, \quad F = 3E, \quad X_3 = A \cdot (B - F), \quad G = (B + F)/2,$$
$$Y_3 = G^2 - 3E^2, \quad H = (Y_1 + Z_1)^2 - (B + C), \tag{3}$$
$$Z_3 = B \cdot H, \quad I = E - B, \quad J = X_1^2$$
$$l_{0,0,0} = I_0 - I_1, \quad l_{0,0,1} = I_0 + I_1, \quad l_{1,1} = H \cdot \overline{y_P}, \quad l_{0,2} = 3J \cdot x_P.$$

We point out that in practice we have observed that $\tilde{m} - \tilde{s} \approx 3\tilde{a}$. Hence, it is more efficient to compute X_1Y_1 directly than using $(X_1 + Y_1)^2, B$ and J. If this was not the case, the formula could be computed with cost $2\tilde{m}+7\tilde{s}+23\tilde{a}+4m$.

Remarkably, the technique proposed in Section 3 for delaying reductions can also be applied to the point arithmetic over a quadratic extension field. Reductions can be delayed to the end of each $\mathbb{F}_{p^2}$ multiplication/squaring and then delayed further for those sums of products that allow reducing the number of reductions. Although not plentiful (given the nature of most curve arithmetic formulas which have consecutive and redundant multiplications/squarings), there are a few places where this technique can be applied. For instance, doubling formula (2) requires 25 $\mathbb{F}_p$ reductions (3 per $\mathbb{F}_{p^2}$ multiplication using Karatsuba, 2 per $\mathbb{F}_{p^2}$ squaring and 1 per $\mathbb{F}_p$ multiplication). First, by delaying reductions inside $\mathbb{F}_{p^2}$ arithmetic the number of reductions per multiplication goes down to only 2, with 22 reductions in total. Moreover, reductions corresponding to G^2

and $3E^2$ in Y_3 (see execution (3)) can be further delayed and merged, eliminating the need of two reductions. In total, the number of reductions is now 20. Similar optimizations can be applied to other point/line evaluation formulas (see extended version [29] for optimizations to formulas using Jacobian and homogeneous coordinates).

For accumulating line evaluations into the Miller variable, $\mathbb{F}_{p^{12}}$ is represented using the towering $\mathbb{F}_{p^2} \rightarrow \mathbb{F}_{p^4} \rightarrow \mathbb{F}_{p^{12}}$ and a special (dense×sparse)-multiplication costing $13\tilde{m}_u + 6\tilde{r} + 61\tilde{a}$ is used. During the first iteration of the loop, a squaring in $\mathbb{F}_{p^{12}}$ can be eliminated since the Miller variable is initialized as 1 (line 1 in Algorithm 1) and a special (sparse×sparse) multiplication costing $7\tilde{m}_u + 5\tilde{r} + 30\tilde{a}$ is used to multiply the first two line evaluations, resulting in the revised Algorithm 6. This sparser multiplication is also used for multiplying the two final line evaluations in step 10 of the algorithm.

5 Final Exponentiation

The fastest way known for computing the final exponentiation is described in [30]. The power $\frac{p^{12}-1}{n}$ is factored into an easy exponent (p^6-1) which requires a conjugation and an inversion; another easy exponent (p^2+1) which requires a p^2-power Frobenius and a multiplication; and a hard exponent $(p^4-p^2+1)/n$ which can be performed in the cyclotomic subgroup $\mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})$. For computing this last power, one can write the hard exponent as follows [12]:

$$(p^4 - p^2 + 1)/n = \lambda_3 p^3 + \lambda_2 p^2 + \lambda_1 p + \lambda_0,$$

where

$$\lambda_3(u) = 1\ ,\ \lambda_2(u) = 6u^2 + 1,$$
$$\lambda_1(u) = -36u^3 - 18u^2 - 12u + 1\ ,\ \lambda_0(u) = -36u^3 - 30u^2 - 18u - 2,$$

and compute the individual powers by a multi-addition chain, requiring three consecutive exponentiations by the absolute value of the curve parameter $|u|$, 13 multiplications, 4 squarings, 4 p-power Frobenius, 2 p^2-power Frobenius and a single p^3-power Frobenius in $\mathbb{F}_{p^{12}}$. These powers of Frobenius can be efficiently computed with the formulas in [7]. In the following subsections, we explain how to remove the expensive inversion in $\mathbb{F}_{p^{12}}$ mentioned at the end of Section 2; and how the cyclotomic subgroup structure allows faster compressed squarings and consequently faster exponentiation by $|u|$.

