Implementing Cryptographic Pairings on Smartcards

Michael Scott, Neil Costigan, and Wesam Abdulwahab

School of Computer Applications
Dublin City University
Ballymun, Dublin 9, Ireland.
mike@computing.dcu.ie [**]

Abstract. Pairings on elliptic curves are fast coming of age as cryptographic primitives for deployment in new security applications, particularly in the context of implementations of Identity-Based Encryption (IBE). In this paper we describe the implementation of various pairings on a contemporary 32-bit smart-card, the Philips HiPerSmart$^{\text{TM}}$, an instantiation of the MIPS-32 based SmartMIPS$^{\text{TM}}$architecture. Three types of pairing are considered, first the standard Tate pairing on a nonsupersingular curve $E(\mathbb{F}_p)$, second the Ate pairing, also on a nonsupersingular curve $E(\mathbb{F}_p)$, and finally the η_T pairing on a supersingular curve $E(\mathbb{F}_{2^m})$. We demonstrate that pairings can be calculated as efficiently as classic cryptographic primitives on this architecture, with a calculation time of as little as 0.15 seconds.

Keywords: Elliptic curves, pairing-based cryptosystems, Fast implementations

1 Introduction

The appreciation that the Weil and Tate pairings can be used for constructive cryptographic application has caused a minor revolution in cryptography. After a flurry of research results involving new protocols based on new but plausible security assumptions, it is time for the first commercial applications to start appearing. The final, and perhaps most demanding, niche for the implementation of many cryptographic protocols is in the smart-card, a constrained computing environment in which private keys can be adequately protected. It is the purpose of this paper to demonstrate that such implementations are perfectly feasible.

In the beginning it was original research by Menezes, Okamoto and Vanstone [27], and by Frey et al. [16], which pointed out that the Weil and Tate pairings could be used for cryptanalytic purposes, undermining the security of certain types of elliptic curves, some of which had been suggested as suitable vehicles for the implementation of Elliptic Curve Cryptography (ECC). However this was followed by a prolonged hiatus before Sakai, Ohgishi and Kasahara [33] and Joux [23] independently observed that these very same condemned elliptic

[**] research supported by Enterprise Ireland grant IF/2005/053

curves had in fact useful cryptographic properties. Almost immediately Boneh and Franklin famously came up with a very simple solution to the problem of Identity-Based Encryption [10], an open problem in cryptography since the idea was first mooted by Shamir [37].

Since then there has been a veritable flood of ideas, of new protocols for identity-based encryption [10], [32], short signatures [11] and identity-based signcryption [26] to mention but a few. We do not attempt to provide a complete history here, but instead refer the interested reader to the pairing-based crypto lounge [2].

There have been two previous reported implementations of pairings on smartcards, the first in the form of an announcement by Gemplus [17], and the second in a paper by Bertoni et al. [8]. There have also been proposals for implementations, such as that by Granger et al. [18], which would require special supporting hardware. Bertoni et al. report a timing of 752 milliseconds on a 33MHz ST22 32-bit smartcard [8].

2 Pairing-friendly elliptic curves

When it comes to the selection of elliptic curves suitable for pairing-based cryptography, one is currently limited to either the supersingular curves or certain special non-supersingular curves of prime characteristic. A basic requirement is that the selected elliptic curve should have a small *embedding degree*, or *security multiplier*, denoted as k. In this paper it will be assumed that k is even.

So for cryptographic purposes a pairing-friendly elliptic curve over a finite field consists of the finite set of points (including a point at infinity) on a curve which can be described by one of

$$\begin{aligned} E(\mathbb{F}_{p^m}): &\quad y^2 = x^3 + Ax + B \\ E(\mathbb{F}_{2^m}): &\quad y^2 + y = x^3 + x + b \\ E(\mathbb{F}_{3^m}): &\quad y^2 = x^3 - x + b \end{aligned}$$

In the first case the curve can be either supersingular, with an embedding degree of $k = 2$, or nonsupersingular with $m = 1$ and any finite embedding degree [9]. In the second case the curve is supersingular and has a maximum embedding degree of $k = 4$, where $b = 0, 1$. In the third case the curve is also supersingular with a maximum embedding degree of $k = 6$, and where $b = \pm 1$.

