Affine Pairings on ARM

Tolga Acar, Kristin Lauter, Michael Naehrig, and Daniel Shumow

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
`{tolga, klauter, mnaehrig, danshu}@microsoft.com`

Abstract. Pairings on elliptic curves are being used in an increasing number of cryptographic applications on many different devices and platforms, but few performance numbers for cryptographic pairings have been reported on embedded and mobile devices.
In this paper we give performance numbers for affine and projective pairings on a dual-core Cortex A9 ARM processor and compare performance of the same implementation across three platforms: x86, x86-64 and ARM. Using a fast inversion in the base field and doing inversion in extension fields by using the norm map to reduce to inversions in smaller fields, we find a very low ratio of inversion-to-multiplication costs. In our implementation, this favors using affine coordinates on all three platforms, even for the current 128-bit minimum security level specified by NIST. We use Barreto-Naehrig (BN) curves and report on the performance of an optimal ate pairing for curves covering security levels roughly between 128 and 192 bits. We compare with other reported performance numbers for pairing computation on ARM processors.
Keywords: Pairing computation, affine coordinates, optimal ate pairing, pairing cost, ARM architecture.

1 Introduction

Cryptographic protocols based on bilinear pairings are proliferating, as more and more interesting and useful applications of bilinear maps are discovered, for example non-interactive proof systems [14,8], homomorphic encryption [10], and attribute-based encryption [27]. These developments have led to an increasing need for efficient implementations of pairing-based protocols on a wide range of platforms and devices. The only currently known way to implement bilinear pairings in cryptography which is both efficient and secure is on elliptic curves (or higher dimensional analogues) over finite fields with optimal versions of the Weil or Tate pairings [20,21,5,6,12,18,30]. Security is based on the hardness of the discrete logarithm problem, pairing inversion, and related assumptions [4,29].

In implementations of pairing-based protocols, message values are hashed into points on elliptic curves, and elliptic curve scalar multiplications and pairing operations are computed. The performance of the underlying finite field operations such as addition, multiplication, and inversion, determine the best choices for how to do the elliptic curve and pairing operations. When the inversion-to-multiplication ratio for field arithmetic is low, it favors using affine instead of projective coordinates for representing elliptic curve points and doing elliptic curve and pairing operations, as was recently investigated in [17].

For Elliptic Curve Cryptography (ECC) applications on NIST P-curves where prime fields are chosen with Generalized Mersenne primes for fast modular reduction and multiplication, the base field inversion-to-multiplication ratio is often reported to be 80 : 1 or higher. However, for pairing applications which require fixed embedding degrees to control efficiency and security, special primes like Mersenne primes cannot be used because there is no known way to generate pairing-friendly curves over those particular fields. Instead, more general prime fields arise, and much of the arithmetic in pairing computation is done in extension fields, whose degree is 12 when using Barreto-Naehrig (BN) curves [7,26]. Computing in general prime fields using fast inversion techniques, a typical inversion-to-multiplication ratio can be much lower than 80 : 1, for example 13 : 1 (resp. 25 : 1) in our x86 (resp. x86-64) implementation of arithmetic over 256-bit prime fields reported in Section 4.

In [17], inversion-to-multiplication ratios in extension fields were given which reflect faster inversion in extension fields by taking the norm down to smaller fields and doing inversion there. For example, in an extension field of degree 12, the inversion-to-multiplication ratio was reported as 1.7 : 1 for a 256-bit prime base field. Even for implementations with much faster field multiplies, using that technique, the ratio decreases dramatically as the field extension degree increases, which leads to the argument made in [17] that for any implementation, as the security requirements and thus the field extension degrees grow, there exists a cross-over point after which it becomes more efficient to use affine coordinates in the pairing algorithm rather than projective coordinates. For our implementation of field arithmetic in 256-bit prime fields on the x86 and x86-64 platforms, we find this cross-over point already when considering extension degree $k = 2$. Our implementation targets a minimum 128-bit security level. It thus works with BN curves with embedding degree 12, and involves curve arithmetic in a degree-2 extension field by taking advantage of sextic twists as usual.

This led us to wonder what the cross-over point might be on other platforms, and how it would vary with different intrinsics or instruction sets. In this paper, we give performance numbers for affine and projective pairings on a dual-core Cortex A9 ARM processor and compare performance of the same implementation across three platforms: x86, x86-64, and ARM. On all three platforms, we use intrinsics and assembly for the Montgomery multiplication implementation.

We find that, in our implementation, affine coordinates are the better choice for pairing computation also on the ARM processor, and examine the variation in performance of pairings on ARM with different base field multiplications. Other implementations presented in the literature recently have faster field multiplies that are hand-optimized for specific processor architectures to obtain pairing speed records [24,9,3]. We do not aim at comparable optimizations of ARM pairing implementations; instead we aim to keep more generality in the base field size and base field multiplication implementation, with the goal of producing product-quality code. Thus our implementation can be further optimized for specific architectures, but already we find that our implementation of affine pairings compares favorably with all other reported ARM pairing timings we have found in the literature (see Section 5).

