# Software Implementation of Pairings[1]

Darrel HANKERSON [a], Alfred MENEZES [b] and Michael SCOTT [c]
[a] *Auburn University, hankedr@auburn.edu*
[b] *University of Waterloo, ajmeneze@uwaterloo.ca*
[c] *Dublin City University, mike@computing.dcu.ie*

**Abstract.** This chapter describes and compares the software implementation of popular elliptic curve pairings on two architectures, of which the Intel Pentium 4 and Core2 are representatives.

**Keywords.** Elliptic curve, Tate pairing, ate pairing, software implementation

## Introduction

Researchers have been studying methods for the efficient and secure implementation of conventional elliptic and hyperelliptic curve cryptographic schemes for over twenty years (see [24,15]). Even though the arithmetic in low-genus hyperelliptic curves is conceptually quite simple and well understood, discoveries are still being made that significantly improve performance; two recent examples are the use of Edwards coordinates [8] and Theta functions [20] to accelerate the addition rule for elliptic curves and genus 2 curves.

Since the arithmetic of pairings is considerably more complicated than conventional elliptic and hyperelliptic curve arithmetic, it is not surprising that there is substantial ongoing research on improving the performance of pairing-based protocols. There have been numerous proposals for defining and computing cryptographically-suitable pairings $G_1 \times G_2 \rightarrow G_T$. As a result, implementers of pairing-based protocols are faced with a bewildering selection of parameters choices. Among these choices are the embedding degree, the genus of the curve, the type of curve (supersingular or ordinary), the characteristic of the underlying field, and the prime-order groups $G_1$ and $G_2$. A particular selection of parameters can influence the functionality, efficiency, and security of the pairing application (see [19]). This chapter focuses on the software implementation of pairings at the 128-bit security level. We provide detailed analyses and comparisons of pairing algorithms on three specific elliptic curves: an embedding degree 4 supersingular curve defined over $\mathbb{F}_{2^{1223}}$, an embedding degree 6 supersingular curve defined over $\mathbb{F}_{3^{509}}$, and an embedding degree 12 ordinary curve defined over a 256-bit prime field; these elliptic curves were previously considered in [1] and [18]. While our focus is on the pairing operation, we emphasize that a pairing-based protocol may

[1] Draft of a chapter to appear in *Identity-Based Cryptography*, edited by Marc Joye and Gregory Neven. Date: May 2, 2008.

involve other computationally-intensive operations such as point multiplication and field exponentiation.

Our implementations were done primarily on two architectures, of which the Intel Pentium 4 and Core2 are representatives. Although these processors are superficially similar, there are significant feature and performance differences, and we will examine how well the features of a particular platform can be exploited to accelerate a pairing implementation.

## 1. Symmetric Pairings

Let $E$ be a supersingular elliptic curve defined over $\mathbb{F}_q$ with embedding degree $k > 1$. Let $r$ be a prime divisor of $\#E(\mathbb{F}_q)$, let $P \in E(\mathbb{F}_q)$ be a point of order $r$, and let $\mu_r$ denote the order-$r$ subgroup of $\mathbb{F}_{q^k}^*$. The symmetric pairing associated with $E$ is a bilinear map $e_r : \langle P \rangle \times \langle P \rangle \to \mu_r$ defined by $e_r(P_1, P_2) = e(P_1, \psi(P_2))$, where $e$ is a bilinear map and $\psi$ is a distortion map.

§1.1 and §1.2 describe two specific symmetric pairings derived from supersingular elliptic curves defined over the characteristic two field $\mathbb{F}_{2^{1223}}$ and the characteristic three field $\mathbb{F}_{3^{509}}$. These elliptic curves have embedding degrees 4 and 6, respectively. Both pairings attain the 128-bit security level because Pollard's rho method for computing discrete logarithms in the order-$r$ subgroup of $E(\mathbb{F}_q)$ has running time at least $2^{128}$, as do the index-calculus algorithms for computing discrete logarithms in the extension fields $\mathbb{F}_{q^k}$ [29].

### *1.1. Characteristic 2 Field ($k = 4$)*

*Field representation.* Let $q = 2^{1223}$. We chose the following polynomial basis representation for $\mathbb{F}_{2^{1223}}$:

$$\mathbb{F}_{2^{1223}} = \mathbb{F}_2[x]/(x^{1223} + x^{255} + 1).$$

That is, the elements of $\mathbb{F}_{2^{1223}}$ are the polynomials in $\mathbb{F}_2[x]$ of degree at most 1222, with multiplication performed modulo the irreducible trinomial $x^{1223} + x^{255} + 1$. Since $\mathbb{F}_{2^{1223}}$ has characteristic 2, squaring in $\mathbb{F}_{2^{1223}}$ is inexpensive relative to multiplication. Furthermore, since $\sqrt{c} = \sum c_{2i}x^i + \sqrt{x}\sum c_{2i+1}x^i$ for $c = \sum c_i x^i \in \mathbb{F}_{2^{1223}}$ and $\sqrt{x} = x^{612} + x^{128}$, square roots can also be computed inexpensively. The extension field $\mathbb{F}_{q^4}$ is represented using tower extensions $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$ and $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2 + v + u)$, whence a basis for $\mathbb{F}_{q^4}$ over $\mathbb{F}_q$ is $\{1, u, v, uv\}$.

*Elliptic curve.* The supersingular elliptic curve

$$E_1/\mathbb{F}_{2^{1223}} : Y^2 + Y = X^3 + X$$

has embedding degree $k = 4$. We have $\#E_1(\mathbb{F}_{2^{1223}}) = 5r$ where $r = (2^{1223} + 2^{612} + 1)/5$ is a 1221-bit prime. A distortion map is $\psi : (x, y) \mapsto (x + u^2, y + xu + v)$. Addition of two $E_1(\mathbb{F}_q)$ points using mixed affine-projective coordinates can be accomplished in 9 field multiplications. The doubling formula is $(x, y) \mapsto (x^4 + 1, x^4 + y^4 + 1)$, and hence the cost of doubling a point is relatively small.

*Pairing.* Barreto, Galbraith, Ó' hÉigeartaigh and Scott [3] presented the following algorithm for computing the $\eta_T$ pairing:

**Algorithm 1.** Computing the $\eta_T$ pairing for $E_1/\mathbb{F}_{2^{1223}}$

INPUT: $P = (x_1, y_1)$ and $Q = (x_2, y_2) \in E_1(\mathbb{F}_{2^{1223}})[r]$.
OUTPUT: $\eta_T(P, Q)$.

1. $T \leftarrow x_1 + 1$.
2. $f \leftarrow T \cdot (x_1 + x_2 + 1) + y_1 + y_2 + (T + x_2)u + v$.
3. For $i$ from 1 to 612 do
   3.1 $T \leftarrow x_1$, $x_1 \leftarrow \sqrt{x_1}$, $y_1 \leftarrow \sqrt{y_1}$.
   3.2 $g \leftarrow T \cdot (x_1 + x_2) + y_1 + y_2 + x_1 + 1 + (T + x_2)u + v$.
   3.3 $f \leftarrow f \cdot g$.
   3.4 $x_2 \leftarrow x_2^2$, $y_2 \leftarrow y_2^2$.
4. Return$(f^{(q^2-1)(q-\sqrt{2q}+1)})$.

*Analysis.* Step 3.2 costs 1 $\mathbb{F}_q$ multiplication. In step 3.3, write $f = f_1 + f_2u + f_3v + f_4uv$ and $g = g_1 + g_2u + v$, where $f_i, g_j \in \mathbb{F}_q$. Then

$$\begin{aligned} f \cdot g = {} & (f_1g_1 + f_2g_2 + f_4) + (f_1g_2 + f_2g_1 + f_2g_2 + f_3 + f_4)u \\ & + (f_1 + f_3 + f_3g_1 + f_4g_2)v + (f_2 + f_3g_2 + f_4 + f_4g_1 + f_4g_2)uv, \end{aligned}$$

which can be computed at a cost of 6 $\mathbb{F}_q$ multiplications. Now, $q$th-powering an element in $\mathbb{F}_{q^4}$ is essentially free because if $f = f_1 + f_2u + f_3v + f_4uv$ then

$$f^q = (f_1 + f_2 + f_3) + (f_2 + f_3 + f_4)u + (f_3 + f_4)v + f_4uv.$$

Note also that if $h = f^{q^2-1}$, then $h^{-1} = h^{q^2}$. It follows that the total cost of step 4 is 1 inversion in $\mathbb{F}_{q^4}$, 3 multiplications in $\mathbb{F}_{q^4}$, and 612 squarings in $\mathbb{F}_q$ for the powering by $\sqrt{2q}$. Since these costs are dominated by the cost of step 3, a reasonable approximation for the overall cost of Algorithm 1 is the cost of step 3, namely $612 \times 7 = 4284$ $\mathbb{F}_q$ multiplications.