As is common in elliptic curve cryptography over $E(\mathbb{F}_q)$, one wants to work with a group of points of prime order r, where $r \mid q + 1 - t$ the total number of points on the curve (denoted $\#E$), and where t is the trace of the Frobenius, with $|t| \leqslant 2\sqrt{q}$ (the Hasse condition) [27]. These points then form a prime order cyclic abelian group. This group size needs to be large enough to avoid various generic attacks on the elliptic curve discrete logarithm problem, and therefore at a minimum r should be 160-bits. The embedding degree k is related to this group of points on the elliptic curve by the condition that k is the smallest positive integer such that $r \mid (q^k - 1)$. A further security requirement for these elliptic curves is that $\mathbb{F}_{q^k}$, where $q = p, 2^m$ or 3^m, should be an extension field of sufficient size to prevent an index calculus attack on the discrete logarithm problem in that field. So at a minimum $k.\lg(q)$ should be 1024 bits.

So we have the interesting constraints that r can at most be about as big as q (due to the Hasse condition), with $\lg(r)$ a minimum of 160, and that $k.\lg(q)$ should then be at least 1024. One obvious feasible solution would be to choose $\lg(r) \approx 170$, $r = q + 1 - t$, and $k = 6$ so that $6.\lg(q) \approx 1024$. This explains the early popularity of curves of characteristic 3 with $k = 6$. This also has the advantage of keeping the size of the elliptic curve as small as those required for standard ECC while still attaining the minimum levels of index calculus security. However another valid and popular choice would be to use a supersingular [10] or non-supersingular curve [34] over $\mathbb{F}_p$, with $\lg(r) = 160$, $\lg(p) = 512$ and $k = 2$.

In the case of fields of low characteristic the security situation is rather unclear. As first pointed out by Coppersmith [14], the discrete logarithm problem in $\mathbb{F}_{2^m}$ is somewhat easier than it is over a prime characteristic field. According to the current record holder [38], who was able to calculate discrete logarithms for $m = 607$, it would require $m \approx 1200$ to obtain a greater level of security than 1024-bit RSA. Interpolating into the tables provided by Lenstra [24] would suggest that 1300 bits would be sufficient. Page, Smart and Vercauten [31] have observed that since the record for prime field discrete logarithms is 398 bits [25], $607/298 = 1.53$. So perhaps 50% more bits for characteristic 2 might be about right. We believe that our choice of $m = 379$ bits and hence $4m = 1516$ bits, is an appropriately conservative one.

A pairing is denoted as $e(P, Q)$, where P is taken as a point of order r, usually on $E(\mathbb{F}_q)$, and Q is a point on $E(\mathbb{F}_{q^k})$ linearly independent of P. The pairing evaluates naturally as an element of order r in $\mathbb{F}_{p^k}$. Its most important cryptographic property is its *bilinearity*

$$e(aP, bQ) = e(P, Q)^{ab}$$

If Q should be linearly dependent on P, then the pairing is degenerate and $e(P, Q) = 1$, and so for example $e(P, P) = 1$. On a supersingular curve it is usual to exploit the existence of a distortion map $\psi(.)$, which maps a point from $E(\mathbb{F}_q)$ to a linearly independent point on $E(\mathbb{F}_{q^k})$. Now both P and Q can be linearly dependent points from the same group of order r on $E(\mathbb{F}_q)$, and the distorted pairing can be calculated as $\hat{e}(P, Q) = e(P, \psi(Q))$. This pairing has the additional and sometimes useful property that $\hat{e}(P, Q) = \hat{e}(Q, P)$ which implies that $\hat{e}(P, P) \neq 1$. As our chosen smart-card has special support for multiprecision arithmetic over $\mathbb{F}_p$, and over $\mathbb{F}_{2^m}$, we will restrict our attention here to these two cases, although the field $\mathbb{F}_{3^m}$ has undoubted advantages (with its nice embedding degree $k = 6$) and has received considerable attention in the context of pairing based cryptography [18].