In Section 2, we give background on BN curves and the optimal ate pairing. Section 3 explains our improvements to the multiply on ARM using intrinsics and assembly code. Section 4 gives a description of our implementation and performance numbers including a subsection which summarizes trends we observe across platforms and security levels. Finally, in Section 5 we compare our results with related work in the literature.

2 BN curves and the optimal ate pairing

In this section, we give a brief overview of the background on Barreto-Naehrig (BN) curves [7] and optimal ate pairings [30] as used in our implementation. Let $u \in \mathbb{Z}$ be an integer such that $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and $n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$ are prime numbers. The pair (p, n) is called a BN prime pair. A BN curve is an elliptic curve E over $\mathbb{F}_p$ such that $n = \#E(\mathbb{F}_p)$ and (p, n) is a BN prime pair. In that case, the curve has an equation $E : y^2 = x^3 + b$, $b \in \mathbb{F}_p$ and embedding degree $k = 12$.

To compute a pairing on a BN curve, we identify two groups G_1 and G_2 that are determined by the eigenspaces of the Frobenius endomorphism ϕ_p on the n-torsion $E[n]$. The first one is the 1-eigenspace, i.e. it is simply $G_1 = E(\mathbb{F}_p)[n] = E(\mathbb{F}_p)$. The second one, G_2, is the p-eigenspace which is contained in $E(\mathbb{F}_{p^{12}})[n]$ (see [15]). For efficiency reasons, we represent G_2 by another group G_2' of points on a different curve. This other curve is a particular twist E' of E which is defined over $\mathbb{F}_{p^2}$. Furthermore, we require that $n \mid \#E'(\mathbb{F}_{p^2})$. We take the group G_2' to be the n-torsion on E' over $\mathbb{F}_{p^2}$, i.e. $G_2' = E'(\mathbb{F}_{p^2})[n]$. There is a twisting isomorphism $\psi : E' \to E,\ (x, y) \mapsto (\omega^2 x, \omega^3 y)$ for some $\omega \in \mathbb{F}_{p^{12}}$ that gives us an easy-to-compute group isomorphism $G_2' \to G_2$. The twist has an equation $E' : y^2 = x^3 + b/\xi$, where $\xi = \omega^6 \in \mathbb{F}_{p^2}$ (for more details see [23, Chapter 2]). Having fixed the groups for pairing computation, we now give an algorithm to compute pairings on BN curves.

The most efficient pairing algorithms on BN curves occur when computing optimal ate pairings [30]. An optimal ate pairing is given by

$$a_{\text{opt}} : G_2' \times G_1 \to \mathbb{F}_{p^{12}}^*, (Q', P) \mapsto (f_{6u+2,Q}(P) \cdot h_{6u+2,Q}(P))^{(p^{12}-1)/n},$$

where $Q = \psi(Q')$ and $f_{m,Q}(P)$ is a Miller function with respect to m and Q, evaluated at P. Furthermore, the function h is given by $h_{6u+2,Q}(P) = l_{[6u+2]Q,Q_1}(P) \cdot l_{[6u+2]Q+Q_1,-Q_2}(P)$ with $Q_1 = \phi_p(Q)$, $Q_2 = \phi_p(Q_1)$ and $l_{R,S}$ is the function given by the line through points R and S.

Miller's algorithm [20,21] gives us a way to evaluate such functions. To compute the above optimal ate pairing, we use Algorithm 1. We give a version of the algorithm that exploits the scalar $6u + 2$ in non-adjacent form (NAF). For certain curves in the particular implementation-friendly subfamily of BN curves that has been introduced recently in [26], the best performance is obtained when using the NAF version of Miller's algorithm.

Algorithm 1 Optimal ate pairing for BN curves.

```
Input: $P \in G_1 = E(\mathbb{F}_p)$, $Q' \in G_2'$, $Q = \psi(Q')$, $m = 6u + 2 = (1, m_{s-1}, \ldots, m_0)_{\mathrm{NAF}}$.
Output: $a_{\mathrm{opt}}(Q', P)$.
 1: $R' \leftarrow Q'$, $f \leftarrow 1$
 2: for $(i \leftarrow s - 1;\ i \geq 0;\ i - -)$ do
 3:     $f \leftarrow f^2 \cdot l_{\psi(R'),\psi(R')}(P)$, $R' \leftarrow [2]R'$
 4:     if $(m_i = \pm 1)$ then
 5:         $f \leftarrow f \cdot l_{\psi(R'),\pm Q}(P)$, $R' \leftarrow R' \pm Q'$
 6:     end if
 7: end for
 8: if $u < 0$ then
 9:     $f \leftarrow f^{p^6}$, $R' \leftarrow -R'$
10: end if
11: $Q_1 = \phi_p(Q)$, $Q_2 = \phi_{p^2}(Q)$
12: $f \leftarrow f \cdot l_{\psi(R'),Q_1}(P)$, $R' \leftarrow R' + \psi^{-1}(Q_1)$
13: $f \leftarrow f \cdot l_{\psi(R'),-Q_2}(P)$
14: $f \leftarrow f^{p^6-1}$
15: $f \leftarrow f^{p^2+1}$
16: $f \leftarrow f^{(p^4-p^2+1)/n}$
17: return $f$
```

