High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves

Abstract. This paper describes the design of a fast software library for the computation of the optimal ate pairing on a Barreto–Naehrig elliptic curve. Our library is able to compute the optimal ate pairing over a 254-bit prime field $\mathbb{F}_p$, in just 2.63 million of clock cycles on a single core of an Intel Core i7 2.8GHz processor, which implies that the pairing computation takes 0.942msec. We are able to achieve this performance by a careful implementation of the base field arithmetic through the usage of the customary Montgomery multiplier for prime fields. The prime field is constructed via the Barreto–Naehrig polynomial parametrization of the prime p given as, $p = 36t^4 + 36t^3 + 24t^2 + 6t + 1$, with $t = 2^{62} - 2^{54} + 2^{44}$. This selection of t allows us to obtain important savings for both the Miller loop as well as the final exponentiation steps of the optimal ate pairing.

Keywords: Tate pairing, optimal pairing, Barreto–Naehrig curve, ordinary curve, finite field arithmetic, bilinear pairing software implementation.

1 Introduction

The protocol solutions provided by pairing-based cryptography can only be made practical if one can efficiently compute bilinear pairings at high levels of security. Back in 1986, Victor Miller proposed in [26, 27] an iterative algorithm that can evaluate rational functions from scalar multiplications of divisors, thus allowing to compute bilinear pairings at a linear complexity cost with respect to the size of the input. Since then, several authors have found further algorithmic improvements to decrease the complexity of the *Miller's Algorithm* by reducing its loop length [3, 4, 12, 20, 21, 38], and by constructing pairing-friendly elliptic curves [5, 14, 29] and pairing-friendly tower extensions of finite fields [6, 24].

Roughly speaking, an asymmetric bilinear pairing can be defined as the non-degenerate bilinear mapping, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$, where both $\mathbb{G}_1$, $\mathbb{G}_2$ are finite cyclic additive groups with prime order r, whereas $\mathbb{G}_3$ is a multiplicative cyclic group whose order is also r. Additionally, as it was mentioned above, for cryptographic applications it is desirable that pairings can be computed efficiently. When $\mathbb{G}_1 = \mathbb{G}_2$, we say that the pairing is symmetric, otherwise, if $\mathbb{G}_1 \neq \mathbb{G}_2$, the pairing is asymmetric [15].

Arguably the η_T pairing [3] is the most efficient algorithm for symmetric pairings that are always defined over supersingular curves. In the case of asymmetric pairings, recent breakthroughs include the ate pairing [21], the R-ate pairing [25], and the optimal ate pairing [38].

Several authors have presented software implementations of bilinear pairings targeting the 128-bit security level [1,8,10,16,18,23,31,32]. By taking advantage of the eight cores of a dual quad-core Intel Xeon 45nm, the software library presented in [1] takes 3.02 millions of cycles to compute the η_T pairing on a supersingular curve defined over $\mathbb{F}_{2^{1223}}$. Authors in [8] report 5.42 millions of cycles to compute the η_T pairing on a supersingular curve defined over $\mathbb{F}_{3^{509}}$ on an Intel Core i7 45nm processor using eight cores. The software library presented in [32] takes 4.470 millions of cycles to compute the optimal ate pairing on a 257-bit BN curve using only one core of an Intel Core 2 Quad Q6600 processor.

This paper addresses the efficient software implementation of asymmetric bilinear pairings at high security levels. We present a library[1] that performs the optimal ate pairing over a 254-bit Barreto–Naehrig (BN) curve in just 2.63 million of clock cycles on a single core of an Intel i7 2.8GHz processor, which implies that the optimal ate pairing is computed in 0.942msec. To the best of our knowledge, this is the first time that a software or a hardware accelerator reports a high security level pairing computation either symmetric or asymmetric, either on one core or on a multi-core platform, in less than one millisecond. After a careful selection of a pairing-friendly elliptic curve and the tower field (Sections 2 and 3), we describe the computational complexity associated to the execution of the optimal ate pairing (Section 4). Then, we describe our approach to implement arithmetic over the underlying field $\mathbb{F}_p$ and to perform tower field arithmetic (Section 5), and we give benchmarking results of our software library (Section 6).

2 Optimal Ate Pairing over Barreto–Naehrig Curves

Barreto and Naehrig [5] described a method to construct pairing-friendly ordinary elliptic curves over a prime field $\mathbb{F}_p$. Barreto–Naehrig curves (or BN curves) are defined by the equation $E : y^2 = x^3 + b$, where $b \neq 0$. Their embedding degree k is equal to 12. Furthermore, the number of $\mathbb{F}_p$-rational points of E, denoted by r in the following, is a prime. The characteristic p of the prime field, the group order r, and the trace of Frobenius t_r of the curve are parametrized as follows [5]:

$$\begin{aligned} p(t) &= 36t^4 + 36t^3 + 24t^2 + 6t + 1, \\ r(t) &= 36t^4 + 36t^3 + 18t^2 + 6t + 1, \\ t_r(t) &= 6t^2 + 1, \end{aligned} \tag{1}$$

where $t \in \mathbb{Z}$ is an arbitrary integer such that $p = p(t)$ and $r = r(t)$ are both prime numbers. Additionally, t must be large enough to guarantee an adequate security level. For a security level equivalent to AES-128, we should select t such

[1] An open source code for benchmarking our software library is available at http://homepage1.nifty.com/herumi/crypt/ate-pairing.html

that $\log_2(r(t)) \geq 256$ and $3000 \leq k \cdot \log_2(p(t)) \leq 5000$ [14]. For this to be possible t should have roughly 64 bits.

Let $E[r]$ denote the r-torsion subgroup of E and π_p be the Frobenius endomorphism $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. We define $\mathbb{G}_1 = E[r] \cap \mathrm{Ker}(\pi_p - [1]) = E(\mathbb{F}_p)[r]$, $\mathbb{G}_2 = E[r] \cap \mathrm{Ker}(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^{12}})[r]$, and $G_3 = \mu_r \subset \mathbb{F}^*_{p^{12}}$ (*i.e.* the group of r-th roots of unity). Since we work with a BN curve, every point on $E(\mathbb{F}_p)$ has order r and $\mathbb{G}_1 = E(\mathbb{F}_p)[r] = E(\mathbb{F}_p)$. The optimal ate pairing on the BN curve E is a non-degenerate and bilinear pairing given by the map [30, 32, 38]:

$$
\begin{aligned}
a_{\mathrm{opt}} : \mathbb{G}_2 \times \mathbb{G}_1 &\longrightarrow \mathbb{G}_3 \\
(Q, P) &\longmapsto \big(f_{6t+2,Q}(P) \cdot l_{[6t+2]Q,\pi_p(Q)}(P) \cdot \\
&\qquad\qquad l_{[6t+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P)\big)^{\frac{p^{12}-1}{r}},
\end{aligned}
$$

where

- $f_{s,Q}$, for $s \in \mathbb{N}$ and $Q \in \mathbb{G}_2$, is a family of normalized $\mathbb{F}_{p^{12}}$-rational functions with divisor $(f_{s,Q}) = s(Q) - ([s]Q) - (s-1)(\mathcal{O})$, where $\mathcal{O}$ denotes the point at infinity.
- l_{Q_1,Q_2} is the equation of the line corresponding to the addition of $Q_1 \in \mathbb{G}_2$ with $Q_2 \in \mathbb{G}_2$.

