Fast Software Implementations of Bilinear Pairings

Reza Azarderakhsh, Dieter Fishbein, Gurleen Grewal, Shi Hu, David Jao, Patrick Longa, and Rajeev Verma

Abstract—Advancement in pairing-based protocols has had a major impact on the applicability of cryptography to the solution of more complex real-world problems. However, the computation of pairings in software still needs to be optimized for different platforms including emerging embedded systems and high-performance PCs. Few works in the literature have considered implementations of pairings on the former applications despite their growing importance in a post-PC world. In this paper, we investigate the efficient computation of the Optimal-Ate pairing over special class of pairing friendly Barreto-Naehrig curves in software at different security levels. We target both applications and perform our implementations on ARM-powered processors (with and without NEON instructions) and PC processors. We exploit state-of-the-art techniques and propose new optimizations to speed up the computation in the different levels including 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 and D-type sextic twists in the pairing computation and carry out a detailed comparison between affine, Jacobian, and homogeneous coordinate systems. Our implementations on various mass-market emerging embedded devices significantly improve the state-of-the-art of pairing computation on ARM-powered devices and x86-64 PC platforms. For ARM implementations we achieved considerably faster computations in comparison to the counterparts.

Index Terms—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. Various works in the literature have considered the computation of fast pairings on PCs, for instance one can refer to [1], [2], [3], [4], [5] to name a few. Most recently, Aranha et al. [1] have applied a combination of improvements to accelerate the entire pairing and computed the O-Ate pairing at the 128-bit security level in under 2 million cycles on various 64-bit PC processors. Moreover, recent articles (see for instance [6], [7], and [8]) have considered efficient software implementations of pairings for 128-bit security level on ARM-based platforms such as hand-held smartphones and tablets. These platforms are widely predicted to become a dominant computing platform in the near future. As detailed in [9] and [10], the focus is now beginning to shift to optimizing pairings at higher security levels both for PCs and ARM-based platforms. Therefore, efficient implementation of pairing-based protocols for higher level security on these devices is crucial for deployment of pairing-based cryptography.

In [6], Acar et al. presented implementations of bilinear pairings on ARM-based platforms. They 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, most recent results presented in [11] demonstrate a clear advantage for homogeneous projective coordinates at the 128-bit security level, although affine coordinates remain faster at the 192-bit security level. Moreover, in [11], efficient and hand-optimized assembly implementations are presented for the computation of field arithmetic for the 128-bit security level. A follow-up work has been carried out in [8] for the computation of pairings using special instruction vectors known as NEON instructions for 128-bit security level. It is observed that without employing NEON instructions the implementation results in this paper and [11] outperform the previous work.

In this paper, we investigate efficient pairing computations at multiple security levels across different generations of ARM-based processors and 64-bit PC processors. We extend the work of [11] and [1] to different BN curves and higher security levels. In addition, we make several further optimizations and analyze different options available for

- R. Azarderakhsh and R. Verma are with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY. E-mail: {rxaeec, rv4560}@rit.edu.
- D. Fishbein, D. Jao, and G. Grewal are with the Center for Applied Cryptographic Research (CACR), Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada. E-mail: {dfishbei, ggrewal, djao}@math.uwaterloo.ca.
- S. Hu is with the Department of Computer Science, Cornell Univeristy, NY. E-mail: s3hu@stanford.edu.
- P. Longa is with the Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: plonga@microsoft.com.

Manuscript received 28 May 2015; revised 13 Sept. 2015; accepted 4 Dec. 2015. Date of publication 9 Dec. 2015; date of current version 10 Nov. 2017. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2015.2507120

implementation at various stages of the pairing computation. We summarize our contributions as follows:

- First, we extend the concept of lazy reduction employed by Aranha et al. [1] (see also Longa [12, 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 [3]. 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 [13] 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.
- We examine the efficiency of different coordinate systems including affine, Jacobian projective, and homogeneous projective coordinates. We realized that affine coordinates and homogeneous coordinates result in the best computation timings depending on the platforms.
- Finally, we implement the proposed algorithms for computing the O-Ate pairing over BN curves on different ARM-based platforms and x86-64 PCs and compare our measured timing results to their counterparts in the literature. We performed our implementations for different different field sizes and security levels. For ARM implementations we obtained 20 percent faster computations without employing NEON instructions in comparison to the previous works. Our NEON implementations are as competitive as the previous works, depending on the security level. For PC implementations we follow closely the state-of-the-art results for 128-, 164-, and 192-bit security levels.

2 Preliminaries

Barreto and Naehrig [14] 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 [15] on E is defined by:

$$a_{\text{opt}} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mu_n, (Q, P) \mapsto (f_{6x+2,Q}(P) \cdot h(P))^{\frac{q^{12}-1}{n}}, \tag{2}$$

where $\mu_n = \{u \in \bar{\mathbb{F}}_q : u^n = 1\}$ is a multiplicative group of order n formed by the nth roots of unity and $h(P) = l_{[6x+2]Q,qQ}(P) l_{[6x+2]Q+qQ,-q^2Q}(P)$ 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 [16]. We use a NAF representation of x when computing Miller's algorithm 1 [16].

Algorithm 1. Miller's Algorithm for the Tate Pairing [16]

```
Inputs: Points $Q, P \in E[n]$ and integer
$n = (n_{l-1}, n_{l-2}, \dots, n_1, n_0)_2 \in \mathbb{N}$ and $n_{l-1} = 1$
Output: $f_{n,Q}(P)$.
 1: $T \leftarrow Q,\ f \leftarrow 1$
 2: for $i = l - 2$ down to $0$ do
 3:     $f \leftarrow f^2 \cdot l_{T,T}(P)$
 4:     $T \leftarrow 2T$
 5:     if $n_i \neq 0$
 6:         $f \leftarrow f \cdot l_{T,Q}(P)$
 7:         $T \leftarrow T + Q$
 8:     end if
 9: end for
10: $f \leftarrow f^{\frac{q^{12}-1}{n}}$
11: return $f$
```

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 divisible by n [1]. 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 in $\mathbb{G}_2$, which considerably speeds up operations in the Miller loop.

2.1 Notations and Definitions

All other notations used in this paper with their definition are summarized in Table 1.

3 Representation of Extension Fields

Efficient implementation of the underlying extension fields is crucial to achieve fast pairing results. The IEEE P1363.3 standard [17] recommends using towers to represent $\mathbb{F}_{q^{12}}$. BN-curves are defined over prime fields, which means the computation of a pairing over a BN-curve relies on arithmetic over finite fields. Arithmetic over $\mathbb{F}_{q^2}$ is required for manipulating points on the twisted curve, and the computation of the Miller function. Moreover, squaring and multiplying values to compute $f_{n,Q}$ and the final exponentiation involves arithmetic over $\mathbb{F}_{q^{12}}$. In the following we discuss the arithmetic over extension fields.

3.1 Towering Scheme for q Congruent to 3 Modulo 8

For primes q congruent to $3 \pmod 8$, we employ the following construction of Benger and Scott [18] to construct tower fields.

TABLE 1
Notations and Their Definitions

Notation	Definition
m	Multiplication in $\mathbb{F}_q$
s	Squaring in $\mathbb{F}_q$
a	Addition in $\mathbb{F}_q$
i	Inversion in $\mathbb{F}_q$
r	Reduction in $\mathbb{F}_q$
$\tilde{m}$	Multiplication in $\mathbb{F}_{q^2}$
$\tilde{s}$	Squaring in $\mathbb{F}_{q^2}$
$\tilde{a}$	Addition in $\mathbb{F}_{q^2}$
$\tilde{i}$	Inversion in $\mathbb{F}_{q^2}$
$\tilde{r}$	Reduction in $\mathbb{F}_{q^2}$
m_u	Multiplication in $\mathbb{F}_q$ without reduction
s_u	Squaring in $\mathbb{F}_q$ without reduction
$\tilde{m}_u$	Multiplication in $\mathbb{F}_{q^2}$ without reduction
$\tilde{s}_u$	Squaring in $\mathbb{F}_{q^2}$ without reduction
m_b	Multiplication by b
m_i	Multiplication by i
m_ξ	Multiplication by ξ
m_v	Multiplication by v
$\times$	Multiplication without reduction
$\otimes$	Multiplication with reduction.

Proposition 1. *For approximately 2/3rds of the BN-primes $q \equiv 3 \pmod 8$, 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 $\mathbb{F}_{q^2}$.