### *1.2. Characteristic 3 Field ($k = 6$)*

*Field representation.* Let $q = 3^{509}$. We chose the following two polynomial basis representations for $\mathbb{F}_{3^{509}}$:

$$\mathbb{F}_3[x]/(x^{509} - x^{477} + x^{445} + x^{32} - 1) \text{ and } \mathbb{F}_3[x]/(x^{509} - x^{318} - x^{191} + x^{127} + 1).$$

Since $\mathbb{F}_{3^{509}}$ has characteristic 3, cubing in $\mathbb{F}_{3^{509}}$ is inexpensive relative to multiplication. Furthermore, the choice of the reduction polynomials enables cube roots to be computed significantly faster than an $\mathbb{F}_q$ multiplication (cf. §4.1.3). The extension field $\mathbb{F}_{q^6}$ is represented using tower extensions $\mathbb{F}_{q^3} = \mathbb{F}_q[u]/(u^3 - u - 1)$ and $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[v]/(v^2 + 1)$, whence a basis for $\mathbb{F}_{q^6}$ over $\mathbb{F}_q$ is $\{1, u, u^2, v, uv, u^2v\}$.

*Elliptic curve.* The supersingular elliptic curve

$$E_2/\mathbb{F}_{3^{509}} : Y^2 = X^3 - X + 1$$

has embedding degree $k = 6$. We have $\#E_2(\mathbb{F}_{3^{509}}) = 7r$ where $r = (3^{509} - 3^{255} + 1)/7$ is an 804-bit prime. A distortion map is $\psi : (x, y) \mapsto (u - x, yv)$. Addition of two $E_2(\mathbb{F}_q)$ points using mixed affine-projective coordinates can be accomplished in 9 field multiplications. The tripling formula is $(x, y) \mapsto (x^9 - 1, -y^9)$, and hence the cost of tripling a point is relatively small.

*Pairing.* Barreto, Galbraith, Ó' hÉigeartaigh and Scott [3] presented the following algorithm for computing the $\eta_T$ pairing:

**Algorithm 2.** Computing the $\eta_T$ pairing for $E_2/\mathbb{F}_{3^{509}}$

INPUT: $P = (x_1, y_1)$ and $Q = (x_2, y_2) \in E_2(\mathbb{F}_{3^{509}})[r]$.
OUTPUT: $\eta_T(P, Q)$.

1. $f \leftarrow (-x_1 - x_2 + 1) \cdot y_1 + y_1 u + y_2 v$.
2. For $i$ from 1 to 255 do
   2.1 $T \leftarrow x_1 + x_2 + 1$.
   2.2 $g \leftarrow -T^2 - Tu - u^2 + y_1 \cdot y_2 v$.
   2.3 $f \leftarrow f \cdot g$.
   2.4 $x_1 \leftarrow \sqrt[3]{x_1}$, $y_1 \leftarrow \sqrt[3]{y_1}$, $x_2 \leftarrow x_2^3$, $y_2 \leftarrow y_2^3$.
3. Return$(f^{(q^3-1)(q+1)(q-\sqrt{3q}+1)})$.

*Analysis.* The running time analysis of Algorithm 2 is similar to that of Algorithm 1. The dominant calculation is step 2, each iteration of which costs 14 $\mathbb{F}_q$ multiplications — 2 to compute $g$ and 12 to compute $f \cdot g$. The overall cost of Algorithm 2 is thus approximately $255 \times 14 = 3570$ $\mathbb{F}_q$ multiplications.

## 2. Asymmetric Pairings

Let $E$ be an ordinary elliptic curve defined over $\mathbb{F}_q$ having even embedding degree $k$ with respect to a prime divisor $r$ of $\#E(\mathbb{F}_q)$. Suppose further that $r^3 \nmid \#E(\mathbb{F}_{q^k})$ and $r^2 \nmid q^k - 1$. Let $P \in E(\mathbb{F}_q)$ be a point of order $r$, let $G_1 = \langle P \rangle$, and let $\mu_r$ denote the order-$r$ subgroup of $\mathbb{F}_{q^k}^*$. Suppose that $E$ admits a twist $E'$ of degree $d$ over $\mathbb{F}_{q^e}$, where $e = k/d$. Let $E'$ be such a twist for which $r \mid \#E'(\mathbb{F}_{q^e})$; the existence of $E'$ is guaranteed by Theorem 9 of [26]. Let $Q' \in E'(\mathbb{F}_{q^e})$ be a point of order $r$, and let $G_2' = \langle Q' \rangle$. Then there is an efficiently computable group monomorphism $\phi_d : G_2' \to E(\mathbb{F}_{q^k})$ such that $Q = \phi_d(Q') \notin E(\mathbb{F}_q)$. The group $G_2 = \langle Q \rangle$ is the Trace-0 subgroup of $E(\mathbb{F}_{q^k})[r]$. The asymmetric pairings considered in this section are the (reduced) Tate pairing $t_r : G_1 \times G_2 \to \mu_r$, the ate pairing $a_r : G_2 \times G_1 \to \mu_r$, and the R-ate pairing $R_r : G_2 \times G_1 \to \mu_r$.

We only consider these pairings for the Barreto-Naehrig (BN) [5] elliptic curves. These elliptic curves $E$ are defined over prime fields $\mathbb{F}_p$, have prime order $\#E(\mathbb{F}_p)$, and have embedding degree $k = 12$. They are especially well suited for the 128-bit security level because if $p$ is a 256-bit prime then Pollard's rho method for computing discrete logarithms in $E(\mathbb{F}_p)$ has running time approximately $2^{128}$,

as does the number field sieve algorithm for computing discrete logarithms in the extension fields $\mathbb{F}_{p^{12}}$. The BN curves also admit sextic twists ($d = 6$), which means that many computations can be restricted to the field $\mathbb{F}_{p^2}$ by working with the points in $G_2'$ rather than with points in $G_2$.

The BN curve we work with is

$$E_3/\mathbb{F}_p : Y^2 = X^3 + 3$$

with BN parameter $z = 6000000000001F2D$ (in hexadecimal) [18]. For this choice of BN parameter, $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ is a 256-bit prime of Hamming weight 87, $r = \#E_3(\mathbb{F}_p) = 36z^4 + 36z^3 + 18z^2 + 6z + 1$ is a 256-bit prime of Hamming weight 91, and $t - 1 = p - r = 6z^2 + 1$ is a 128-bit integer of Hamming weight 28; here $t = p + 1 - r$ is the trace of $E_3/\mathbb{F}_p$. Note that $p \equiv 7 \pmod 8$ (whence $-2$ is a nonsquare modulo $p$) and $p \equiv 1 \pmod 6$.

*Field representation.* The extension field $\mathbb{F}_{p^{12}}$ is represented using tower extensions $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 + 2)$, $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$ where $\xi = -u - 1$, and $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$. We also have the representation $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[W]/(W^6 - \xi)$ where $W = w$. Hence an element $\alpha \in \mathbb{F}_{p^{12}}$ can be represented in any of the following three ways:

$$\begin{aligned}
\alpha &= a_0 + a_1 w, \quad \text{where } a_0, a_1 \in \mathbb{F}_{p^6} \\
&= (a_{0,0} + a_{0,1} v + a_{0,2} v^2) + (a_{1,0} + a_{1,1} v + a_{1,2} v^2) w \quad \text{where } a_{i,j} \in \mathbb{F}_{p^2} \\
&= a_{0,0} + a_{1,0} W + a_{0,1} W^2 + a_{1,1} W^3 + a_{0,2} W^4 + a_{1,2} W^5.
\end{aligned}$$

We let $(m, s, i)$, $(\tilde{m}, \tilde{s}, \tilde{i})$, $(M, S, I)$ denote the cost of multiplication, squaring, inversion in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^{12}}$, respectively. Experimentally, we have $s \approx 0.9m$ and $i \approx 41m$.[2] In our cost estimates that follow, we will make the simplifying assumption $s \approx m$. If $a \in \mathbb{F}_p$ and $\alpha \in \mathbb{F}_{p^n}$ for $n \in \{2, 6, 12\}$, then the cost of computing $a \cdot \alpha$ is $nm$. For $\mathbb{F}_{p^2}$ arithmetic, we have $\tilde{m} \approx 3m$ (using Karatsuba's method which reduces a multiplication in a quadratic extension to 3 (rather than 4) small field multiplications), $\tilde{s} \approx 2m$ (using the complex method: $(a+bu)^2 = (a-b)(a+2b) - ab + (2ab)u$), and $\tilde{i} \approx i + 2m + 2s$ (since $(a + bu)^{-1} = (a - bu)/(a^2 + 2b^2)$). Note also that $p$-th powering is free in $\mathbb{F}_{p^2}$ since $(a + bu)^p = a - bu$.

Karatsuba's method reduces a multiplication in a cubic extension to 6 (rather than 9) multiplications in the smaller field.[3] Hence a multiplication in $\mathbb{F}_{p^6}$ costs $18m$. Squaring in $\mathbb{F}_{p^6}$ costs $2\tilde{m} + 3\tilde{s} = 12m$ via the following formulae [14]: if $\beta = b_0 + b_1 v + b_2 v^2 \in \mathbb{F}_{p^6}$ where $b_i \in \mathbb{F}_{p^2}$, then $\beta^2 = (A + D\xi) + (B + E\xi)v + (B + C + D - A - E)v^2$ where $A = b_0^2$, $B = 2b_0 b_1$, $C = (b_0 - b_1 + b_2)^2$, $D = 2b_1 b_2$, and $E = b_2^2$.[4] Finally, as shown in [36, Section 3.2], inversion in $\mathbb{F}_{p^6}$ can be reduced to 1 inversion, 9 multiplications, and 3 squarings in $\mathbb{F}_{p^2}$.