Algorithm 1 shows how we compute the optimal ate pairing in this work. Our approach can be seen as a signed-digit version of the algorithm utilized in [32], where both point additions and subtractions are allowed. The Miller loop (lines 3–10) calculates the value of the rational function $f_{6t+2,Q}$ at point P. In lines 11–13 the product of the line functions $l_{[6t+2]Q,\pi_p(Q)}(P) \cdot l_{[6t+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P)$ is multiplied by $f_{6t+2,Q}(P)$. The so-called final exponentiation is computed in line 14. A detailed summary of the computational costs associated to Algorithm 1 can be found in Section 4.

The BN curves admit a sextic twist $E'/\mathbb{F}_{p^2} : y^2 = x^3 + b/\xi$ defined over $\mathbb{F}_{p^2}$, where $\xi \in \mathbb{F}_{p^2}$ is an element that is neither a square nor a cube in $\mathbb{F}_{p^2}$, and that has to be carefully selected such that $r | \#E'(\mathbb{F}_{p^2})$ holds. This means that pairing computations can be restricted to points P and Q' that belong to $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^2})$, respectively [5, 21, 38].

3 Tower Extension Field Arithmetic

Since $k = 12 = 2^2 \cdot 3$, the tower extensions can be created using irreducible binomials only. This is because $x^k - \beta$ is irreducible over $\mathbb{F}_p$ provided that $\beta \in \mathbb{F}_p$ is neither a square nor a cube in $\mathbb{F}_p$ [24]. Hence, the tower extension can be constructed by simply adjoining a cube or square root of such element β and then the cube or square root of the previous root. This process should be repeated until the desired extension of the tower has been reached.

Accordingly, we decided to represent $\mathbb{F}_{p^{12}}$ using the same tower extension of [18], namely, we first construct a quadratic extension, which is followed by

Algorithm 1 Optimal ate pairing over Barreto–Naehrig curves.

Input: $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$.
Output: $a_{\text{opt}}(Q, P)$.

Write $s = 6t + 2$ as $s = \sum_{i=0}^{L-1} s_i 2^i$, where $s_i \in \{-1, 0, 1\}$;
$T \leftarrow Q$, $f \leftarrow 1$;
for $i = L - 2$ to 0 **do**
  $f \leftarrow f^2 \cdot l_{T,T}(P)$; $T \leftarrow 2T$;
  if $s_i = -1$ **then**
    $f \leftarrow f \cdot l_{T,-Q}(P)$; $T \leftarrow T - Q$;
  else if $s_i = 1$ **then**
    $f \leftarrow f \cdot l_{T,Q}(P)$; $T \leftarrow T + Q$;
  end if
end for
$Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow \pi_{p^2}(Q)$;
$f \leftarrow f \cdot l_{T,Q_1}(P)$; $T \leftarrow T + Q_1$;
$f \leftarrow f \cdot l_{T,-Q_2}(P)$; $T \leftarrow T - Q_2$;
$f \leftarrow f^{(p^{12}-1)/r}$;
return f;

a cubic extension and then by a quadratic one, using the following irreducible binomials:

$$\begin{aligned} \mathbb{F}_{p^2} &= \mathbb{F}_p[u]/(u^2 - \beta), \text{ where } \beta = -5, \\ \mathbb{F}_{p^6} &= \mathbb{F}_{p^2}[v]/(v^3 - \xi), \text{ where } \xi = u + 12, \\ \mathbb{F}_{p^{12}} &= \mathbb{F}_{p^6}[w]/(w^2 - v). \end{aligned} \tag{2}$$

We adopted the tower extension of Equation (2), mainly because field elements $f \in \mathbb{F}_{p^{12}}$ can be seen as a quadratic extension of $\mathbb{F}_{p^6}$, and hence they can be represented as $f = g + hw$, with $g, h \in \mathbb{F}_{p^6}$. This towering will help us to exploit the fact that in the hard part of the final exponentiation we will deal with field elements $f \in \mathbb{F}_{p^{12}}$ that become *unitary* [35, 36], *i.e.*, elements that belong to the cyclotomic subgroup $\mathbb{G}_{\Phi_2}(\mathbb{F}_{p^6})$ as defined in [17]. Such elements satisfy, $f^{p^6+1} = 1$, which means that $f^{-1} = f^{p^6} = g - hw$. In other words, inversion of such elements can be accomplished by simple conjugation. This nice feature opens the door for using addition-subtraction chains in the final exponentiation step, which is especially valuable for our binary signed choice of the parameter t.[2]

3.1 Computational Costs of the Tower Extension Field Arithmetic

The tower extension arithmetic algorithms used in this work were directly adopted from [18]. Let (a, m, s, i), $(\tilde{a}, \tilde{m}, \tilde{s}, \tilde{i})$, (A, M, S, I) denote the cost of

[2] It is worth mentioning, that in [6] the tower choice quadratic then quadratic then cubic was recommended. This tower extension could benefit from the highly efficient squaring formulae given in [17], but field inversion over $\mathbb{F}_{p^{12}}$ would not be essentially free anymore.

Table 1. Computational costs of the tower extension field arithmetic.

Field	**Add./Sub.**	**Mult.**	**Squaring**	**Inversion**
$\mathbb{F}_{p^2}$	$\tilde{a} = 2a$	$\tilde{m} = 3m + 3a + m_\beta$	$\tilde{s} = 2m + 3a + m_\beta$	$\tilde{i} = 4m + m_\beta$ $+2a + i$
$\mathbb{F}_{p^6}$	$3\tilde{a}$	$6\tilde{m} + 2m_\xi + 15\tilde{a}$	$2\tilde{m} + 3\tilde{s} + 2m_\xi + 8\tilde{a}$	$9\tilde{m} + 3\tilde{s} + 4m_\xi$ $+4\tilde{a} + \tilde{i}$
$\mathbb{F}_{p^{12}}$	$6\tilde{a}$	$18\tilde{m} + 6m_\xi + 54\tilde{a}$	$12\tilde{m} + 4m_\xi + 45\tilde{a}$	$25\tilde{m} + 9\tilde{s} + 12m_\xi$ $+56\tilde{a} + \tilde{i}$
$\mathbb{G}_{\Phi_2}(\mathbb{F}_{p^6})$	$6\tilde{a}$	$18\tilde{m} + 6m_\xi + 54\tilde{a}$	$4\tilde{m} + 6\tilde{s} + 4m_\xi$ $+28\tilde{a} + a$	Conjugation

field addition, multiplication, squaring, and inversion in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, and $\mathbb{F}_{p^6}$, respectively. From our implementation (see Section 5), we observed experimentally that $m = s = 8a$ and $i = 46.9m$. We summarize the towering arithmetic costs as follows:

