Efficient Implementation of Bilinear Pairings on ARM Processors

Gurleen Grewal[1], Reza Azarderakhsh[1], Patrick Longa[2], Shi Hu[3], and David Jao[1]

[1] Department of Combinatorics and Optimization,
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
grewal.gurleen@ymail.com
{razarder and djao}@math.uwaterloo.ca
[2] Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
plonga@microsoft.com
[3]Stanford University, Stanford, California, USA
s3hu@stanford.edu

Abstract. As hardware capabilities increase, low-power devices such as smartphones represent a natural environment for the efficient implementation of cryptographic pairings. Few works in the literature have considered such platforms despite their growing importance in a post-PC world. In this paper, we investigate the efficient computation of the Optimal-Ate pairing over Barreto-Naehrig curves in software at different security levels on ARM processors. We exploit state-of-the-art techniques and propose new optimizations to speed up the computation in the tower field and curve arithmetic. In particular, we extend the concept of lazy reduction to inversion in extension fields, analyze an efficient alternative for the sparse multiplication used inside the Miller's algorithm and reduce further the cost of point/line evaluation formulas in affine and projective homogeneous coordinates. In addition, we study the efficiency of using M-type sextic twists in the pairing computation and carry out a detailed comparison between affine and projective coordinate systems. Our implementations on various mass-market smartphones and tablets significantly improve the state-of-the-art of pairing computation on ARM-powered devices, outperforming by at least a factor of 3.5 the best previous results in the literature.

Keywords: Optimal-Ate pairing, Barreto-Naehrig curves, ARM processor, pairing implementation.

1 Introduction

In the past decade, bilinear pairings have found a range of constructive applications in areas such as identity-based encryption and short signatures. Naturally, implementing such protocols requires efficient computation of the pairing function. Considerable work has been done to compute fast pairings on

PCs [3,6,8,15,17]. Most recently, Aranha et al. [3] have computed the O-Ate pairing at the 128-bit security level in under 2 million cycles on various 64-bit PC processors. In contrast, relatively few articles [1,10] have considered efficient software implementations of pairings on ARM-based platforms such as handheld smartphones and tablets. These platforms are widely predicted to become a dominant computing platform in the near future. Therefore, efficient implementation of pairing-based protocols for these devices is crucial for deployment of pairing-based cryptography in a mobile world and represents a natural area of research.

In this paper, we investigate efficient pairing computations at multiple security levels across different generations of ARM-based processors. We extend the work of Aranha et al. [3] to different BN curves and higher security levels. In addition, we make several further optimizations and analyze different options available for implementation at various stages of the pairing computation. We summarize our contributions as follows:

- Firstly, we extend the concept of lazy reduction employed by Aranha et al. [3] (see also Longa [13, Chapter 6]) to inversion in extension fields. We also optimize the sparse multiplication algorithm in the degree 12 extension.
- We examine different choices of towers for extension field arithmetic over various prime fields including BN-254, BN-446, and BN-638 [8]. We determine the most efficient implementation of extension fields in the context of pairing computation over BN-curves from the various choices available.
- The M-type sextic twist [16] has been largely ignored for use in pairing computations, most likely due to the inefficient untwisting map. We demonstrate that by computing the pairing on the twisted curve, we can bypass the inefficient untwisting. As a result, for the purposes of optimization one can use either M-type or D-type twists, thus roughly doubling the available choice of curves.
- Finally, we implement the proposed algorithms for computing the O-Ate pairing over BN curves on different ARM-based platforms and compare our measured timing results to their counterparts in the literature. Our experimental results are 3 to 5 times faster than the fastest available in prior literature, depending on the security level.

Acar et al. [1] have recently raised the question of whether affine coordinates or projective coordinates are a better choice for curve arithmetic in the context of software implementation of pairings. Their conclusion is that affine coordinates are faster at all security levels at or above 128 bits on the ARM platform. In contrast, our results (Section 6.2) demonstrate a clear advantage for homogeneous projective coordinates at the 128-bit security level, although affine coordinates remain faster at the 256-bit security level. We believe that our findings are more reliable since they represent a more realistic amount of optimization of the underlying field arithmetic implementation. We stress that, except for the work described in Section 6.1, our code does not contain any hand-optimized assembly or any overly aggressive optimizations that would compromise portability or maintainability.

The rest of this paper is organized as follows. In Section 2, we provide some background on the O-Ate pairing. In Section 3, we discuss the representation of extension fields. In Section 4, we describe arithmetic on BN curves including point addition and doubling presented in different coordinates. We provide operation counts for our algorithms in Section 5. In Section 6, we present the results of our implementation of the proposed scheme for computing O-Ate pairings on different ARM processors, and compare them with prior work.

2 Preliminaries

Barreto and Naehrig [4] describe a family of pairing friendly curves $E : y^2 = x^3 + b$ of order n with embedding degree 12 defined over a prime field $\mathbb{F}_q$ where q and n are given by the polynomials:

$$\begin{aligned} q &= 36x^4 + 36x^3 + 24x^2 + 6x + 1 \\ n &= 36x^4 + 36x^3 + 18x^2 + 6x + 1, \end{aligned} \tag{1}$$

for some integer x such that both q and n are prime and $b \in \mathbb{F}_q^*$ such that $b + 1$ is a quadratic residue.

Let $\Pi_q : E \to E$ be the q-power Frobenius. Set $\mathbb{G}_1 = E[n] \cap \ker(\Pi_q - [1])$ and $\mathbb{G}_2 = E[n] \cap \ker(\Pi_q - [q])$. It is known that points in $\mathbb{G}_1$ have coordinates in $\mathbb{F}_q$, and points in $\mathbb{G}_2$ have coordinates in $\mathbb{F}_{q^{12}}$. The Optimal-Ate or O-Ate pairing [18] on E is defined by:

$$a_{\text{opt}} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mu_n, (Q, P) \to f_{6x+2,Q}(P) \cdot h(P) \tag{2}$$

where $h(P) = l_{[6x+2]Q,qQ}(P) l_{[6x+2]Q+qQ,-q^2Q}(P)$ and $f_{6x+2,Q}(P)$ is the appropriate Miller function. Also, $l_{Q_1,Q_2}(P)$ is the line arising in the addition of Q_1 and Q_2 at point P. This function can be computed using Miller's algorithm [14]. A modified version of the algorithm from [14] which uses a NAF representation of x is given in Algorithm 1.