3 Platform-specific improvements on ARM

We used the same C source code on all three platforms. The implementations differ only in a few places where performance matters the most in implementation: unsigned integer multiplication. Regardless of the higher level algorithms, we chose three approaches to implementing the inner-most multiplication routines utilizing multiply, multiply-accumulate, and Montgomery multiplication [22]. Our choice of compiler is Microsoft Visual C++ on x86 (32-bit Intel), x86-64 (64-bit Intel), and ARM (Thumb-2).

We used a Tegra 2 development platform from NVidia to obtain the benchmark figures on Table 1. This system features a dual-core Cortex A9 ARM CPU running at 1GHz with 32KB/32KB (I/D) L1 cache per core, 1MB L2 cache, and 1GB DDR2-667 main memory. The entire benchmark program fits in the 1MB L2 cache, and the core routines executed in tight loops fit in the 32KB instruction cache.

The Montgomery multiplication function implements the CIOS method in [16], and its performance numbers are tabulated in Table 1 for a set of moduli length in bits. The C implementation relies on the compiler's support for double-length unsigned integers (`unsigned __int64`). The intrinsics method uses a few compiler-supported ARM assembly instructions: `umull, umaal, umlal` while other operations are implemented in C. The `umull` is an unsigned 32-bit integer multiplication instruction that generates an unsigned 64-bit product. The `umlal` is a 32-bit multiply and 64-bit accumulate, and the `umaal` is a 32-bit multiply and double 32-bit accumulate instruction. The assembly row reports the benchmark figures where the CIOS method is implemented in ARM Thumb-2 assembly language.

The difference between the assembly and the C-with-intrinsics implementation is in the Montgomery multiplication routine. Both implementations use the above instrinsics in other primitive functions (e.g., multiply and multiply-accumulate) for other purposes, such as inversion.

	Modulus length in bits					
Implementation	**160**	**224**	**288**	**480**	**640**	**3168**
Intrinsics	2.07	2.55	3.17	5.66	9.26	147
Assembly	1.97	2.41	2.93	5.15	9.04	128

Table 1. Montgomery multiplication implementation choices and benchmark figures in micro seconds.

While the use of intrinsics provides an improvement over the C version, the assembly implementation provides an incremental improvement over intrinsics. We experimented with several implementation approaches such as loop unrolling, different instruction ordering, conditional instructions, and multi-word load/stores. None of these approaches provided a measurable performance improvement on our reference platform. Thus, we did not use any of these techniques to generate the numbers on the table. Instead, we carefully crafted a straightforward assembly implementation of the Montgomery multiplication CIOS algorithm in [16] to form base reference benchmark numbers.

Our intention was to generate product quality software, and we adhered to sound software engineering guidelines including maintainability, robustness, and defensive programming. While our interest is clearly to create good benchmark results, this goal does not disregard qualities we expect from a product-quality code. As a result, we think it is possible to further optimize our code for speed (i.e. hand-optimize the code).

The assembly implementation and intrinsics only leverage the core ARM instruction set, but do not utilize SIMD and NEON instructions. In the future, we intend to experiment with the ARM SIMD and NEON technologies for better performance.

4 Implementation and performance

In this section, we present the timing results of our pairing implementation on BN curves for the ARM, x86, and x86-64 instruction sets, aiming at security levels of at least 128 bits or higher.

We use BN curves since they not only suit the 128-bit security level extremely well, but are also promising candidates for being the most efficient choice of pairing-friendly curves with security of up to 192 bits. Our pairing code can be used to compute pairings on all 16 curves recently introduced in [26]. In particular, the code is not tailored for one specific curve. These curves are easy to generate, have a very compact representation and were chosen to provide very efficient implementation. The loop order $6u + 2$ for all curves is very sparse when represented in non-adjacent form. Furthermore, the curve of size 254 bits has recently been used to obtain the current software speed record for pairings as outlined in [3].

Due to space constraints, we present performance results for only three of the curves in [26], namely the curves bn254, bn446, and bn638 over prime fields of respective bit sizes 254, 446, and 638 bits. We further give results for the same 256 bit curve bn256 that has been used in [17]. The curve bn254 roughly provides 128 bits of security and bn638 yields about 192 bits.