3.2 Towering Scheme for q Congruent to 7 Modulo 8

We now illustrate explicit towering schemes for primes congruent to $7 \pmod 8$ using the 446-bit prime given by $q(x)$, where $x = 2^{110} + 2^{36} + 1$. This prime was used in [6] to implement the O-Ate pairing. In this case, the above scheme does not work because $1 + i$ is not a cubic non-residue in $\mathbb{F}_{p^2}$. Therefore, the following towering scheme is used

$$\begin{cases} \mathbb{F}_{q^2} = \mathbb{F}_q[i]/(i^2 - \beta), & \text{where } \beta = -1. \\ \mathbb{F}_{q^6} = \mathbb{F}_{q^2}[v]/(v^3 - \xi), & \text{where } \xi = 16 + i. \\ \mathbb{F}_{q^{12}} = \mathbb{F}_{q^6}[w]/(w^2 - v). \end{cases}$$

Here β is minimal, however ξ is slightly large. Minimizing β might be beneficial because a large chunk of the arithmetic during pairing computation is performed over $\mathbb{F}_{q^2}$. Multiplication by i requires a negation; multiplication by ξ requires five additions in $\mathbb{F}_{q^2}$. Then, the following construction which can be proven using the same ideas as those in Benger and Scott [18] can be employed. For approximately 2/3rds of the BN-primes $q \equiv 7 \pmod 8$, 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:

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

Here, multiplication by i requires one addition and one negation in $\mathbb{F}_q$, and multiplication by ξ requires two additions in $\mathbb{F}_{q^2}$. Note that here we have taken the opposite approach to the one suggested in [3]. Instead of choosing a curve that gives the right twist, and then letting these choices dictate the towering scheme, we first choose a towering scheme that optimizes extension field arithmetic and deal with the curve later. All things being equal, the towering scheme derived from Proposition 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 $q \equiv 7 \pmod 8$ in order to optimize other aspects such as the Hamming weight of x. In particular, the curves BN-446 and BN-638 [3] have $q \equiv 7 \pmod 8$. In such cases, Proposition 1 does not apply, so we use the towering scheme derived from Proposition 3.2. We also considered other approaches to construct tower extensions as suggested in [3], but found the above schemes consistently resulted in faster pairings compared to the other options.

3.3 Finite Field Operations and Lazy Reduction

Aranha et al. [1] 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 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 versus no lazy reduction for our choices of finite fields 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 [1] 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. [3], 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

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

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 [3], our sparse multiplication algorithm requires $13\tilde{m}$ and $44\tilde{a}$ when a D-type twist is involved. 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 [1] (lazy reduction versions).

Algorithm 3. Inversion Over $\mathbb{F}_{q^6}$ Employing Lazy Reduction Technique

```
Inputs: $a = a_0 + a_1 v + a_2 v^2;\ a_0, a_1, a_2 \in \mathbb{F}_{q^2}$
Output: $c = a^{-1} \in \mathbb{F}_{q^6}$
1: $T_0 \leftarrow a_0 \times a_0$
2: $t_0 \leftarrow \xi a_1$
3: $T_1 \leftarrow t_0 \times a_2$
4: $T_0 \leftarrow T_0 - T_1$
5: $t_1 \leftarrow T_0 \bmod q$
6: $T_0 \leftarrow a_2 \times a_2$
7: $T_0 \leftarrow \xi T_0$
8: $T_1 \leftarrow a_0 \times a_1$
9: $T_0 \leftarrow T_0 - T_1$
10: $t_2 \leftarrow T_0 \bmod q$
11: $T_0 \leftarrow a_1 \times a_1$
12: $T_1 \leftarrow a_0 \times a_2$
13: $T_0 \leftarrow T_0 - T_1$
14: $t_3 \leftarrow T_0 \bmod q$
15: $T_0 \leftarrow t_0 \times t_3$
16: $T_1 \leftarrow a_0 \times t_1$
17: $T_0 \leftarrow T_0 + T_1$
18: $t_0 \leftarrow \xi a_2$
19: $T_1 \leftarrow t_0 \times t_2$
20: $T_0 \leftarrow T_0 + T_1$
21: $t_0 \leftarrow T_0 \bmod q$
22: $t_0 \leftarrow t_0^{-1} \bmod q$
23: $c_0 \leftarrow t_1 \otimes t_0$
24: $c_1 \leftarrow t_2 \otimes t_0$
25: $c_2 \leftarrow t_3 \otimes t_0$
26: return $c = c_1 + c_2 v + c_3 v^2$
```

Algorithm 4. Inversion over $\mathbb{F}_{q^{12}}$ Employing Lazy Reduction Technique

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

3.4 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:

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

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:

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

The untwisting map in (4) is not as 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.

Algorithm 5. D-type Sparse-Dense Multiplication in $\mathbb{F}_{q^{12}}$

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

Algorithm 6. Fq6SparseMul, used in Algorithm 5

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

3.5 Final Exponentiation Scheme

On supersingular curves, the final exponentiation is relatively easy compared to the Miller step. However, on ordinary curves, the computation of the final exponentiation is crucial. We use the final exponentiation scheme proposed in [19], 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 [19] it is shown that

$$2x(6x^2+3x+1)d(x) = \lambda_3 q^3 + \lambda_2 q^2 + \lambda_1 q + \lambda_0, \tag{5}$$

where

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

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

$$f \mapsto f^x \mapsto f^{2x} \mapsto f^{4x} \mapsto f^{6x} \mapsto f^{6x^2} \mapsto f^{12x^2} \mapsto f^{12x^3}. \tag{7}$$

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 three multiplications. The final pairing value is obtained as

$$a f^{6x^2} f b^q a^{q^2} (bf^{-1})^{q^3}, \tag{8}$$

which costs six multiplications and six Frobenius operations. In total, this method requires three exponentiations by x, 3 squarings, 10 multiplications, and three Frobenius operations. In comparison, the technique used in [1] requires three additional multiplications and an additional squaring, and thus is slightly slower.

3.6 Exponentiation by x

The final exponentiation requires three exponentiations by x. This is traditionally done using a square-and-multiply method. Before we raise the output of the Miller loop to x, we exponentiate it to $(q^6-1)(q^2+1)$. This ensures that the value we need to exponentiate to x lies in the cyclotomic subgroup $\mathbb{G}_{\phi_6}(\mathbb{F}_{q^2})$.

Definition 2. *We denote by $\mathbb{G}_{\phi_6}(\mathbb{F}_{q^2})$ the cyclotomic subgroup (w.r.t. $\mathbb{F}_{q^{12}}/\mathbb{F}_{q^2}$) of $\mathbb{F}^*_{q^{12}}$. This is the subgroup of all elements $\alpha \in \mathbb{F}_{q^{12}}$ such that $\alpha^{q^4-q^2+1}=1$.*

Now, we have,

$$\begin{aligned} (q^6-1)(q^2+1) &= (q^6-1)\frac{(q^6+1)}{q^4-q^2+1} \\ &= \frac{q^{12}-1}{q^4-q^2+1}. \end{aligned}$$

Thus, an element raised to $(q^6-1)(q^2+1)$ lies in $\mathbb{G}_{\phi_6}(\mathbb{F}_{q^2})$. Fast formulas for computing squarings in $\mathbb{G}_{\phi_6}(\mathbb{F}_{q^2})$ are given in [1] which we use in our implementation. To compute a square, an element is first compressed, then squared in compressed form, and then decompressed. It is not known how to perform multiplication of compressed elements. Hence, when raising an element to the exponent x, one may keep squaring in compressed form, but when multiplication is required, one needs to decompress the elements. A compressed squaring requires $6\tilde{s}$, $28\tilde{a}$, and $3m_\xi$. A decompression requires $1\tilde{i}, 2\tilde{m}, 3\tilde{s}$, $9\tilde{a}$, and $2m_\xi$. Let h be the Hamming weight of x and l be the bit-length of x. Using Montgomery's simultaneous inversion trick [20], an exponentiation by x requires l compressed squarings, $h-1$ multiplications in $\mathbb{F}_{q^{12}}$, and $h(3\tilde{m}+3\tilde{s}+9\tilde{a}+2m_\xi)+3(h-1)\tilde{m}+\tilde{i}$.

3.7 The Frobenius Operator

Let $a = \alpha + i\beta \in \mathbb{F}_{q^2}$. Then,

$$\begin{aligned} a^q &= (\alpha+i\beta)^q \\ &= \alpha^q + i^q\beta^q \\ &= \alpha + i^3\beta, \qquad \text{since } q \equiv 3 \pmod 4 \\ &= \alpha - i\beta. \end{aligned}$$

Thus, computing a^q is almost free.

Now, suppose $A = \sum_{i=0}^5 a_i w^i \in \mathbb{F}_{q^{12}}$, with each $a_i \in \mathbb{F}_{q^2}$. By examining the polynomial $q(x)$, we note that $q \equiv 1 \pmod 6$ for any x. Then,

$$\begin{aligned} A^q &= \left(\sum_{i=0}^5 a_i w^i\right)^q \\ &= \sum_{i=0}^5 a_i^q w^{i\cdot q} \\ &= \sum_{i=0}^5 a_i^q w^{i\cdot(q-1)} \cdot w^i \\ &= \sum_{i=0}^5 (a_i^q \xi^{i\cdot\frac{q-1}{6}}) \cdot w^i, \text{since } q \equiv 1 \pmod 6. \end{aligned}$$

$\xi^0 = 1$, and we precompute $\xi^{i\frac{q-1}{6}}$ for $i = \{1,2,\ldots,5\}$. Hence, applying the Frobenius operator in $\mathbb{F}_{q^{12}}$ costs $5\tilde{m}$.