Let ξ be a quadratic and cubic non-residue over $\mathbb{F}_{q^2}$. Then the curves $E' : y^2 = x^3 + \frac{b}{\xi}$ (D-type) and $E'' : y^2 = x^3 + b\xi$ (M-type) are sextic twists of E over $\mathbb{F}_{q^2}$, and exactly one of them has order dividing n [3]. For this twist, the image $\mathbb{G}_2^{'}$ of $\mathbb{G}_2$ under the twisting isomorphism lies entirely in $E'(\mathbb{F}_{q^2})$. Instead of using a degree 12 extension, the point Q can now be represented using only elements in a quadratic extension field. In addition, when performing curve arithmetic and computing the line function in the Miller loop, one can perform the arithmetic in $\mathbb{G}_2^{'}$ and then map the result to $\mathbb{G}_2$, which considerably speeds up operations in the Miller loop.

2.1 Notations and Definitions

Throughout this paper, lower case variables denote single-precision integers, upper case variables denote double-precision integers. The operation $\times$ represents

Algorithm 1 Miller's Algorithm for the O-Ate Pairing [14].

```
Input: Points $P, Q \in E[n]$ and integer $n = (n_{l-1}, n_{l-2}, \cdots, n_1, n_0)_2 \in \mathbb{N}$.
Output: $f_{n,P}(Q)^{\frac{q^k-1}{n}}$.
 1: $T \leftarrow P,\ f \leftarrow 1$
 2: for $i = l-2$ down to $0$ do
 3:     $f \leftarrow f^2 \cdot l_{T,T}(Q)$
 4:     $T \leftarrow 2T$
 5:     if $l_i \neq 0$ then
 6:         $f \leftarrow f \cdot l_{T,P}(Q)$
 7:         $T \leftarrow T + P$
 8:     end if
 9: end for
10: $f \leftarrow f^{\frac{q^k-1}{n}}$
11: return $f$
```

multiplication without reduction, and $\otimes$ represents multiplication with reduction. The quantities m, s, a, i, and r denote the times for multiplication, squaring, addition, inversion, and modular reduction in $\mathbb{F}_q$, respectively. Likewise, $\tilde{m}, \tilde{s}, \tilde{a}$, $\tilde{i}$, and $\tilde{r}$ denote times for multiplication, squaring, addition, inversion, and reduction in $\mathbb{F}_{q^2}$, respectively, and m_u, s_u, $\tilde{m}_u$, and $\tilde{s}_u$ denote times for multiplication and squaring without reduction in the corresponding fields. Finally, m_b, m_i, m_ξ, and m_v denote times for multiplication by the quantities b, i, ξ, and v from Section 3.

3 Representation of Extension Fields

Efficient implementation of the underlying extension fields is crucial to achieve fast pairing results. The IEEE P1363.3 standard [9] recommends using towers to represent $\mathbb{F}_{q^k}$. For primes q congruent to 3 mod 8, we employ the following construction of Benger and Scott [5] to construct tower fields:

Property 1. For approximately 2/3rds of the BN-primes $q \equiv 3 \bmod 8$, the polynomial $y^6 - \alpha$, $\alpha = 1 + \sqrt{-1}$ is irreducible over $\mathbb{F}_{q^2} = \mathbb{F}_q(\sqrt{-1})$.

This gives the following towering scheme:

$$\begin{cases} \mathbb{F}_{q^2} = \mathbb{F}_q[i]/(i^2 - \beta), & \text{where } \beta = -1. \\ \mathbb{F}_{q^6} = \mathbb{F}_{q^2}[v]/(v^3 - \xi), & \text{where } \xi = 1 + i. \\ \mathbb{F}_{q^{12}} = \mathbb{F}_{q^6}[w]/(w^2 - v). & \end{cases}$$

Based on this scheme, multiplication by i requires one negation over $\mathbb{F}_q$, and multiplication by ξ requires only one addition over $\mathbb{F}_{q^2}$. For primes congruent to 7 mod 8, we use the following construction which can be proven using the same ideas as those in Benger and Scott [5]:

Property 2. For approximately 2/3rds of the BN-primes $q \equiv 7 \bmod 8$, the polynomial $y^6 - \alpha$, $\alpha = 1 + \sqrt{-2}$ is irreducible over $\mathbb{F}_{q^2} = \mathbb{F}_q(\sqrt{-2})$.

This gives the following towering scheme:

$$\begin{cases} \mathbb{F}_{q^2} = \mathbb{F}_q[i]/(i^2 - \beta), & \text{where } \beta = -2. \\ \mathbb{F}_{q^6} = \mathbb{F}_{q^2}[v]/(v^3 - \xi), & \text{where } \xi = 1 + i. \\ \mathbb{F}_{q^{12}} = \mathbb{F}_{q^6}[w]/(w^2 - v). & \end{cases}$$

All things being equal, the towering scheme derived from Property 1 is slightly faster for a given bit size. However, in practice, desirable BN-curves are rare, and it is sometimes necessary to use primes $q \equiv 7 \bmod 8$ in order to optimize other aspects such as the Hamming weight of x. In particular, the curves BN-446 and BN-638 [8] have $q \equiv 7 \bmod 8$. In such cases, Property 1 does not apply, so we use the towering scheme derived from Property 2. We also considered other approaches to construct tower extensions as suggested in [8], but found the above schemes consistently resulted in faster pairings compared to the other options.

3.1 Finite Field Operations and Lazy Reduction

Aranha et al. [3] proposed a lazy reduction scheme for efficient pairing computation in tower-friendly fields and curve arithmetic using projective coordinates. We extensively exploit their method and extend it to field inversion and curve arithmetic over affine coordinates. The proposed schemes using lazy reduction for inversion are given in the Appendix as Algorithms 2, 3 and 4 for $\mathbb{F}_{q^2}$, $\mathbb{F}_{q^6}$ and $\mathbb{F}_{q^{12}}$, respectively. The total savings with lazy reduction vs. no lazy reduction are one $\mathbb{F}_q$-reduction in $\mathbb{F}_{q^2}$-inversion, and 36 $\mathbb{F}_q$-reductions in $\mathbb{F}_{q^{12}}$-inversion (improving upon [3] by 16 $\mathbb{F}_q$-reductions). Interestingly enough, if one applies the lazy reduction technique to the recent $\mathbb{F}_{q^{12}}$ inversion algorithm of Pereira et al. [8], it replaces two $\tilde{m}_u$ by two $\tilde{s}_u$ but requires five more $\tilde{r}$ operations, which ultimately makes it slower in practice in comparison with the proposed scheme.