Our implementation uses Algorithm 1 to compute the optimal ate pairing on BN curves. For the projective version we used the explicit formulas in [11], but we obtained better results for the affine version. It uses the tower of field extensions $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}/\mathbb{F}_{p^2}/\mathbb{F}_p$ to realize field arithmetic in $\mathbb{F}_{p^{12}}$. The final exponentiation is done as usual using the Frobenius action and the addition chain from [28] as well as the special squaring functions from [13].

Our implementation results are shown in Tables 2 to 5 for the above mentioned curves. We give timings for the finite field additions (**add**), subtractions (**sub**), multiplications (**M**), squarings (**S**) and inversions (**I**) as well as the inversion-to-multiplication ratio ($\mathbf{R} = \mathbf{I}/\mathbf{M}$) for all fields in the tower of extensions.

We give timings for several pairing functions that use different optimizations for different computing scenarios. The line entitled "20 at once (per pairing)" gives the average timing for one pairing out of 20 that have been computed at the same time. This function uses Montgomery's inversion-sharing trick as described in [17, Section 4.3]. The function corresponding to the line "product of 20" computes the product of 20 pairings. The lines with the attribute "1st arg. fixed" mean functions that compute multiple pairings or a product of pairings, where the first input point is fixed for all pairings, and only the second point varies. In this case, the operations depending only on the first argument are done only once. We list separately the final exponentiation timings. They are included in the pairing timings of the other lines.

We do not give cycle counts for the ARM implementation in the tables since high-frequency counters are currently not supported in our development environment on the ARM. However, estimates for cycle counts can be easily read off from the values given in μs and s by multiplying them by 10^3 and 10^6, respectively (note the clock frequency of 1GHz for the ARM processor).

ARM, dual-core Cortex A9 @ 1GHz, Windows

254-bit prime field	**add** µs	**sub** µs	**M** µs	**S** µs	**I** µs	$\mathbf{R = I/M}$
$\mathbb{F}_p$	0.67	0.61	1.72	1.68	18.35	10.67
$\mathbb{F}_{p^2}$	1.42	1.24	8.18	5.20	26.61	3.25
$\mathbb{F}_{p^6}$	4.43	3.96	69.83	48.24	136.68	1.96
$\mathbb{F}_{p^{12}}$	9.00	8.32	228.27	161.43	379.09	1.66

optimal ate pairings bn254		ARM ms
projective		55.19
affine	single pairing	51.01
	20 at once (per pairing)	50.71
	20 at once, 1st argument fixed (per pairing)	46.06
	product of 20 (per pairing)	17.44
	product of 20, 1st argument fixed (per pairing)	12.72
single final exponentiation		24.69

x86, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)

254-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R = I/M}$
prime field	cyc	µs	cyc	µs	cyc	µs	cyc	µs	cyc	µs	
$\mathbb{F}_p$	291	0.12	268	0.11	967	0.40	965	0.40	13224	5.44	13.60
$\mathbb{F}_{p^2}$	548	0.23	526	0.22	4297	1.78	2709	1.12	17658	7.30	4.10
$\mathbb{F}_{p^6}$	1604	0.66	1488	0.63	35476	14.78	24460	10.24	74338	31.13	2.11
$\mathbb{F}_{p^{12}}$	3106	1.30	2933	1.24	116109	49.13	82600	33.84	201266	84.37	1.72

optimal ate pairings		x86	
bn254		cyc	ms
projective		28,371,661	11.78
affine	single pairing	26,570,757	10.96
	20 at once (per pairing)	26,195,898	10.84
	20 at once, 1st argument fixed (per pairing)	23,811,273	9.81
	product of 20 (per pairing)	8,959,470	3.74
	product of 20, 1st argument fixed (per pairing)	6,556,946	2.74
single final exponentiation		12,876,582	5.24

x86-64, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)

254-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R = I/M}$
prime field	cyc	µs	cyc	µs	cyc	µs	cyc	µs	cyc	µs	
$\mathbb{F}_p$	191	0.08	163	0.07	405	0.18	403	0.17	10803	4.50	25.00
$\mathbb{F}_{p^2}$	336	0.14	299	0.13	2131	0.88	1318	0.55	12774	5.33	6.06
$\mathbb{F}_{p^6}$	942	0.39	825	0.35	19103	7.91	13053	5.43	41710	17.48	2.21
$\mathbb{F}_{p^{12}}$	1807	0.76	1624	0.68	61927	25.97	43646	18.47	107857	45.46	1.75

optimal ate pairings		x86-64	
bn254		cyc	ms
projective		14,989,039	6.31
affine	single pairing	14,125,439	5.92
	20 at once (per pairing)	13,697,623	5.73
	20 at once, 1st argument fixed (per pairing)	12,491,536	5.20
	product of 20 (per pairing)	4,688,080	1.97
	product of 20, 1st argument fixed (per pairing)	3,466,350	1.45
single final exponentiation		6,633,846	2.77

Table 2. Field arithmetic timings in a 254-bit prime field and optimal ate pairing timings on a 254-bit BN curve. Field timings average over 1000 operations, pairing timings average over 20 pairings. Timings given in cpucycles (cyc) and milliseconds (ms).