5.1 Removing the Inversion Penalty

From Algorithm 1, the Optimal Ate pairing when $u < 0$ can be computed as

$$a_{opt}(Q,P) = \left[g^{-1} \cdot h\right]^{\frac{p^{12}-1}{n}}, \tag{4}$$

with $r = 6u+2, g = f_{|r|,Q}(P)$ and $h = l_{[-|r|]Q,\pi_p(Q)}(P) \cdot l_{[-|r|Q]+\pi_p(Q),-\pi_p^2(Q)}(P)$. Lemma 1 below allows one to replace the expensive inversion g^{-1} with a simple conjugation with no change in the result. This is depicted in line 9 of Algorithm 6.

Lemma 1. *The pairing $a_{opt}(Q,P)$ can be computed as $\left[g^{p^6}\cdot h\right]^{\frac{p^{12}-1}{n}}$, with g,h defined as above.*

Proof. By distributing the power $(p^{12}-1)/n$ in terms g,h in Equation (4):

$$a_{opt}(Q,P) = g^{\frac{1-p^{12}}{n}}\cdot h^{\frac{p^{12}-1}{n}} = g^{\frac{(1-p^6)(1+p^6)}{n}}\cdot h^{\frac{p^{12}-1}{n}}$$
$$= g^{\frac{(p^{12}-p^6)(1+p^6)}{n}}\cdot h^{\frac{p^{12}-1}{n}} = g^{\frac{p^6(p^6-1)(p^6+1)}{n}}\cdot h^{\frac{p^{12}-1}{n}} = \left[g^{p^6}\cdot h\right]^{\frac{p^{12}-1}{n}} \quad \square$$

5.2 Computing u-th Powers in $\mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})$

Let

$$g=\sum_{i=0}^{2}(g_{2i}+g_{2i+1}s)t^i \in \mathbb{G}_{\phi_6}(\mathbb{F}_{p^2}) \text{ and } g^2=\sum_{i=0}^{2}(h_{2i}+h_{2i+1}s)t^i$$

with $g_i,h_i \in \mathbb{F}_{p^2}$. In [31], it was shown that one can compress g to $\mathcal{C}(g)=[g_2,g_3,g_4,g_5]$, and the compressed representation of g^2 is computed as $\mathcal{C}(g^2)=[h_2,h_3,h_4,h_5]$, where h_i is computed as follows:

$$\begin{aligned} h_2 &= 2(g_2+3\xi B_{4,5}), & h_3 &= 3(A_{4,5}-(\xi+1)B_{4,5})-2g_3,\\ h_4 &= 3(A_{2,3}-(\xi+1)B_{2,3})-2g_4, & h_5 &= 2(g_5+3B_{2,3}), \end{aligned} \tag{5}$$

where $A_{i,j}=(g_i+g_j)(g_i+\xi g_j)$ and $B_{i,j}=g_ig_j$. The above formula requires 4 multiplications in $\mathbb{F}_{p^2}$. Considering the lazy reduction technique discussed in Section 3.3, we propose another formula that is slightly faster and has a cost of $6\tilde{s}_u+4\tilde{r}+31\tilde{a}$. The formula is given as follows:

$$\begin{aligned} h_2 &= 2g_2+3(S_{4,5}-S_4-S_5)\xi, & h_3 &= 3(S_4+S_5\xi)-2g_3,\\ h_4 &= 3(S_2+S_3\xi)-2g_4, & h_5 &= 2g_5+3(S_{2,3}-S_2-S_3), \end{aligned} \tag{6}$$

where $S_{i,j}=(g_i+g_j)^2$ and $S_i=g_i^2$; also see extended version [29] for the correctness of our formula and an explicit implementation.