4 Curve Arithmetic

In this section, we discuss our optimizations to curve arithmetic over affine, Jacobian, and homogeneous projective coordinates.

4.1 Affine Coordinates

4.1.1 Doubling

Let the points $T=(x,y)$ and $Q=(x_2,y_2) \in E'(\mathbb{F}_{q^2})$ 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:

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

To compute the above, we precompute $\bar{x}_P = -x_P$ (to save the cost of computing the negation on-the-fly) and we propose use the following sequence of operations which requires $1\tilde{i},\ 3\tilde{m},\ 2\tilde{s},\ 6\tilde{a}$, and $2m$ if $T = Q$,

$$A = \frac{1}{y},\ B = \frac{x^2}{2} + x^2,\ C = AB,\ D = 2x,\ x_3 = C^2 - D;$$
$$E = Cx - y,\ y_3 = E - Cx_3,\ F = C\bar{x}_P,$$

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

which reduces the number of additions to 6. Lauter et al.'s formula [21] now costs four more additions in comparison to the one proposed here.

If we are working with an M-type twist then the tangent line evaluated at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is given by the following equation:

$$l_{2T}(\Psi^{-1}(P)) = y_P w^3 - m x_P w^2 - (y - mx).$$

This can be computed in a manner similar to above, requiring the same sequence of operations.

4.1.2 Addition

Let the points $T = (x, y)$ and $Q = (x_2, y_2) \in E'(\mathbb{F}_{q^2})$ be in affine coordinates. To compute $T + Q = (x_3, y_3)$ and the line passing through them, we use the following formulae:

$$m = \frac{y_2 - y}{x_2 - x}$$
$$x_3 = m^2 - x - x_2$$
$$y_3 = m(x - x_3) - y.$$

If we are working with a D-type twist, then the secant line evaluated at $P = (x_P, y_P)$ is given by the following equation:

$$l_{\Psi(T+Q)}(P) = y_P - m x_P w - (y - mx)w^3.$$

To compute the above, we use the following sequence of operations which requires $1\tilde{i},\ 3\tilde{m},\ 1\tilde{s},\ 7\tilde{a}$, and $2m$ as

$$A = \frac{1}{x_2 - x},\ B = y_2 - y,\ C = AB,\ D = x + x_2,$$
$$x_3 = C^2 - D,\ E = C(x - x_3),$$
$$y_3 = E - y,\ F = Cx_P,\ G = y - Cx$$

$$l_{\Psi(T+Q)}(P) = y_P - Fw - Gw^3.$$

If we are working with an M-type twist then the secant line evaluated at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is given by the following equation:

$$l_{T+Q}(\Psi^{-1}(P)) = y_P w^3 - m x_P w^2 - (y - mx).$$

This can be computed in a manner similar to above, requiring the same sequence of operations.

4.2 Homogeneous Coordinates

4.2.1 Doubling

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, Aranha et al. [1] presented optimized formulas for point doubling/line evaluation. 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:

$$X_3 = \frac{XY}{2}(Y^2 - 9(b/\xi)Z^2),$$
$$Y_3 = \left[\frac{1}{2}(Y^2 + 9(b/\xi)Z^2)\right]^2 - 27(b/\xi)^2 Z^4,$$
$$Z_3 = 2Y^3 Z.$$

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

$$l_{2\Psi(T)}(P) = -2YZy_P + 3X^2 x_P w + (3(b/\xi)Z^2 - Y^2)w^3.$$

We compute this value using the following sequence of operations:

$$A = \frac{XY}{2},\ B = Y^2,\ C = Z^2,\ E = 3(b/\xi)C,$$
$$F = 3E,\ X_3 = A \cdot (B - F), H = (Y + Z)^2 - (B + C),$$
$$G = \frac{B + f}{2},\ Z_3 = B \cdot H,\ Y_3 = G^2 - 3E^2,$$

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

Aranha et al. [1] 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. In the case of an M-type twist, the corresponding line function evaluated at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is:

$$l_{2T}(\Psi^{-1}(P)) = -2YZy_P w^3 + 3X^2 x_P w^2 + (3(b\xi)Z^2 - Y^2).$$

4.2.2 Addition

Let $T = (X, Y, Z)$ and $Q = (X_2, Y_2, 1) \in E'(\mathbb{F}_{q^2})$ be in homogeneous coordinates with $T \neq Q$. Then $T + Q = (X_3, Y_3, Z_3)$ is given by:

$$\theta = Y - Y_2 Z,\ \lambda = X - X_2 Z$$
$$X_3 = \lambda(\lambda^3 + Z\theta^2 - 2X\lambda^2)$$
$$Y_3 = \theta(3X\lambda^2 - \lambda^3 - Z\theta^2) - Y\lambda^3$$
$$Z_3 = Z\lambda^3.$$

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

$$l_{\Psi(T+Q)}(P) = \lambda y_P - \theta x_P w + (\theta X_2 - \lambda Y_2)w^3.$$

In [1] computing the above formula requires $11\tilde{m}, 2\tilde{s},\ 12\tilde{a}$, and $4m$. We use the following sequence of operations which uses four fewer $\mathbb{F}_{q^2}$ additions. Note that $\bar{x}_P$ and $\bar{y}_P$ are precomputed to save again the cost of computing $-x_P$ and $-y_P$.

TABLE 2
Doubling and Addition Costs Comparison for Different Coordinate Systems

Coordinates	Doubling Cost	Addition Cost
Affine	$\tilde{i} + 3\tilde{m} + 2\tilde{s} + 8\tilde{a} + 2m$	$\tilde{i} + 3\tilde{m} + 1\tilde{s} + 7\tilde{a} + 2m$
Projective (Homogeneous)	$2\tilde{m} + 7\tilde{s} + 22\tilde{a} + 4m$	$11\tilde{m} + 2\tilde{s} + 8\tilde{a} + 4m$
Projective (Jacobian)	$6\tilde{m} + 4\tilde{s} + 13\tilde{a} + 4m$	$10\tilde{m} + 3\tilde{s} + 6\tilde{a} + 4m$

$$\begin{gathered} A = Y_2Z,\ B = X_2Z,\ \theta = Y - A,\ \lambda = X - B, \\ C = \theta^2,\ D = \lambda^2,\ E = \lambda^3,\ F = ZC,\ G = XD, \\ H = E + F - 2G,\ X_3 = \lambda H,\ I = YE, \\ Y_3 = \theta(G - H) - I,\ Z_3 = ZE,\ J = \theta X_2 - \lambda Y_2, \\ l_{\Psi(T+Q)}(P) = \lambda y_P - \theta x_P w + J w^3. \end{gathered}$$

In the case of an M-type twist the corresponding line computation can be executed using the same sequence of operations as above. The equation for the secant line computed at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is:

$$l_{T+Q}(\Psi^{-1}(P)) = \lambda y_P w^3 - \theta x_P w^2 + (\theta X_2 - \lambda Y_2).$$

As in [1], we also use lazy reduction techniques to optimize the above formulae.

4.3 Jacobian Coordinates

4.3.1 Doubling

The point T on the twisted curve is traditionally stored and manipulated using Jacobian coordinates. The formula presented here is derived from [1], and is revised to minimize the number of $\mathbb{F}_{q^2}$ squarings. Let $T = (X, Y, Z) \in E'(\mathbb{F}_{q^2})$ be in Jacobian coordinates. Then $2T = (X_3, Y_3, Z_3)$ is given by:

$$\begin{aligned} X_3 &= X\left(\frac{9}{4}X^3 - 2Y^2\right) \\ Y_3 &= \frac{9}{4}X^3\left(2Y^2 - \frac{6}{4}X^3\right) - Y^4 \\ Z_3 &= YZ. \end{aligned}$$