ARM, dual-core Cortex A9 @ 1GHz, Windows						
256-bit	**add**	**sub**	**M**	**S**	**I**	$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	μs	μs	μs	μs	μs	
$\mathbb{F}_p$	0.68	0.62	1.71	1.67	18.44	10.78
$\mathbb{F}_{p^2}$	1.36	1.23	8.13	6.27	26.73	3.29
$\mathbb{F}_{p^6}$	4.46	3.88	69.60	48.21	137.12	1.97
$\mathbb{F}_{p^{12}}$	8.91	8.34	227.86	159.89	379.15	1.66

optimal ate pairings		ARM
bn256		ms
projective		58.10
	single pairing	54.19
	20 at once (per pairing)	53.88
affine	20 at once, 1st argument fixed (per pairing)	49.07
	product of 20 (per pairing)	17.95
	product of 20, 1st argument fixed (per pairing)	13.16
single final exponentiation		27.44

x86, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
256-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	288	0.12	266	0.12	969	0.41	965	0.40	13067	5.42	13.22
$\mathbb{F}_{p^2}$	544	0.23	523	0.22	4266	1.78	2707	1.15	17722	7.32	4.11
$\mathbb{F}_{p^6}$	1580	0.66	1491	0.62	35365	14.89	24460	10.30	75722	31.48	2.11
$\mathbb{F}_{p^{12}}$	3097	1.29	2627	1.24	116630	49.20	81316	33.97	201589	84.65	1.72

optimal ate pairings		x86	
bn256		cyc	ms
projective		30,042,916	12.31
	single pairing	28,404,966	11.73
	20 at once (per pairing)	27,985,410	11.47
affine	20 at once, 1st argument fixed (per pairing)	25,508,592	10.42
	product of 20 (per pairing)	9,271,505	3.87
	product of 20, 1st argument fixed (per pairing)	6,811,275	2.84
single final exponentiation		14,368,538	5.81

x86-64, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
256-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	187	0.08	162	0.07	407	0.18	408	0.17	10929	4.53	25.17
$\mathbb{F}_{p^2}$	328	0.13	299	0.12	2121	0.89	1319	0.55	12878	5.39	6.06
$\mathbb{F}_{p^6}$	941	0.39	825	0.35	19099	7.87	13064	5.43	41806	17.43	2.21
$\mathbb{F}_{p^{12}}$	1803	0.76	1623	0.68	61950	25.84	43720	18.22	108664	45.06	1.74

optimal ate pairings		x86-64	
bn256		cyc	ms
projective		15,601,367	6.61
	single pairing	15,151,212	6.30
	20 at once (per pairing)	14,519,677	6.09
affine	20 at once, 1st argument fixed (per pairing)	13,315,367	5.57
	product of 20 (per pairing)	5,039,186	2.02
	product of 20, 1st argument fixed (per pairing)	3,598,167	1.50
single final exponentiation		7,422,983	3.07

Table 3. Field arithmetic timings in a 256-bit prime field and optimal ate pairing timings on a 256-bit BN curve. Field timings average over 1000 operations, pairing timings average over 20 pairings. Timings given in cpucycles (cyc) and milliseconds (ms).

ARM, dual-core Cortex A9 @ 1GHz, Windows						
446-bit	**add**	**sub**	**M**	**S**	**I**	$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	μs	μs	μs	μs	μs	
$\mathbb{F}_p$	1.17	1.03	4.01	3.92	35.85	8.94
$\mathbb{F}_{p^2}$	2.37	2.07	17.24	10.84	54.23	3.15
$\mathbb{F}_{p^6}$	7.77	7.15	152.79	109.74	302.34	1.98
$\mathbb{F}_{p^{12}}$	15.65	14.88	498.58	364.34	846.21	1.70

optimal ate pairings		ARM
bn446		ms
projective		195.56
	single pairing	184.28
	20 at once (per pairing)	183.54
affine	20 at once, 1st argument fixed (per pairing)	167.83
	product of 20 (per pairing)	62.33
	product of 20, 1st argument fixed (per pairing)	46.50
single final exponentiation		86.75

x86, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
446-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	449	0.19	415	0.17	1890	0.79	1877	0.78	25400	10.53	13.33
$\mathbb{F}_{p^2}$	839	0.35	809	0.34	7768	3.24	5166	2.09	33890	14.10	4.35
$\mathbb{F}_{p^6}$	2500	1.04	2353	0.98	67611	28.35	48732	20.45	145554	60.94	2.15
$\mathbb{F}_{p^{12}}$	4933	2.06	4723	1.95	220707	92.51	161286	67.80	388908	162.81	1.76