When g is raised to a power via a square-and-multiply exponentiation algorithm, full representation of elements (decompression) is required because, if $\mathcal{C}$ is used as the compression map, it is not known how to perform multiplication given the compressed representation of elements. Given a compressed representation of $g\in\mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})\setminus\{1\}$, $\mathcal{C}(g)=[g_2,g_3,g_4,g_5]$, the decompression map $\mathcal{D}$ is evaluated as follows (see [31] for more details):

$$\mathcal{D}([g_2,g_3,g_4,g_5])=(g_0+g_1s)+(g_2+g_3s)t+(g_4+g_5s)t^2,$$

$$\begin{cases} g_1=\frac{g_5^2\xi+3g_4^2-2g_3}{4g_2}, \quad g_0=(2g_1^2+g_2g_5-3g_3g_4)\xi+1, & \text{if } g_2\neq 0;\\ g_1=\frac{2g_4g_5}{g_3}, \quad g_0=(2g_1^2-3g_3g_4)\xi+1, & \text{if } g_2=0. \end{cases}$$

In particular, $g^{|u|}$ can be computed in three steps:

1. Compute $\mathcal{C}(g^{2^i})$ for $1 \leq i \leq 62$ using (6) and store $\mathcal{C}(g^{2^{55}})$ and $\mathcal{C}(g^{2^{62}})$.
2. Compute $\mathcal{D}(\mathcal{C}(g^{2^{55}})) = g^{2^{55}}$ and $\mathcal{D}(\mathcal{C}(g^{2^{62}})) = g^{2^{62}}$.
3. Compute $g^{|u|} = g^{2^{62}} \cdot g^{2^{55}} \cdot g$.

Step 1 requires 62 squarings in $\mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})$. Using Montgomery's simultaneous inversion trick [32], Step 2 requires $9\tilde{m} + 6\tilde{s} + 22\tilde{a} + \tilde{i}$. Step 3 requires 2 multiplications in $\mathbb{F}_{p^{12}}$. The total cost is:

$$\begin{aligned} Exp &= 62 \cdot (6\tilde{s}_u + 4\tilde{r} + 31\tilde{a}) + (9\tilde{m} + 6\tilde{s} + 22\tilde{a} + \tilde{i}) + 2 \cdot (18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}) \\ &= 45\tilde{m}_u + 378\tilde{s}_u + 275\tilde{r} + 2164\tilde{a} + \tilde{i}, \end{aligned}$$

Granger-Scott's [33] formula for squaring can be implemented at a cost of $9\tilde{s}_u + 6\tilde{r} + 46\tilde{a}$ if lazy reduction techniques are employed. With this approach, an exponentiation costs:

$$\begin{aligned} Exp' &= 62 \cdot (9\tilde{s}_u + 6\tilde{r} + 46\tilde{a}) + 2 \cdot (18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}) \\ &= 36\tilde{m}_u + 558\tilde{s}_u + 399\tilde{r} + 3072\tilde{a}. \end{aligned}$$

Hence, the faster compressed squaring formulas reduce by 33% the number of squarings and by 30% the number of additions in $\mathbb{F}_{p^2}$.

Algorithm 6. Revised Optimal Ate pairing on BN curves (generalized for $u < 0$).

Input: $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, r = |6u + 2| = \sum_{i=0}^{\log_2(r)} r_i 2^i$
Output: $a_{opt}(Q, P)$

1. $d \leftarrow l_{Q,Q}(P)$, $T \leftarrow 2Q, e \leftarrow 1$
2. **if** $r_{\lfloor \log_2(r) \rfloor - 1} = 1$ **then** $e \leftarrow l_{T,Q}(P), T \leftarrow T + Q$
3. $f \leftarrow d \cdot e$
4. **for** $i = \lfloor \log_2(r) \rfloor - 2$ **downto** 0 **do**
5. $\quad f \leftarrow f^2 \cdot l_{T,T}(P), T \leftarrow 2T$
6. $\quad$ **if** $r_i = 1$ **then** $f \leftarrow f \cdot l_{T,Q}(P), T \leftarrow T + Q$
7. **end for**
8. $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p^2(Q)$
9. **if** $u < 0$ **then** $T \leftarrow -T, f \leftarrow f^{p^6}$
10. $d \leftarrow l_{T,Q_1}(P), T \leftarrow T + Q_1, e \leftarrow l_{T,-Q_2}(P), T \leftarrow T - Q_2$, $f \leftarrow f \cdot (d \cdot e)$
11. $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/n}$
12. **return** f

6 Computational Cost

We now consider all the improvements described in the previous sections and present a detailed operation count. Table 1 shows the exact operation count for each operation executed in Miller's Algorithm.