Algorithm 2 Inversion over $\mathbb{F}_{q^2}$ employing lazy reduction technique

```
Input: $a = a_0 + a_1 i;\ a_0, a_1 \in \mathbb{F}_q;\ \beta$ is a quadratic non-residue over $\mathbb{F}_q$
Output: $c = a^{-1} \in \mathbb{F}_{q^2}$
  $T_0 \leftarrow a_0 \times a_0$
  $T_1 \leftarrow -\beta \cdot (a_1 \times a_1)$
  $T_0 \leftarrow T_0 + T_1$
  $t_0 \leftarrow T_0 \bmod p$
  $t_0 \leftarrow t_0^{-1} \bmod p$
  $c_0 \leftarrow a_0 \otimes t_0$
  $c_1 \leftarrow -(a_1 \otimes t_0)$
  return $c = c_0 + c_1 i$
```

Algorithm 3 Inversion over $\mathbb{F}_{q^6}$ employing lazy reduction technique

```
Input: $a = a_0 + a_1 v + a_2 v^2;\ a_0, a_1, a_2 \in \mathbb{F}_{q^2}$
Output: $c = a^{-1} \in \mathbb{F}_{q^6}$
  $T_0 \leftarrow a_0 \times a_0$
  $t_0 \leftarrow \xi a_1$
  $T_1 \leftarrow t_0 \times a_2$
  $T_0 \leftarrow T_0 - T_1$
  $t_1 \leftarrow T_0 \bmod p$
  $T_0 \leftarrow a_2 \times a_2$
  $T_0 \leftarrow \xi T_0$
  $T_1 \leftarrow a_0 \times a_1$
  $T_0 \leftarrow T_0 - T_1$
  $t_2 \leftarrow T_0 \bmod p$
  $T_0 \leftarrow a_1 \times a_1$
  $T_1 \leftarrow a_0 \times a_2$
  $T_0 \leftarrow T_0 - T_1$
  $t_3 \leftarrow T_0 \bmod p$
  $T_0 \leftarrow t_0 \times t_3$
  $T_1 \leftarrow a_0 \times t_1$
  $T_0 \leftarrow T_0 + T_1$
  $t_0 \leftarrow \xi a_2$
  $T_1 \leftarrow t_0 \times t_2$
  $T_0 \leftarrow T_0 + T_1$
  $t_0 \leftarrow T_0 \bmod p$
  $t_0 \leftarrow t_0^{-1}$
  $c_0 \leftarrow t_1 \otimes t_0$
  $c_1 \leftarrow t_2 \otimes t_0$
  $c_2 \leftarrow t_3 \otimes t_0$
  return $c = c_1 + c_2 v + c_3 v^2$
```

Algorithm 4 Inversion over $\mathbb{F}_{q^{12}}$ employing lazy reduction technique

```
Input: $a = a_0 + a_1 w;\ a_0, a_1 \in \mathbb{F}_{q^6}$
Output: $c = a^{-1} \in \mathbb{F}_{q^{12}}$
  $T_0 \leftarrow a_0 \times a_0$
  $T_1 \leftarrow v \cdot (a_1 \times a_1)$
  $T_0 \leftarrow T_0 - T_1$
  $t_0 \leftarrow T_0 \bmod p$
  $t_0 \leftarrow t_0^{-1} \bmod p$
  $c_0 \leftarrow a_0 \otimes t_0$
  $c_1 \leftarrow -a_1 \otimes t_0$
  return $c = c_0 + c_1 w$
```

The line function in the Miller loop evaluates to a sparse $\mathbb{F}_{q^{12}}$ element containing only three of the six basis elements over $\mathbb{F}_{q^2}$. Thus, when multiplying the line function output with $f_{i,Q}(P)$, one can utilize the sparseness property to avoid full $\mathbb{F}_{q^{12}}$ arithmetic (Algorithm 5). For the BN-254 curve [8], our sparse multiplication algorithm requires $13\tilde{m}$ and $44\tilde{a}$ when a D-type twist is involved.

Algorithm 5 D-type sparse-dense multiplication in $\mathbb{F}_{q^{12}}$

```
Input: $a = a_0 + a_1 w + a_2 vw;\ a_0, a_1, a_2 \in \mathbb{F}_{q^2}, b = b_0 + b_1 w;\ b_0, b_1 \in \mathbb{F}_{q^6}$
Output: $ab \in \mathbb{F}_{q^{12}}$
  $A_0 \leftarrow a_0 \times b_0[0],\ A_1 \leftarrow a_0 \times b_0[1],\ A_2 \leftarrow a_0 \times b_0[2]$
  $A \leftarrow A_0 + A_1 v + A_2 v^2$
  $B \leftarrow \mathrm{Fq6SparseMul}(a_1 w + a_2 vw, b_1)$
  $c_0 \leftarrow a_0 + a_1, c_1 \leftarrow a_2, c_2 \leftarrow 0$
  $c \leftarrow c_0 + c_1 v + c_2 v^2$
  $d \leftarrow b_0 + b_1$
  $E \leftarrow \mathrm{Fq6SparseMul}(c, d)$
  $F \leftarrow E - (A + B)$
  $G \leftarrow Bv$
  $H \leftarrow A + G$
  $c_0 \leftarrow H \bmod p$
  $c_1 \leftarrow F \bmod p$
  return $c = c_0 + c_1 w$
```

Algorithm 6 Fq6SparseMul, used in Algorithm 5

```
Input: $a = a_0 + a_1 v;\ a_0, a_1 \in \mathbb{F}_{q^2}, b = b_0 + b_1 v + b_2 v^2;\ b_0, b_1, b_2 \in \mathbb{F}_{q^2}$
Output: $ab \in \mathbb{F}_{q^6}$
  $A \leftarrow a_0 \times b_0,\ B \leftarrow a_1 \times b_1$
  $C \leftarrow a_1 \times b_2 \xi$
  $D \leftarrow A + C$
  $e \leftarrow a_0 + a_1, f \leftarrow b_0 + b_1$
  $E \leftarrow e \times f$
  $G \leftarrow E - (A + B)$
  $H \leftarrow a_0 \times b_2$
  $I \leftarrow H + B$
  return $D + Gv + Iv^2$
```

A similar dense-sparse multiplication algorithm works for M-type twists, and requires an extra multiplication by v. We note that our approach requires 13 fewer additions over $\mathbb{F}_{q^2}$ compared to the one used in [3] (lazy reduction versions).