optimal ate pairings		x86	
bn446		cyc	ms
projective		87,212,586	36.49
	single pairing	83,336,787	34.95
	20 at once (per pairing)	81,812,153	34.30
affine	20 at once, 1st argument fixed (per pairing)	77,889,927	31.26
	product of 20 (per pairing)	28,082,417	11.78
	product of 20, 1st argument fixed (per pairing)	20,882,757	8.72
single final exponentiation		38,388,060	16.08

x86-64, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
446-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	275	0.11	233	0.10	850	0.36	851	0.35	19188	8.04	22.33
$\mathbb{F}_{p^2}$	493	0.20	457	0.19	3821	1.60	2445	1.01	22957	9.68	6.05
$\mathbb{F}_{p^6}$	1443	0.61	1276	0.53	36683	14.81	26306	10.82	79514	33.28	2.25
$\mathbb{F}_{p^{12}}$	2929	1.22	2886	1.10	116564	48.31	85535	35.60	206192	86.05	1.78

optimal ate pairings		x86-64	
bn446		cyc	ms
projective		45,515,035	19.05
	single pairing	44,152,307	18.46
	20 at once (per pairing)	42,817,218	17.91
affine	20 at once, 1st argument fixed (per pairing)	39,195,813	16.43
	product of 20 (per pairing)	14,465,284	6.01
	product of 20, 1st argument fixed (per pairing)	10,840,389	4.53
single final exponentiation		20,156,765	8.47

Table 4. Field arithmetic timings in a 446-bit prime field and optimal ate pairing timings on a 446-bit BN curve. Field timings average over 1000 operations, pairing timings average over 20 pairings. Timings given in cpucycles (cyc) and milliseconds (ms).

ARM, dual-core Cortex A9 @ 1GHz, Windows						
638-bit	**add**	**sub**	**M**	**S**	**I**	$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	μs	μs	μs	μs	μs	
$\mathbb{F}_p$	1.71	1.53	8.22	8.18	56.09	6.82
$\mathbb{F}_{p^2}$	3.48	3.17	31.81	20.55	91.92	2.89
$\mathbb{F}_{p^6}$	10.63	10.09	261.87	186.21	535.42	2.04
$\mathbb{F}_{p^{12}}$	21.04	20.28	840.07	607.36	1454.38	1.73

optimal ate pairings		ARM
bn638		ms
projective		768.06
	single pairing	649.85
	20 at once (per pairing)	650.08
affine	20 at once, 1st argument fixed (per pairing)	609.45
	product of 20 (per pairing)	164.82
	product of 20, 1st argument fixed (per pairing)	124.08
single final exponentiation		413.37

x86, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
638-bit	**add**		**sub**		**M**		**S**		**I**		**R**
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	597	0.25	547	0.23	3205	1.36	3197	1.37	38882	16.17	11.89
$\mathbb{F}_{p^2}$	1141	0.48	1089	0.46	12453	5.23	8048	3.46	37111	21.99	4.20
$\mathbb{F}_{p^6}$	3388	1.42	3188	1.34	103040	43.08	73492	30.94	227431	95.20	2.21
$\mathbb{F}_{p^{12}}$	6713	2.80	6373	2.66	331261	139.57	240109	100.24	592520	242.72	1.74

optimal ate pairings		x86	
bn638		cyc	ms
projective		310,613,286	131.62
	single pairing	263,677,987	113.17
	20 at once (per pairing)	265,516,406	111.92
affine	20 at once, 1st argument fixed (per pairing)	245,536,930	105.21
	product of 20 (per pairing)	68,437,346	27.57
	product of 20, 1st argument fixed (per pairing)	49,751,199	20.81
single final exponentiation		167,124,395	72.70

x86-64, dual-core Intel Core2 E6600 @ 2.4 GHz, Windows 7 (64-bit)											
638-bit	**add**		**sub**		**M**		**S**		**I**		$\mathbf{R} = \mathbf{I}/\mathbf{M}$
prime field	cyc	μs	cyc	μs	cyc	μs	cyc	μs	cyc	μs	
$\mathbb{F}_p$	366	0.15	302	0.13	1501	0.64	1471	0.62	28451	11.90	18.59
$\mathbb{F}_{p^2}$	659	0.27	620	0.25	6176	2.58	3961	1.64	34935	14.64	5.67
$\mathbb{F}_{p^6}$	1961	0.83	1737	0.75	52965	22.17	37768	15.85	122208	51.14	2.31
$\mathbb{F}_{p^{12}}$	3858	1.61	3584	1.50	171676	71.60	124590	52.01	313769	130.92	1.83

optimal ate pairings		x86-64	
bn638		cyc	ms
projective		157,309,156	65.95
	single pairing	136,534,428	56.88
	20 at once (per pairing)	133,301,871	55.79
affine	20 at once, 1st argument fixed (per pairing)	125,428,050	52.36
	product of 20 (per pairing)	33,485,833	13.97
	product of 20, 1st argument fixed (per pairing)	25,288,926	10.57
single final exponentiation		85,037,269	35.61

Table 5. Field arithmetic timings in a 638-bit prime field and optimal ate pairing timings on a 638-bit BN curve. Field timings average over 1000 operations, pairing timings average over 20 pairings. Timings given in cpucycles (cyc) and milliseconds (ms).