Table 1. Operation counts for arithmetic required by Miller's Algorithm. (†) Work [7] counts these additions in a different way. Considering their criteria, costs for multiplication and squaring in $\mathbb{F}_{p^2}$ are $3m_u + 2r + 4a$ and $2m_u + 2r + 2a$, respectively.

$E'(\mathbb{F}_{p^2})$-Arithmetic	Operation Count
Doubling/Eval.	$3\tilde{m}_u + 6\tilde{s}_u + 8\tilde{r} + 22\tilde{a} + 4m$
Addition/Eval.	$11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 12\tilde{a} + 4m$
p-power Frobenius	$6m_u + 4r + 18a$
p^2-power Frobenius	$2m + 2a$
Negation	$\tilde{a}$
$\mathbb{F}_{p^2}$-Arithmetic	Operation Count
Add./Sub./Neg.	$\tilde{a} = 2a$
Conjugation	a
Multiplication	$\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r + 8a^{\dagger}$
Squaring	$\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r + 3a^{\dagger}$
Multiplication by β	a
Multiplication by ξ	$2a$
Inversion	$\tilde{i}$

$\mathbb{F}_{p^{12}}$-Arithmetic	Operation Count
Add./Sub.	$6\tilde{a}$
Conjugation	$3\tilde{a}$
Multiplication	$18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}$
Sparse Mult.	$13\tilde{m}_u + 6\tilde{r} + 61\tilde{a}$
Sparser Mult.	$7\tilde{m}_u + 5\tilde{r} + 30\tilde{a}$
Squaring	$12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}$
Cyc. Squaring	$9\tilde{s}_u + 6\tilde{r} + 46\tilde{a}$
Comp. Squaring	$6\tilde{s}_u + 4\tilde{r} + 31\tilde{a}$
Simult. Decomp.	$9\tilde{m} + 6\tilde{s} + 22\tilde{a} + \tilde{i}$
p-power Frobenius	$15m_u + 10r + 46a$
p^2-power Frobenius	$10m + 2\tilde{a}$
Inversion	$25\tilde{m}_u + 9\tilde{s}_u + 24\tilde{r} + 112\tilde{a} + \tilde{i}$

For the selected parameters and with the presented improvements, the Miller Loop in Algorithm 6 executes 64 point doublings with line evaluations, 6 point additions with line evaluations (4 inside Miller Loop and 2 more at the final steps), 1 negation in $\mathbb{F}_{p^2}$ to precompute $\overline{y_P}$, 1 p-power Frobenius, 1 p^2-power Frobenius and 2 negations in $E(\mathbb{F}_{p^2})$; and 1 conjugation, 1 multiplication, 66 sparse multiplications, 2 sparser multiplications and 63 squarings in $\mathbb{F}_{p^{12}}$. The cost of the Miller Loop is:

$$\begin{aligned} ML &= 64 \cdot (3\tilde{m}_u + 6\tilde{s}_u + 8\tilde{r} + 22\tilde{a} + 4m) + 6 \cdot (11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 12\tilde{a} + 4m) \\ &\quad + \tilde{a} + 6m_u + 4r + 18a + 2m + 2a + 2\tilde{a} + 3\tilde{a} + (18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}) \\ &\quad + 66 \cdot (13\tilde{m}_u + 6\tilde{r} + 61\tilde{a}) + 2 \cdot (7\tilde{m}_u + 5\tilde{r} + 30\tilde{a}) + 63 \cdot (12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}) \\ &= 1904\tilde{m}_u + 396\tilde{s}_u + 1368\tilde{r} + 10281\tilde{a} + 282m + 6m_u + 4r + 20a. \end{aligned}$$