- In the field $\mathbb{F}_{p^2}$, we used Karatsuba multiplication and the complex method for squaring, at a cost of 3 and 2 field multiplications in $\mathbb{F}_p$, respectively. Inversion of an element $A = a_0 + a_1 u \in \mathbb{F}_{p^2}$, can be found from the identity, $(a_0 + a_1 u)^{-1} = (a_0 - a_1 u)/(a_0^2 - \beta a_1^2)$. Using once again the Karatsuba method, field multiplication in $\mathbb{F}_{p^6}$ can be computed at a cost of $6\tilde{m}$ plus several addition operations. All these three operations require the multiplication in the base field by the constant coefficient $\beta \in \mathbb{F}_p$ of the irreducible binomial $u^2 - \beta$. Additionally, we sometimes need to compute the multiplication of an arbitrary element in $\mathbb{F}_{p^2}$ times the constant $\xi \in \mathbb{F}_{p^2}$ at a cost of two field multiplications and two field additions in $\mathbb{F}_p$ plus one multiplication by the constant β. We refer to this operation as m_ξ.
- Squaring in $\mathbb{F}_{p^6}$ can be computed via the formula derived in [9] at a cost of $2\tilde{m} + 3\tilde{s}$ plus some addition operations. Inversion in the sextic extension can be computed at a cost of $9\tilde{m} + 3\tilde{s} + 4m_\xi + 4\tilde{a} + \tilde{i}$ [34].
- Since our field towering constructed $\mathbb{F}_{p^{12}}$ as a quadratic extension of $\mathbb{F}_{p^6}$, the arithmetic costs of the quadratic extension apply. Hence, a field multiplication, squaring and inversion costs in $\mathbb{F}_{p^{12}}$ are, $3M + 3A$, $2M + 5A$ and $2M + 2S + 2A + I$, respectively. However, if $f \in \mathbb{F}_{p^{12}}$, belongs to the cyclotomic subgroup $\mathbb{G}_{\Phi_2}(\mathbb{F}_{p^6})$, its field squaring f^2 can be reduced to two squarings in $\mathbb{F}_{p^6}$ [17,18].

Table 1 lists the computational costs of the tower extension field arithmetic in terms of the $\mathbb{F}_{p^2}$ field arithmetic operations, namely, $(\tilde{a}, \tilde{m}, \tilde{s}, \tilde{i})$.

3.2 Frobenius Operator

Raising an element $f \in \mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - \gamma)$ to the p-power, is an arithmetic operation needed in the final exponentiation (line 14) of the optimal ate pairing (Algorithm 1). We briefly describe in the following how to compute f^p efficiently.

We first remark that the field extension $\mathbb{F}_{p^{12}}$ can be also represented as a sextic extension of the quadratic field, *i.e.*, $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6 - \xi)$. Hence, we can write $f = g + hw \in \mathbb{F}_{p^{12}}$, with $g,\ h \in \mathbb{F}_{p^6}$ such that, $g = g_0 + g_1 v + g_2 v^2$, $h = h_0 + h_1 v + h_2 v^2$, where $g_i, h_i \in \mathbb{F}_{p^2}$, for $i = 1, 2, 3$. This means that f can be equivalently written as, $f = g + hw = g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5$.

We note that the p-power of an arbitrary element in the quadratic extension field $\mathbb{F}_{p^2}$ can be computed essentially free of cost as follows. Let $b \in \mathbb{F}_{p^2}$ be an arbitrary element that can be represented as $b = b_0 + b_1 u$. Then, $(b)^{p^{2i}} = b$ and $(b)^{p^{2i-1}} = \bar{b}$, with $\bar{b} = b_0 - b_1 u$, for $i \in \mathbb{N}$.

Let $\bar{g}_i$, $\bar{h}_i$, denote the conjugates of g_i, h_i, for $i = 1, 2, 3$ respectively. Then, using the identity $W^p = \xi^{(p-1)/6} W$, we can write, $(W^i)^p = \gamma_i W^i$, with $\gamma_{1,i} = \xi^{i(p-1)/6}$, for $i = 1, \ldots, 5$. From the definitions given above, we can compute f^p as,

$$
\begin{aligned}
f^p &= \left(g_0 + h_0 W + g_1 W^2 + h_1 W^3 + g_2 W^4 + h_2 W^5\right)^p \\
&= \bar{g}_0 + \bar{h}_0 W^p + \bar{g}_1 W^{2p} + \bar{h}_1 W^{3p} + \bar{g}_2 W^{4p} + \bar{h}_2 W^{5p} \\
&= \bar{g}_0 + \bar{h}_0 \gamma_{1,1} W + \bar{g}_1 \gamma_{1,2} W^2 + \bar{h}_1 \gamma_{1,3} W^3 + \bar{g}_2 \gamma_{1,4} W^4 + \bar{h}_2 \gamma_{1,5} W^5.
\end{aligned}
$$

The equation above has a computational cost of 5 multiplications and 5 conjugations. We can follow a similar procedure for computing f^{p^2} and f^{p^3}, which are arithmetic operations required in the hard part of the final exponentiation of Algorithm 1. For that, we must pre-compute and store the per-field constants $\gamma_{1,i} = \xi^{i \cdot (p-1)/6}$, $\gamma_{2,i} = \gamma_{1,i} \cdot \bar{\gamma}_{1,i}$, and $\gamma_{3,i} = \gamma_{1,i} \cdot \gamma_{2,i}$ for $i = 1, \ldots, 5$.

4 Computational Cost of the Optimal Ate Pairing

In this paper we considered several choices of the parameter t, required for defining $p(t)$, $r(t)$, and $t_r(t)$ of Equation (1). We found 64-bit values of t with Hamming weight as low as 2 that yield the desired properties for p, r, and t_r. For example, the binomial $t = 2^{63} - 2^{49}$ guarantees that p and r as defined in Equation (1) are both 258-bit prime numbers. However, due to the superior efficiency on its associated base field arithmetic, we decided to use the trinomial $t = 2^{62} - 2^{54} + 2^{44}$, which guarantees that p and r as defined in Equation (1) are 254-bit prime numbers. Since the automorphism group $\mathrm{Aut}(E)$ is a cyclic group of order 6 [30], it is possible to slightly improve Pollard's rho attack and get a speedup of $\sqrt{6}$ [11]. Therefore, we achieve a 126-bit security level with our choice of parameters. The curve equation is $E : Y^2 = X^3 + 5$ and we followed the procedure outlined in [6, 36] in order to find a generator $P = (x_P, y_P) = (1, \sqrt{6})$ for the group $E(\mathbb{F}_p)$, and one generator $Q' = (x_{Q'}, y_{Q'})$ for the group $E'(\mathbb{F}_{p^2})[r]$, given as,

$$
\begin{aligned}
x_{Q'} &= \mathrm{0x}1119BE15924F635DB946C36805C8C2504EDEBD0AD2EB3FDA236325D9D3EBB8D \\
&\quad + \mathrm{0x}5u, \\
y_{Q'} &= \mathrm{0x}1FFC8D5578C017FF79B1F10F5006A7DC33470F93E079AD3684484FC53BB9F73.
\end{aligned}
$$

In this Section, we show that our selection of t yields important savings in the Miller loop and the hard part of the final exponentiation step of Algorithm 1.

4.1 Miller loop

We remark that the parameter $6t+2$ of Algorithm 1 has a bitlength $L = 65$, with a Hamming weight of 7. This implies that the execution of the Miller loop requires 64 doubling step computations in line 4, and 6 addition/subtraction steps in lines 6 and 8.