4.1 Summary of our implementation performance results

Here we summarize some of the results given in Tables 2, 3, 4, and 5 on the performance of our implementation across platforms and security levels. As high-level points of comparison, we note that:

1. At all security levels, there is roughly 10-fold speed-up when switching from the ARM to the x86-64 architecture in clock-unadjusted form. The speed-up when switching from the ARM to the x86 architecture is about 5-fold (clock unadjusted). For example at the 128-bit security level, Table 2 shows an affine optimal ate pairing on the ARM processor at 51 milliseconds vs. 5.9 milliseconds on x86-64. Using the same word-length (unit of operation for unsigned integers) of 32 bits, a still rough but almost apples-to-apples comparison between ARM and x86 when adjusted for clock speed can be given by dividing the ARM timing by 2.4 to get $51/2.4 = 21.25$ milliseconds. This compares to 13 milliseconds on x86, slightly less than twice the speed of the ARM platform. We conclude that, architecturally, computing a pairing on x86 is about twice faster than on our ARM platform. We think that the major reason for this difference is the use of SSE2 instructions on x86. The clock-adjusted difference should narrow when we use SIMD instructions on ARM.
2. Affine coordinates are better than projective coordinates for optimal ate pairing computation in all cases shown: all platforms, all security levels. The trend is toward bigger differences at higher security levels. On all three platforms, the affine pairing is roughly 20% better at the 192-bit security level instead of 10% better at the 128-bit security level.

   For example, on ARM for the 254-bit curve, an affine pairing takes 51 milliseconds while the projective pairing takes 55 milliseconds, whereas on ARM for the 638-bit curve, an affine pairing takes 650 milliseconds while the projective pairing takes 768 milliseconds. On x86-64 for the 254-bit curve, an affine pairing takes 5.9 milliseconds while the projective pairing takes 6.3 milliseconds, whereas on x86-64 for the 638-bit curve, an affine pairing takes 56.9 milliseconds while the projective pairing takes 65.9 milliseconds.
3. The inversion-to-multiplication ratio is lower in the base fields on the ARM platform at all security levels. For example, for 254-bit fields, the I/M ratio on ARM is 10.7 versus 25 for the x86-64 platform. For 638-bit fields, the I/M ratio on ARM is 6.8 versus 18.6 for the x86-64 platform.
4. The inversion-to-multiplication ratio is lower in larger base fields on all platforms. This can be seen in the example just given for the previous observation. Also, it largely explains observation 2 above.
5. In the degree-12 extension fields, the inversion-to-multiplication ratio is close to 1.7:1 on all platforms at all security levels. There is very little variation in that, despite big differences in ratios in the base fields, as observed in the last two points.
6. The percentage of the computation time spent on the final exponentiation goes up at the higher security levels, and this is true across platforms:

For example, on ARM for the 254-bit curve, an affine pairing spends 48% of the time on the final exponentiation, whereas on ARM for the 638-bit curve, an affine pairing spends 63% of the time on the final exponentiation. On x86-64 for the 254-bit curve, an affine pairing spends 47% of the time on the final exponentiation, whereas on x86-64 for the 638-bit curve, an affine pairing spends 63% of the time on the final exponentiation.

5 Related work

In this section, we compare the performance of our pairing implementation on ARM to related work previously presented in the literature. However, we note that different implementations are often hard to compare because they use very different underlying hardware, different processors that may operate at totally different clock frequencies, use different operating systems, aim at a different security level or use other pairing-friendly curves with different properties. We still feel that it is worth contrasting our implementation with previous work.

For applications of pairings to privacy of electronic medical records using Attribute-Based Encryption for key management, some recent performance numbers for pairings on ARM processors were reported in [1]. The comments in [1, Section 6.1] give rough performance numbers for pairings on ARM: the Pairing-Based Crypto (PBC) [19] library computes pairings in 135 milliseconds on an ARM processor running on Apple A4 chip-based iPhone 4, running iOS 4 with 512MB of RAM and computing on a 224-bit MNT elliptic curve.

As mentioned above, it is hard to compare across different hardware and operating systems, but as a point of reference, our implementation of affine optimal ate pairings computes pairings on curves of comparable security level, 222-bit BN curves, in 53 milliseconds, on the hardware Tegra 2 NVidia, Dual-core ARM Cortex A9, 1GHz, 1MB L2 cache, 32KB/32KB (I/D) L1 per core, DDR2-667. Note that MNT curves have embedding degree 6 instead of 12 as for BN curves, which means less security and faster extension field operations and final exponentiation.