3.2 Mapping from the Twisted Curve to the Original Curve

Suppose we take ξ (from the towering scheme) to be the cubic and quadratic non-residue used to generate the sextic twist of the BN-curve E. After manipulating points on the twisted curve, they need to be mapped to the original curve. In the case of a D-type twist, the untwisting isomorphism is given by:

$$\Psi : (x, y) \to (\xi^{\frac{1}{3}} x, \xi^{\frac{1}{2}} y) = (w^2 x, w^3 y), \tag{3}$$

where both w^2 and w^3 are basis elements, and hence the untwisting map is almost free. If one uses a M-type twist the untwisting isomorphism is given as follows:

$$\Psi : (x, y) \to (\xi^{-\frac{2}{3}} x, \xi^{-\frac{1}{2}} y) = (\xi^{-1} w^4 x, \xi^{-1} w^3 y). \tag{4}$$

Untwisting of (4) is not efficient as the one given in (3). However, if we compute the pairing value on the twisted curve instead of the original curve, then we do not need to use the untwisting map. Instead, we require the inverse map which is almost free. Therefore, we compute the pairing on the original curve E when a D-type twist is involved, and on the twisted curve E' when an M-type twist is involved. Using this approach, we have found that both twist types are equivalent in performance up to point/line evaluation. The advantage of being able to consider both twist types is the immediate availability of many more useful curves for pairing computation.

3.3 Final Exponentiation Scheme

We use the final exponentiation scheme proposed in [7], which represents the current state-of-the-art for BN curves. In this scheme, first $\frac{q^{12}-1}{n}$ is factored into q^6-1, q^2+1, and $\frac{q^4-q^2+1}{n}$. The first two factors are easy to exponentiate. The remaining exponentiation $\frac{q^4-q^2+1}{n}$ can be performed in the cyclotomic subgroup. Using the fact that any fixed non-degenerate power of a pairing is a pairing, we raise to a multiple of the remaining factor. Recall that q and n are polynomials in x, and hence so is the final factor. We denote this polynomial as $d(x)$. In [7] it is shown that

$$\begin{aligned}2x(6x^2+3x+1)d(x) = {}& \lambda_3 q^3 + \lambda_2 q^2 + \lambda_1 q + \lambda_0 1 + 6x + 12x^2 + 12x^3 \\ & + (4x+6x^2+12x^3)p(x) + (6x+6x^2+12x^3)p(x)^2 \\ & + (-1+4x+6x^2+12x^3)p(x)^3, \qquad (5)\end{aligned}$$

where

$$\begin{aligned}\lambda_3(x) &= -1+4x+6x^2+12x^3, \\ \lambda_2(x) &= 6x+6x^2+12x^3 \\ \lambda_1(x) &= 4x+6x^2+12x^3 \\ \lambda_0(x) &= 1+6x+12x^2+12x^3. \qquad (6)\end{aligned}$$

To compute (6), the following exponentiations are performed:

$$f \mapsto f^x \mapsto f^{2x} \mapsto f^{4x} \mapsto f^{6x} \mapsto f^{6x^2} \mapsto f^{12x^2} \mapsto f^{12x^3}. \qquad (7)$$

The cost of computing (7) is 3 exponentiations by x, 3 squarings and 1 multiplication. We then compute the terms $a = f^{12x^3} f^{6x^2} f^{6x}$ and $b = a(f^{2x})^{-1}$, which require 3 multiplications. The final pairing value is obtained as

$$af^{6x^2} f b^p a^{p^2} (bf^{-1})^{p^3}, \qquad (8)$$

which costs 6 multiplications and 6 Frobenius operations. In total, this method requires 3 exponentiations by x, 3 squarings, 10 multiplications, and 3 Frobenius operations. In comparison, the technique used in [3] requires 3 additional multiplications and an additional squaring, and thus is slightly slower.

4 Curve Arithmetic

In this section, we discuss our optimizations to curve arithmetic over affine and homogeneous projective coordinates. We also evaluated other coordinate systems such as Jacobian coordinates but found that none were faster than homogeneous coordinates for our application.

4.1 Affine Coordinates

Let the points $T = (x, y)$ and $Q = (x_2, y_2) \in E'(\mathbb{F}_q)$ be given in affine coordinates, and let $T + Q = (x_3, y_3)$ be the sum of the points T and Q. When $T = Q$ we have

$$\begin{aligned} m &= \frac{3x^2}{2y} \\ x_3 &= m^2 - 2x \\ y_3 &= (mx - y) - mx_3 \end{aligned}$$

For D-type twists, the secant or tangent line evaluated at $P = (x_P, y_P)$ is given by:

$$l_{2\Psi(T)}(P) = y_P - mx_Pw + (mx - y)w^3. \tag{9}$$

To compute the above, we precompute $\bar{x}_P = -x_P$ (to save the cost of computing the negation on-the-fly) and use the following sequence of operations which requires $1\tilde{i}$, $3\tilde{m}$, $2\tilde{s}$, $7\tilde{a}$, and $2m$ if $T = Q$. In comparison, the doubling formula in Lauter et al. [12] costs 3 additional $\tilde{a}$.

$$A = \frac{1}{2y} \qquad B = 3x^2 \qquad C = AB \qquad D = 2x \qquad x_3 = C^2 - D$$

$$E = Cx - y \qquad y_3 = E - Cx_3 \qquad F = C\bar{x}_P$$

$$l_{2\Psi(T)}(P) = y_P + Fw + Ew^3$$

Similarly, when $T \neq Q$ we use the following sequence of operations which requires $1\tilde{i}$, $3\tilde{m}$, $1\tilde{s}$, $6\tilde{a}$, and $2m$ – saving $2\tilde{a}$ compared to the addition formula in Lauter et al. [12].

$$A = \frac{1}{y_2 - y} \qquad B = x_2 - x \qquad C = AB \qquad D = x + x_2 \qquad x_3 = C^2 - D$$

$$E = Cx - y \qquad y_3 = E - Cx_3 \qquad F = C\bar{x}_P$$

$$l_{2\Psi(T)}(P) = y_P + Fw + Ew^3$$

In an M-type twist, the tangent line evaluated at $\Psi(P) = (x_Pw^2, y_Pw^3)$ is given by:

$$l_{2T}(\Psi(P)) = y_Pw^3 - mx_Pw^2 + (mx - y), \tag{10}$$

and can be computed in a similar way.

