Affine Pairings on ARM

Tolga Acar[1], Kristin Lauter[1], Michael Naehrig[1,2,⋆], and Daniel Shumow[1]

[1] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{tolga,klauter,danshu}@microsoft.com
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands
michael@cryptojedi.org

Abstract. We report on relative performance numbers for affine and projective pairings on a dual-core Cortex A9 ARM processor. 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, 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 between 128 and 192 bits. We compare with other reported performance numbers for pairing computation on ARM CPUs.

Keywords: Optimal ate pairing, BN curves, ARM architecture.

1 Introduction

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 with degree 12 when using Barreto-Naehrig (BN) curves [3,14]. Computing in general prime fields using fast inversion techniques, a typical inversion-to-multiplication ratio can be much lower than 80 : 1.

In [9], 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, an inversion-to-multiplication ratio of 1.7 : 1 was reported for a 256-bit prime base field. Even for implementations with much faster field multiplies, using that

⋆ Acknowledges funding from the Netherlands Organisation for Scientific Research (NWO) under project PACE - Pairing Acceleration for Cryptography using Elliptic Curves.

technique, the ratio decreases dramatically as the field extension degree increases, which leads to the argument made in [9] 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 the implementation of field arithmetic in 256-bit prime fields discussed in [9] on the x86 and x86-64 platforms, this cross-over point already occured when considering extension degree 2.

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. 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. The high-level pairing implementation is very close to that reported in [9]. We use intrinsics and assembly for the Montgomery multiplication implementation. Improving the underlying field multiplication algorithm would certainly increase the degree at which one would switch from projective to affine pairings.

In our implementation, affine coordinates are the better choice for pairing computation also on the ARM processor. Other implementations presented in the literature recently have faster field multiplies that are optimized for specific processor architectures to obtain pairing speed records [12,4,2], whereas our code is not optimized for any particular architecture. Our implementation of affine pairings compares favorably with all other reported ARM pairing timings we have found in the literature (see Section 4).

2 Platform-Specific Improvements on ARM

The multiplication routines in our implementation utilize multiply, multiply-accumulate, and Montgomery multiplication [11], and the compiler is Microsoft Visual C++ on ARM (Thumb-2).

We used a Tegra 2 development platform from NVidia to obtain the benchmark figures in 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 [8], and its performance is given in Table 1 for various moduli lengths. The C implementation relies on the compiler 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), such as inversion.

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

	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

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 [8] to form base reference benchmark numbers. The assembly implementation and intrinsics only leverage the core ARM instruction set, but do not utilize SIMD and NEON instructions.

3 Implementation and Performance

Here we present the timing results of our pairing implementation on BN curves for the ARM instruction sets, for security levels of 128 bits or higher. Our pairing code can be used to compute pairings on all 16 curves recently introduced in [14]. 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 [2].

Due to space constraints, we present performance results for only three of the curves in [14], namely the curves bn254, bn446, and bn638 over prime fields of respective bit sizes 254, 446, and 638 bits. The curve bn254 roughly provides 128 bits of security and bn638 yields about 192 bits.

Our implementation uses the optimal ate pairing on BN curves. For the projective version we used the explicit formulas in [6], but we obtained better results for the affine version. It uses the tower of field extensions $\mathbb{F}_{p^{12}}/\mathbb{F}_{p^6}/\mathbb{F}_{p^2}/\mathbb{F}_p$ via the method described in [9] to realize field arithmetic in $\mathbb{F}_{p^{12}}$. Fast methods for doing inversion in the base field were described in [5, Appendix D]. The final exponentiation is done using the Frobenius action and the addition chain from [15] as well as the special squaring functions from [7].

Our implementation results are shown in Table 2. 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 [9, 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, 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 ms by multiplying them by 10^3 and 10^6, respectively (note the clock frequency of 1GHz for the ARM processor).