The final exponentiation executes in total 1 inversion, 4 conjugations, 15 multiplications, 3 u-th powers, 4 cyclotomic squarings, 5 p-power Frobenius, 3 p^2-power Frobenius:

$$\begin{aligned} FE &= 25\tilde{m}_u + 9\tilde{s}_u + 24\tilde{r} + 112\tilde{a} + \tilde{i} + 4 \cdot 3\tilde{a} + 15 \cdot (18\tilde{m}_u + 6\tilde{r} + 110\tilde{a}) \\ &\quad + 3 \cdot Exp + 4 \cdot (9\tilde{s}_u + 6\tilde{r} + 46\tilde{a}) + 5 \cdot (15m_u + 10r + 46a) + 3 \cdot (10m + 2\tilde{a}) \\ &= 430\tilde{m}_u + 1179\tilde{s}_u + 963\tilde{r} + 8456\tilde{a} + 4\tilde{i} + 30m + 75m_u + 50r + 230a. \end{aligned}$$

Table 2 gives a first-order comparison between our implementation and the best implementation available in the literature of the Optimal Ate pairing at the 128-bit security level in the same platform. For the related work, we suppose that lazy reduction is always used in $\mathbb{F}_{p^2}$ and then each multiplication or squaring essentially computes a modular reduction (that is, $\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r$ and $\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r$). Note that our generalization of the lazy reduction techniques to the whole pairing computation brings the number of modular reductions from the expected 7818 (if lazy reduction was only used for $\mathbb{F}_{p^2}$ arithmetic) to just 4662, avoiding more than 40% of the total required modular reductions. The number of multiplications is also reduced by 13% and the

number of additions is increased by 26% due to lazy reduction trade-offs. Our operation count for the pairing computation is apparently more expensive than Pereira *et al.* [10]. However, the reader should note that, when we consider the real cost of additions in $\mathbb{F}_p$, we cannot exploit the squaring formula in $\mathbb{F}_{p^{12}}$ by [28] (see Section 3.3) and a point doubling formula with fewer multiplications (see Section 4), given the significant increase in the number of additions.

Table 2. Comparison of operation counts for different implementations of the Optimal Ate pairing at the 128-bit security level

Work	Phase	**Operations in $\mathbb{F}_{p^2}$**	**Operations in $\mathbb{F}_p$**
Beuchat *et al.*[7]	ML	$1952(\tilde{m}_u + \tilde{r}) + 568(\tilde{s}_u + \tilde{r}) + 6912\tilde{a}$	$6992m_u + 5040r$
	FE	$403(\tilde{m}_u + \tilde{r}) + 1719(\tilde{s}_u + \tilde{r}) + 7021\tilde{a}$	$4647m_u + 4244r$
	ML+FE	$2355(\tilde{m}_u + \tilde{r}) + 2287(\tilde{s}_u + \tilde{r}) + 13933\tilde{a}$	$11639m_u + 9284r$
This work	ML	$1904\tilde{m}_u + 396\tilde{s}_u + 1368\tilde{r} + 10281\tilde{a}$	$6504m_u + 2736r$
	FE	$430\tilde{m}_u + 1179\tilde{s}_u + 963\tilde{r} + 8456\tilde{a}$	$3648m_u + 1926r$
	ML+FE	$2334\tilde{m}_u + 1575\tilde{s}_u + 2331\tilde{r} + 18737\tilde{a}$	$10152m_u + 4662r$

7 Implementation Results

A software implementation was realized to confirm the performance benefits resulting from the introduced techniques. We implemented $\mathbb{F}_{p^2}$ arithmetic directly in Assembly, largely following advice from [7] to optimize carry handling and eliminate function call overheads. Higher-level algorithms were implemented using the C programming language compiled with the GCC compiler using `-O3` optimization level. Table 3 presents the relevant timings in millions of cycles. Basic Implementation employs homogeneous projective coordinates and lazy reduction below $\mathbb{F}_{p^2}$. Faster arithmetic in cyclotomic subgroups accelerates the Basic Implementation by 5%-7% and, in conjunction with generalized lazy reduction, it improves the Basic Implementation by 18%-22%.

Table 3. Cumulative performance improvement when using new arithmetic in cyclotomic subgroups (Section 5.2) and generalized lazy reduction (Section 3.1) on several Intel and AMD 64-bit architectures. Improvements are calculated relatively to the Basic Implementation. Timings are presented in millions of clock cycles.