It is noted that the equation of the tangent line at $T \in \mathbb{G}_2$ evaluated at P defines a sparse element in $\mathbb{F}_{p^{12}}$ (half of the coefficients are equal to zero). The same observation holds for the equation of the line through the points T and $\pm Q$ evaluated at P. This sparsity allows us to reduce the number of operations on the underlying field when performing accumulation steps (lines 4, 6, 8, 12, and 13 of Algorithm 1).

We perform an interleaved computation of the tangent line at point T (respectively, the line through the points T and Q) with a point doubling (respectively, point addition) using the formulae given in [2] as explained next.

Doubling step (line 4). We represent the point $T \in E'(\mathbb{F}_{p^2})$ in Jacobian coordinates as $T = (X_T, Y_T, Z_T)$. The formulae for doubling T, *i.e.*, the equations that define the point $R = 2T = (X_R, Y_R, Z_R)$ are,

$$X_R = 9X_T^4 - 8X_T Y_T^2,\; Y_R = 3X_T^2(4X_T Y_T^2 - X_R) - 8Y_T^4,\; Z_R = 2Y_T Z_T.$$

Let the point $P \in E(\mathbb{F}_p)$ be represented in affine coordinates as $P = (x_P, y_P)$. Then, the tangent line at T evaluated at P can be calculated as,

$$l = 2Z_R Z_T^2 y_P \;-\; (6X_T^2 Z_T^2 x_P)W \;+\; (6X_T^3 - 4Y_T^2)W^2 \in \mathbb{F}_{p^{12}}.$$

Hence, the computational cost of the interleaving computation of the tangent line and the doubling of the point T is, $3\tilde{m}+8\tilde{s}+16\tilde{a}+2m+2a$. Other operations included in line 4 are f^2 and the product $f^2 \cdot l_{T,T}(P)$, which can be computed at a cost of, $12\tilde{m} + 45\tilde{a} + 4m_\xi$ and $13\tilde{m} + 39\tilde{a} + 2m_\xi$, respectively. In summary, the computational cost associated to line 4 of Algorithm 1 is given as, $28\tilde{m} + 8\tilde{s} + 100\tilde{a} + 4m + 6m_\xi + 2a$.

Addition step (lines 6 and 8). Let $Q = (X_Q, Y_Q, Z_Q)$ and $T = (X_T, Y_T, Z_T)$ represent the points Q and $T \in E'(\mathbb{F}_{p^2})$ in Jacobian coordinates. Then the point $R = T + Q = (X_R, Y_R, Z_R)$, can be computed as,

$$\begin{aligned}
X_R &= (2Y_Q Z_T^3 - 2Y_T)^2 - 4(X_Q Z_T^2 - X_T)^3 - 8(X_Q Z_T^2 - X_T)^2 X_T,\\
Y_R &= (2Y_Q Z_T^3 - 2Y_T)(4(X_Q Z_T^2 - X_T)^2 X_T - X_R) - 8Y_T(X_Q Z_T^2 - X_T)^3,\\
Z_R &= 2Z_T(X_Q Z_T^2 - X_T).
\end{aligned}$$

Once again, let the point $P \in E(\mathbb{F}_p)$ be represented in affine coordinates as $P = (x_P, y_P)$. Then, the line through T and Q is given as,

$$l = 2Z_R y_P \;-\; 4x_P(Y_Q Z_T^3 + Y_T)W \;+\; (4X_Q(Y_Q Z_T^3 X_Q - Y_T) - 2Y_Q Z_R)W^2 \in \mathbb{F}_{p^{12}}.$$

The combined cost of computing the line through T and Q and the point addition $R = T + Q$ is, $7\tilde{m} + 7\tilde{s} + 25\tilde{a} + 2m + 2a$. Finally we must accumulate the value of l by performing the product $f \cdot l$ at a cost of, $13\tilde{m} + 39\tilde{a} + 2m_\xi$.

Therefore, the computational cost associated to line 6 of Algorithm 1 is given as, $20\tilde{m} + 7\tilde{s} + 64\tilde{a} + 2m + 2m_\xi + 2a$. This is the same cost of line 8.

Frobenius application and final addition step (lines 11–13). In this step we add to the value accumulated in $f = f_{6t+2,Q}(P)$, the product of the lines through the points $Q_1, -Q_2 \in E'(\mathbb{F}_{p^2})$, namely, $l_{[6t+2]Q,Q_1}(P) \cdot l_{[6t+2]Q+Q_1,-Q_2}(P)$.

The points Q_1, Q_2 can be found by applying the Frobenius operator as, $Q_1 = \pi_p(Q)$, $Q_2 = \pi_p^2(Q)$. The total cost of computing lines 11–13 is given as, $42\tilde{m} + 14\tilde{s} + 128\tilde{a} + 4m_\xi$.

Let us recall that from our selection of t, $6t + 2$ is a 65-bit number with a low Hamming weight of 7.[3] This implies that the Miller loop of the optimal ate pairing can be computed using only 64 point doubling steps and 6 point addition/subtraction steps. Therefore, the total cost of the Miller loop portion of Algorithm 1 is approximately given as,

$$\begin{aligned}\text{Cost of Miller loop} &= 64 \cdot (28\tilde{m} + 8\tilde{s} + 100\tilde{a} + 4m + 6m_\xi + 2a) + \\ &\quad 6 \cdot (20\tilde{m} + 7\tilde{s} + 64\tilde{a} + 2m + 2m_\xi + 2a) + \\ &\quad 42\tilde{m} + 14\tilde{s} + 128\tilde{a} + 4m_\xi \\ &= 1954\tilde{m} + 568\tilde{s} + 6912\tilde{a} + 2m + 400m_\xi.\end{aligned}$$

4.2 Final Exponentiation

Line 14 of Algorithm 1 performs the final exponentiation step, by raising $f \in \mathbb{F}_{p^{12}}$ to the power $e = (p^{12} - 1)/r$. We computed the final exponentiation by following the procedure described by Scott *et al.* in [36], where the exponent e is split into three coefficients as,

$$e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}. \tag{3}$$

As it was discussed in Section 3, we can take advantage of the fact that raising f to the power p^6 is equivalent to one conjugation. Hence, one can compute $f^{(p^6-1)} = \bar{f} \cdot f^{-1}$, which costs one field inversion and one field multiplication in $\mathbb{F}_{p^{12}}$. Moreover, after raising to the power $p^6 - 1$, the resulting field element becomes a member of the cyclotomic subgroup $\mathbb{G}_{\Phi_2}(\mathbb{F}_{p^6})$, which implies that inversion of such elements can be computed by simply conjugation (see Table 1). Furthermore, from the discussion in Section 3.2 raising to the power $p^2 + 1$, can be done with five field multiplications in the base field $\mathbb{F}_p$, plus one field

[3] We note that in the binary signed representation with digit set $\{-1, 0, 1\}$, the integers $t = 2^{62} - 2^{54} + 2^{44}$ and $6t + 2 = 2^{64} + 2^{63} - 2^{56} - 2^{55} + 2^{46} + 2^{45} + 2$ have a signed bitlength of 63 and 65, respectively.