3 The SmartMIPS$^{\mathrm{TM}}$architecture

The SmartMIPS$^{\mathrm{TM}}$specification is of an instruction-set enhanced version of the popular RISC MIPS32 architecture [1]. The enhancements are designed to improve the performance of popular cryptographic algorithms, and are largely those envisaged and described by Großschädl and Savas [19]. It is interesting to note that this new generation of 32-bit smartcards do not employ a classic cryptographic co-processor, with its restricted and specialised set of operations, but rather use carefully selected instruction set enhancements, which when combined with the improved overall performance of the 32-bit chip, permit standard cryptographic algorithms to be executed with sufficient speed. It is also fortunately flexible enough to efficiently support new algorithms that were not envisaged when the processor was being designed.

The main idea is that an extended `ACX|HI|LO` triple of registers can be used to accumulate the partial products that arise when employing the popular Comba/Montgomery technique for multi-precision multiplication [19]. This is supported by a modified `MADDU` instruction which carries out an unsigned integer multiplication and addition to the triple register. Another important addition to the instruction set is the inclusion of a `MADDP` instruction which supports binary polynomial multiplication, and which therefore supports field multiplication over $\mathbb{F}_{2^m}$. For many years algorithms over this field have been disadvantaged with respect to the field $\mathbb{F}_p$ by the absence of such an instruction in standard processors. The addition of this instruction finally "levels the playing field", and allows the full potential of fast arithmetic over the field $\mathbb{F}_{2^m}$ to be realised.

One disadvantage of the MIPS architecture for multi-precision integer arithmetic is the lack of a carry flag, and specifically an add-with-carry `ADC` instruction. In fact it takes 5 instructions just to process one digit in a multi-precision integer addition in order to handle the carry-in and carry-out correctly, not including memory loads and stores. Note however that this is not an issue in $\mathbb{F}_{2^m}$ as in this context addition is carry-free.

When considering the performance of any processor the CPU performance equation [20] is relevant

$$\mathtt{CPU\ Time} = \frac{\mathtt{Number\ of\ Instructions} \times \mathtt{Average\ Clocks\ Per\ Instruction}}{\mathtt{Clock\ Speed\ in\ cycles\ per\ second}}$$

As instantiated by the Philips HiPerSmart$^{\mathrm{TM}}$our targeted processor is characterised by

- A five stage pipeline
- Maximum clock speed of 36MHz
- 2k Instruction cache
- 256k Flash memory
- 16k RAM memory

One of the most significant attributes from a programming point of view is the small size of the 2-way associative instruction cache. The MIPS processor as described in [20] is very much designed as a classic RISC processor, which can benefit enormously from loop-unrolling as is indeed the default behaviour of `GCC -O3` compiler optimization. However this is entirely inappropriate with such a small instruction cache. Cache misses are very expensive, and are the main reason for increased CPI (Clocks-Per-Instruction), leading to poorer performance. Ruthless loop unrolling can dramatically decrease overall instruction count, but only at the cost of much poorer CPI.

While the majority of instructions can complete one pipeline stage per clock tick, certain combinations of instructions will cause a stall in the pipeline. Most of these stalls can be identified and avoided by instruction scheduling (re-ordering). A typical cause for such a stall might be the latency of a multiply instruction like `MADDU`. However as pointed out in [19] these potential performance hits can be avoided if we use the right algorithm. While such pipeline stalls increase CPI, they do so in a fashion which is independent of the clock speed. Cache capacity misses must happen given the small size of the cache, and furthermore conflict misses are inevitable give that the cache is only 2-way associative. These cache misses exact a cost in wasted cycles which can increase dramatically with clock speed, as the access time of main memory becomes much slower than the 1-cycle access time of a cache hit.