Another paper with performance numbers for pairings on small processors for embedded devices is [25], which reports the performance of Tiny PBC. TinyPBC is an efficient implementation of pairing-based cryptography (PBC) primitives, suited for example for 8-bit processors used in wireless sensor networks (WSNs). TinyPBC is open source code based on RELIC [2], a publicly available C library which contains highly efficient code for optimized pairing computations, providing about 70 bits of security. For 8-bit processors, working on elliptic curves over binary fields GF(2^{271}) and using embedding degree 4, their implementation is able to compute pairings in about 1.9 seconds on the MICAz Mote sensor node, an ATmega128L microprocessor at 7.3828-MHz, with 4KB SRAM and 128KB ROM. On processors more comparable to the ones considered here, on the Imote2 platform (13MHz PXA271, a 32-bit ARMv5TE with 32 KB data cache and 32 KB instruction cache), their implementation showed a pairing computation in 140 milliseconds [25, Table 3]. Again these computations are not really comparable because of the 70-bit security level, different hardware and operating system, binary fields, different curve and embedding degree.

In [31] the authors report a performance of some optimal pairings on supersingular elliptic curves in characteristic 3, using the BREW emulator on 150 MHz and 225 MHz ARM9 processors. Their implementation achieves a pairing computation in 401 and 262 milliseconds respectively over the base field GF(3^{193}) on curves claimed to be at the 80-bit security level.

Acknowledgements. We thank Patrick Longa and Diego F. Aranha for valuable comments on an earlier version of this work and interesting discussions.

References

1. J. A. Akinyele, C. U. Lehmanny, M. D. Green, M. W. Pagano, Z. N. J. Peterson, and A. D. Rubin. Self-protecting electronic medical records using attribute-based encryption. Cryptology ePrint Archive, Report 2010/565, 2010. `http://eprint.iacr.org/2010/565/`.
2. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.
3. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology – EUROCRYPT 2011*, Lecture Notes in Computer Science, Tallinn, Estonia, 2011. Springer. To appear.
4. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management - part 1: General (revised). Technical report, NIST National Institute of Standards and Technology, 2007. Published as NIST Special Publication 800-57, `http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf`.
5. P. S. L. M. Barreto, H. Y. Kim, B. L., and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.
6. P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.
7. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
8. M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009.
9. 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 *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.
10. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography – TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

11. C. Costello, T. Lange, and M. Naehrig. Faster pairing computations on curves with high-degree twists. In *Public-Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 224–242. Springer, 2010.
12. S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In Claus Fieker and David R. Kohel, editors, *ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2002.
13. R. Granger and M. Scott. Faster squaring in the cyclotomic group of sixth degree extensions. In *Public-Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2010.
14. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.
15. F. Heß, N. P. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52:4595–4602, 2006.
16. Ç. K. Koç and T. Acar. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16:26–33, 1996.
17. K. Lauter, P. L. Montgomery, and M. Naehrig. An analysis of affine coordinates for pairing computation. In *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2010.
18. E. Lee, H. S. Lee, and C.-M. Park. Efficient and generalized pairing computation on Abelian varieties. *IEEE Trans. on Information Theory*, 55(4):1793–1803, 2009.
19. B. Lynn. The Pairing-Based Cryptography Library (PBC). available at http://crypto.stanford.edu/pbc/.
20. V. S. Miller. Short programs for functions on curves, 1986. Unpublished manuscript. `http://crypto.stanford.edu/miller/`.
21. V. S. Miller. The Weil pairing and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
22. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
23. M. Naehrig. *Constructive and Computational Aspects of Cryptographic Pairings*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2009.
24. M. Naehrig, R. Niederhagen, and P. Schwabe. New software speed records for cryptographic pairings. In *Progress in Cryptology – Latincrypt 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2010. Corrected version: `http://www.cryptojedi.org/papers/dclxvi-20100714.pdf`.
25. L. B. Oliveira, D. F. Aranha, C. P. L. Gouvêa, M. Scott, D. F. Câmara, J. López, and R. Dahab. TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. *Computer Communications*, 34(3):485–493, 2011.
26. G. C. C. F. Pereira, M. A. Simplício Jr, M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 2011. To appear, doi:10.1016/j.jss.2011.03.083.
27. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.
28. M. Scott, N. Benger, M. Charlemagne, Ls J. Dominguez Perez, and E. J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography – Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 78–88. Springer, 2009.

29. N. Smart (editor). ECRYPT II yearly report on algorithms and keysizes (2009-2010). Technical report, ECRYPT II – European Network of Excellence in Cryptology, EU FP7, ICT-2007-216676, 2010. Published as deliverable D.SPA.13, `http://www.ecrypt.eu.org/documents/D.SPA.13.pdf`.
30. F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.
31. M. Yoshitomi, T. Takagi, S. Kiyomoto, and T. Tanaka. Efficient implementation of the pairing on mobilephones using brew. In *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 203–214. Springer, 2007.