multiplication in $\mathbb{F}_{p^{12}}$. The processing of the third coefficient in Equation (3) is referred as the *hard part* of the final exponentiation, *i.e*, the task of computing $m^{(p^4-p^2+1)/r}$, with $m \in \mathbb{F}_{p^{12}}$. In order to accomplish that, Scott *et al.* described in [36] a clever procedure that requires the calculation of ten temporary values, namely,

$$m^t,\ m^{t^2},\ m^{t^3},\ m^p,\ m^{p^2},\ m^{p^3},\ m^{(tp)},\ m^{(t^2p)},\ m^{(t^3p)},\ m^{(t^2p^2)},$$

which are the building blocks required for constructing a vectorial addition chain whose evaluation yields the final exponentiation f^e, by performing 13 and 4 field multiplication and squaring operations over $\mathbb{F}_{p^{12}}$, respectively.[4] Taking advantage of the Frobenius operator efficiency, the temporary values m^p, m^{p^2}, m^{p^3}, $m^{(tp)}$, $m^{(t^2p)}$, $m^{(t^3p)}$, and $m^{(t^2p^2)}$ can be computed at a cost of just 35 field multiplications over $\mathbb{F}_p$ (see Section 3.2). Therefore, the most costly computation of the hard part of the final exponentiation is the calculation of $m^t, m^{t^2} = (m^t)^t, m^{t^3} = (m^{t^2})^t$. From our choice, $t = 2^{62} - 2^{54} + 2^{44}$, we can compute these three temporary values at a combined cost of $62 \cdot 3 = 186$ cyclotomic squarings plus $2 \cdot 3 = 6$ field multiplications over $\mathbb{F}_{p^{12}}$. This is cheaper than the t selection used in [32] that requires $5 \cdot 3 = 15$ more field multiplications over $\mathbb{F}_{p^{12}}$.

Consulting Table 1, we can approximately estimate the total computational cost associated to the Final exponentiation as,

$$\begin{aligned}\text{F. Exp. cost} = {} & (25\tilde{m} + 9\tilde{s} + 12m_\xi + 4m + 56\tilde{a} + i) + (18\tilde{m} + 6m_\xi + 54\tilde{a}) + \\ & 5m + (18\tilde{m} + 6m_\xi + 54\tilde{a}) + \\ & 35m + 13 \cdot (18\tilde{m} + 6m_\xi + 54\tilde{a}) + 4 \cdot (4\tilde{m} + 6\tilde{s} + 4m_\xi + 28\tilde{a}) + \\ & 186 \cdot (4\tilde{m} + 6\tilde{s} + 4m_\xi + 28\tilde{a}) + 6 \cdot (18\tilde{m} + 6m_\xi + 54\tilde{a}) \\ = {} & 1163\tilde{m} + 1149\tilde{s} + 6510\tilde{a} + 898m_\xi + i.\end{aligned}$$

Table 2 presents a comparison of $\mathbb{F}_{p^2}$ arithmetic operations of our work against the reference pairing software libraries [18, 32]. From Table 2, we observe that our approach saves about 20% and 10% $\mathbb{F}_{p^2}$ multiplications when compared against [18] and [32], respectively.

5 Software Implementation of Field Arithmetic

In this work, we target the `x86_64` instruction set [22]. Our software library is written in C++ and can be used on several platforms: 64-bit Windows 7 with Visual Studio 2008 Professional, 64-bit Linux 2.6 and Mac OS X 10.5 with gcc 4.4.1 or later, etc. In order to improve the runtime performance of our pairing library, we made an extensive use of Xbyak [28], a x86/x64 just-in-time assembler for the C++ language.

---

[4] We remark that the cost of the field squaring operations is that of the elements in the cyclotomic subgroup $\mathbb{G}_{\Phi_2}(\mathbb{F}_{p^6})$ listed in the last row of Table 1.

Table 2. A Comparison of arithmetic operations required by the computation of the ate pairing variants.

		$\tilde{m}$	$\tilde{s}$	$\tilde{a}$	$\tilde{i}$	m_ξ
Hankerson *et al.* [18] R-ate pairing	Miller Loop	2272	356	6706	1	412
	Final Exp.	1616	1197	7922	1	1062
	Total	3893	1553	14628	2	1474
Naehrig *et al.* [32] Optimal ate pairing	Miller Loop	2022	590	7140		410
	Final Exp.	1433	1149	7320	1	988
	Total	3455	1739	14460	1	1398
This work Optimal ate pairing	Miller Loop	1954	568	6912		400
	Final Exp.	1163	1149	6510	1	898
	Total	3117	1717	13422	1	1298

5.1 Implementation of Prime Field Arithmetic

The `x86_64` instruction set has a **mul** operation which multiplies two 64-bit unsigned integers and returns a 128-bit unsigned integer. The execution of this operation takes about 3 cycles on Intel Core i7 and AMD Opteron processors. Compared to previous architectures, the gap between multiplication and addition/subtraction in terms of cycles is much smaller. This means that we have to be careful when selecting algorithms to perform prime field arithmetic: the schoolbook method is for instance faster than Karatsuba multiplication in the case of 256-bit operands.

An element $x \in \mathbb{F}_p$ is represented as $x = (x_3, x_2, x_1, x_0)$, where $x_i, 0 \leq i \leq 3$, are 64-bit integers. The addition and the subtraction over $\mathbb{F}_p$ are performed in a straightforward manner, *i.e.*, we add/subtract the operands followed by reduction into $\mathbb{F}_p$. Multiplication and inversion over $\mathbb{F}_p$ are accomplished according to the well-known Montgomery multiplication and Montgomery inversion algorithms, respectively [19].

5.2 Implementation of Quadratic Extension Field Arithmetic

This section describes our optimizations for some operations over $\mathbb{F}_{p^2}$ defined in Equation group (2).

Multiplication. We implemented the multiplication over the quadratic extension field $\mathbb{F}_{p^2}$ using a Montgomery multiplication scheme split into two steps:

1. The straightforward multiplication of two 256-bit integers (producing a 512-bit integer), denoted as, **mul256**.
2. The Montgomery reduction from a 512-bit integer to a 256-bit integer. This operation is denoted by **mod512**.

According to our implementation, **mul256** (resp. **mod512**) contains 16 (resp. 20) **mul** operations and its execution takes about 55 (resp. 100) cycles.

Let $P(u) = u^2 + 5$ be the irreducible binomial defining the quadratic extension $\mathbb{F}_{p^2}$. Let $A, B, C \in \mathbb{F}_{p^2}$ such that, $A = a_0 + a_1 u$, $B = b_0 + b_1 u$, and $C = c_0 + c_1 u = A \cdot B$. Then, $c_0 = a_0 b_0 - 5a_1 b_1$ and $c_1 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$. Hence, in order to obtain the field multiplication over the quadratic extension field, we must compute three multiplications over $\mathbb{F}_p$, and it may seem that three **mod512** operations are necessary. However, we can keep the results of the products **mul256**(a_0, b_0), **mul256**(a_1, b_1), and **mul256**$(a_0 + a_1, b_0 + b_1)$ in three temporary 512-bit integer values. Then, we can add or subtract them without reduction, followed by a final call to **mod512** in order to get $c_0, c_1 \in \mathbb{F}_p$. This approach yields the saving of one **mod512** operation as shown in Algorithm 2. We stress that the **addNC/subNC** functions in lines 1, 2, 6, and 7 of Algorithm 2, stand for addition/subtraction between 256-bit or 512-bit integers without carry operation. We explain next the rationale for using addition/subtraction without carry.