4 Calculating the Pairing

We consider the scenario in which a smart-card is required to carry out IBE decryption, using either the IBE method of Boneh and Franklin [10] or the method of Sakai and Kasahara as described in [12]. In both cases the critical calculation to recover the plaintext is of the pairing $e(A, B)$, where A is the recipient's private and constant key, and B is a public and variable value associated with the ciphertext. For provable chosen ciphertext security an additional point multiplication is required in both cases, but this is multiplication of a constant point and so fast methods can be used. We omit a formal description of either scheme and instead refer the interested reader to the referenced material.

Much effort has been made to optimize the Tate pairing. In this work we will describe an implementation of the pairing over a prime order finite field $\mathbb{F}_p$ using the BKLS algorithm [4], as described by Scott [34], an implementation of the recently discovered Ate pairing [21], and an implementation over the small characteristic field $\mathbb{F}_{2^m}$ using the η_T pairing approach described in [3]. In all cases we will exploit the setting in which the pairing is to be calculated to maximize performance.

4.1 The BKLS pairing algorithm

All algorithms for calculating a pairing are elaborations and improvements of the basic Miller algorithm [29]. This particular variation [4] has general applicability to pairing-friendly elliptic curves $E(\mathbb{F}_p)$, either supersingular or non-supersingular. In this case we choose to use an embedding degree of 2 with a non-supersingular curve, very much following the description given in [34]. We use the same non-supersingular curve as described there, where p is a 512-bit prime number and r is the low-hamming weight Solinas prime $2^{159}+2^{17}+1$. The point Q is handled as a point on the twisted curve $E'(\mathbb{F}_p)$. Since $p = 3 \bmod 4$, elements of the extension field $\mathbb{F}_{p^2}$ such as m can be described as $m_R + im_I$, where i is the "imaginary" square root of the quadratic non-residue -1.

The helper function $g(.)$ calculates the line functions required by Miller's algorithm, and returns a value in $\mathbb{F}_{p^2}$. This function in turn requires a function *A.add(B)* which adds the elliptic curve points $A = A + B$ using standard methods, and returns the slope of the line joining A and B.

Algorithm 1 Function $g(.)$

```
INPUT: $A, B, Q$
let $A = (x_i, y_i)$, $Q = (x_Q, y_Q)$
$\lambda_i = A.add(B)$
return $y_i - \lambda_i(x_Q + x_i) - i.y_Q$
```

Algorithm 2 Computation of the Tate pairing $e(P, Q)$ on $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$ where P is a point of prime order r on $E(\mathbb{F}_p)$ and Q is a point on the twisted curve $E'(\mathbb{F}_p)$

```
INPUT: $P, Q$
$m = 1$
$A = P$
$n = r - 1$
for $i \leftarrow \lfloor \lg(r) \rfloor - 1$ downto $0$ do
    $m = m^2 \cdot g(A, A, Q)$
    if $n_i = 1$ then $m = m \cdot g(A, P, Q)$
end for
$m = \bar{m}/m$
return $V_{(p+1)/r}(m_R)$
```

After the Miller loop, the value of m needs to be subject to a final exponentiation to the power of $(p-1)(p+1)/r$. This is done in two parts – first we calculate m^{p-1} using a conjugation and a division, and then we use a Lucas sequence to raise this value to the power of $(p+1)/r$. The returned value is thus compressed to a single element in $\mathbb{F}_p$ [36].

Observe that the parameter P is in effect being multiplied by its group order r using a standard double-and-add method. The points generated as a result of this process (the x_i and y_i in the $g(.)$ function), and the associated line slopes λ_i, can be precalculated and stored if P is a constant, which it will be in the context under consideration here – in fact its the IBE private key of the card-holder.

Therefore we will precompute and store the points (x_i, y_i, λ_i) that arise in the multiplication of P by r. This results in a much simplified algorithm, where the expensive *A.add(B)* function is no longer required and curve points can be represented using simple affine coordinates.