Since $\mathbb{F}_{p^{12}}$ is a tower of quadratic, cubic, and quadratic extensions, Karatsuba's method gives $M \approx 54m$. By using the complex method for squaring in $\mathbb{F}_{p^{12}}$

[2] We observed $i \approx 41m$ on a Pentium 4 and $i \approx 85m$ on a Core2.

[3] The Toom-Cook method requires 5 multiplications, but is slower in practice [17].

[4] Squaring in $\mathbb{F}_{p^6}$ can also be accomplished in $11m$ using the SQU3 formulae in [14], but at the expense of several additions, subtractions, and a division by 2.

and Karatsuba for multiplication in $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^2}$, we have $S \approx 36m$. Note, however, that if $\alpha = a + bw \in \mathbb{F}_{p^{12}}$ satisfies $\alpha^{p^6+1} = 1$ (and hence $a^2 - b^2v = 1$), then we have $\alpha^2 = (a+bw)^2 = (a^2+b^2v)+(2ab)w = (2b^2v+1)+[(a+b)^2-b^2-b^2v-1]w$. Hence squaring such $\alpha$, an operation denoted by $S'$, can be reduced to 2 squarings in $\mathbb{F}_{p^6}$ and so $S' \approx 24m$ [39]. Inverting such $\alpha$ is essentially free because $\alpha^{-1} = \alpha^{p^6}$. Since inversion in $\mathbb{F}_{p^{12}}$ can be reduced to 1 inversion, 2 multiplications, and 2 squarings in $\mathbb{F}_{p^6}$, it follows that $I \approx i + 97m$.

$E_3$ has a degree-6 twist over $\mathbb{F}_{p^2}$, namely $E'/\mathbb{F}_{p^2} : Y^2 = X^3 + 3/\xi$. The monomorpism $\phi_6 : G_2' \to G_2$ is given by $(x, y) \mapsto (xW^2, yW^3)$.

### *2.1. Tate Pairing*

Algorithm 3 computes the Tate pairing $t_r : G_1 \times G_2 \to \mu_r$ defined by

$$t_r(P, Q) = f_{r,P}(Q)^{(p^{12}-1)/r},$$

where $f_{r,P}$ is the Miller function [31]. Step 3 for computing $f_{r,P}(Q)$ (called a Miller operation) is from [4], while the technique for the final exponentiation (step 4) is from [18].

**Algorithm 3.** Computing the Tate pairing for $E_3/\mathbb{F}_p$

INPUT: $P \in G_1$ and $Q \in G_2$.
OUTPUT: $t_r(P, Q)$.

1. Write $r$ in binary: $r = \sum_{i=0}^{L-1} r_i 2^i$.
2. $T \leftarrow P$, $f \leftarrow 1$.
3. For $i$ from $L-2$ downto 0 do: {Miller operation}
   - 3.1 Let $\ell$ be the tangent line at $T$.
   - 3.2 $T \leftarrow 2T$.
   - 3.3 $f \leftarrow f^2 \cdot \ell(Q)$.
   - 3.4 If $r_i = 1$ and $i \neq 0$ then
     Let $\ell$ be the line through $T$ and $P$.
     $T \leftarrow T + P$.
     $f \leftarrow f \cdot \ell(Q)$.
4. Compute $f^{(p^{12}-1)/r}$ as follows: {Final exponentiation}
   - 4.1 $f \leftarrow f^{p^6-1}$.
   - 4.2 $f \leftarrow f^{p^2+1}$.
   - 4.3 $a \leftarrow f^{-(6z+5)}$, $b \leftarrow a^p$, $b \leftarrow a \cdot b$.
   - 4.4 Compute $f^p$, $f^{p^2}$, $f^{p^3}$.
   - 4.5 $f \leftarrow f^{p^3} \cdot [b \cdot (f^p)^2 \cdot f^{p^2}]^{6z^2+1} \cdot b \cdot (f^p \cdot f)^9 \cdot a \cdot f^4$.
5. Return($f$).

*Analysis.* A point $(X, Y, Z)$ in jacobian coordinates corresponds to the point $(x, y)$ in affine coordinates with $x = X/Z^2$ and $y = Y/Z^3$. In jacobian coordinates the formulae for doubling a point $T = (X, Y, Z)$ are $2T = (X_3, Y_3, Z_3)$ where $X_3 = 9X^4 - 8XY^2$, $Y_3 = (3X^2)(4XY^2 - X_3) - 8Y^4$ and $Z_3 = 2YZ$. The tangent line at $T$, after clearing denominators, is $\ell(x, y) = Z_3Z^2y - 2Y^2 - 3X^2(Z^2x - X) \in$

**Table 1.** Operation costs for the final exponentiation.

| Operation | Cost |
|---|---|
| $f^{p^6-1}$ | $I + M$ |
| $f^{p^2+1}$ | $M + 5\tilde{m}$ |
| $f^{-(6z+5)}$ | $10M + 65S'$ |
| $a^p$, $a \cdot b$, $f^p$, $f^{p^2}$, $f^{p^3}$ | $M + 20\tilde{m}$ |
| $T \leftarrow b \cdot (f^p)^2 \cdot f^{p^2}$ | $2M + S'$ |
| $T \leftarrow T^{6z^2+1}$ | $21M + 127S'$ |
| $f^{p^3} \cdot T \cdot b \cdot (f^p \cdot f)^9 \cdot f^4$ | $7M + 5S'$ |

$\mathbb{F}_p[x, y]$ [13]. The cost of steps 3.2[5] and 3.3 are $3m + 4s$ for computing $2T$, $7m + s$ for evaluating $\ell(Q)$ (note that the computation of $2T$ yields $X^2$, $Y^2$ and $Z_3$), $36m$ for computing $f^2$, and $39m$ for computing the product $f^2 \cdot \ell(Q)$ (note that $\ell(Q)$ has the form $a + bW^2 + cW^3$ with $a \in \mathbb{F}_p$ and $b, c \in \mathbb{F}_{p^2}$).

The formulae for mixed jacobian-affine addition are the following: if $P = (X_1, Y_1, Z_1)$ is in jacobian coordinates and $Q = (X_2, Y_2)$ is in affine coordinates, then $P + Q = (X_3, Y_3, Z_3)$ where $X_3 = (Y_2 Z_1^3 - Y_1)^2 - (X_2 Z_1^2 - X_1)^2 (X_1 + X_2 Z_1^2)$, $Y_3 = (Y_2 Z_1^3 - Y_1)[X_1 (X_2 Z_1^2 - X_1)^2 - X_3] - Y_1 (X_2 Z_1^2 - X_1)^3$, $Z_3 = (X_2 Z_1^2 - X_1) Z_1$. The line through $T$ and $P$ is $\ell(x, y) = (y - Y_2) Z_3 - (Y_2 Z_1^3 - Y_1)(x - X_2) \in \mathbb{F}_p[x, y]$ [13]. The cost of step 3.4 is $8m + 3s$ for computing $T + P$, $6m$ for evaluating $\ell(Q)$, and $39m$ for computing the product $f \cdot \ell(Q)$ (where again $\ell(Q)$ has the form $a + bW^2 + cW^3$ with $a \in \mathbb{F}_p$ and $b, c \in \mathbb{F}_{p^2}$).

Table 1 lists the operation costs for step 4. Observe that exponentiation by $p^6$ is free in $\mathbb{F}_{p^{12}}$. Also, since $W^p = (W^6)^{(p-1)/6} W = \xi^{(p-1)/6} W$, we have $(\sum_{i=0}^{5} a_i W^i)^p = \sum (a_i^p \cdot \gamma_i) W^i$ where $\gamma_i = \xi^{i(p-1)/6} \in \mathbb{F}_{p^2}$. Thus, if the $\gamma_i$ are precomputed, then powering an element in $\mathbb{F}_{p^{12}}$ by $p$ can be accomplished with 5 $\mathbb{F}_{p^2}$ multiplications. Similarly, powering by $p^2$ and $p^3$ each costs 5 $\mathbb{F}_{p^2}$ multiplications. The exponentiations $f^{6z+5}$, $T^z$ and $(T^z)^{6z}$ are performed by repeated square-and-multiply; to derive the costs note that $6z + 5$ and $6z$ have bitlength 66 and Hamming weight 11, while $z$ has bitlength 63 and Hamming weight 11.

Thus our estimated costs of the Miller operation and the final exponentiation are $255(85m + 5s) + 89(53m + 3s)$ and $I + 43M + 198S' + 25\tilde{m}$. Setting $s = m$ yields the estimated costs $27934m$, $7246m + i$, and $35180m + i$ for the Miller operation, the final exponentiation, and Algorithm 3, respectively.