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

$$l_{2\Psi(T)}(P) = 2Z_3Z^2y_P - 3X^2Z^2x_Pw + (3X^3 - 2Y^2)w^3.$$

As compared to [1], the line evaluation has been multiplied by 2 to save an addition in $\mathbb{F}_{q^2}$. This extra factor is eliminated by the final exponentiation. We use the following sequence of operations to compute the point doubling and line evaluation in $6\tilde{m}$, $4\tilde{s}$, $13\tilde{a}$, and $4m$:

$$\begin{gathered} A = 3X^2,\ E = 3X^3,\ B_0 = \frac{6X^3}{4}, B = \frac{9X^3}{4},\ C = Y^2, \\ D = 2Y^2, X_3 = X(B - D),\ F = C^2,\ Y_3 = B(D - B_0), \\ Z_3 = YZ,\ G = Z^2,\ H = 2Z_3G,\ I = -AGx_P,\ J = E - D \end{gathered}$$

$$l_{2\Psi(T)}(P) = Hy_P + Iw + Jw^3.$$

In the case of an M-type twist the corresponding line computation can be executed using the same sequence of operations as above. The equation for the tangent line computed at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is:

$$l_{2T}(\Psi^{-1}(P)) = 2Z_3Z^2y_Pw^3 - 3X^2Z^2x_Pw^2 + (3X^3 - 2Y^2).$$

4.3.2 Addition

Let $T = (X, Y, Z)$ and $Q = (X_2, Y_2, 1) \in E'(\mathbb{F}_{q^2})$ be in Jacobian coordinates with $T \neq Q$. Then $T + Q = (X_3, Y_3, Z_3)$ is given by:

$$\begin{aligned} \theta &= Y_2Z^3 - Y,\ \lambda = X_2Z^2 - X \\ X_3 &= \theta^2 - 2X\lambda^2 - \lambda^3 \\ Y_3 &= \theta(X\lambda^2 - X_3) - Y\lambda^3 \\ Z_3 &= Z\lambda. \end{aligned}$$

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

$$l_{\Psi(T+Q)}(P) = Z_3y_P - \theta x_P w + (\theta X_2 - Y_2Z_3)w^3.$$

In [1] computing the above formula requires $10\tilde{m}, 3\tilde{s},\ 8\tilde{a}$, and $4m$. We use the following sequence of operations which uses two fewer $\mathbb{F}_{q^2}$ additions,

$$\begin{gathered} A = Z^2,\ B = Z^3,\ \theta = Y_2B - Y,\ \lambda = X_2A - X,\ C = \theta^2 \\ D = \lambda^2,\ E = \lambda^3,\ F = C - E,\ G = XD,\ X_3 = F - 2G \\ Y_3 = \theta(G - X_3) - YE,\ Z_3 = Z\lambda,\ J = \theta X_2 - Y_2Z_3 \end{gathered}$$

$$l_{\Psi(T+Q)}(P) = Z_3y_P - \theta x_P w + Jw^3.$$

In the case of an M-type twist the corresponding line computation can be computed using the same sequence of operations as above. The equation for the secant line computed at $\Psi^{-1}(P) = (x_P w^2, y_P w^3)$ is:

$$l_{T+Q}(\Psi^{-1}(P)) = Z_3y_Pw^3 - \theta x_P w^2 + (\theta X_2 - Y_2Z_3).$$

4.4 Comparison of Coordinates

In Table 2, we list the operation counts for each of the above operations for affine, Jacobian, and homogeneous coordinates for easier comparison. For doubling, homogeneous coordinates are the fastest choice. However, for addition, Jacobian coordinates require less computations and hence are faster. It should be noted that the gain is not sufficient to warrant the use of Jacobian coordinates over homogeneous coordinates for pairing computation. Therefore, in this paper, we only consider implementing pairings using affine and projective (homogeneous) coordinates.

5 Operation Counts

5.1 Miller Loop Operations

We provide here detailed operation counts for our algorithms on the BN-254, BN-446, and BN-638 curves used in Acar et al. [6] and defined in [3]. Table 3 provides the operation counts for all component operations. Numbers for BN-446 and BN-638 are the same except where

TABLE 3
Operation Counts for 254-bit, 446-bit, and 638-bit Prime Fields

$E'(\mathbb{F}_{q^2})$ **Arithmetic**	254-bit	446-bit/638-bit
Doubl/Eval (Projective) ARM	$2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 25\tilde{a} + 4m$	$2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 34\tilde{a} + a + 4m$
Doubl/Eval (Projective) x86-64	$3\tilde{m}_u + 6\tilde{s}_u + 8\tilde{r} + 21\tilde{a} + 4m$	$3\tilde{m}_u + 6\tilde{s}_u + 8\tilde{r} + 30\tilde{a} + a + 4m$
Doubl/Eval (Affine)	$\tilde{i} + 3\tilde{m}_u + 2\tilde{s}_u + 5\tilde{r} + 7\tilde{a} + 2m$	$\tilde{i} + 3\tilde{m}_u + 2\tilde{s}_u + 5\tilde{r} + 7\tilde{a} + 2m$
Add./Eval (Projective)	$11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 10\tilde{a} + 4m$	$11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 10\tilde{a} + 4m$
Add/Eval (Affine)	$\tilde{i} + 3\tilde{m}_u + \tilde{s}_u + 4\tilde{r} + 6\tilde{a} + 2m$	$\tilde{i} + 2\tilde{m}_u + \tilde{s}_u + 3\tilde{r} + 6\tilde{a} + 2m$
First doubl./Eval	$3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 14\tilde{a} + 4m$	$3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 23\tilde{a} + a + 4m$
q-power Frobenius	$2\tilde{m} + 2a$	$8\tilde{m} + 2a$
q^2- power Frobenius	$4m$	$16\tilde{m} + 4a$
$\mathbb{F}_{q^2}$ **Arithmetic**	254-bit	446-bit/638-bit
Add/Subtr./Nega.	$\tilde{a} = 2a$	$\tilde{a} = 2a$
Multiplication	$\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r + 8a$	$\tilde{m} = \tilde{m}_u + \tilde{r} = 3m_u + 2r + 10a$
Squaring	$\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r + 3a$	$\tilde{s} = \tilde{s}_u + \tilde{r} = 2m_u + 2r + 5a$
Multiplication by β	$m_b = a$	$m_b = 2a$
Multiplication by ξ	$m_\xi = 2a$	$m_\xi = 3a$
Inversion	$\tilde{i} = i + 2m_u + 2s_u + 3r + 3a$	$\tilde{i} = i + 2m_u + 2s_u + 3r + 5a$
$\mathbb{F}_{q^{12}}$ **Arithmetic**	254-bit	446-bit/638-bit
Multiplication	$18\tilde{m}_u + 110\tilde{a} + 6\tilde{r}$	$18\tilde{m}_u + 117\tilde{a} + 6\tilde{r}$
Sparse Multiplication	$13\tilde{m}_u + 6\tilde{r} + 48\tilde{a}$	$13\tilde{m}_u + 6\tilde{r} + 54\tilde{a}$
Sparser Multiplication	$6\tilde{m}_u + 6\tilde{r} + 13\tilde{a}$	$6\tilde{m}_u + 6\tilde{r} + 14\tilde{a}$
Affine Sparse Multiplication	$10\tilde{m}_u + 6\tilde{r} + 47\tilde{a} + 6m_u + a$	$10\tilde{m}_u + 53\tilde{a} + 6\tilde{r} + 6m_u + a$
Squaring	$12\tilde{m}_u + 6\tilde{r} + 73\tilde{a}$	$12\tilde{m}_u + 6\tilde{r} + 78\tilde{a}$
Cyclotomic Squaring	$9\tilde{s}_u + 46\tilde{a} + 6\tilde{r}$	$9\tilde{s}_u + 49\tilde{a} + a + 6\tilde{r}$
Simult. Decompression	$9\tilde{m} + 6\tilde{s} + 22\tilde{a} + \tilde{i}$	$9\tilde{m} + 6\tilde{s} + 24\tilde{a} + \tilde{i}$ (BN-446) $16\tilde{m} + 9\tilde{s} + 35\tilde{a} + \tilde{i}$ (BN-638)
q-power Frobenius	$5\tilde{m} + 6a$	$5\tilde{m} + 6a$
q^2-power Frobenius	$10m + 2\tilde{a}$	$10m + 2\tilde{a}$
Exponentiation 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 Squaring	$6\tilde{s}_u + 31\tilde{a} + 4\tilde{r}$	$6\tilde{s}_u + 33\tilde{a} + a + 4\tilde{r}$

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, six point additions with line evaluations, one q-power Frobenius in $E'(\mathbb{F}_{q^2})$, one q^2-power Frobenius in $E'(\mathbb{F}_{q^2})$, 66 sparse multiplications, 63 squarings in $\mathbb{F}_{q^2}$, 1 negation in $E'(\mathbb{F}_{q^2})$, 2 sparser (i.e., sparse-sparse) multiplications [1], and 1 multiplication in $\mathbb{F}_{q^{12}}$. Using Table 3, we compute the total number of operations required in the Miller loop using homogeneous projective coordinates to be

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

For the curve BN-446, the Miller loop executes one negation in $\mathbb{F}_q$ to precompute $\overline{y_P}$, one first doubling with line evaluation, 111 point doublings with line evaluations, 6 point additions with line evaluations, 2 q-power Frobenius in $E'(\mathbb{F}_{q^2})$, 114 sparse multiplications, 111 squarings in $\mathbb{F}_{q^2}$, 1 negation in $E'(\mathbb{F}_{q^2})$, 2 sparser multiplications, and 1 multiplication in $\mathbb{F}_{q^{12}}$. Thus, the cost of the Miller loop on an ARM processor using projective coordinates is:

$$\begin{aligned}\text{ML446P} &= a + 3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 23\tilde{a} + a + 4m\\ &+ 111(2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 34\tilde{a} + a + 4m)\\ &+ 6(11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 11\tilde{a} + 4m)\\ &+ 2(8\tilde{m} + 2a) + 114(13\tilde{m}_u + 6\tilde{r} + 54\tilde{a})\\ &+ 111(12\tilde{m}_u + 6\tilde{r} + 78\tilde{a}) + \tilde{a}\\ &+ 2(6\tilde{m}_u + 6\tilde{r} + 14\tilde{a}) + 18\tilde{m}_u + 117\tilde{a} + 6\tilde{r}\\ &= 3151\tilde{m}_u + 793\tilde{s}_u + 2345\tilde{r} + 18601\tilde{a} + 472m + 117a.\end{aligned}$$