	This work							
Method	Phenom II	Impr.	Core i5	Impr.	Opteron	Impr.	Core 2	Impr.
Basic Implementation	1.907	-	2.162	-	2.127	-	2.829	-
Cyclotomic Formulas	1.777	7%	2.020	7%	2.005	6%	2.677	5%
Lazy Reduction	1.562	18%	1.688	22%	1.710	20%	2.194	22%

Table 4 compares our implementation with related work. To ensure that machines with different configurations but belonging to the same microarchitecture had compatible performance (as is the case with Core i5 and Core i7), software from [7] was benchmarked and the results compared with the ones reported in [7].

Machines considered equivalent by this criteria are presented in the same column. We note that Phenom II was not considered in the original study and that we could not find a Core 2 Duo machine producing the same timings as in [7]. For this reason, timings for these two architectures were taken independently by the authors using the available software. Observe that the Basic Implementation in Table 3 consistently outperforms Beuchat *et al.* due to our careful implementation of an optimal choice of parameters $(E(\mathbb{F}_p) : y^2 = x^3 + 2, p = 3 \mod 4)$ [10] combined with optimized curve arithmetic in homogeneous coordinates [9]. When lazy reduction and faster cyclotomic formulas are enabled, pairing computation becomes faster than the best previous result by 28%-34%. For extended benchmark results and comparisons with previous works on different 64-bit processors, the reader is referred to our online database [34].

Table 4. Comparison between implementations on 64-bit architectures. Timings are presented in clock cycles.

	Work/Platform			
	Beuchat *et al.* [7]			
Operation	Phenom II	Core i7	Opteron	Core 2 Duo
Multiplication in $\mathbb{F}_{p^2}$	440	435	443	590
Squaring in $\mathbb{F}_{p^2}$	353	342	355	479
Miller Loop	1,338,000	1,330,000	1,360,000	1,781,000
Final Exponentiation	1,020,000	1,000,000	1,040,000	1,370,000
Optimal Ate Pairing	2,358,000	2,330,000	2,400,000	3,151,000
	This work			
Operation	Phenom II	Core i5	Opteron	Core 2 Duo
Multiplication in $\mathbb{F}_{p^2}$	368	412	390	560
Squaring in $\mathbb{F}_{p^2}$	288	328	295	451
Miller Loop	898,000	978,000	988,000	1,275,000
Final Exponentiation	664,000	710,000	722,000	919,000
Optimal Ate Pairing	1,562,000	1,688,000	1,710,000	2,194,000
Improvement	34%	28%	29%	30%

8 Conclusion

In this work, we revisited the problem of computing optimal pairings on ordinary pairing-friendly curves over prime fields. Several new techniques were introduced for pairing computation, comprised mainly in the generalization of lazy reduction techniques to arithmetic in extensions above $\mathbb{F}_{p^2}$ and inside curve arithmetic; and improvements to the final exponentiation consisting of a formula for compressed squaring in cyclotomic subgroups and an arithmetic trick to remove penalties from negative curve parameterizations. The faster arithmetic in the cyclotomic subgroup improved pairing performance by 5%-7% and the generalized lazy reduction technique was able to eliminate 40% of the required modular reductions, improving pairing performance by further 11%-17%. The introduced techniques allow for the first time a pairing computation under 2 million cycles on 64-bit

desktop computing platforms, improving the state-of-the-art by 28%-34%. The performance improvements are expected to be even higher on embedded architectures, where the ratio between multiplication and addition is typically higher.

Acknowledgements

We would like to express our gratitude to Alfred Menezes, Craig Costello, Michael Scott, Paulo S. L. M. Barreto, Geovandro C. C. F. Pereira and Conrado P. L. Gouvêa for useful discussions during the preparation of this work. The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Centres of Excellence (OCE), CNPq, CAPES and FAPESP for partially supporting this work.