4.2 The Ate pairing algorithm

The Ate pairing [21] is calculated faster than the Tate pairing over non-supersingular curves $E(\mathbb{F}_p)$ if $\lg(t)/\lg(r)$ is less than one, as it uses a truncated Miller loop of length $\lg(t)$ instead of the $\lg(r)$ as required above. It was once considered "natural" when implementing the Tate pairing on non-supersingular curves with embedding degree $k \geq 4$, that the first parameter P should be on the the curve defined over the base field $E(\mathbb{F}_p)$ and that the second parameter Q should be a point on a twist of the curve $E'(\mathbb{F}_{p^{k/d}})$, where d can always be 2 [6], but can be as high as 6 for certain curves, such as the BN curves [7]. The authors of [21] however observed that, rather counter-intuitively, the Ate pairing idea works best with P on $E'(\mathbb{F}_{p^{k/d}})$ and Q on $E(\mathbb{F}_p)$. In our application this swapping of roles is not an important issue, as P will be fixed and its multiples can be precalculated and stored as above. More important is the fact that we can get away with a possibly much shorter Miller loop, and still calculate a viable bilinear pairing.

To exploit the Ate pairing we first need a family of elliptic curves which have the required properties. Not only must they be pairing-friendly, but to get the full advantage we want $\lg(t) < \lg(r)$. The best that can be hoped for is that $\lg(t)/\lg(r) = 1/\deg(\Phi_k(x))$, where $\Phi_k(x)$ is the k-th cyclotomic polynomial [21]. So for a $k = 12$ curve such as that described in [5], the loop may be shortened to as little as one-quarter size. However for our targeted level of security, $k = 12$ is too big. Consider instead the family of elliptic elliptic curves defined by

$$x = (Dz^2 - 3)/4,\ t = x + 1,\ r = x^2 + 1$$
$$p = (x^3 + 13x^2 + 26x + 13)/25,\quad \#E = ((x + 13)r)/25$$

It can easily be verified that these parameters define a family of pairing-friendly elliptic curve with embedding degree $k = 4$, and with complex multiplication by $-D$. Note that $r = \Phi_4(x)$, and that $\lg(t)/\lg(r) = 0.5$ which is optimal, and so we can leverage the maximum advantage from the Ate pairing idea with a half-length loop. The actual parameters of a curve in the form $y^2 = x^3 + Ax + B$ can then be found using the method of complex multiplication [22]. By choosing random z such that p is prime and 256 bits in length, then we can easily find a value for r which has a 160-bit prime divisor. In this way the conditions that $k.\lg(p) = 1024$ and $\lg(r) = 160$ can be satisfied. For our particular curve (Appendix A), $t-1$ has a relatively low hamming weight of 31, and the discriminant $D = 259$. The full algorithm can now be given

In this case the function $g(.)$ returns a value in $\mathbb{F}_{p^4}$ and the Ate pairing returns a compressed value in $\mathbb{F}_{p^2}$. Since we choose $p = 5 \bmod 8$, -2 is a quadratic non-residue in $\mathbb{F}_p$ and $\sqrt{-2}$ is a quadratic non-residue in $\mathbb{F}_{p^2}$, elements in $\mathbb{F}_{p^4}$ can be

Algorithm 3 Function $g(.)$

```
INPUT: $A, B, Q$
let $A = (x_i, y_i)$, $Q = (x_Q, y_Q)$
$\lambda_i = A.add(B)$
return $i^2 y_Q - i(i^2 y_i/2 + \lambda_i(i^2 x_i/2 + x_Q))$
```

Algorithm 4 Computation of the Ate pairing $a(P, Q)$ on $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$ where P is a point of prime order r on the twisted curve $E'(\mathbb{F}_{p^2})$ and Q is a point on the curve $E(\mathbb{F}_p)$

```
INPUT: $P, Q$
$m = 1$
$A = P$
$n = t - 1$
for $i \leftarrow \lfloor \lg(n) \rfloor - 1$ downto $0$ do
    $m = m^2 \cdot g(A, A, Q)$
    if $n_i = 1$ then $m = m \cdot g(A, P, Q)$
end for
$m = \bar{m}/m$
return $V_{(p^2+1)/r}(m_R)$
```

represented as a pair of elements in $\mathbb{F}_{p^2}$, $m = m_R + im_I$ with $i = (-2)^{1/4}$ [30]. In the function $g(.)$, points on the twisted curve $E'(\mathbb{F}_{p^2})$ must first be converted to coordinates on $E(\mathbb{F}_{p^4})$, which explains the apparent complexity of this function. However given that these can all be precalculated, this is not an issue in practise.