Finally, for the curve BN-638, we use the NAF Miller algorithm. Hence, the Miller loop executes one negation in $\mathbb{F}_q$ to precompute $\overline{y_P}$, one first doubling with line evaluation, 160 point doublings with line evaluations, 8 point additions with line evaluations, 2 q-power Frobenius in $E'(\mathbb{F}_{q^2})$, 167 sparse multiplications, 160 squarings in $\mathbb{F}_{q^2}$, 2 negations in $E'(\mathbb{F}_{q^2})$, 2 sparser multiplications, and 1 multiplication in $\mathbb{F}_{q^{12}}$. Thus, the cost of the Miller loop is:

TABLE 4
Cost of the Computation of O-Ate Pairings Using Various Coordinates on ARM

Curve bit-level	Coordinates ML	Cost
BN-254	Projective	$1841\tilde{m}_u + 457\tilde{s}_u + 1371\tilde{r} + 9516\tilde{a} + 284m + 3a$
	Affine	$70\tilde{i} + 1658\tilde{m}_u + 134\tilde{s}_u + 942\tilde{r} + 8398\tilde{a} + 540m + 132a$
BN-446	Projective	$3151\tilde{m}_u + 793\tilde{s}_u + 2345\tilde{r} + 18601\tilde{a} + 472m + 117a$
	Affine	$118\tilde{i} + 2872\tilde{m}_u + 230\tilde{s}_u + 1610\tilde{r} + 15790\tilde{a} + 920m + 231a$
BN-638	Projective	$4548\tilde{m}_u + 1140\tilde{s}_u + 3557\tilde{r} + 27206\tilde{a} + 676m + 166a$
	Affine	$169\tilde{i} + 4143\tilde{m}_u + 330\tilde{s}_u + 2324\tilde{r} + 22830\tilde{a} + 1340m + 333a$

$$\begin{aligned}\text{ML638P} &= a + 3\tilde{m}_u + 4\tilde{s}_u + 7\tilde{r} + 23\tilde{a} + a + 4m \\ &\quad + 160(2\tilde{m}_u + 7\tilde{s}_u + 8\tilde{r} + 34\tilde{a} + a + 4m) \\ &\quad + 8(11\tilde{m}_u + 2\tilde{s}_u + 11\tilde{r} + 11\tilde{a} + 4m) \\ &\quad + 2(8\tilde{m} + 2a) + 167(13\tilde{m}_u + 6\tilde{r} + 54\tilde{a}) \\ &\quad + 160(12\tilde{m}_u + 6\tilde{r} + 78\tilde{a}) + 2\tilde{a} \\ &\quad + 2(6\tilde{m}_u + 6\tilde{r} + 14\tilde{a}) + 18\tilde{m}_u + 117\tilde{a} + 6\tilde{r} \\ &= 4548\tilde{m}_u + 1140\tilde{s}_u + 3557\tilde{r} + 27206\tilde{a} \\ &\quad + 676m + 166a.\end{aligned}$$

The Miller loop operation costs for BN-254, BN-446 and BN-638 and for projective and affine coordinates are given in Tables 4 and 5, for ARM and x86-64 platforms, respectively.

5.2 Final Exponentiation Operations

We also compute the operation count for the final exponentiation. For BN-254, the final exponentiation requires 6 conjugations in $\mathbb{F}_{q^{12}}$, one negation in $E'(\mathbb{F}_{q^2})$, one inversion in $\mathbb{F}_{q^{12}}$, 12 multiplications in $\mathbb{F}_{q^{12}}$, two q-power Frobenius in $\mathbb{F}_{q^{12}}$, 3 q^2-power Frobenius in $\mathbb{F}_{q^{12}}$, 3 exponentiations by x, and 3 cyclotomic squarings. Based on these costs, we compute the total number of operations required in the final exponentiation as follows:

$$\begin{aligned}\text{FE254} &= 6(3\tilde{a}) + \tilde{a} + 25\tilde{m}_u + 9\tilde{s}_u + 24\tilde{r} + +112\tilde{a} + \tilde{i} \\ &\quad + 12(18\tilde{m}_u + 110\tilde{a} + 6\tilde{r}) + 2(5\tilde{m} + 6a) \\ &\quad + 3(10m + 2\tilde{a}) + 3(45\tilde{m}_u + 378\tilde{s}_u + 275\tilde{r} + 2164\tilde{a} + \tilde{i}) \\ &\quad + 3(9\tilde{s}_u + 49\tilde{a} + a + 6\tilde{r}) \\ &= 386\tilde{m}_u + 1164\tilde{s}_u + 949\tilde{r} + 4\tilde{i} + 7978\tilde{a} + 30m + 15a.\end{aligned}$$

In the case of BN-446, the final exponentiation requires 5 conjugations in $\mathbb{F}_{q^{12}}$, 1 inversion in $\mathbb{F}_{q^{12}}$, 12 multiplications in $\mathbb{F}_{q^{12}}$, 2 q-power Frobenius in $\mathbb{F}_{q^{12}}$, 3 q^2-power Frobenius in $\mathbb{F}_{q^{12}}$, 3 exponentiations by x, and 3 cyclotomic squarings. Hence, the total cost of the final exponentiation is:

$$\begin{aligned}\text{FE446} &= 5(3\tilde{a}) + 25\tilde{m}_u + 9\tilde{s}_u + 24\tilde{r} + 123\tilde{a} \\ &\quad + \tilde{i} + 12(18\tilde{m}_u + 117\tilde{a} + 6\tilde{r}) + 2(5\tilde{m} + 6a) \\ &\quad + 3(45\tilde{m}_u + 666\tilde{s}_u + 467\tilde{r} + 3888\tilde{a} + 110a + \tilde{i}) \\ &\quad + 3(10m + 2\tilde{a}) + 3(9\tilde{s}_u + 49\tilde{a} + a + 6\tilde{r}) \\ &= 386\tilde{m}_u + 2034\tilde{s}_u + 1525\tilde{r} + 4\tilde{i} + 13359\tilde{a} + 30m + 345a.\end{aligned}$$

In the case of BN-638, the high level operations are the same as the previous case. The Hamming weight of x is 4, hence simultaneous decompression requires $16\tilde{m} + 9\tilde{s} + 35\tilde{a} + \tilde{i}$. Hence, exponentiation by x requires $70\tilde{m} + 948\tilde{s} + 675\tilde{r} + 5606\tilde{a} + 158a + \tilde{i}$. As a result, the total cost of the final exponentiation is:

$$\begin{aligned}\text{FE638} &= 5(3\tilde{a}) + 25\tilde{m}_u + 9\tilde{s}_u + 24\tilde{r} + 123\tilde{a} + \tilde{i} \\ &\quad + 2(5\tilde{m} + 6a) + 3(10m + 2\tilde{a}) \\ &\quad + 3(70\tilde{m} + 948\tilde{s} + 675\tilde{r} + 5606\tilde{a} + 158a + \tilde{i}) \\ &\quad + 3(9\tilde{s}_u + 49\tilde{a} + a + 6\tilde{r}) \\ &= 436\tilde{m}_u + 2880\tilde{s}_u + 2149\tilde{r} + 4\tilde{i} + 18513\tilde{a} + 30m + 489a.\end{aligned}$$

In Table 4, we summarize the cost of computing final exponentiation operation for BN-254, BN-446, and BN-638. Note that the total operation count for the pairing computation is the cost of the Miller loop plus the final exponentiation.