4.2 Homogeneous Coordinates

During the first iteration of the Miller loop, the Z-coordinate of the point Q has value equal to 1. We use this fact to eliminate a multiplication and three squarings using a special first doubling routine in the first iteration. Recently, Arahna et al. [3], presented optimized formulas for point doubling/line evaluation. We note that the twisting point P given by $(x_P/w^2, y_P/w^3)$ is better represented by $(x_P w, y_P)$, which is obtained by multiplying by w^3. This eliminates the multiplication by ξ and gives the following revised formula. Let $T = (X, Y, Z) \in E'(\mathbb{F}_{q^2})$ be in homogeneous coordinates. Then $2T = (X_3, Y_3, Z_3)$ is given by:

$$\begin{aligned} X_3 &= \frac{XY}{2}(Y^2 - 9b'Z^2) \\ Y_3 &= \left[\frac{1}{2}(Y^2 + 9b'Z^2)\right]^2 - 27b'^2Z^4 \\ Z_3 &= 2Y^3Z \end{aligned}$$

In the case of a D-type twist, the corresponding line function evaluated at $P = (x_P, y_P)$ is given by:

$$l_{2\Psi(T)}(P) = -2YZy_P + 3X^2x_Pw + (3b'Z^2 - Y^2)w^3$$

We compute this value using the following sequence of operations.

$$A = \frac{XY}{2} \quad B = Y^2 \quad C = Z^2 \quad E = 3b'C \quad F = 3E \quad X_3 = A \cdot (B - F)$$

$$G = \frac{B + f}{2} \quad Y_3 = G^2 - 3E^2 \quad H = (Y + Z)^2 - (B + C) \quad Z_3 = B \cdot H$$

$$l_{2\Psi(T)}(P) = H\bar{y}_P + 3X^2x_Pw + (E - B)w^3$$

Aranha et al. [3] observe $\tilde{m} - \tilde{s} \approx 3\tilde{a}$ and hence computing XY directly is faster than using $(X + Y)^2$, Y^2 and X^2 on a PC. However, on ARM processors, we have $\tilde{m} - \tilde{s} \approx 6\tilde{a}$. Thus, the latter technique is more efficient on ARM processors. The overall cost of point doubling and line evaluation is $2\tilde{m}$, $7\tilde{s}$, $22\tilde{a}$, and $4m$, assuming that the cost of division by two and multiplication by b' are equivalent to the cost of addition. Similarly, we compute point addition and line function evaluation using the following sequence of operations which uses $11\tilde{m}$, $2\tilde{s}$, $8\tilde{a}$, and $4m$ (saving 2 $\mathbb{F}_{q^2}$ additions over [3]). Note that $\bar{x}_P$ and $\bar{y}_P$ are precomputed to save again the cost of computing $-x_P$ and $-y_P$.

$$A = Y_2Z \quad B = X_2Z \quad \theta = Y - A \quad \lambda = X - B \quad C = \theta^2$$

$$D = \lambda^2 \quad E = \lambda^3 \quad F = ZC \quad G = XD \quad H = E + F - 2G$$

$$X_3 = \lambda H \quad I = YE \quad Y_3 = \theta(G - H) - I \quad Z_3 = ZE \quad J = \theta X_2 - \lambda Y_2$$

$$l_{\Psi(T+Q)}(P) = \lambda\bar{y}_P + \theta\bar{x}_Pw + Jw^3$$

In the case of an M-type twist the corresponding line computation can be computed using the same sequences of operations as above. As in [3], we also use lazy reduction techniques to optimize the above formulae (see Table 1).

Table 1. Operation counts for 254-bit, 446-bit, and 638-bit prime fields

$E'(\mathbb{F}_{p^2})$ **Arith.**	254-bit	446-bit/638-bit
Doubl/Eval (Proj)	$2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 25\tilde{a} + 4m$	$2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 34\tilde{a} + a + 4m$
Doubl/Eval (Affi)	$\tilde{i} + 3\tilde{m}_u + 2\tilde{s}_u + 5\tilde{r} + 7\tilde{a} + 2m$	$\tilde{i} + 3\tilde{m}_u + 2\tilde{s}_u + 5\tilde{r} + 7\tilde{a} + 2m$
Add./Eval (Pro)	$11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 10\tilde{a} + 4m$	$11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 10\tilde{a} + 4m$
Add/Eval (Affi)	$\tilde{i} + 3\tilde{m}_u + \tilde{s}_u + 4\tilde{r} + 6\tilde{a} + 2m$	$\tilde{i} + 2\tilde{m}_u + \tilde{s}_u + 3\tilde{r} + 6\tilde{a} + 2m$
First doubl./Eval	$3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 14\tilde{a} + 4m$	$3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 23\tilde{a} + a + 4m$
p-power Frob.	$2\tilde{m} + 2a$	$8\tilde{m} + 2a$
p^2- power Frob.	$4m$	$16\tilde{m} + 4a$
$\mathbb{F}_{p^2}$ **Arith.**	254-bit	446-bit/638-bit
Add/Subtr./Nega.	$\tilde{a} = 2a$	$\tilde{a} = 2a$
Mult.	$\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r + 8a$	$\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r + 10a$
Squaring	$\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r + 3a$	$\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r + 5a$
Mult. by β	$m_b = a$	$m_b = 2a$
Mult. by ξ	$m_\xi = 2a$	$m_\xi = 3a$
Inversion	$\tilde{i} = i + 2m_u + 2s_u + 3r + 3a$	$\tilde{i} = i + 2m_u + 2s_u + 3r + 5a$
$\mathbb{F}_{p^{12}}$ **Arith.**	254-bit	446-bit/638-bit
Multi.	$18\tilde{m}_u + 110\tilde{a} + 6\tilde{r}$	$18\tilde{m}_u + 117\tilde{a} + 6\tilde{r}$
Sparse Mult.	$13\tilde{m}_u + 6\tilde{r} + 48\tilde{a}$	$13\tilde{m}_u + 6\tilde{r} + 54\tilde{a}$
Sparser Mult.	$6\tilde{m}_u + 6\tilde{r} + 13\tilde{a}$	$6\tilde{m}_u + 6\tilde{r} + 14\tilde{a}$
Affi. Sparse Mult.	$10\tilde{m}_u + 6\tilde{r} + 47\tilde{a} + 6m_u + a$	$10\tilde{m}_u + 53\tilde{a} + 6\tilde{r} + 6m_u + a$
Squaring	$12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}$	$12\tilde{m}_u + 6\tilde{r} + 78\tilde{a}$
Cyclotomic Sqr.	$9\tilde{s}_u + 46\tilde{a} + 6\tilde{r}$	$9\tilde{s}_u + 49\tilde{a} + a + 6\tilde{r}$
Simult. Decomp.	$9\tilde{m} + 6\tilde{s} + 22\tilde{a} + \tilde{i}$	$9\tilde{m} + 6\tilde{s} + 24\tilde{a} + \tilde{i}$ (BN-446) $16\tilde{m} + 9\tilde{s} + 35\tilde{a} + \tilde{i}$ (BN-638)
p-power Frob.	$5\tilde{m} + 6a$	$5\tilde{m} + 6a$
p^2-power Frob.	$10m + 2\tilde{a}$	$10m + 2\tilde{a}$
Expon. by x	$45\tilde{m}_u + 378\tilde{s}_u +$ $275\tilde{r} + 2164\tilde{a} + \tilde{i}$	$45\tilde{m}_u + 666\tilde{s}_u + 467\tilde{r}_u +$ $3943\tilde{a} + \tilde{i}$ (BN-446)
		$70\tilde{m} + 948\tilde{s} + 675\tilde{r} +$ $5606\tilde{a} + 158a + \tilde{i}$ (BN-638)
Inversion	$25\tilde{m}_u + 9\tilde{s}_u + 16\tilde{r} + 121\tilde{a} + \tilde{i}$	$25\tilde{m}_u + 9\tilde{s}_u + 18\tilde{r} + 138\tilde{a} + \tilde{i}$
Compressed Sqr.	$6\tilde{s}_u + 31\tilde{a} + 4\tilde{r}$	$6\tilde{s}_u + 33\tilde{a} + a + 4\tilde{r}$