### *2.2. Ate Pairing*

The ate pairing $a_r : G_2 \times G_1 \to \mu_r$, as proposed by Hess, Smart and Vercauteren [26], is defined to be

$$a_r(Q, P) = f_{t-1,Q}(P)^{(p^{12}-1)/r}.$$

[5]The first one or two iterations of step 3.2 are cheaper that subsequent iterations because $f$ may be sparse; for example $f = 1$ at the beginning of the first iteration of step 3.3. Our counts will ignore these small differences.

Algorithm 4 for computing the ate pairing modifies Algorithm 3 by interchanging the roles of $P$ and $Q$, and by using $t-1$ (instead of $r$) to determine the number of iterations in the Miller operation. Since $t \approx \sqrt{r}$ for the BN curve $E_3$, the number of iterations in the Miller operation is halved.

**Algorithm 4.** Computing the ate pairing for $E_3/\mathbb{F}_p$

INPUT: $P \in G_1$ and $Q \in G_2$.
OUTPUT: $a_r(Q, P)$.

1. Write $t-1$ in binary: $t-1 = \sum_{i=0}^{L-1} t_i 2^i$.
2. $T \leftarrow Q$, $f \leftarrow 1$.
3. For $i$ from $L-2$ downto 0 do: {Miller operation}
   3.1 Let $\ell$ be the tangent line at $T$.
   3.2 $T \leftarrow 2T$.
   3.3 $f \leftarrow f^2 \cdot \ell(P)$.
   3.4 If $t_i = 1$ then
   Let $\ell$ be the line through $T$ and $Q$.
   $T \leftarrow T + Q$.
   $f \leftarrow f \cdot \ell(P)$.
4. Return($f^{(p^{12}-1)/r}$), where $f^{(p^{12}-1)/r}$ is computed as in Algorithm 3.

*Analysis.* The analysis of Algorithm 4 is similar to that of Algorithm 3. The doubling and addition formulae are the same, however they are actually applied to points in $E'(\mathbb{F}_{p^2})$, thus ensuring that elliptic curve arithmetic is over $\mathbb{F}_{p^2}$ (instead of over $\mathbb{F}_{p^{12}}$); a jacobian point $(X, Y, Z) \in E'(\mathbb{F}_{p^2})$ conveniently maps to the jacobian point $(XW^2, YW^3, Z) \in E(\mathbb{F}_{p^{12}})$. In step 2, $T$ is initialized to the jacobian point $(xW^2, yW^3, 1)$, where $Q = (xW^2, yW^3) \in G_2$. The point doubling in step 3.2 costs $3\tilde{m} + 4\tilde{s}$. The tangent line at the affine point corresponding to $T = (XW^2, YW^3, Z)$ is $\ell(x, y) = Z_3Z^2y - 2Y^2W^3 - 3X^2W(Z^2x - XW^2) \in \mathbb{F}_{p^{12}}[x, y]$, where $2T = (X_3W^2, Y_3W^3, Z_3)$. Computing $\ell(P)$ and $f^2$ in step 3.3 costs $3\tilde{m} + \tilde{s} + 4m$ and $36m$, respectively. Noting that $\ell(P)$ is of the form $a + bW + cW^3$ with $a, b, c \in \mathbb{F}_{p^2}$, we can write $\ell(P) = a + (b + cv)w$ where $a$ and $(b + cv)$ are considered to be elements of $\mathbb{F}_{p^6}$. It follows that the product of $\ell(P)$ and $f^2 = f_0 + f_1w$ (where $f_0, f_1 \in \mathbb{F}_{p^6}$) can be computed using Karatsuba's technique at a cost of $13\tilde{m}$. The cost of step 3.4 is $8\tilde{m} + 3\tilde{s}$ for computing $T + Q$, $2\tilde{m} + 4m$ for evaluating $\ell(P)$, and $13\tilde{m}$ for computing $f \cdot \ell(P)$. Here, the line (after clearing denominators) through the affine point corresponding to $T = (X_1W^2, Y_1W^3, Z_1)$ and the point $Q = (X_2W^2, Y_2W^3)$ is $\ell(x, y) = (y - Y_2W^3)Z_3 - (Y_2Z_1^3 - Y_1)W(x - X_2W^2) \in \mathbb{F}_{p^{12}}[x, y]$ where $T + Q = (X_3W^2, Y_3W^3, Z_3)$; and $\ell(P)$ is of the form $a + bW + cW^3$ with $a, b, c \in \mathbb{F}_{p^2}$. Setting $s = m$ yields the estimated costs $15801m$, $7246m + i$, and $23047m + i$ for the Miller operation, the final exponentiation, and Algorithm 4, respectively.

### *2.3. R-ate Pairing*

The R-ate pairing $R_r : G_2 \times G_1 \rightarrow \mu_r$ is a generalization of the ate pairing due to Lee, Lee and Park [28]. For the BN curve $E_3$, the R-ate pairing is defined by

$$R_r(Q,P) = \left(f \cdot (f \cdot \ell_{aQ,Q}(P))^p \cdot \ell_{\pi(aQ+Q),aQ}(P)\right)^{(p^{12}-1)/r},$$

where $a = 6z + 2$, $f = f_{a,Q}(P)$, $\ell_{A,B}$ denotes the line through $A$ and $B$, and $\pi : (x, y) \mapsto (x^p, y^p)$ is the Frobenius map. Since $a \approx \sqrt{t}$, the Miller operation in Algorithm 5 has half as many iterations as in Algorithm 4.

**Algorithm 5.** Computing the R-ate pairing for $E_3/\mathbb{F}_p$

INPUT: $P \in G_1$ and $Q \in G_2$.
OUTPUT: $R_r(Q, P)$.

1. Write $a = 6z + 2$ in binary: $a = \sum_{i=0}^{L-1} a_i 2^i$.
2. $T \leftarrow Q$, $f \leftarrow 1$.
3. For $i$ from $L - 2$ downto 0 do
   3.1 Let $\ell$ be the tangent line at $T$.
   3.2 $T \leftarrow 2T$.
   3.3 $f \leftarrow f^2 \cdot \ell(P)$.
   3.4 If $a_i = 1$ then
   Let $\ell$ be the line through $T$ and $Q$.
   $T \leftarrow T + Q$.
   $f \leftarrow f \cdot \ell(P)$.
4. $f \leftarrow f \cdot (f \cdot \ell_{T,Q}(P))^p \cdot \ell_{\pi(T+Q),T}(P)$.
5. Return($f^{(p^{12}-1)/r}$), where $f^{(p^{12}-1)/r}$ is computed as in Algorithm 3.

*Analysis.* The analysis of step 3 is the same as for Algorithm 4. Hence, since $a$ has bitlength 66 and Hamming weight 9, the cost of step 3 is $7587m$. The cost of step 4 is: $10\tilde{m}+3\tilde{s}+4m$ to compute $T+Q$ and evaluate $\ell_{T,Q}(P)$; $2\tilde{m}$ to compute $\pi(T+Q)$; $\tilde{\imath} + 3\tilde{m} + \tilde{s}$ to convert $T$ to affine coordinates, and $10\tilde{m} + 3\tilde{s} + 4m$ to compute $\pi(T+Q)+T$ and evaluate $\ell_{\pi(T+Q),T}(P)$; $30\tilde{m}$ to multiply the two line evaluations into the accumulator; $5\tilde{m}$ for the $p$th-power; and $M$ for the multiplication by $f$. Setting $s = m$ yields an estimated cost of $7847m + i$ for steps 3 and 4, $7246m + i$ for the final exponentiation, and $15093m + 2i$ for Algorithm 5.

## 3. Platform and Algorithm Notes

In algorithm analysis, operation counts (for the more expensive of the basic operations) typically suffices for rough comparisons. However, a significant portion of experimental results is often not explained by this higher-level analysis, in part because such analysis fails to adequately capture platform characteristics such as cache size and speed, number of registers, and pipelining. In this section, we provide context and technical details for the platforms and algorithms used in the timings. In particular, we discuss features of platforms that have been widely used for experimental data and their influence on algorithm, field, and curve selection.

### *3.1. Platform Selection*

The selection of a specific processor can significantly affect experimental results and algorithm selection, even among processors that are of similar class or possibly even in the same family. In the past dozen years, processors from the Intel Pentium

family have been the favourite for benchmarks, in large part because of their dominance in the consumer market. Among these processors, we will restrict our attention to the 32-bit "P6 family" (e.g., Pentium II, III) and Pentium 4 models 0–2.[6]

This choice has meant 32-bit platforms with only eight general-purpose registers and relatively fast access to memory. Although these processors are instruction-set similar, there are significant differences that are easily seen in practice. The Pentium 4 promised to scale to very high clock speeds, but the design had a significant penalty in cycle counts for integer multiplication, add-with-carry, and branch misprediction compared with earlier P6 designs. The penalty for arithmetic in general-purpose registers was offset by extensions in the single-instruction multiple-data (SIMD) instruction set that permit implementation of large integer multiplication that is cycle-competitive with the Pentium III.