The addition $x + y$, and subtraction $x - y$, of two elements $x, y \in \mathbb{F}_p$ include an unpredictable branch check to figure out whether $x + y \geq p$ or $x < y$. This is a costly check that is convenient to avoid as much as possible. Fortunately, our selected prime p satisfies $7p < N$, $(N = 2^{256})$ and the function **mod512** can reduce operands x, whenever, $x < pN$. This implies that we can add up to seven times without performing a carry check. In line 8, d_0 is equal to $(a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1 = a_0 b_1 + a_1 b_0 < 2p^2 < pN$. Hence, we can use **addNC/subNC** for step 1, 2, 6, and 7. In line 9, we multiply d_2 by the constant value 5, which can be computed with no carry operation. By applying these modifications, we manage to reduce the cost of the field multiplication over $\mathbb{F}_{p^2}$ from about 640 cycles (required by a non-optimized procedure) to just 440 cycles.

In line 10, $d_1 = a_0 b_0 - 5a_1 b_1$. We perform this operation as a 512-bit integer subtraction with carry operation followed by a **mod512** reduction. Let x be a 512-bit integer such that $x = a_0 b_0 - 5a_1 b_1$ and let t be a 256-bit integer. The aforementioned carry operation can be accomplished as follows: if $x < 0$, then $t \leftarrow p$, otherwise $t \leftarrow 0$, then $d_1 \leftarrow x + tN$, where this addition operation only uses the 256 most significant bits of x.

Squaring. Algorithm 3 performs field squaring where some carry operations have been reduced, as explained next. Let $A = a_0 + a_1 u \in \mathbb{F}_{p^2}$, $C = A^2 = c_0 + c_1 u$, and let $x = (a_0 + p - a_1)(a_0 + 5a_1)$. Then $c_0 = x \bmod p$. However, we observe that $x \leq 2p \cdot 6p = 12p^2 < N^2$ where $N = 2^{256}$. Also we have that,

$$x - 4a_0 a_1 \geq a_0(a_0 + 5a_1) - 4a_0 a_1 = a_0(a_0 + a_1) \geq 0,$$

which implies,

$$\begin{aligned} \max(x - 4a_0 a_1) &= \max(a_0(a_0 + p) + 5a_1(p - a_1)) \\ &< p \cdot 2p + 5(p/2)(p - p/2) < pN. \end{aligned}$$

Algorithm 2 Optimized multiplication over $\mathbb{F}_{p^2}$.

Input: A and $B \in \mathbb{F}_{p^2}$ such that $A = a_0 + a_1 u$ and $B = b_0 + b_1 u$.
Output: $C = A \cdot B \in \mathbb{F}_{p^2}$.
1. $s \leftarrow$ **addNC**(a_0, a_1);
2. $t \leftarrow$ **addNC**(b_0, b_1);
3. $d_0 \leftarrow$ **mul256**(s, t);
4. $d_1 \leftarrow$ **mul256**(a_0, b_0);
5. $d_2 \leftarrow$ **mul256**(a_1, b_1);
6. $d_0 \leftarrow$ **subNC**(d_0, d_1);
7. $d_0 \leftarrow$ **subNC**(d_0, d_2);
8. $c_1 \leftarrow$ **mod512**(d_0);
9. $d_2 \leftarrow 5d_2$;
10. $d_1 \leftarrow d_1 - d_2$;
11. $c_0 \leftarrow$ **mod512**(d_1);
12. **return** $C \leftarrow c_0 + c_1 u$;

We conclude that we can safely add/subtract the operands in Algorithm 3 without carry check.

Algorithm 3 Optimized squaring over $\mathbb{F}_{p^2}$.

Input: $A \in \mathbb{F}_{p^2}$ such that $A = a_0 + a_1 u$.
Output: $C = A^2 \in \mathbb{F}_{p^2}$.
1. $t \leftarrow$ **addNC**(a_1, a_1);
2. $d_1 \leftarrow$ **mul256**(t, a_0);
3. $t \leftarrow$ **addNC**(a_0, p);
4. $t \leftarrow$ **subNC**(t, a_1);
5. $c_1 \leftarrow 5a_1$;
6. $c_1 \leftarrow$ **addNC**(c_1, a_0);
7. $d_0 \leftarrow$ **mul256**(t, c_1);
8. $c_1 \leftarrow$ **mod512**(d_1);
9. $d_1 \leftarrow$ **addNC**(d_1, d_1);
10. $d_0 \leftarrow$ **subNC**(d_0, d_1);
11. $c_0 \leftarrow$ **mod512**(d_0);
12. **return** $C \leftarrow c_0 + c_1 u$;

Multiplication by ξ. Algorithm 4 shows the procedure that we followed for computing the field multiplication of an arbitrary element $A \in \mathbb{F}_{p^2}$ by the field constant $\xi = 12 + u \in \mathbb{F}_{p^2}$.

We first remark that Algorithm 4 requires the calculation of field multiplications by the constant values 5 and 12. Computing these operations using shift-and-add expressions such as $5n = n + (n \ll 2)$ and $12n = (n \ll 3) + (n \ll 2)$ for $n \in \mathbb{F}_p$ may be tempting as a means to avoid full multiplication calculations. Nevertheless, in our implementation we preferred to compute those multiplication-by-constant operations using the `x86_64` **mul** instruction, since the cost in clock cycles of **mul** is almost the same or even a little cheaper than the one associated to the shift-and-add method.

Algorithm 4 Multiplication by ξ.

Input: $A \in \mathbb{F}_{p^2}$ such that $A = a_0 + a_1 u$. Let $\xi \in \mathbb{F}_{p^2}$ which is defined in Equation (2).
Output: $C = A \cdot \xi \in \mathbb{F}_{p^2}$.
1. $t_0 \leftarrow 12a_0$;
2. $t_1 \leftarrow 5a_1$;
3. $t_0 \leftarrow t_0 - t_1$;
4. $c_0 \leftarrow t_0 \bmod p$; (use Algorithm 5)
5. $t_0 \leftarrow 12a_1$;
6. $t_0 \leftarrow$ **addNC**(a_0, t_0);
7. $c_1 \leftarrow t_0 \bmod p$; (use Algorithm 5)
8. **return** $C \leftarrow c_0 + c_1 u$;

Our multiplication by ξ algorithm requires the reduction modulo p of an integer x smaller than $13p$ (lines 4 and 7). Note that we need five 64-bit registers to store $x = (x_4, x_3, x_2, x_1, x_0)$. However, one can easily see that $x_4 = 0$ or $x_4 = 1$, and then one can prove that $x \text{ div } 2^{253} = (x_4 \ll 3)|(x_3 \gg 61)$. Division by 2^{253} involves only three logical operations and is efficiently performed on our target processor. Furthermore, the prime p we selected has the following nice property:

$$(ip) \text{ div } 2^{253} = \begin{cases} i & \text{if } 0 \le i \le 9, \\ i+1 & \text{if } 10 \le i \le 14. \end{cases}$$

We build a small look-up table p-Tbl defined as follows:

$$p\text{-Tbl}[i] = \begin{cases} ip & \text{if } 0 \le i \le 9, \\ (i-1)p & \text{if } 10 \le i \le 14. \end{cases} \qquad (4)$$

We then get $|x - p\text{-Tbl}[x \gg 253]| < p$. Algorithm 5 summarizes how we apply this strategy to perform a modulo p reduction.