5 Operation Counts

We provide here detailed operation counts for our algorithms on the BN-254, BN-446, and BN-638 curves used in Acar et. al [1] and defined in [8]. Table 1 provides the operation counts for all component operations. Numbers for BN-446 and BN-638 are the same except where indicated.

For BN-254, using the techniques described above, the projective pairing Miller loop executes one negation in $\mathbb{F}_q$, one first doubling with line evaluation, 63 point doublings with line evaluations, 6 point additions with line evaluations, one p-power Frobenius in $E'(\mathbb{F}_{p^2})$, one p^2-power Frobenius in $E'(\mathbb{F}_{p^2})$, 66 sparse multiplications, 63 squarings in $\mathbb{F}_{p^2}$, 1 negation in $E'(\mathbb{F}_{p^2})$, 2 sparser (i.e.

Table 2. Cost of the computation of O-Ate pairings using various coordinates

Curve	Coord.	Cost
BN-254	Proj. Miller loop	$1841\tilde{m}_u + 457\tilde{s}_u + 1371\tilde{r} + 9516\tilde{a} + 284m + 3a$
	Affi. Miller loop	$70\tilde{i} + 1658\tilde{m}_u + 134\tilde{s}_u + 942\tilde{r} + 8292\tilde{a} + 540m + 132a$
	Final exponen.	$386\tilde{m}_u + 1164\tilde{s}_u + 943\tilde{r} + 4\tilde{i} + 7989\tilde{a} + 30m + 15a$
BN-446	Proj. Miller loop	$3151\tilde{m}_u + 793\tilde{s}_u + 2345\tilde{r} + 18595\tilde{a} + 472m + 117a$
	Affi. Miller loop	$118\tilde{i} + 2872\tilde{m}_u + 230\tilde{s}_u + 1610\tilde{r} + 15612\tilde{a} + 920m + 230a$
	Final exponen.	$386\tilde{m}_u + 2034\tilde{s}_u + 1519\tilde{r} + 4\tilde{i} + 13374\tilde{a} + 30m + 345a$
BN-638	Proj. Miller loop	$4548\tilde{m}_u + 1140\tilde{s}_u + 3557\tilde{r} + 27198\tilde{a} + 676m + 166a$
	Affi. Miller loop	$169\tilde{i} + 4143\tilde{m}_u + 330\tilde{s}_u + 2324\tilde{r} + 22574\tilde{a} + 1340m + 333a$
	Final exponen.	$436\tilde{m}_u + 2880\tilde{s}_u + 2143\tilde{r} + 4\tilde{i} + 18528\tilde{a} + 30m + 489a$

sparse-sparse) multiplications [3], and 1 multiplication in $\mathbb{F}_{p^{12}}$. Using Table 1, we compute the total number of operations required in the Miller loop using homogeneous projective coordinates to be

$$\begin{aligned}\text{ML254P} &= a + 3\tilde{m}_u + 7\tilde{r} + 14\tilde{a} + 4m + 63(2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 25\tilde{a} + 4m) + \\ &\quad 6(11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 10\tilde{a} + 4m) + 2\tilde{m} + 2a + 4m + 66(\tilde{m}_u + 6\tilde{r} + 48\tilde{a}) + \\ &\quad 63(12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}) + \tilde{a} + 2(6\tilde{m}_u + 6\tilde{r} + 13\tilde{a}) + 18\tilde{m}_u + 110\tilde{a} + 6\tilde{r} \\ &= 1841\tilde{m}_u + 457\tilde{s}_u + 1371\tilde{r} + 9516\tilde{a} + 284m + 3a.\end{aligned}$$

Similarly, we also compute the Miller loop operation costs for BN-446 and BN-638 and for projective and affine coordinates, and give the results in Table 2.

We also compute the operation count for the final exponentiation. For BN-254, the final exponentiation requires 6 conjugations in $\mathbb{F}_{p^{12}}$, one negation in $E'(\mathbb{F}_{p^2})$, one inversion in $\mathbb{F}_{p^{12}}$, 12 multiplications in $\mathbb{F}_{p^{12}}$, two p-power Frobenius in $\mathbb{F}_{p^{12}}$, 3 p^2-power Frobenius in $\mathbb{F}_{p^{12}}$, 3 exponentiations by x, and 3 cyclotomic squarings. Based on these costs, we compute the total number of operations required in the final exponentiation and give the result in Table 2. Similarly, we also compute the final exponentiation operation cost for BN-446 and BN-638. The total operation count for the pairing computation is the cost of the Miller loop plus the final exponentiation.

6 Implementation Results

To evaluate the performance of the proposed schemes for computing the O-Ate pairing in practice, we implemented them on various ARM processors. We used the following platforms in our experiments.

- A Marvell Kirkwood 6281 ARMv5 CPU processor (Feroceon 88FR131) operating at 1.2 GHz. In terms of registers it has 16 32-bit registers $\mathtt{r}_0$ to $\mathtt{r}_{15}$ of which two are for the stack pointer and program counter, leaving only 14 32-bit registers for general use.