The existence of a de facto reference platform has aided comparisons, but the industry has moved decisively to 64-bit platforms. These have been the standard on workstation-class systems for years, but are now commonplace on commodity hardware. The Core2 (and Xeon) from Intel and the Athlon64 (and Opteron) from AMD are, roughly speaking, extensions of the Pentium family processors to 64-bit instruction sets. Interestingly, Intel abandoned the Pentium 4 architecture, in part due to the success of the competing AMD Athlon. As a specific consequence, integer multiplication in general-purpose registers no longer suffers the significant penalty of the Pentium 4. Our goal here, in part, is to discuss issues specific to these 64-bit systems and to contrast with existing results on 32-bit systems.

The work by Avanzi and Thériault [2] provides a concrete example where the choice of "similar" platform significantly influences conclusions. In this case, the comparison is for scalar multiplication on elliptic vs hyperelliptic curves. The processors are the Motorola PowerPC (G4) and the Intel Core2, superficially similar in the sense that both can be described as "workstation class." For 32-bit code, they observe that the "Core2 offers better multiplication performance...especially for larger fields." Times for point multiplication are given for the PowerPC, and genus 4, for example, is competitive with elliptic curves (although point halving methods for elliptic curves were not exploited). The difference in multiplication performance will mean that higher-genus is more attractive on the PowerPC than the Core2 in this scenario. A portion of this discrepancy can be explained by the RISC architecture on the PowerPC that more heavily (compared with the Core2) favours smaller fields where elements occupy a few registers.

### *3.2. Special Hardware*

The processors used in this report all possess "special purpose" hardware in the form of SIMD and floating-point registers. The SIMD registers are easily employed to extend operations in characteristic 2 or 3 fields to 64 or 128 bits. On the register-poor Pentium 4, this hardware supplies eight 64- or 128-bit registers for vector operations. For fields of characteristic 2 or 3, the basic idea is to use the hardware as wide registers. A factor 2 acceleration over conventional registers can

[6] In the transition to newer architectures, Intel confusingly introduced "Pentium 4" processors that were very different from the earlier models.

be expected, although the precise improvement depends on instruction timings and specific operations. For example, many of the instructions on 128-bit registers have latency and throughput that are worse than their 64-bit counterparts. Further, shifting through 128 bits requires two or three (depending on shift amount) instructions unless the amount is a multiple of 8 bits.

For the 32-bit processors, the floating-point approach has been used by Bernstein [6,7] to obtain very fast point multiplication for elliptic curves over prime fields. The technique is not as straightforward as it may appear, in part because conversion to canonical form is expensive. On the Pentium 4 (where integer multiplication is expensive), an alternative is available in the SSE2 extensions to the SIMD registers. Coding with SSE2 integer operations obtains most of the speed improvement of the floating-point registers, but does not require the commitment across code demanded by the floating-point approach.

For the 64-bit processors considered here, the advantage of the 128-bit registers (for field arithmetic) is less clear. These processors have twice the number of general-purpose registers as the Pentium 4, and instruction timings in SIMD vary between the Intel and AMD offerings. On the AMD, for example, several of the operations of interest have better instruction timings with 64-bit general-purpose registers than with 128-bit SIMD registers. Some experimental results are discussed in §4.

For integer multiplication with 64-bit code, the general-purpose registers can directly multiply 64-bit quantities, while the SIMD registers are limited to operands of 32 bits. Similarly, the floating-point registers are of the same size as on 32-bit systems, and so this approach is less attractive (especially on the AMD which has a 5-cycle multiply with 64-bit operands).[7] On the other hand, multiplication with general-purpose registers has restrictive register requirements, while SIMD can perform two 32-bit multiplications per instruction and, along with the floating-point approach, does not place restrictions on registers.

In addition to the special registers, implementers have considered other attached hardware for cryptographic use. For example, some display adapters possess considerable computing power, although implementers have had mixed success in adapting their instruction set and interface for cryptographic use [33,16]. Hardware targeted to cryptography has usually been an add-on, but the recent UltraSPARC T2 from Sun Microsystems may be a harbinger of widespread on-chip cryptographic hardware on common systems. Even a narrow hardware instruction set enhancement to include a characteristic 2 multiplier could have a dramatic effect: in tests on a SmartMIPS, a factor 5 speedup in multiplication was observed with the addition of a polynomial multiplier.

Finally, we note that the 64-bit hardware is commonly configured with multiple processors and/or cores. Algorithms that can be parallelized are perhaps of less interest here, since aggregate throughput (that is, number of pairings per unit time) is probably the measurement of interest on this class of hardware. For scenarios where an expensive pairing is to be calculated by a device with multiple

[7] All the processors have 64-bit double-precision floating point capability, along with 80-bit extended precision. In the present context, the measurement of interest is the size of the significand, which is effectively 53-bit and 64-bit, resp. Since there is no penalty, the wider operand size available with the 80-bit format is preferred for integer arithmetic.

but weak processing units, algorithms that can be parallelized in software would be desirable.

### *3.3. Sixty-Four Versus Thirty-Two*

While 64-bit systems have been standard on workstations for years, the analysis for processors such as the Core2 and Athlon64 relative to the Pentium 4 "reference standard" may not be as straightforward as it appears. For example, the popular 64-bit Sun UltraSPARC was introduced without a full 64-bit multiplier [41]. In contrast, the Core2 and Athlon64 have a relatively fast and full 64-bit integer multiplier. Further, code on all the Intel and AMD offerings considered in this paper can exploit 128-bit SIMD registers and 80-bit extended floating-point capabilities.

In the present comparison against the Pentium 4, the most interesting features of the 64-bit Intel and AMD processors are the increased number of registers and the 64-bit multiplier. All operations can potentially benefit from the 64-bit register size, but characteristic 2 or 3 arithmetic in the Pentium 4 was already exploiting wide operations in SIMD. Instruction timings must be considered for complete analysis, but we expect that the relatively fast 64-bit multiplier will mean that curves over prime fields benefit more in the move to these 64-bit architectures (in part, because the multiplication in characteristic 2 or 3 is still essentially a few bits at a time).

## 4. Implementation

In this section, we discuss specifics of the implementations and compare experimental results against estimates based on operation counts. In part, our goal is to obtain realistic estimates of the performance penalty in using supersingular curves rather than BN curves at the 128-bit security level on platforms with varying hardware features. To be certain, shortcomings in the implementations remain, but it is hoped that the results provide meaningful benchmarks for future work.

### *4.1. Implementation Details*

We begin with details for the field arithmetic implemented for the example pairings. This includes three fields at sizes dictated by the 128-bit security level for the corresponding pairings, namely the 256-bit prime field $\mathbb{F}_p$ (§2), the 1223-bit characteristic 2 field $\mathbb{F}_{2^{1223}}$ (§1.1), and the 807-bit characteristic 3 field $\mathbb{F}_{3^{509}}$ (§1.2). The focus is on the platforms described in §3, although much of the material applies more widely.

#### *4.1.1. Prime fields*

Field multiplication is Montgomery form, and a fully-unrolled "comba" multiplier calculates integer products column-wise. The code is largely in assembler, with the SSE2 registers used on the Pentium 4 (due to slow arithmetic in general-

**Table 2.** Instruction timings for Pentium 4 (32-bit, model 2), Core2 (64-bit), and Athlon64/Opteron (64-bit) [22]. Times are in cycles for latency (L) and throughput (T).

| | Pentium 4 | | Core2 | | Athlon64 | |
|---|---|---|---|---|---|---|
| | L | T | L | T | L | T |
| add-with-carry | 7-8 | 1/6 | 2 | 1 | 1 | 2.3 |
| multiply | 14 | 1/10 | 8 | 1/4 | 5 | 1/2 |
| SSE2 multiply | 6 | 1/2 | 3 | 1 | 3 | 1/2 |

purpose registers) and general-purpose registers used in the 64-bit case. The SSE2 registers can perform multiplication on vectors of operands up to size 32 bits, but does not possess the carry handling common in general-purpose instruction sets. MIRACL [35] uses multiplication on a pair of 32-bit operands, and then performs a "shuffle" to split the 64-bit result across the 128-bit register so that several products can be accumulated. Roughly speaking, the UltraSPARC is treated as a 32-bit processor with a 64-bit accumulator for multiplication, due to limitations on the integer multiplier.[8]

Inversion is via a Euclidean algorithm variant. For the BN case, an inversion has cost equivalent to approximately 85 multiplications on the Core2. Only one or two inversions are performed in the pairings, and so the performance of inversion is not a significant factor.

On processors such as the UltraSPARC and Pentium 4, design "shortcomings" encouraged the use of floating-point hardware for integer multiplication. The strategy isn't new, but the performance obtained by Bernstein [6,7] for point multiplication and other operations was dramatic. The implementation in floating point is decidedly more complicated, in part because the commitment to (redundant) floating point representation is substantial, and bounds conditions on intermediate results must be verified.