6 Implementation Results

We list in Table 3 the timings that we achieved on different architectures. Our library is able to evaluate the optimal ate pairing over a 254-bit prime field $\mathbb{F}_p$, in just 2.63 million of clock cycles on a single core of an Intel Core i7 2.8GHz processor, which implies that the pairing computation takes 0.942msec. To our best knowledge, we are the first to compute a cryptographic pairing in less than one millisecond at this level of security on a desktop computer.

According to the second column of Table 3, the costs (in clock cycles) that were measured for the $\mathbb{F}_{p^2}$ arithmetic when implemented in the Core i7 processor

Algorithm 5 Fast reduction $x \bmod p$.

Input: $x \in \mathbb{Z}$ such that $0 \leq x < 13p$ and represented as $x = (x_4, x_3, x_2, x_1, x_0)$, where $x_i, 0 \leq i \leq 4$, are 64-bit integers. Let p-Tbl be the precomputed look-up table defined in Equation (4).
Output: $z = x \bmod p$.
1. $q \leftarrow (x_4 \ll 3)|(x_3 \gg 61)$; $(q \leftarrow \lfloor x/2^{253} \rfloor)$
2. $z \leftarrow x - p\text{-Tbl}[q]$;
3. **if** $z < 0$ **then**
4. $\quad z \leftarrow z + p$;
5. **end if**
6. **return** z;

are $\tilde{m} = 435$ and $\tilde{s} = 342$. Additionally, we measured $\tilde{a} = 40$. Now, from Table 2, one can see that the predicted computational cost of the optimal ate pairing is given as,

$$\begin{aligned} \text{Optimal ate pairing cost} &= 3117\tilde{m} + 1717\tilde{s} + 13422\tilde{a} + i \\ &= 3117 \cdot 435 + 1717 \cdot 342 + 13422 \cdot 40 \\ &= 2{,}487{,}489. \end{aligned}$$

We conclude that the experimental results presented in Table 3 have a reasonable match with the computational cost prediction given in Section 4.

For comparison purpose, we also report the performance of the software library for BN curves developed by Naehrig *et al.* [32], which is the best software implementation that we know of.[5] Naehrig *et al.* combined several state-of-the art optimization techniques to write a software that is more than twice as fast as the previous reference implementation by Hankerson *et al.* [18]. The most original contribution is the implementation of arithmetic over $\mathbb{F}_p$. Working in the case of hardware realizations of pairings, Fan *et al.* [13] suggested to take advantage of the polynomial form of $p(t)$ and introduced a new hybrid modular multiplication algorithm. The operands a and $b \in \mathbb{F}_p$ are converted to degree-4 polynomials $a(t)$ and $b(t)$, and multiplied according to Montgomery's algorithm in the polynomial ring. Coefficients of the results must be reduced modulo t. Fan *et al* noticed that, if $t = 2^m + s$, where s is a small constant, this step consists of a multiplication by s instead of a division by t. Naehrig *et al.* adapted this technique to design a software-oriented modular multiplication algorithm implemented by means of double-precision floating-point SIMD instructions.

Table 4 summarizes the best results published in the open literature since 2007. All the works featured in Table 4, targeted a level of security equivalent to that of AES-128. Aranha *et al.* [1] and Beuchat *et al.* [8] considered supersingular elliptic curves in characteristic 2 and 3, respectively. All other authors worked with ordinary curves.

[5] The results on the Core 2 Quad processor are reprinted from [32]. We downloaded the library [33] and made our own experiments on an Opteron platform.

Table 3. Cycle counts of multiplication over $\mathbb{F}_{p^2}$, squaring over $\mathbb{F}_{p^2}$, and optimal ate pairing on different machines.

	Our results			dclxvi [32, 33]	
	Core i7[a]	Opteron[b]	Core 2 Duo[c]	Opteron[b]	Core 2 Quad[d]
Multiplication over $\mathbb{F}_{p^2}$	435	443	558	695	693
Squaring over $\mathbb{F}_{p^2}$	342	355	445	614	558
Miller loop	1,330,000	1,380,000	1,740,000	2,480,000	2,260,000
Final exponentiation	1,300,000	1,300,000	1,580,000	2,520,000	2,210,000
Optimal ate pairing	2,630,000	2,680,000	3,320,000	5,000,000	4,470,000

[a] Intel Core i7 860 (2.8GHz), Windows 7, Visual Studio 2008 Professional
[b] Quad-Core AMD Opteron 2376 (2.3GHz), Linux 2.6.18, gcc 4.4.1
[c] Intel Core 2 Duo T7100 (1.8GHz), Windows 7, Visual Studio 2008 Professional
[d] Intel Core 2 Quad Q6600 (2394MHz), Linux 2.6.28, gcc 4.3.3

Several authors studied multi-core implementations of a cryptographic pairing [1, 8, 16]. In the light of the results reported in Table 4, it seems that the acceleration achieved by an n-core implementation is always less than the ideal $n\times$ speedup. This is related to the extra arithmetic operations needed to combine the partial results generated by each core, and the dependencies between the different operations involved in the final exponentiation. The question that arises is therefore: how many cores should be utilized to compute a cryptographic pairing? We believe that the best answer is the one provided by Grabher *et al.*: "if the requirement is for two pairing evaluations, the slightly moronic conclusion is that one can perform one pairing on each core [. . .], doubling the performance versus two sequential invocations of any other method that does not already use multi-core parallelism internally" [16].

7 Conclusion

In this paper we have presented a software library that implements the optimal ate pairing over a Barreto–Naehrig curve at the 126-bit security level. To the best of our knowledge, we are the first to have reported the computation of a bilinear pairing at a level of security roughly equivalent to that of AES-128 in less than one millisecond on a single core of an Intel Core i7 2.8GHz processor. The speedup achieved in this work is a combination of two main factors:

– A careful programming of the underlying field arithmetic based on Montgomery multiplication that allowed us to perform a field multiplication over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ in just 160 and 435 cycles, respectively, when working in an Opteron-based machine. We remark that in contrast with [32], we did not make use of the 128-bit multimedia arithmetic instructions.
– A binary signed selection of the parameter t that allowed us to obtain significant savings in both the Miller loop and the final exponentiation of the optimal ate pairing.

Table 4. A comparison of cycles and timings required by the computation of the ate pairing variants. The frequency is given in GHz and the timings are in milliseconds.