- An iPad 2 (Apple A5) using an ARMv7 Cortex-A9 MPCore processor operating at 1.0 GHz clock frequency. It has 16 128-bit vector registers which are available as 32 64-bit vector registers, as these registers share physical space with the 128-bit vector registers.
- A Samsung Galaxy Nexus (1.2 GHz TI OMAP 4460 ARM Cortex-A9). The CPU microarchitecture is identical to the Apple A5. We included it to examine whether different implementations of the Cortex-A9 core have comparable performance in this application.

Our software is based on version 0.2.3 of the RELIC toolkit [2], with the GMP 5.0.2 backend, modified to include our optimizations. Except for the work described in Section 6.1, all of our software is platform-independent C code, and the same source package runs unmodified on all the above ARM platforms as well as both x86 and x86-64 Linux and Windows PCs. Our implementation also supports and includes BN curves at additional security levels beyond the three presented here. For each platform, we used the standard operating system and development environment that ships with the device, namely Debian Squeeze (native C compiler), XCode 4.3.0, and Android NDK (r7c) for the Kirkwood, iPad, and Galaxy Nexus respectively.

We present the results of our experiments in Table 3. For ease of comparison we have also included the numbers from [1] in Table 3. Roughly speaking, our timings are over three times faster than the results appearing in [1], which itself represents the fastest reported times prior to our work. Specifically, examining our iPad results, which were obtained on an identical micro-architecture and clock speed, we find that our implementation is 3.7, 3.7, and 5.4 times faster on BN-254, BN-446, and BN-638, respectively. Some, but not all, of the improvement can be attributed to faster field arithmetic; for example, $\mathbb{F}_q$-field multiplication on the RELIC toolkit is roughly 1.4 times as fast on the iPad 2 compared to [1]. A more detailed comparison based on operation counts is difficult because [1] does not provide any operation counts, and also because our strategy and our operation counts rely on lazy reduction, which does not play a role in [1].

6.1 Assembly Optimization

In order to investigate the potential performance gains available from hand optimized machine code, we implemented the two most commonly used field arithmetic operations (addition and multiplication) for the BN-254 curve in ARM assembly instructions. Due to the curve-specific and platform-specific nature of this endeavor, we performed this work only for the BN-254 curve and only on the Linux platforms (Marvell Kirkwood and Galaxy Nexus).

The main advantage of assembly language is that it provides more control for lower level arithmetic computations. Although the available C compilers are quite good, they still produce inefficient code since in the C language it is infeasible to express instruction priorities. Moreover, one can use hand-optimized assembly code to decompose larger computations into small pieces suitable for

Table 3. Timings for affine and projective pairings on different ARM processors and comparisons with prior literature. Times for the Miller loop (ML) in each row reflect those of the faster pairing.

Marvell Kirkwood (ARM v5) Feroceon 88FR131 at 1.2 GHz [This work]														
Field Size	Lang	Operation Timing [μs]												
		a	m	r	i	I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	O-A(a)	O-A(p)
254-bit	ASM	0.12	1.49	1.12	17.53	11.8	0.28	4.08	3.44	23.57	9,722	6,176	16,076	15,898
	C	0.18	1.74	1.02	17.40	10.0	0.35	4.96	4.01	24.01	11,877	7,550	19,427	19,509
446-bit		0.20	3.79	2.25	34.67	9.1	0.38	10.74	8.57	48.90	42,857	23,137	65,994	65,958
638-bit		0.27	6.82	3.83	52.33	7.7	0.51	18.23	14.93	77.11	98,044	51,351	149,395	153,713

iPad 2 (ARM v7) Apple A5 Cortex-A9 at 1.0 GHz [This work]														
Field Size	Lang	Operation Timing [μs]												
		a	m	r	i	I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	O-A(a)	O-A(p)
254-bit	C	0.16	1.28	0.93	13.44	10.5	0.25	3.48	2.88	19.19	8,338	5,483	14,604	13,821
446-bit		0.16	2.92	1.62	27.15	9.3	0.26	8.03	6.46	37.95	32,087	17,180	49,365	49,267
638-bit		0.20	5.58	2.92	43.62	7.8	0.34	15.07	12.09	64.68	79,056	40,572	119,628	123,410

Galaxy Nexus (ARM v7) TI OMAP 4460 Cortex-A9 at 1.2 GHz [This work]														
Field Size	Lang	Operation Timing [μs]												
		a	m	r	i	I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	O-A(a)	O-A(p)
254-bit	ASM	0.05	0.93	0.55	9.42	10.1	0.10	2.46	2.07	13.79	6,147	3,758	10,573	9,905
	C	0.07	0.98	0.53	9.62	9.8	0.13	2.81	2.11	14.05	6,859	4,382	11,839	11,241
446-bit		0.12	2.36	1.27	23.08	9.8	0.22	6.29	5.17	32.27	25,792	13,752	39,886	39,544
638-bit		0.19	4.87	3.05	38.45	7.9	0.45	12.20	10.39	56.78	65,698	33,658	99,356	99,466

NVidia Tegra 2 (ARM v7) Cortex-A9 at 1.0 GHz [1]														
Field Size	Lang	Operation Timing [μs]												
		a	m	r	i	I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML	FE	O-A(a)	O-A(p)
254-bit	C	0.67	1.72	n/a	18.35	10.7	1.42	8.18	5.20	26.61	26,320	24,690	51,010	55,190
446-bit		1.17	4.01	n/a	35.85	8.9	2.37	17.24	10.84	54.23	97,530	86,750	184,280	195,560
638-bit		1.71	8.22	n/a	56.09	6.8	3.48	31.81	20.55	91.92	236,480	413,370	649,850	768,060

vectorization. We employ the following techniques to optimize our implementation in assembly:

- Loop unrolling: since the maximum number of bits of the operands are known, it makes sense to unroll all loops in order to provide us the ability to avoid conditional branches (which basically eliminates branch prediction misses in the pipeline), reorder the instructions, and insert carry propagate codes at desired points.
- Instruction re-ordering: by careful reordering of non-dependent instructions (in terms of data and processing units), it is possible to minimize the number of pipeline stalls and therefore execute the code faster. Two of the most frequent multi-cycle instructions used in our code are word multiplication and memory reads. Each 32-bit word multiplication takes between 3 and 6 cycles and each memory read needs 2 cycles. By applying loop unrolling, it is possible to load the data required for the next multiplication while the pipeline is performing the current multiplication. Also, lots of register clean-ups and carry propagation codes are performed while the pipeline is doing a multiplication.
- Register allocation: all of the available registers were used extensively in order to eliminate the need to access memory for fetching the operands or store partial results. This improves overall performance considerably.