The case for floating-point arithmetic on the Core2 and Athon64 is different, due to the existence of a relatively fast 64-bit multiplier. A "quadratic complexity" estimate suggests a factor 4 acceleration in integer multiplication in general purpose registers, although this is admittedly less than convincing. A more complete analysis can be made from experimental data and instruction timings in Table 2. Roughly speaking, latency is the number of cycles that must pass before the results can be used, and throughput is the number of the instructions that can issue per cycle.[9]

Compared to the Pentium 4, the 64-bit systems have twice the operand size and also significantly better instruction timings for integer operations in general-purpose registers. In contrast, the operand size for floating point and SSE2 multiplication is the same across these systems. On the downside, the 64-bit multiplier retains the restrictive register requirements of the Pentium 4. The floating-point

[8] The UltraSPARC has 64-bit addition and corresponding condition codes, but add-with-carry uses the 32-bit condition code [41]. Hence, multi-precision addition involves more instructions than on systems with conventional 64-bit carry handling.

[9] The definition of throughput is the reciprocal of Intel's use in [27]. Under the current definitions, small latency and large throughput are desirable.

approach can operate (in a stack-based fashion) on any pair of inputs. On the Athlon, floating-point was especially attractive since a multiply and add could be issued in the same cycle. Nonetheless, the smaller operand size is a significant penalty relative to the 64-bit multiplier.

Experimentally, the Core2 can perform multi-precision integer multiplication of 256-bit inputs at a cost of approximately 8 cycles to calculate and accumulate each product of 64-bit inputs. On this system, floating point multiplies can be issued only every two cycles. The analysis is somewhat different on the AMD due to the timings and pipeline properties, but we expect that the 64-bit multiplier will be preferred over floating-point on both these processors.

Finally, we note that SSE2 multiplication is more interesting in the Core2 than the Athlon64 due to integer and SSE2 instruction timings. The SSE2 hardware can in fact perform two 32-bit multiplications per instruction. However, arranging the data for this vector operation is inelegant, and earlier experiments (on a Pentium 4) with Intel's demonstration code were no faster than a scalar approach on 224-bit integers [24, Section 5.4]. We also note that the SSE2 registers do not have the usual carry handling of conventional instruction sets, a consideration in accumulation (the 224-bit example split the input into eight 28-bit segments). Our examination is not conclusive, but we suspect SSE2 will not improve on our timings for integer multiplication on the 64-bit systems.

#### *4.1.2. Characteristic 2 fields*

Most of the material in this section applies more generally, but we will focus on the example field represented as $\mathbb{F}_{2^{1223}} = \mathbb{F}_2[x]/(f)$ where $f(x) = x^{1223} + x^{255} + 1$. As noted in §1.1, square roots are inexpensive in this representation since $\sqrt{c} = \sum c_{2i} x^i + \sqrt{x} \sum c_{2i+1} x^i$ for $c \in \mathbb{F}_{2^{1223}}$, where $\sqrt{x} = x^{612} + x^{128}$. Note that the product in this expression is obtained with a few shifts and additions, and does not require reduction since $\deg \sqrt{x} \leq 612$. The even and odd coefficients in $c$ are extracted simultaneously via lookup on 8 bits.

Several approaches were tested for multiplication. The comb method [30] has generally been among the fastest, in part because of reduced shifting and more efficient use of precomputation compared with a traditional Karatsuba-style approach. For efficiency, it is necessary to code for a specific size multiplication (where size is in words), although code expansion can be controlled by using Karatsuba down to a few fixed sizes. For the platforms considered, there are multiple register sizes and we selected comb sizes according to the following table:

| Register size | Karatsuba depth | Comb size | Notes |
|---|---|---|---|
| 32 | 2 | 10 | 32-bit only |
| 64 | 1 | 10 | via MMX on Pentium 4 |
| 128 | 1 | 5 | via SSE2 |

The MMX and SSE2 registers in the table are SIMD, but MMX was introduced earlier in the family while SSE first appeared on the Pentium 3 and the SSE2 extensions on the Pentium 4. Capabilities and instruction timings differ between MMX and SSE2; in particular, the SSE2 registers require multiple instructions to shift across 128 bits unless the amount is divisible by 8. Further, the instruction timings differ between platforms. The Intel and AMD 64-bit offerings

have the same instruction timings for logical operations in general-purpose registers, but the difference in the SSE2 timings partially explain our results: the AMD has fastest multiplication with general-purpose registers while Intel can efficiently exploit SSE2.

The multiplication in MIRACL is based on a Karatsuba approach down to word-sized operands where a traditional polynomial multiplier is used with data-dependent precomputation. If the comparison of interest is with general-purpose registers, then combing gives less than 20% improvement to the MIRACL approach. MIRACL optionally implements SIMD register use at the word level (rather than at the size of the SIMD register), giving approximately 20% acceleration on the Pentium 4. Compared to this approach, combing was significantly faster in our tests, with acceleration of 47% and 60% with MMX and SSE2, resp., over combing with general-purpose registers on the Pentium 4.

Coding was primarily in C with intrinsics for the SIMD instructions (see §4.2). Inversion was via a Euclidean algorithm variant, with general-purpose registers only and without aggressive optimizations but with an assembly fragment to aid in finding the degree of a polynomial. Comb width 4 (16 elements of data-dependent precomputation) was used in all the multiplication routines. To exploit the cheaper shifting by multiples of 8 bits in the 128-bit SSE2 registers, we used two passes through the multiplicand with shifts by 8 and a 4-bit shift between passes.

### *4.1.3. Characteristic 3 fields*

The specific example considered is represented as $\mathbb{F}_{3^{509}} = \mathbb{F}_3[x]/(f)$ where $f(x) = x^{509} + r(x)$ is an irreducible pentanomial. There are irreducible trinomials for this extension; however, none give $\sqrt[3]{x}$ with few terms. There is a tetranomial with a 17-term root, but the pentanomials were chosen so that $\deg r$ is at most $509-32$ or $509-64$, resp., and so that the combined number of terms in $x^{1/3}$ and $x^{2/3}$ is as small as possible (to speed the cube root calculation $\sqrt[3]{c} = \sum c_{3i}x^i + x^{1/3}\sum c_{3i+1}x^i + x^{2/3}\sum c_{3i+2}x^i$). Under these criteria, we looked for examples where $f$ and the roots had terms where the exponents differed by a multiple of the word size. We selected:

| | 32-bit | 64-bit |
|---|---|---|
| $f(x)$ | $x^{509} - x^{477} + x^{445} + x^{32} - 1$ | $x^{509} - x^{318} - x^{191} + x^{127} + 1$ |
| $x^{1/3}$ | $x^{361} - x^{329} + x^{297} - x^{202} - x^{170} + x^{43}$ | $x^{467} + x^{361} - x^{276} + x^{255} + x^{170} + x^{85}$ |
| $x^{2/3}$ | $x^{181} + x^{149} + x^{22}$ | $-x^{234} + x^{128} - x^{43}$ |

These choices have $\deg r$ relatively large, although this is not a concern for our environment.

As in [25], each coefficient $a_i \in \mathbb{F}_3$ is represented uniquely in $\{0, 1, -1\}$ using a pair $(a_i^0, a_i^1)$ of bits, where $a_i = a_i^0 - a_i^1$ and not both bits are 1. Elements $a$ are represented by vectors $a^j = (a_{m-1}^j, \ldots, a_0^j)$, $j \in \{0, 1\}$. Addition $c = a + b$ is

$$t \leftarrow (a^0 \vee b^1) \oplus (a^1 \vee b^0), \quad c^0 \leftarrow (a^1 \vee b^1) \oplus t, \quad c^1 \leftarrow (a^0 \vee b^0) \oplus t.$$

The seven operations involve only bitwise "or" ($\vee$) and "exclusive-or" ($\oplus$), and it is easy to order the instructions to cooperate with processor pipelining. Negation

is $-a = (a^1, a^0)$. Techniques from characteristic 2 fields extend directly in this representation; in particular, our multiplication is via comb:

| Register size | Karatsuba depth | Comb size | Notes |
|---|---|---|---|
| 32 | 1 | 8 | 32-bit only |
| 64 | 0 | 8 | via MMX on Pentium 4 |
| 128 | 0 | 4 | via SSE2 |

The "comb size" is in pairs of registers, and a depth of 0 means that combing was on field elements.

Coding considerations are similar to those of characteristic 2. However, addition was written in assembly in order to coerce better sequences from compilers. Comb multiplication in SSE2 uses the same strategy to exploit faster shifting by multiples of 8; however, our comb width of 3 (27 points of data-dependent precomputation) means that there are more passes and "fixups" than in the binary case.

Precomputation is less expensive than it appears, since half the elements are obtained by simple negation. We note that Takahashi, Hoshino, and Kobayashi [40] report substantial savings from sharing precomputations from $\mathbb{F}_q$ multiplications in $fg$ of step 2.3 in Algorithm 2. However, the amount of re-use is for approaches using more than the 12 multiplications described in §1.2, and the proportion of re-use decreases as the number of multiplications for $fg$ decreases. For the approach with 15 multiplications in $fg$, only 7 precomputations are done, saving a reported 25% in this product. Their multiplication is for $q = 3^{97}$, and their precomputation cost, as a proportion of an $\mathbb{F}_q$ product, is higher than our estimates for $q = 3^{509}$. A more direct proposal to accelerate Algorithm 2 appears in [11,9], where an "unrolling" technique adapted from [21] reduces the multiplication count in step 2 from 14 to 12.5 (and with 11 cubings where Algorithm 2 has 2 cubings and 2 roots).