	Algo.	**Architecture**	**Cycles**	**Freq.**	**Calc. time**
Devegili *et al.* [10]	ate	Intel Pentium IV	69,600,000	3.0	23.20
Naehrig *et al.* [31]	ate	Intel Core 2 Duo	29,650,000	2.2	13.50
Grabher *et al.* [16]	ate	Intel Core 2 Duo (1 core)	23,319,673	2.4	9.72
		Intel Core 2 Duo (2 cores)	14,429,439		6.01
Aranha *et al.* [1]	η_T	Intel Xeon 45nm (1 core)	17,400,000	2.0	8.70
		Intel Xeon 45nm (8 cores)	3,020,000		1.51
Beuchat *et al.* [8]	η_T	Intel Core i7 (1 core)	15,138,000	2.9	5.22
		Intel Core i7 (8 cores)	5,423,000		1.87
Hankerson *et al.* [18]	R-ate	Intel Core 2	10,000,000	2.4	4.10
Naehrig *et al.* [32]	a_{opt}	Intel Core 2 Quad Q6600	4,470,000	2.4	1.80
This work	a_{opt}	Intel Core i7	2,630,000	2.8	0.94

Our selection of t yields a prime $p = p(t)$ that has a bitlength of just 254 bits. This size is slightly below than what Freeman *et al.* [14] recommend for achieving a high security level. If for certain scenarios, it becomes strictly necessary to meet or exceed the 128-bit level of security, we recommend to select $t = 2^{63} - 2^{49}$ that produces a prime $p = p(t)$ with a bitlength of 258 bits. However, we warn the reader that since a 258-bit prime implies that more than four 64-bit register will be required to store field elements, the performance of the arithmetic library will deteriorate.

Consulting the cycle count costs listed in Table 3, one can see that for our implementation the cost of the Miller loop is slightly but consistently higher than that of the final exponentiation step.

Authors in [13,32] proposed to exploit the polynomial parametrization of the prime p as a means to speed up the underlying field arithmetic. We performed extensive experiments trying to apply this idea to our particular selection of t with no success. Instead, the customary Montgomery multiplier algorithm appears to achieve a performance that is very hard to beat by other multiplication schemes, whether integer-based or polynomial-based multipliers.

The software library presented in this work computes a bilinear pairing at a high security level at a speed that is faster than the best hardware accelerators published in the open literature (see for instance [7, 13, 23, 37]). We believe that this situation is unrealistic and therefore we will try to design a hardware architecture that can compute 128-bit security bilinear pairing in shorter timings. Our future work will also include a study of the parallelization possibilities on pairing-based protocols that specify the computation of many bilinear pairing during their execution.

References

1. D.F. Aranha, J. López, and D. Hankerson. High-speed parallel software implementation of the η_T pairing. In J. Pieprzyk, editor, *Topics in Cryptology–CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010.
2. C. Arene, T. Lange, M. Naehrig, and C. Ritzenthaler. Faster computation of the Tate pairing. Cryptology ePrint Archive, Report 2009/155, 2009. Available at `http://eprint.iacr.org/2009/155.pdf`.
3. P.S.L.M. Barreto, S.D. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.
4. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology–CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.
5. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography–SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
6. N. Benger and M. Scott. Constructing tower extensions for the implementation of pairing-based cryptography. Cryptology ePrint Archive, Report 2009/556, 2009. Available at `http://eprint.iacr.org/2009/556.pdf`.
7. J.-L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves. Cryptology ePrint Archive, Report 2009/398, 2009. Available at `http://eprint.iacr.org/2009/398.pdf`.
8. J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core implementation of the Tate pairing over supersingular elliptic curves. In J.A. Garay, A. Miyaji, and A. Otsuka, editors, *Cryptology and Network Security–CANS 2009*, number 5888 in Lecture Notes in Computer Science, pages 413–432. Springer, 2009.
9. J. Chung and M.A. Hasan. Asymmetric squaring formulae. In P. Kornerup and J.-M. Muller, editors, *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 113–122. IEEE Computer Society, 2007.
10. A.J. Devegili, M. Scott, and R. Dahab. Implementing cryptographic pairings over Barreto–Naehrig curves. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, *Pairing-Based Cryptography–Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 197–207. Springer, 2007.
11. I. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology–ASIACRYPT 1999*, volume 1716 of *Lecture Notes in Computer Science*, pages 103–121. Springer, 1999.
12. I. Duursma and H.S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology–ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.
13. J. Fan, F. Vercauteren, and I. Verbauwhede. Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto–Naehrig curves. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems–CHES 2009*, number 5747 in Lecture Notes in Computer Science, pages 240–253. Springer, 2009.

14. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, April 2010.
15. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, January 2008.
16. P. Grabher, J. Großschädl, and D. Page. On software parallel implementation of cryptographic pairings. In *Selected Areas in Cryptography–SAC 2008*, number 5381 in Lecture Notes in Computer Science, pages 34–49. Springer, 2008.
17. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. Cryptology ePrint Archive, Report 2009/565, 2009. Available at `http://eprint.iacr.org/2009/565.pdf`.
18. D. Hankerson, A. Menezes, and M. Scott. Software implementation of pairings. In M. Joye and G. Neven, editors, *Identity-based Cryptography*, Cryptology and Information Security Series, chapter 12, pages 188–206. IOS Press, 2009.
19. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, Inc., 2004.
20. F. Hess. Pairing lattices. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography–Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 18–38. Springer, 2008.
21. F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, October 2006.
22. Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manuals.* Available at `http://www.intel.com/products/processor/manuals/`.
23. D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar. Designing an ASIP for cryptographic pairings over Barreto–Naehrig curves. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems–CHES 2009*, number 5747 in Lecture Notes in Computer Science, pages 254–271. Springer, 2009.
24. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. Cryptology ePrint Archive, Report 2005/076, 2005. Available at `http://eprint.iacr.org/2005/076.pdf`.
25. E. Lee, H.-S. Lee, and C.-M. Park. Efficient and generalized pairing computation on abelian varieties. Cryptology ePrint Archive, Report 2008/040, 2008. Available at `http://eprint.iacr.org/2008/040.pdf`.
26. V.S. Miller. Short programs for functions on curves. Available at `http://crypto.stanford.edu/miller`, 1986.
27. V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, September 2004.
28. S. Mitsunari. Xbyak: JIT assembler for C++. Available at `http://homepage1.nifty.com/herumi/soft/xbyak_e.html`.
29. A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84:1234–1243, 2001.
30. M. Naehrig. *Constructive and Computational Aspects of Cryptographic Pairings.* PhD thesis, Technische Universiteit Eindhoven, 2009. Available at `http://www.cryptojedi.org/users/michael/data/thesis/2009-05-13-diss.pdf`.
31. M. Naehrig, P.S.L.M. Barreto, and P. Schwabe. On compressible pairings and their computation. In S. Vaudenay, editor, *Progress in Cryptology–AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2008.
32. M. Naehrig, R. Niederhagen, and P. Schwabe. New software speed records for cryptographic pairings. Cryptology ePrint Archive, Report 2010/186, 2010. Available at `http://eprint.iacr.org/2010/186.pdf`.

33. P. Schwabe. Software library of "New software speed records for cryptographic pairings", Accessed June, 4, 2010. Available at `http://cryptojedi.org/crypto/#dclxvi`.
34. M. Scott. Implementing cryptographic pairings. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, *Pairing-Based Cryptography–Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 177–196. Springer, 2007.
35. M. Scott and P.S.L.M. Barreto. Compressed pairings. In M.K. Franklin, editor, *Advances in Cryptology–CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2004.
36. M. Scott, N. Benger, M. Charlemagne, L.J. Dominguez Perez, and E.J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. Cryptology ePrint Archive, Report 2008/490, 2008. Available at `http://eprint.iacr.org/2008/490.pdf`.
37. C. Shu, S. Kwon, and K. Gaj. Reconfigurable computing approach for Tate pairing cryptosystems over binary fields. *IEEE Transactions on Computers*, 58(9):1221–1237, September 2009.
38. F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, January 2010.