– Multiple stores: ARM processors are capable of loading and storing multiple words from or to the memory by one instruction. By storing the final result at once instead of writing a word back to memory each time when a new result is ready, we minimize the number of memory access instructions. Also, we do some register clean-ups (cost-free) when the pipeline is performing the multiple store instruction. It is worth mentioning that while it was possible to write 8 words at once, only 4 words are written to memory at each time because the available non-dependent instructions to re-order after the multiple store instruction are limited.

Table 3 includes our measurements of pairing computation times using our assembly implementation alongside the results for the C implementation. We find that the BN-254 pairing using hand-optimized assembly code is roughly 20% faster than the C implementation. In all cases, the projective pairing benefits more than the affine pairing, because we did not hand-optimize the inversion routine in assembly.

6.2 Affine vs. Homogeneous Coordinates

Acar et al. [1] assert that on ARM processors, small inversion to multiplication (I/M) ratios over $\mathbb{F}_q$ render it more efficient to compute a pairing using affine coordinates. If we are using a prime q congruent to $3 \bmod 8$, then compared to a projective doubling step, an affine doubling step costs an extra $\tilde{i}$ and an unreduced multiplication, and saves $5\tilde{s}_u + 3\tilde{r} + 16.5\tilde{a} + 2m$. Compared to a first doubling, it costs an extra $\tilde{i}$ and saves $2\tilde{s}_u + 2\tilde{r} + 6.5\tilde{a} + 2m$. An addition step costs an extra $\tilde{i}$ and saves $8\tilde{m}_u + \tilde{s}_u + 7\tilde{r} + 3\tilde{a} + 2m$; and a dense-sparse multiplication needs an additional $6m$ and saves $3\tilde{m}_u + 3\tilde{r} + 0.5\tilde{a}$. Thus, the difference between an affine and projective pairing is $70i - 659m_u - 396r - 4417a$.

From Table 3, we observe that the two pairings are roughly equal in performance at about the 446-bit field size. At this field size, we have $m \approx 1.5r$ and $m \approx 15a$. Plugging these estimates into the expression $70i - 659m_u - 396r - 4417a$, we find that an affine pairing is expected to be faster than a projective pairing whenever the I/M ratio in the base field falls below about 10.0. The results of Table 3 indicate that our I/M ratios cross this point slightly above 446 bits. We observe that both affine and projective pairings achieve similar performance in our implementation on ARM processors, with an advantage in the range of 1%-6%) for projective coordinates on BN-254, and a slight advantage for affinr coordinates on BN-638, with slightly better results for projective coordinates on BN-446. and a similar advantage for affine coordinates on BN-638, with mixed results on BN-446. These results overturn those of Acar et al. [1] which show too much advantage in favor of affine coordinates (well above 5%). However, there are situations in which affine coordinates would be preferable even at the 128-bit security level (e.g., products of pairings). Different conclusions may hold for assembly-optimized variants at higher security levels, which are not included in our analysis.

7 Conclusions

In this paper, we present high speed implementation results of the Optimal-Ate pairing on BN curves for different security levels. We extend the concept of lazy reduction to inversion in extension fields and optimize the sparse multiplication algorithm in the degree 12 extension. Our work indicates that D-type and M-type twists achieve equivalent performance for point/line evaluation computation, with only a very slight advantage in favor of D-type when computing sparse multiplications. In addition, we include an efficient method from [7] to perform final exponentiation and reduce its computation time. Finally, we measure the Optimal-Ate pairing over BN curves on different ARM-based platforms and compare the timing results to the leading ones available in the open literature. Our timing results are over three times faster than the previous fastest results appearing in [1]. Although the authors in [1] find affine coordinates to be faster on ARM in all cases, based on our measurements we conclude that homogeneous projective coordinates are unambiguously faster than affine coordinates for O-Ate pairings at the 128-bit security level when higher levels of optimization are used.

8 Acknowledgment

The authors would like to thank the reviewers of SAC 2012 for their comments. This work has been supported by NSERC CRD Grant CRDPJ 405857-10. The work by P. Longa was carried out when he was a postdoctoral researcher in the Department of Combinatorics and Optimization at the University of Waterloo.

References

1. Tolga Acar, Kristin Lauter, Michael Naehrig, and Daniel Shumow. Affine Pairings on ARM. *IACR Cryptology ePrint Archive*, 2011:243, 2011. Pairing 2012 (to appear).
2. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.
3. Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2011.
4. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.
5. Naomi Benger and Michael Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In M. Anwar Hasan and Tor Helleseth, editors, *WAIFI*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2010.

6. Jean-Luc Beuchat, Jorge Enrique González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In Joye et al. [11], pages 21–39.
7. Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. Faster hashing to $\mathbb{G}_2$. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 412–430. Springer, 2011.
8. C. C. F. Pereira Geovandro, Marcos A. Simplício Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.
9. IEEE Std. 1363-2000. "IEEE Standard Specifications for Public-Key Cryptography". January 2000.
10. Tadashi Iyama, Shinsaku Kiyomoto, Kazuhide Fukushima, Toshiaki Tanaka, and Tsuyoshi Takagi. Efficient implementation of pairing on BREW mobile phones. In Isao Echizen, Noboru Kunihiro, and Ryôichi Sasaki, editors, *IWSEC*, volume 6434 of *Lecture Notes in Computer Science*, pages 326–336. Springer, 2010.
11. Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors. *Pairing-Based Cryptography - Pairing 2010 - 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings*, volume 6487 of *Lecture Notes in Computer Science*. Springer, 2010.
12. Kristin Lauter, Peter L. Montgomery, and Michael Naehrig. An analysis of affine coordinates for pairing computation. In Joye et al. [11], pages 1–20.
13. Patrick Longa. *High-Speed Elliptic Curve and Pairing-Based Cryptography*. PhD thesis, University of Waterloo, April 2011.
14. Victor S. Miller. The Weil pairing, and its efficient calculation. *J. Cryptology*, 17(4):235–261, 2004.
15. Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2010.
16. M. Scott. A note on twists for pairing friendly curves. 2009. Personal webpage: `ftp://ftp.computing.dcu.ie/pub/resources/crypto/twists.pdf`.
17. Michael Scott. On the efficient implementation of pairing-based protocols. In Liqun Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 296–308. Springer, 2011.
18. Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.