The reduction polynomial is more favourable in the 32-bit case than in the 64-bit case, since there are several exponents that differ by a multiple of 32; see also [37] for related material. As an alternative, the pairing algorithm can be constructed so that root calculations are avoided [10], in which case a trinomial can be selected.

### *4.2. Development Environment*

Development was done on a variety of systems, including Sun Solaris (Blade 2000 with UltraSPARC III and X4200 with Opteron), Linux/x86, and Microsoft Windows with the Sun Studio (5.9), GNU C (gcc 3.4 and 4.1), Intel (6.0, 32-bit only), and Microsoft (6.0) tools. Compilers exhibit a frustrating amount of sensitivity to the precise way the code is written and can produce object code of quite different quality. The use of multiple compilers provided useful sanity tests and debugging help.

Most of the code was written in C with some assembly language for critical sections and to work around compiler shortcomings; an exception is the prime field arithmetic where substantial portions are in assembler. The code for small-characteristic fields involving SIMD registers was primarily with compiler intrin-

sics. Roughly speaking, programming with intrinsics is similar to assembly language, but register allocation is managed by the compiler and code optimizations can be performed. This is a double-edged sword, however, and compilers sometimes substitute awful code sequences rather than a direct translation. For example, we were surprised to see that gcc 4.1 and 4.2 produced dramatically different code for shifting sequences involving SSE2 intrinsics in 64-bit code, where version 4.2 chose an expensive strategy involving moves to conventional registers and double-register shifts.

We chose the GNU compilers (gcc) for the timings because these are widely available across systems and have been a common choice for benchmarks. These compilers generally produce fairly good code and have an excellent interface for insertion of assembly language fragments (adopted by Intel and recent Sun compilers). Compared to the Sun compiler, gcc can require more code tuning to coerce better sequences and register allocation. In particular, gcc does not optimize as well when the data is written in structures or arrays (even when the indices are known at compile-time), and we tuned critical sections for gcc by breaking aggregates into scalars. The Sun and Intel compilers are much less sensitive to this scalar-vs-aggregate issue.

### *4.3. Timings*

The estimates in §1 and §2 obtained by counting field multiplications ignore other operations and overheads. These omissions can be significant and also hard to estimate, and so experimental data is often an important part of the analysis. Field and pairing timings appear in Table 3. The times are given in units involving the clock speed; to obtain the elapsed time for the test machines, divide by 2.8 GHz for the Pentium 4 and Opteron, 2.4 GHz for the Core2, and 1.2 GHz for the UltraSPARC.

The entries involving use of SIMD facilities require some explanation. On the Pentium 4, the 128-bit SSE2 capabilities give significant improvement across the three fields. However, the improvement for characteristic 3 is less dramatic than for characteristic 2. As discussed in §4.1.3, a portion of this difference is explained by restrictions on shifts in these registers. The current implementation has some assembly language enhancements, but also has has excessive memory operations and more careful tuning of this code could offer some acceleration. The Core2 timings for small characteristic illustrate the faster SSE2 operations relative to the Opteron (where SSE2 was not effective). Although the entries for SIMD on the UltraSPARC are empty, the processor does in fact have such capabilities in the VIS instruction set. Unlike SSE2, these registers are 64-bit and the multiplication is 8-bit by 16-bit, limiting their usefulness in our context. The MMX entries for prime fields are also omitted since these registers do not possess the 32-bit multiplier introduced in the SSE2 extensions.

We consider the ratio of the estimated time from multiplication counts over the actual time (and so ratios much less than 1 indicate that significant time is not represented in the counts). For pairing in the characteristic 2 case (where field additions are very inexpensive compared to multiplication), the ratio is approximately 0.9 across systems when multiplication is via general-purpose registers,

**Table 3.** Timings (in clock cycles) for field operations and pairings on a Pentium 4 model 2, Opteron, and Core2. Registers for field multiplication are GP (general-purpose), MMX (64-bit SIMD), and SSE (128-bit SIMD, SSE2 extensions). $\tau$ denotes the field characteristic. Compilers are Sun 5.9 on UltraSPARC and GNU 4.1 on the others.

| | Multiplication | | | | | | $\eta_T$/Ate | | R-ate | |
|---|---|---|---|---|---|---|---|---|---|---|
| Field | GP | MMX | SSE | $a^\tau$ | $\sqrt[\tau]{a}$ | $a^{-1}$ | GP | SSE | GP | SSE |
| 32-bit, Pentium 4 | | | | | | | | | | |
| $\mathbb{F}_{p256}$ | 2.2 | | 1.4 | — | — | 56 | 81 | 58 | 54 | 38 |
| $\mathbb{F}_{2^{1223}}$ | 43.4 | 23.0 | 16.2 | 1.7 | 1.1 | 627 | 201 | 81 | — | — |
| $\mathbb{F}_{3^{509}}$ | 32.5 | 22.1 | 19.0 | 2.0 | 5.0 | 518 | 139 | 86 | — | — |
| 64-bit, Opteron | | | | | | | | | | |
| $\mathbb{F}_{p256}$ | 0.25 | | | — | — | 32 | 15 | | 10 | |
| $\mathbb{F}_{2^{1223}}$ | 10.4 | 13.4 | 14.6 | 0.7 | 0.6 | 200 | 48 | | — | — |
| $\mathbb{F}_{3^{509}}$ | 10.6 | 12.9 | 15.1 | 1.1 | 1.3 | 116 | 46 | | — | — |
| 64-bit, Core2 | | | | | | | | | | |
| $\mathbb{F}_{p256}$ | 0.31 | | | — | — | 25 | 15 | | 10 | |
| $\mathbb{F}_{2^{1223}}$ | 10.3 | 12.5 | 8.2 | 0.6 | 0.5 | 162 | 48 | 39 | — | — |
| $\mathbb{F}_{3^{509}}$ | 10.8 | 11.0 | 7.7 | 0.9 | 1.2 | 98 | 46 | 33 | — | — |
| UltraSPARC III | | | | | | | | | | |
| $\mathbb{F}_{p256}$ | 2.4 | | | — | — | 40 | 77 | | 52 | |
| $\mathbb{F}_{2^{1223}}$ | 11.8 | | | 0.8 | 0.7 | 217 | 57 | | — | — |
| $\mathbb{F}_{3^{509}}$ | 14.5 | | | 1.4 | 1.8 | 145 | 63 | | — | — |
| Units: | $10^3$ cycles | | | | | | $10^6$ cycles | | | |

and falls to 0.8 when multiplication is via SIMD (in the cases where SIMD is advantageous). Part of the explanation for this difference is that SIMD was applied only to field multiplication, and other operation costs remain unchanged. For the characteristic 3 case, the ratio falls to 0.76–0.8 depending on registers employed.

The more troublesome case is for BN, where field additions are now a more significant part of the overall cost of the pairing. For the Tate pairing, if the fastest multiplication is considered on the Pentium 4, then the ratio of interest drops to 0.62. On the Core2, it is even lower at 0.52. However, good estimates from operation counts can be obtained if field additions are included—the corresponding ratios rise to 0.9 and 0.83, resp. As illustration, the code from [18] on a 1.66 GHz Core2 performs a Tate pairing in 14.5 ms, using 39824 field multiplications and 132893 additions. In isolation, the multiplications run at 7.6 ms and the additions at 4.5 ms.

The relatively strong showing for small characteristic on the UltraSPARC is due primarily to a weak integer multiplier that is restricted to 64-bit output. The other data is cycle-competitive with the newer Opteron and Core2 designs, although it should be noted that the UltraSPARC III was typically at half the clock speed of common Opteron and Core2 systems.

At the 128-bit security level, the R-ate pairing for BN curves is also substantially faster than other pairings on elliptic curves defined over large prime fields,

including the MNT curves [32]. A recent IETF standard [12] for identity-based encryption mandates the use of supersingular elliptic curves over prime fields — these curves have embedding degree $k = 2$. Our implementation of the Tate pairing on such an elliptic curve $E$ defined over a 1536-bit prime field $\mathbb{F}_p$ (and where $\#E(\mathbb{F}_p)$ has a 256-bit prime divisor) took 111 ms on a 32-bit 3GHz Pentium 4, versus versus 14.2 ms for the R-ate pairing on the BN curve of §2.

## 5. Conclusions

In this report, we have attempted to quantify pairing performance on popular systems at the 128-bit security level. The selected platforms have relatively fast integer multiplication and no special support for small characteristic fields. As expected, the BN curves hold a substantial edge, thanks to fast multiplication, "ideal" match of embedding degree and security level, and accelerations in the R-ate algorithm.

However, the difference is smaller than earlier reports may have suggested. In some cases, this has been due in part to overly pessimistic timings for characteristic 2 and especially characteristic 3 fields. Estimates based on multiplication counts can also be misleading. Operation counts often provide meaningful relative comparisons, even if fairly rough when used to estimate actual performance. However, the common comparison involving multiplications gets coarser when other operations have very different cost relative to field multiplication (as in the case of multiplication to addition for prime fields versus characteristic 2 fields). This kind of estimate inflates the advantage of the BN case over characteristic 2 and 3.