5.3 Comparison

In Table 6, we compare the operation counts for the computation of ML, FE, and total pairings for BN-254 curves (on x86-64 platforms) to the previous work available in

TABLE 5
Cost of the Computation of O-Ate Pairings Using Various Coordinates on x86-64

Curve bit-level	Coordinates ML	Cost
BN-254	Projective	$1904\tilde{m}_u + 394\tilde{s}_u + 1371\tilde{r} + 9306\tilde{a} + 284m + 3a$
	Affine	$70\tilde{i} + 1658\tilde{m}_u + 134\tilde{s}_u + 942\tilde{r} + 8398\tilde{a} + 540m + 132a$
BN-446	Projective	$3262\tilde{m}_u + 682\tilde{s}_u + 2345\tilde{r} + 18268\tilde{a} + 472m + 117a$
	Affine	$118\tilde{i} + 2872\tilde{m}_u + 230\tilde{s}_u + 1610\tilde{r} + 15790\tilde{a} + 920m + 231a$
BN-638	Projective	$4708\tilde{m}_u + 980\tilde{s}_u + 3557\tilde{r} + 26556\tilde{a} + 676m + 166a$
	Affine	$169\tilde{i} + 4143\tilde{m}_u + 330\tilde{s}_u + 2324\tilde{r} + 22830\tilde{a} + 1340m + 333a$

TABLE 6
Comparison of Operations Count for Pairing Computation on x86-64 on the BN-254 Curve with the Leading Works Available in the Literature

Work		Total Cost of Pairing in $\mathbb{F}_{q^2}$	Total Cost of Pairing in $\mathbb{F}_q$
Aranha et. al [1]	ML	$1904\tilde{m}_u + 396\tilde{s}_u + 1368\tilde{r} + 10281\tilde{a}$	$6796m + 2736r + 20436a$
	FE	$430\tilde{m}_u + 1179\tilde{s}_u + 963\tilde{r} + 8456\tilde{a}$	$3753m + 1926r + 17025a$
	ML+FE	$2334\tilde{m}_u + 1575\tilde{s}_u + 2331\tilde{r} + 18737\tilde{a}$	$10549m + 4662r + 37461a$
Beuchat et. al [2]	ML	$1952(\tilde{m}_u + \tilde{r}) + 568(\tilde{s}_u + \tilde{r}) + 6912\tilde{a}$	–
	FE	$403(\tilde{m}_u + \tilde{r}) + 1719(\tilde{s}_u + \tilde{r}) + 7021\tilde{a}$	–
	ML+FE	$2355(\tilde{m}_u + \tilde{r}) + 2287(\tilde{s}_u + \tilde{r}) + 13933\tilde{a}$	–
Arahna et. al [10]	ML	$1337\tilde{m}_u + 1152\tilde{s}_u + 1388\tilde{r} + 10462\tilde{a}$	–
	FE	$384\tilde{m}_u + 1172\tilde{s}_u + 941\tilde{r} + 8085\tilde{a}$	–
	ML+FE	$1721\tilde{m}_u + 2324\tilde{s}_u + 2329\tilde{r} + 4\tilde{i} + 18547\tilde{a}$	–
This work (Projective)	ML	$1841\tilde{m}_u + 457\tilde{s}_u + 1371\tilde{r} + 9516\tilde{a}$	$6784m + 2742r + 18615a$
	FE	$386\tilde{m}_u + 1164\tilde{s}_u + 949\tilde{r} + 4\tilde{i} + 7978\tilde{a}$	$3516m + 1898r + 15971a$
	ML+FE	$2227\tilde{m}_u + 1621\tilde{s}_u + 2320\tilde{r} + 4\tilde{i} + 17494\tilde{a}$	$10300m + 4640r + 34586a$
This work (Affine)	ML+FE	$70\tilde{i} + 1658\tilde{m}_u + 134\tilde{s}_u + 942\tilde{r} + 8398\tilde{a}$	–

the literature. As one can see, the cost of computing pairings based on our algorithms outperforms the previously proposed work.

6 Implementations

In this section, we implement proposed algorithms on various platforms including ARM processors and x86-64 processors aiming at a security level of 128 bits and higher.

6.1 ARM-Based Platforms

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. (i) 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. (ii) 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. (iii) A Samsung Galaxy Nexus (1.2 GHz TI OMAP 4460 ARM Cortex-A9) which implements same CPU microarchitecture as Apple A5 Cortex-A9. We included it to examine whether different implementations of the Cortex-A9 core have comparable performance in this application. (iv) An Arndale Board using an ARMv7 Cortex-A15 processor operating at 1.7 GHz. The CPU microarchitecture is similar to the Apple A5 Cortex-A9. However, using a development board like an Arndale board allows for complete control of the processes running on the CPU. This allows for more precise benchmarking. We also employed NEON instructions in this setting to further speed up the computation of pairings. Our software is based on version 0.2.3 of the RELIC toolkit [22], with the GMP 5.0.2 backend, modified to include our optimizations. Except for the work described in Section 6.1.1, 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.

6.1.1 *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 and field multiplication for the BN-446 curve in ARM assembly instructions. Due to the curve-specific and platform-specific nature of this endeavor, we performed this work only on certain curves and platforms. Field addition and multiplication were implemented for the BN-254 curve on the Marvell Kirkwood and Galaxy Nexus. Field multiplication was implemented for both the BN-254 and BN-446 curves on the Arndale Board. 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 vectorization. We employ the following techniques to optimize our implementation in assembly: (a) Loop unrolling: since the maximum number of bits of the operands is known, it makes sense to unroll all loops in order to give 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. (b) 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. 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. (c) 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. (d) 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. While it was possible to write eight words at once, only four words are written to memory at each time because the available non-dependent instructions to re-order after the multiple store instruction are limited.

6.1.2 NEON Implementations

As mentioned above and based on our previous work in [11] it is observed that in most of ARM-powered smartphone processors, field operations over large finite fields suffer speed penalty owing to the smaller register size and relative lack of Single-Instruction-Multiple-Data (SIMD) instructions compared to a desktop processor. One feature of SIMD instructions is to allow multiple multiplication instructions to be executed at the same time with virtually the same cost as a single multiplication instruction. If applied in an intelligent way, these techniques could likely improve performance in both the key-exchange and isogeny computations. NEON instructions as a vector instruction set are included in a large fraction of new ARM-based tablets and smartphones. NEON is a combined 64- and 128-bit SIMD instruction set that provides standardized acceleration for cryptographic applications [24]. NEON allows a better exploitation of the inherent parallelism present in several lower level and higher level arithmetic operations and accelerate arithmetic computations. However, data loading and storing is, in general, costly since the NEON registers have to be fed by storing data into consecutive 32-bit ARM registers. Hence, in order to take a real advantage of NEON, load/store instructions should be avoided. In this paper, we include the implementation of pairings on ARMv7 Cortex-A15 processor operating at 1.7 GHz employing NEON instructions and compare the results to the one available in the literature.

6.1.3 Results and Comparison

We present the results of our experiments on ARM-based platforms in Table 7. For ease of comparison we also include the implementation results from [6], [11], and [8] in Table 7. We applied hand-optimized assembly optimizations on BN-254 and BN-446 and timing results are compared to the ones generated by C. We find that the BN-254 pairing using hand-optimized assembly code is roughly 20 percent faster than the C implementation. Also our assembly optimization gives a 7-8 percent speed improvement in the computation of both the affine and projective 446-bit Optimal Ate pairing. Roughly speaking, our timings are over three times faster than the results appearing in [6]. 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 [6]. A more detailed comparison based on operation counts is difficult because [6] 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 [6]. In comparison to the work presented in [8], our non-NEON implementations (ASM) are 20 percent faster on the same platform for BN-254 curves. The reason why our implementation results outperform this work is that, our field arithmetic computations on extension field $\mathbb{F}_{q^2}$ are always faster and more efficient. In particular, timing results for an inversion on $\mathbb{F}_{q^2}$ for this work is 66 percent faster than the one presented in [8]. Unfortunately, the results for higher security level are not reported in [8], so we cannot compare our results for BN-446 and BN-638 curves. For the NEON implementations, our results are as competitive as the ones provided in [8] over the same platform. It should be noted that the difference between NEON implementations in Cortex-A9 and Cortex-A15 is huge.

6.2 x86-64 Implementations

We also implement our proposed schemes on an AMD Opteron II 2.4 GHz x86-64 platforms to have a fair comparison to the leading works available in the literature as the counterparts. For x86-64 implementations reported in Table 8, hand-optimized assembly provides 2.3 times faster computations in comparison to C implementations. For BN-254, our assembly results are 9.4 times faster than the results presented in [6] and follows closely the state-of-the-art results presented in [1] and [10]. We also implemented our pairings on larger field sizes for BN-446 and BN-638 curves for 164- and 192-bit security levels, respectively.

6.3 Cross-over I/M Ratio and Affine versus Projective (Homogeneous) Coordinates

In [6], Acar et al. 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. Referring to addition and doubling costs discussed in this paper, we see that if inversions are cheap enough, then the addition and doubling steps using affine coordinates will be faster. Using affine coordinates also leads to a faster dense-sparse multiplication algorithm. In Table 7, we provide the cross-over I/M ratio over various platforms. In the following we objectively compare the two coordinates based on the various parameters.