4.3 The BGOhES pairing algorithm

On the supersingular curve

$$E(\mathbb{F}_{2^m}) : y^2 + y = x^3 + x + 1$$

where m is prime and $m = 3 \bmod 8$, the number of points is $2^m + 2^{(m+1)/2} + 1$ [3]. For our choice of $m = 379$, this value is a prime. A suitable irreducible polynomial for the field $\mathbb{F}_{2^{379}}$ is $x^{379} + x^{315} + x^{301} + x^{287} + 1$. This supersingular curve has an embedding degree of $k = 4$. To represent the quartic extension field $\mathbb{F}_{2^{4m}}$, we use the irreducible polynomial $X^4 + X + 1$.

Recall that in a characteristic 2 field with a polynomial basis, field squarings are of linear complexity. Furthermore on this supersingular curve, point doublings require only cheap field squarings (using affine coordinates - see Appendix B). Therefore we can anticipate that calculations on this curve will be very efficient.

A distortion map for this particular supersingular curve is $\psi(x, y) = (x + s^2.y + sx + t)$, where $t = X$ and $s = X + X^2$ [27]. A major insight from [3] is that the Tate pairing can be calculated from the more primitive η_T pairing, which requires a half-length loop compared to the Duursma-Lee method [15], with considerable computational savings. The algorithm as described benefits from unrolling the loops times 2, in which case each iteration costs just seven base field multiplications. The final exponentiation looks a little complex, but in fact can be accomplished with only 4 extension field multiplications, $(m+1)/2$ cheap extension field squarings and some nearly-free Frobenius operations.

Algorithm 5 Computation of $\hat{e}(P,Q)$ on $E(\mathbb{F}_{2^m}) : y^2 + y = x^3 + x + b : m \equiv 3 \pmod 8$ case

```
INPUT: $P, Q$
OUTPUT: $\hat{e}(P,Q)$
let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$
$u \leftarrow x_P + 1$
$f \leftarrow u \cdot (x_P + x_Q + 1) + y_P + y_Q + b + 1 + (u + x_Q)s + t$
for $i \leftarrow 1$ to $(m+1)/2$ do
    $u \leftarrow x_P$, $x_P \leftarrow \sqrt{x_P}$,  $y_P \leftarrow \sqrt{y_P}$
    $g \leftarrow u \cdot (x_P + x_Q) + y_P + y_Q + x_P + (u + x_Q)s + t$
    $f \leftarrow f \cdot g$
    $x_Q \leftarrow x_Q^2$,  $y_Q \leftarrow y_Q^2$
end for
return $f^{(2^{2m}-1)(2^m-2^{(m+1)/2)}+1)(2^{(m+1)/2}+1)}$
```

Since P will be fixed, all the square roots in this algorithm can be precalculated and stored with some savings. With this modification, our implementation is largely the same as that described in [3].

5 Implementation Issues

Our implementation makes use of the MIRACL multiprecision library [35]. The current version (5.01) of this library is fortunately friendly towards those attempting implementations in a constrained environment, like a smartcard. Typically a big number library forces allocation of memory for big variables from the heap. In a constrained environment however a heap is a luxury that often cannot be afforded. Therefore allocation from the stack is appropriate, and is supported. Header file definitions were used to cut down the amount of code required. This was supplemented with some manual pruning of unwanted functionality.

For optimal performance MIRACL includes a mechanism for generating unrolled Comba code for modular multiplication, squaring, and reduction with respect to a fixed modulus, including specific support for the SmartMIPS$^{\text{TM}}$processor. However as pointed out above, fully unrolled code is inappropriate in an environment where the instruction cache is very small. Therefore we found it necessary to take the automatically generated (and correct) code, and to roll it up again into tight loops, much as described in [19]. Extra manually written inline assembly code was provided to support fast squaring in $\mathbb{F}_{2^m}$ using the `MADDP` instruction, and short unrolled assembly language code was provided for fast field addition in $\mathbb{F}_{2^m}$. With these exceptions, the rest of the code was written in standard C.