We remark briefly on extrapolating from the experimental data. A common practice is to implement in the general-purpose register set, under the assumption that results may be more widely applicable across "similar" systems. This of course gets the absolute performance wrong, but may adequately capture relative differences. As an example, for characteristic 3 on the Pentium 4, the restriction to general-purpose registers gives roughly the same "BN vs small characteristic" conclusions as allowing SIMD. On the other hand, this characterization is wrong for the preceding generation Pentium III, where there is a relatively fast integer multiplier (but not the SSE2 32-bit multiplication of the Pentium 4) and only small characteristic benefits from SIMD. In the Core2 vs Opteron comparison, we see that the Opteron has faster integer multiplication, and the two are similar when using general-purpose registers for small characteristic. However, the Core2 has faster SIMD timings that can be exploited for small characteristic. Necessarily, the small characteristic case looks somewhat less bleak against BN on the Core2 if SIMD is permitted.

On other devices, small characteristic can of course look more attractive. For example, favourable reviews for characteristic 2 implementations on smartcards and wireless sensor networks can be found in [38] and [34]. As noted earlier, the addition of a hardware characteristic 2 polynomial multiplier could dramatically change the performance. Intel has announced a characteristic 2 multiplier for its next generation processors, with multiplication on 64-bit operands [23]. Imple-

menting small characteristic across systems is arguably easier, although admittedly the amount of "difficult" coding can typically be limited to a fairly small portion of the arithmetic and still exploit most of the available performance with a fairly generic approach. Small characteristic also seems to lead to smaller code sizes.

## References


[1] O. Ahmadi, D. Hankerson and A. Menezes, Software implementation of arithmetic in $\mathbb{F}_{3^m}$, *International Workshop on Arithmetic of Finite Fields (WAIFI 2007)*, LNCS 4547 (2007), 85–102.

[2] R. Avanzi and N. Thériault, Effects of optimizations for software implementations of small binary field arithmetic, *International Workshop on Arithmetic of Finite Fields (WAIFI 2007)*, LNCS 4547 (2007), 69–84.

[3] P. Barreto, S. Galbraith, C. Ó' hÉigeartaigh and M. Scott, Efficient pairing computation on supersingular abelian varieties, *Designs, Codes and Cryptography*, **42** (2007), 239–271.

[4] P. Barreto, H. Kim, B. Lynn and M. Scott, Efficient algorithms for pairing-based cryptosystems, *Advances in Cryptology–CRYPTO 2002*, LNCS 2442 (2002), 354–368.

[5] P. Barreto and M. Naehrig, Pairing-friendly elliptic curves of prime order, *Selected Areas in Cryptography (SAC 2005)*, LNCS 3897 (2006), 319–331.

[6] D. Bernstein, A software implementation of NIST P-224, presentation at ECC 2001, University of Waterloo, 2001. Slides and software available via http://cr.yp.to/nistp224.html.

[7] D. Bernstein, Curve25519: New Diffie-Hellman speed records, *Public Key Cryptography–PKC 2006*, LNCS 3958 (2006), 207–228.

[8] D. Bernstein and T. Lange, Faster addition and doubling on elliptic curves, *Advances in Cryptology–ASIACRYPT 2007*, LNCS 4833 (2007), 29–50.

[9] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase and T. Takagi, Algorithms and arithmetic operators for computing the $\eta_T$ pairing in characteristic three, Cryptology ePrint Archive, Report 2007/417, 2007.

[10] J.-L. Beuchat, M. Shirase, T. Takagi and E. Okamoto, An algorithm for the $\eta_T$ pairing calculation in characteristic three and its hardware implementation, *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 97–104. IEEE Computer Society, 2007.

[11] J.-L. Beuchat, M. Shirase, T. Takagi and E. Okamoto, A refined algorithm for the $\eta_T$ pairing calculation in characteristic three, Cryptology ePrint Archive, Report 2007/311, 2007.

[12] X. Boyen and L. Martin, Identity-based cryptography standard (IBCS) #1: Supersingular curve implementations of the BF and BB1 cryptosystems, IETF RFC 5091, December 2007.

[13] S. Chatterjee, P. Sarkar and R. Barua, "Efficient computation of Tate pairing in projective coordinate over general characteristic fields", *Information Security and Cryptology–ICISC 2004*, LNCS 3506 (2005), 168–181.

[14] J. Chung and A. Hasan, "Asymmetric squaring formulae", *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, 113–122.

[15] H. Cohen, C. Doche and G. Frey, editors, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, 2005.

[16] D. Cook, J. Ionnidis, A. Keromytis and J. Luck, CryptoGraphics: Secret key cryptography using graphics cards, *Topics in Cryptology–CT-RSA 2005*, LNCS 3376 (2005), 334–350.

[17] A. Devegili, C. Ó' hÉigeartaigh, M. Scott and R. Dahab, Multiplication and squaring on pairing-friendly fields, Cryptology ePrint Archive, Report 2006/471, 2006.

[18] A. Devegili, M. Scott and R. Dahab, Implementing cryptographic pairings over Barreto-Naehrig curves, *Pairing-Based Cryptography–Pairing 2007*, LNCS 4575 (2007), 197–207.

[19] S. Galbraith, K. Paterson and N. Smart, Pairings for cryptographers, Cryptology ePrint Archive, Report 2006/165, 2006.

[20] P. Gaudry, Fast genus 2 arithmetic based on Theta functions, *Journal of Mathematical Cryptology*, **1** (2007), 243–266.
[21] R. Granger, D. Page, and M. Stam, On small characteristic algebraic tori in pairing-based cryptography, *LMS Journal of Computation and Mathematics*, **9** (2006), 64–85.
[22] T. Granlund, Instruction latencies and throughput for AMD and Intel x86 processors, Swox AB, http://swox.com/doc/x86-timing.pdf, 2007.
[23] S. Gueron and M. E. Kounavis, Carry-less multiplication and its usage for computing the GCM mode, Intel Corporation White Paper, 2008.
[24] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2003.
[25] K. Harrison, D. Page and N. Smart, Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems, *LMS Journal of Computation and Mathematics*, **5** (2002), 181–193.
[26] F. Hess, N. Smart and F. Vercauteren, The eta pairing revisited, *IEEE Transactions on Information Theory*, **52** (2006), 4595–4602.
[27] Intel Corporation, *Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual*, 2001. Number 248966-04, available from http://developer.intel.com.
[28] E. Lee, H.-S. Lee and C.-M. Park, Efficient and generalized pairing computation on abelian varieties, Cryptology ePrint Archive, Report 2008/040, 2008.
[29] A. Lenstra, Unbelievable security: Matching AES security using public key systems, *Advances in Cryptology–Asiacrypt 2001*, LNCS 2248 (2001), 67–86.
[30] J. López and R. Dahab, High-speed software multiplication in $\mathbb{F}_{2^m}$, *Progress in Cryptology–Indocrypt 2000*, LNCS 1977 (2000), 203–212.
[31] V. Miller, The Weil pairing and its efficient calculation, *Journal of Cryptology*, **17** (2004), 235–261.
[32] A. Miyaji, M. Nakabayashi and S. Takano, New explicit conditions of elliptic curve traces for FR-reduction, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **E84-A** (2001), 1234–1243.
[33] A. Moss, D. Page and N. Smart, Toward acceleration of RSA using 3D graphics hardware, *Cryptography and Coding 2007*, LNCS 4887 (2007), 364–383.
[34] P. Szczechowiak, L. Oliveira, M. Scott, M. Collier and R. Dahab, NanoECC: Testing the limits of elliptic curve cryptography in sensor networks, *Wireless Sensor Networks (EWSN 2008)*, LNCS 4913 (2008), 305–320.
[35] M. Scott, MIRACL – Multiprecision Integer and Rational Arithmetic C Library, http://www.computing.dcu.ie/~mike/miracl.html.
[36] M. Scott, Implementing cryptographic pairings, *Pairing-Based Cryptography–Pairing 2007*, LNCS 4575 (2007), 177–196.
[37] M. Scott. Optimal irreducible polynomials for $GF(2^m)$ arithmetic, Cryptology ePrint Archive, Report 2007/192, 2007.
[38] M. Scott, N. Costigan and W. Abdulwahab, Implementing cryptographic pairings on smartcards, *Cryptographic Hardware and Embedded Systems–CHES 2006*, LNCS 4249 (2006), 134-147.
[39] M. Stam and A. Lenstra, "Efficient subgroup exponentiation in quadratic and sixth degree extensions", *Cryptographic Hardware and Embedded Systems–CHES 2002*, LNCS 2523 (2003), 159-174.
[40] G. Takahashi, F. Hoshino and T. Kobayashi, Efficient $GF(3^m)$ multiplication algorithm for $\eta_T$ pairing, Cryptology ePrint Archive, Report 2007/463, 2007.
[41] D. Weaver and T. Germond, editors, *The SPARC Architecture Manual, Version 9*, Prentice Hall, 1994.