TABLE 7
Timings for Affine and Projective Pairings on Different ARM Processors and Comparisons with Prior Literature

Marvell Kirkwood (ARM v5) Feroceon 88FR131 at 1.2 GHz [11]

Field Size	Language	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	C	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 [11]

Field Size	Language	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		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	C	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 [11]

Field Size	Language	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	C	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[6]

Field Size	Language	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		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	C	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

Galaxy Note (ARM v7) Exynos 4 Cortex-A9 at 1.4 GHz [8]

Field Size	Language	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	–	–	–	–	–	0.12	2.4	1.7	28.01	5,937	3,763	–	9,727
	NEON	–	–	–	–	–	0.11	1.63	1.42	43.1	4,112	2,710	–	6,769

Arndale Board (ARM v7) Exynos 5 Cortex-A15 at 1.7 GHz [8]

Field Size	Language	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	NEON	–	–	–	–	–	0.08	0.8	0.5	17.01	1,992	1,384	–	3,434
	ASM	–	–	–	–	–	0.048	1.45	0.94	16.7	3,455	2,206	–	5,693

Arndale Board (ARM v7) Cortex-A15 at 1.7 GHz **[11]**and**[This work]**

Field Size	Language	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	0.029	0.39	0.19	5.11	13.1	0.06	1.05	0.81	6.6	2,939	1,742	4,859	**4,639**
	C	0.030	0.45	0.20	5.21	11.6	0.06	1.23	0.92	6.84	3,235	1,936	5,371	5,295
	NEON	0.030	0.41	0.20	5.09	12.4	0.05	0.81	0.46	6.06	2,332	1,383	3,846	**3,582**
446-bit	ASM	0.047	0.93	0.54	11.29	12.13	0.095	2.67	2.15	14.89	6,426	6,003	18,038	17,688
	C	0.048	1.02	0.55	11.39	11.16	0.097	2.95	2.39	15.24	6,934	6,407	19,308	18,764
638-bit	C	0.069	2.10	1.04	18.63	8.87	0.130	5.71	4.63	26.01	12,161	15,893	47,788	46,966

Times for the Miller loop (ML) in each row reflect those of the faster pairing.

6.3.1 BN-254

If we are using a prime q congruent to 3 modulo 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 $3\tilde{m}_u + 3\tilde{r} + 0.5\tilde{a}$. Computing a pairing on BN254 requires 1 first doubling, 63 doublings, 6 additions, and 66 dense-sparse multiplications. Thus, the difference between an affine and projective pairing on BN-254 can be computed as follows:

$$\begin{aligned}254A - 254P &= (\tilde{i} - 2\tilde{s}_u - 2\tilde{r} - 6.5\tilde{a} - 2m) + \\ &= 63(\tilde{i} + \tilde{m}_u - 5\tilde{s}_u - 3\tilde{r} - 16.5\tilde{a} - 2m) \\ &\quad + 6(\tilde{i} - 8\tilde{m}_u - \tilde{s}_u - 7\tilde{r} - 3\tilde{a} - 2m) \\ &\quad + 66(6m - 3\tilde{m}_u - 3\tilde{r} - 0.5\tilde{a}) \\ &= 70\tilde{i} + 256m - 183\tilde{m}_u - 323\tilde{s}_u - 431\tilde{r} - 1097\tilde{a} \\ &= 70(i + 4m_u + 3r + 3a) + 256(m_u + r) \\ &\quad - 183(3m_u + 8a) - 323(2m_u + 3a) - 431(2r) \\ &\quad - 1097(2a) \\ &= 70i - 659m_u - 396r - 4417a.\end{aligned}$$

Based on our timing results reported in Table 7 for ARM, for BN-254, we observe that both affine and projective pairings achieve similar performance in our implementations on ARM processors. However, we note that when we employ hand-optimized assembly implementations, projective coordinate provide faster pairings computation for all platforms. As one can see in Table 7, for assembly implementations we have I/M always falls above 10.0. Therefore, we have an advantage in the range of 1-6 percent for projective coordinates on BN-254.

6.3.2 BN-446

When using a prime congruent to 7 modulo 8, an affine doubling costs an extra $\mathbb{F}_{q^2}$ inversion and unreduced multiplication, and saves $5\tilde{s}_u + 3\tilde{r} + 26\tilde{a} + 2m$ compared to a projective doubling. Compared to a first doubling, it costs an extra $\mathbb{F}_{q^2}$ inversion and saves $2\tilde{s}_u + 2\tilde{r} + 15\tilde{a} + 2m$. An addition costs an extra $\mathbb{F}_{q^2}$ inversion and saves $8\tilde{m}_u + \tilde{s}_u + 7\tilde{r} + 3\tilde{a} + 2m$; and a dense-sparse multiplication needs six additional base field multiplications and saves $3\tilde{m}_u + 3\tilde{r} + 0.5\tilde{a}$. Computing a pairing on BN446 requires 1 first doubling, 111 doublings, 6 additions, and 114 dense-sparse multiplications. Thus, the difference between an affine and projective pairing is as follows:

$$\begin{aligned}446A - 446P &= (\tilde{i} - 2\tilde{s}_u - 2\tilde{r} - 15\tilde{a} - 2m) \\ &\quad + 111(\tilde{i} + \tilde{m}_u - 5\tilde{s}_u - 3\tilde{r} - 26\tilde{a} - 2m) \\ &\quad + 6(\tilde{i} - 8\tilde{m}_u - \tilde{s}_u - 7\tilde{r} - 3\tilde{a} - 2m) \\ &\quad + 114(6m - 3\tilde{m}_u - 3\tilde{r} - 0.5\tilde{a}) \\ &= 118\tilde{i} + 448m - 279\tilde{m}_u - 551\tilde{s}_u - 719\tilde{r} - 2976\tilde{a} \\ &= 118(i + 4m_u + 3r + 5a) + 448(m_u + r) \\ &\quad - 279(3m_u + 10a) - 551(2m_u + 5a) \\ &\quad - 719(2r) - 2976(2a) \\ &= 118i - 1019m_u - 636r - 10907a.\end{aligned}$$

Based on our timing results reported in Table 7 for ARM, for BN-446, we observe that the two pairings are roughly equal in performance having projective coordinates being slightly faster than affine coordinate. At this field size, we have $m \approx 1.5r$ and $m \approx 15a$. It is worth mentioning that when we employ hand-optimized assembly in our implementations, affine coordinates provide slightly faster computations.

6.3.3 BN-638

We use a prime congruent to 7 modulo 8 for this field. Then, computing a pairing on BN-638 requires 1 first doubling, 160 doublings, 8 additions, and 167 dense-sparse multiplications. Thus, the difference between an affine and projective pairing is:

$$\begin{aligned}638A - 638P &= (\tilde{i} - 2\tilde{s}_u - 2\tilde{r} - 15\tilde{a} - 2m) \\ &\quad + 160(\tilde{i} + \tilde{m}_u - 5\tilde{s}_u - 3\tilde{r} - 26\tilde{a} - 2m) \\ &\quad + 8(\tilde{i} - 8\tilde{m}_u - \tilde{s}_u - 7\tilde{r} - 3\tilde{a} - 2m) \\ &\quad + 167(6m - 3\tilde{m}_u - 3\tilde{r} - 0.5\tilde{a}) \\ &= 169\tilde{i} + 664m - 405\tilde{m}_u - 810\tilde{s}_u \\ &\quad - 1039\tilde{r} - 4282.5\tilde{a} \\ &= 169(i + 4m_u + 3r + 5a) + 664(m_u + r) \\ &\quad - 405(3m_u + 10a) - 810(2m_u + 5a) \\ &\quad - 1039(2r) - 4282.5(2a) \\ &= 169i - 1531m_u - 934r - 15865a.\end{aligned}$$

Based on our timing results reported in Table 7 for ARM, for BN-638 we observe that I/M ratio is always less than 10 and affine coordinates always result in faster pairing computation in all implementations with C. Note here that different conclusions may hold for assembly-optimized variants at higher security levels.

For PC x86-64 implementations, we observe that projective coordinates always outperform affine coordinates as shown in Table 8. It should be noted that the most efficient method of curve arithmetic for one platform is not necessarily the most efficient method for all platforms. For example, ARM optimization differs from PC optimization because ARM has different performance characteristics. On the ARM platform, the I/M ratio and the ratio of the cost of field multiplications to field additions is generally lower than on the PC platform. Therefore, the choice of formulas or coordinate systems geared towards one platform may not be optimal for another.