6 Results

We present our results in a series of tables. As well as the timings for the pairings, we include timings for point multiplications and pairing exponentiations, as these as often relevant to pairing based protocols. For each of the three implementations we assume projective coordinates are used for point multiplication, as field inversions which are required for affine point addition are very slow on the smartcard. The point multiplication is taken over the base field $E(\mathbb{F}_q)$ using a random 160-bit multiplier. Field exponentiation is of the pairing value to a random 160-bit exponent. For the $E(\mathbb{F}_p)$ cases we use Lucas exponentiation of the compressed pairing, while for the $E(\mathbb{F}_{2^{379}})$ case we use standard windowed exponentiation, as we believe these to be the fastest methods in each case.

Our hardware emulator is only cycle accurate up to 20.57MHz, and so we estimate the timings for the maximum supported speed of 36MHz, using linear interpolation for CPI. For comparision purposes we include figures for 1024-bit RSA decryption (using the Chinese Remainder Theorem), and timings on a standard PC.

Table 1. Instructions required (% icache misses) - Philips HiPerSmart$^{\text{TM}}$

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	3705344 (10.9%)	7753341 (7.3%)	8156645 (15.8%)
Point Mult.	2589569 (9.6%)	7418768 (6.1%)	2663217 (17.5%)
Field exp.	1551117 (11.4%)	1364124 (7.2%)	1614016 (15.7%)
RSA decryption	4372772 (3.4%)		

Table 2. Clock cycles required/CPI/time in seconds @ 9 MHz

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	4311454/1.16/0.48	9104450/1.17/1.01	10860479/1.33/1.21
Point Mult.	3118344/1.20/0.35	8529176/1.15/0.95	3739596/1.40/0.42
Field exp.	1924596/1.24/0.21	1593313/1.17/0.18	2122221/1.31/0.24
RSA decryption	4740271/1.08/0.53		

Table 3. Clock cycles required/CPI/time in seconds @ 20.57 MHz

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	4590712/1.24/0.22	9755457/1.26/0.47	12207440/1.50/0.59
Point Mult.	3391127/1.31/0.16	9049457/1.22/0.44	4278858/1.61/0.21
Field exp.	2118707/1.37/0.10	1705365/1.25/0.08	2374885/1.47/0.12
RSA decryption	4880323/1.12/0.24		

Table 4. Clock cycles required/CPI/time in seconds @ 36MHz (estimated)

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	4891054/1.32/0.14	10467010/1.35/0.29	13621597/1.67/0.38
Point Mult.	3677188/1.42/0.10	9570210/1.29/0.27	4847055/1.82/0.13
Field exp.	2326675/1.50/0.06	1814285/1.33/0.05	2630846/1.63/0.07
RSA decryption	5072415/1.16/0.14		

Table 5. Timings in milliseconds on 3GHz Pentium IV

	$E(\mathbb{F}_{2^{379}})$ η_T pairing	$E(\mathbb{F}_p)$ Tate pairing	$E(\mathbb{F}_p)$ Ate pairing
Pairing	3.88	2.97	3.16
Point Mult.	1.82	3.08	1.17
Field exp.	1.14	0.54	0.62
RSA decryption	1.92		

The most surprising and significant observation to be made is that the η_T pairing can be calculated just about as quickly as a standard RSA decryption, for approximately the same level of security. As expected CPI goes up as clock speed increases, as we are punished more heavily for cache misses. This has less impact on algorithms that spend more time in tight loops, and hence disadvantages the η_T and Ate pairings with their more elaborate structures and higher extension fields. Note that RSA, due to its simplicity, suffers least from increasing CPI.