3.1 Summary of our Implementation Performance Results

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

1. Affine coordinates are better than projective coordinates for optimal ate pairing computation at all security levels. The trend is toward bigger differences at higher security levels. 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, for the 254-bit curve, an affine pairing takes 51 milliseconds while the projective pairing takes 55 milliseconds, whereas for the 638-bit curve, an affine pairing takes 650 milliseconds while the projective pairing takes 768 milliseconds.
2. The inversion-to-multiplication ratio is lower in larger base fields. This largely explains observation 1 above.
3. In the degree-12 extension fields, the inversion-to-multiplication ratio is close to 1.7 : 1 at all security levels. There is very little variation in that, despite big differences in ratios in the base fields.
4. The percentage of the computation time spent on the final exponentiation goes up at the higher security levels, and this is true across platforms:

Table 2. Field arithmetic timings in 254-, 446-, and 638-bit prime fields and optimal ate pairing timings on corresponding BN curves. Field timings average over 1000 operations, pairing timings average over 20 pairings.

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

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

446-bit prime field	**add** μs	**sub** μs	**M** μs	**S** μs	**I** μs	$\mathbf{R = I/M}$
$\mathbb{F}_p$	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

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
single final exponentiation		86.75

638-bit prime field	**add** μs	**sub** μs	**M** μs	**S** μs	**I** μs	$\mathbf{R = I/M}$
$\mathbb{F}_p$	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

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
single final exponentiation		413.37

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

4 Related 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) [10] 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.

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.

Working on elliptic curves over binary fields $\mathrm{GF}(2^{271})$ and using embedding degree 4 on processors somewhat comparable to the ones considered here, the Imote2 platform (13MHz PXA271, a 32-bit ARMv5TE with 32 KB data cache and 32 KB instruction cache), [13, Table 3] shows a pairing computation in 140 milliseconds. 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 [16] 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 $\mathrm{GF}(3^{193})$ 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. We also thank the anonymous referees for their helpful comments.

References

1. Akinyele, J.A., Lehmanny, C.U., Green, M.D., Pagano, M.W., Peterson, Z.N.J., Rubin, A.D.: Self-protecting electronic medical records using attribute-based encryption. Cryptology ePrint Archive, Report 2010/565 (2010), `http://eprint.iacr.org/2010/565/`
2. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011)

3. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
4. Beuchat, J.-L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010)
5. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading inversions for multiplications in elliptic curve cryptography. Des. Codes Cryptography 39(2), 189–206 (2006)
6. Costello, C., Lange, T., Naehrig, M.: Faster Pairing Computations on Curves with High-Degree Twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010)
7. Granger, R., Scott, M.: Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (2010)
8. Koç, Ç.K., Acar, T.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro 16, 26–33 (1996)
9. Lauter, K., Montgomery, P.L., Naehrig, M.: An Analysis of Affine Coordinates for Pairing Computation. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 1–20. Springer, Heidelberg (2010)
10. Lynn, B.: The Pairing-Based Cryptography Library (PBC), `http://crypto.stanford.edu/pbc/`
11. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
12. Naehrig, M., Niederhagen, R., Schwabe, P.: New Software Speed Records for Cryptographic Pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010), Corrected version `http://www.cryptojedi.org/papers/dclxvi-20100714.pdf`
13. Oliveira, L.B., Aranha, D.F., Gouvêa, C.P.L., Scott, M., Câmara, D.F., López, J., Dahab, R.: TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. Computer Communications 34(3), 485–493 (2011)
14. Pereira, G.C.C.F., Simplício, Jr., M.A., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly BN elliptic curves. Journal of Systems and Software (2011) (to appear), doi:10.1016/j.jss.2011.03.083
15. Scott, M., Benger, N., Charlemagne, M., Dominguez Perez, L.J., Kachisa, E.J.: On the Final Exponentiation for Calculating Pairings on Ordinary Elliptic Curves. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 78–88. Springer, Heidelberg (2009)
16. Yoshitomi, M., Takagi, T., Kiyomoto, S., Tanaka, T.: Efficient Implementation of the Pairing on Mobilephones Using BREW. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 203–214. Springer, Heidelberg (2008)