References

1. Boneh, D., Franklin, M.K.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
2. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems Based on Pairing over Elliptic Curve. In: The 2001 Symposium on Cryptography and Information Security. IEICE, Oiso (2001) (in Japanese)
3. Groth, J., Sahai, A.: Efficient Non-interactive Proof Systems for Bilinear Groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
4. Joux, A.: A One Round Protocol for Tripartite Diffie-Hellman. Journal of Cryptology 17(4), 263–276 (2004)
5. Hankerson, D., Menezes, A., Scott, M.: Identity-Based Cryptography, ch. 12, pp. 188–206. IOS Press, Amsterdam (2008)
6. Naehrig, M., Niederhagen, R., Schwabe, P.: New Software Speed Records for Cryptographic Pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010)
7. Beuchat, J.L., Díaz, J.E.G., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010)
8. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455–461 (2010)
9. Costello, C., Lange, T., Naehrig, M.: Faster Pairing Computations on Curves with High-Degree Twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010)
10. Pereira, G.C.C.F., Simplício Jr, M.A., Naehrig, M., Barreto, P.S.L.M.: A Family of Implementation-Friendly BN Elliptic Curves. To appear in Journal of Systems and Software
11. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)

12. Scott, M.: Implementing Cryptographic Pairings. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 197–207. Springer, Heidelberg (2007)
13. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster F_p-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 240–253. Springer, Heidelberg (2009)
14. Freeman, D., Scott, M., Teske, E.: A Taxonomy of Pairing-Friendly Elliptic Curves. Journal of Cryptology 23(2), 224–280 (2010)
15. Hess, F., Smart, N.P., Vercauteren, F.: The Eta Pairing Revisited. IEEE Transactions on Information Theory 52, 4595–4602 (2006)
16. Lee, E., Lee, H.-S., Park, C.-M.: Efficient and Generalized Pairing Computation on Abelian Varieties. IEEE Transactions on Information Theory 55(4), 1793–1803 (2009)
17. Nogami, Y., Akane, M., Sakemi, Y., Kato, H., Morikawa, Y.: Integer Variable χ-Based Ate Pairing. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 178–191. Springer, Heidelberg (2008)
18. Miller, V.: Uses of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
19. Miller, V.S.: The Weil Pairing, and its Efficient Calculation. Journal of Cryptology 17(4), 235–261 (2004)
20. Barreto, P.S.L.M., Kim, H.Y., 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)
21. IEEE: P1363.3: Standard for Identity-Based Cryptographic Techniques using Pairings (2006), `http://grouper.ieee.org/groups/1363/IBC/submissions/`
22. Devegili, A.J., Scott, M., Dahab, R.: Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 197–207. Springer, Heidelberg (2007)
23. Weber, D., Denny, T.F.: The Solution of McCurley's Discrete Log Challenge. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 458–471. Springer, Heidelberg (1998)
24. Lim, C.H., Hwang, H.S.: Fast Implementation of Elliptic Curve Arithmetic in GF(p^n). In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 405–421. Springer, Heidelberg (2000)
25. 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)
26. Benger, N., Scott, M.: Constructing Tower Extensions of Finite Fields for Implementation of Pairing-Based Cryptography. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 180–195. Springer, Heidelberg (2010)
27. Montgomery, P.L.: Modular Multiplication Without Trial Division. Mathematics of Computation 44(170), 519–521 (1985)
28. Chung, J., Hasan, M.: Asymmetric Squaring Formulae. In: 18th IEEE Symposium on Computer Arithmetic (ARITH-18 2007), pp. 113–122. IEEE Press, Los Alamitos (2007)
29. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves. Cryptology ePrint Archive, Report 2010/526 (2010), `http://eprint.iacr.org/`

30. Scott, M., Benger, N., Charlemagne, M., Perez, L.J.D., Kachisa, E.J.: On the Final Exponentiation for Calculating Pairings on Ordinary Elliptic Curves. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 78–88. Springer, Heidelberg (2009)
31. Karabina, K.: Squaring in Cyclotomic Subgroups. Cryptology ePrint Archive, Report 2010/542 (2010), http://eprint.iacr.org/
32. Montgomery, P.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of Computation 48, 243–264 (1987)
33. Granger, R., Scott, M.: Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (2010)
34. Longa, P.: Speed Benchmarks for Pairings over Ordinary Curves, http://www.patricklonga.bravehost.com/speed_pairing.html#speed