7 Conclusions

This paper presents a very detailed study of implementing pairings on BN curves on the ARM family of processors and x86-64 PC platforms. 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 [19] to perform final exponentiation and reduce its computation time. We present a detailed comparison of affine, projective Jacobian, and projective homogeneous coordinates for the implementation of pairing on both platforms. Finally, we measure the Optimal-Ate pairing over BN curves on different 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 [6]. Although the authors in [6] 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

TABLE 8
Timings for Affine and Projective Pairings on Different x86-64 and Comparisons with the Previous Works Appearing as the Leading Ones in the Open Literature

x86-64 AMD Opteron II @ 2.4 GHz [This work]										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	ASM		50	370	288	10,002	0.892	0.612	–	1.50
	C	29.1	61	676	546	7,221	1.98	1.31	3.81	3.29
446-bit	C	27.1	73	1,384	1,149	14,683	6.34	3.65	11.48	9.99
638-bit		25.0	85	2,222	1913	22,722	14.05	8.11	24.63	22.16
Naehrig et al. [4] x86-64 Phenom II										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	ASM	–	–	737	606	127,067	2.50	–	–	4.98
Beuchat et al. [2] x86-64 Phenom II										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	ASM	–	–	440	353	–	1.33	1.02	–	2.35
Aranha et al. [1] [23] x86-64 Phenom II										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	ASM	–	–	368	288	–	0.898	0.664	–	1.56
638-bit		–	–	–	–	–	16.38	7.21	–	23.77
Aranha et al. [10] x86-64 Phenom II @ 3.0 GHz										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	ASM	–	–	–	–	–	–	–	1.9	1.42
Acar et al. [6] x86-64 Core2 E6600 @ 2.4 GHz										
Field	Language	# of Clock Cycles								
Size		I/M	$\tilde{a}$	$\tilde{m}$	$\tilde{s}$	$\tilde{i}$	ML [$\times 10^6$]	FE [$\times 10^6$]	O-A(a) [$\times 10^6$]	O-A(p) [$\times 10^6$]
254-bit	C	25.0	336	2,131	1,318	12,774	7.49	6.63	14.98	14.12
446-bit		22.33	493	3,821	2,445	22,957	23.99	20.15	44.15	45.51
638-bit		18.59	659	6,176	3,961	34,935	51.49	85.03	136.53	157.30

pairings at the 128-bit security level when higher levels of optimization are used. For PC platform implementations our results follow closely the state-of-the-art results presented in [1] and [10].

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments. This material is based upon work supported by the National Science Foundation under Award No. CNS-1464118 to Reza Azarderakhsh. The work of David Jao is supported by NSERC CRD Grant CRDPJ 405857-10.

REFERENCES

[1] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, “Faster explicit formulas for computing pairings over ordinary curves,” in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2011, pp. 48–68.

[2] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, “High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves,” in *Proc. 4th Int. Conf. Pairing-Based Cryptography*, 2010, pp. 21–39.

[3] C. C. F. G. Pereira, M. A. Simplício Jr, M. Naehrig, and P. S. L. M. Barreto, “A family of implementation-friendly BN elliptic curves,” *J. Syst. Softw.*, vol. 84, no. 8, pp. 1319–1326, 2011.

[4] M. Naehrig, R. Niederhagen, and P. Schwabe, “New software speed records for cryptographic pairings,” in *Proc. 1st Int. Conf. Progress Cryptol.: Cryptol. Inf. Security Latin Am.*, 2010, pp. 109–123.

[5] M. Scott, “On the efficient implementation of pairing-based protocols,” in *Proc. IMA Int. Conf.*, 2011, pp. 296–308.

[6] T. Acar, K. Lauter, M. Naehrig, and D. Shumow, “Affine pairings on ARM,” in *Proc. 5th Int. Conf. Pairing-Based Cryptography*, 2013, pp. 203–209.

[7] T. Iyama, S. Kiyomoto, K. Fukushima, T. Tanaka, and T. Takagi, “Efficient implementation of pairing on BREW mobile phones,” in *Proc. 5th Int. Workshop Security Adv. Inf. Comput. Security* 2010, pp. 326–336.

[8] A. H. Sánchez and F. Rodríguez-Henríquez, “NEON implementation of an attribute-based encryption scheme,” in *Proc. 11th Int. Conf. Appl. Cryptography Netw. Security*, 2013, pp. 322–338.

[9] C. Costello, K. Lauter, and M. Naehrig, “Attractive subfamilies of BLS curves for implementing high-security pairings,” in *Proc. 12th Int. conf. Cryptol. India*, 2011, pp. 320–342.

[10] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini, “The realm of the pairings,” in *Proc. 20th Int. Workshop Select. Areas Cryptography*, 2014, pp. 3–25.

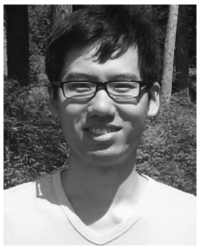
[11] G. Grewal, R. Azarderakhsh, P. Longa, S. Hu, and D. Jao, “Efficient implementation of bilinear pairings on ARM processors,” in *Proc. Select. Areas Cryptography*, 2012, pp. 149–165.

[12] P. Longa. (2011, Apr.). High-speed elliptic curve and pairing-based cryptography, Ph.D. dissertation, Univ. of Waterloo. [Online]. Available: http://hdl.handle.net/10012/5857
[13] M. Scott. (2009). A note on twists for pairing friendly curves [Online]. Available: http://indigo.ie/ mscott/twists.pdf
[14] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proc. 12th Int. Conf. Select. Areas Cryptography*, 2005, pp. 319–331.
[15] F. Vercauteren, "Optimal pairings," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 455–461, Jan. 2010.
[16] V. S. Miller, "The Weil pairing, and its efficient calculation," *J. Cryptol.*, vol. 17, no. 4, pp. 235–261, 2004.
[17] IEEE Std. 1363-3/D1, "Draft standard for identity-based public key cryptography using pairings," Jan. 2008.
[18] N. Benger and M. Scott, "Constructing tower extensions of finite fields for implementation of pairing-based cryptography," in *Proc. 3rd Int. Workshop Arithmetic Finite Fields*, 2010, pp. 180–195.
[19] L. Fuentes-Castañeda, E. Knapp, and F. Rodríguez-Henríquez, "Faster hashing to $\mathbb{G}_2$," in *Proc. Select. Areas Cryptography*, 2011, pp. 412–430.
[20] D. G. Harris. (2008). Simultaneous field divisions: An extension of montgomery's trick, davidgharris29@hotmail.com 14006 received 7 May 2008. [Online]. Available: http://eprint.iacr.org/2008/199
[21] K. Lauter, P. L. Montgomery, and M. Naehrig, "An analysis of affine coordinates for pairing computation," in *Proc. Pairing*, 2010, pp. 1–20.
[22] D. F. Aranha, and C. P. L. Gouvêa. RELIC is an efficient library for cryptography [Online]. Available: http://code.google.com/p/relic-toolkit
[23] D. F. Aranha, L. Fuentes-Castañeda, E. Knapp, A. Menezes, and F. Rodríguez-Henríquez, "Implementing pairings at the 192-bit security level," *IACR Cryptology ePrint Archive*, vol. 2012, p. 232, 2012.
[24] D. J. Bernstein and P. Schwabe, "NEON crypto," in *Proc. 14th Int. Workshop Cryptographic Hardw. Embedded Syst.*, 2012, pp. 320–339.
[25] M. Joye, A. Miyaji, and A. Otsuka, Eds., "Pairing-based cryptography—pairing 2010," in *Proc. 4th Int. Conf.*, Yamanaka Hot Spring, Japan, New York. NY, USA: Springer, vol. 6487, Dec. 2010.

Reza Azarderakhsh received the PhD degree in electrical and computer engineering from the University of Western Ontario in 2011. He received the NSERC Post-Doctoral Research Fellowship. He was with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. Currently, he is an assistant professor at the Department of Computer Engineering at the Rochester Institute of Technology. He is serving as an associate editor of the *IEEE Transactions on Circuits and Systems I*. His current research interests include finite field and its application, elliptic curve cryptography, pairing-based cryptography, and post-quantum cryptography.

Dieter Fishbein received the MSc degree in mathematics from the University of Waterloo, he is working as a security engineer in BMO. His research interests include elliptic curve cryptography, protocol design, and implementation, and post-quantum cryptography.

Gurleen Grewal received the MSc degree in mathematics from the University of Waterloo. She works as a security and cryptographic engineer at CSEC Canada. Her research interests include elliptic curve cryptography, protocol design, and implementation.



Shi Hu received the bachelor's and master's degrees in computer science from the University of Waterloo and Stanford University in 2012 and 2014, respectively. He was working toward the PhD degree in the Computer Science Department at Carnegie Mellon University from 2014 to 2015. He interned at major tech organizations such as Microsoft Research and Amazon before the PhD. His current research interests include machine learning and data science.

David Jao received the PhD degree in mathematics from Harvard University in 2003. From 2003 to 2006, he worked in the Cryptography and Anti-Piracy Group at Microsoft Research, contributing cryptographic software modules for several Microsoft products. He is currently an associate professor in the Mathematics Faculty at the University of Waterloo, and the director of the Centre for Applied Cryptographic Research. His research interests include elliptic curve cryptography, protocol design, and implementation, and post-quantum cryptography.

Patrick Longa received the PhD degree in electrical and computer engineering from the University of Waterloo in 2010. He is currently working in cryptographic group of Microsoft research as a senior engineer. His research interests include elliptic curve cryptography, protocol design, and implementation, and post-quantum cryptography.

Rajeev Verma is currently working toward the MSc degree in the Department of Computer Engineering at the Rochester Institute of Technology. His research interests include elliptic curve cryptography, pairing based cryptography, and efficient implementations.