7 Does pairing delegation make sense?

The idea of securely delegating the calculation of a pairing to the terminal was considered in [13]. This was motivated by the assumption that the pairing calculation might be too resource consuming to be carried out on a smartcard. Here we present a slightly modified version of the method described in section 6.2 of [13]. In the context of IBE decryption the calculation of $e(A, B)$ involves a constant and private A (in fact the IBE private key), and a public B (in fact part of the ciphertext). It is assumed that the smartcard also has stored a random secret point Q and the value of $e(A, Q)$.

- The card generates random x,y,and z, and queries the following pairings to the terminal.

$$\alpha_1 = e(x^{-1}A, B), \quad \alpha_2 = e(yA, z(B + Q))$$

- The card computes

$$e_{AB} = \alpha_1^x$$

- The card checks that

$$\alpha_1^r = 1, \quad \alpha_1^{xyz \bmod r} = \alpha_2 / e(A, Q)^{yz \bmod r}$$

If successful the protocol outputs $e(A, B) = e_{AB}$. Observe that two of the point multiplications are of the fixed point A. These may be calculated offline, or at the very least can benefit from fast methods for fixed-point multiplication. Also $e(A, Q)^{yz}$ can be precalculated, or calculated using fixed-base exponentiation [28]. So the major online cost will be of 3 exponentiations and one point multiplication. From the tables above it is clear that the η_T pairing is so fast that delegation is unlikely to be beneficial. The standard Tate pairing ($k = 2$) implementation suffers badly as point multiplication is over a large 512-bit field. However in the case of our Ate pairing implementation, with its smaller 256-bit field size, it appears that delegation might be beneficial.

8 Conclusions

We have demonstrated for the first time that cryptographic pairings can be implemented just as quickly as classic public key cryptographic operations on a standard smartcard, hence clearing the way for their more widespread adoption. The issue of pairing delegation has been investigated, and it appears that despite the efficiency of our implementations, it may be advantageous in certain circumstances.

References

1. http://www.mips.com/content/Products/Architecture/SmartMIPSASE/ProductCatalog/P_SmartMIPSASE/productBrief.
2. P. S. L. M. Barreto. The pairing-based crypto lounge. http://paginas.terra.com.br/informatica/paulobarreto/pblounge.html.
3. P. S. L. M. Barreto, S. Galbraith, C. O'hEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. http://eprint.iacr.org/2004/375.
4. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer-Verlag, 2002.
5. P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks – SCN'2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 263–273. Springer-Verlag, 2002.
6. Paulo S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography – SAC'2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25, Ottawa, Canada, 2003. Springer-Verlag.
7. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. Cryptology ePrint Archive, Report 2005/133, 2005. http://eprint.iacr.org/2005/133.
8. G. M. Bertoni, L. Chen, P. Fragneto, K. A. Harrison, and G. Pelosi. Computing tate pairing on smartcards, 2005. http://www.st.com/stonline/products/families/smartcard/ches2005_v4.pdf.
9. I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography, Volume 2*. Cambridge University Press, 2005.
10. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
11. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2002.
12. L. Chen and Zhaohui Cheng. Security proof of Sakai-Kasahara's identity-based encryption scheme, 2005. http://eprint.iacr.org/2005/226.
13. B. Chevallier-Mames, J-S. Coron, N. McCullagh, D. Naccache, and M. Scott. Secure delegation of elliptic-curve pairing, 2005. http://eprint.iacr.org/2005/150.
14. D. Coppersmith. Fast evaluation of logarithms in fields of characteristics two. In *IEEE Transactions on Information Theory*, volume 30, pages 587–594, 1984.

15. I. Duursma and H.-S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology – Asiacrypt'2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer-Verlag, 2003.
16. G. Frey, M. Müller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.
17. Gemplus. ID based Cryptography and Smartcards, 2005. `http://www.gemplus.com/smart/rd/publications/pdf/Joy05iden.pdf`.
18. R. Granger, D. Page, and M. Stam. Hardware and software normal basis arithmetic for pairing based cryptography in characteristic three. Cryptology ePrint Archive, Report 2004/157, 2004. `http://eprint.iacr.org/2004/157`.
19. Johann Großschädl and Erkay Savas. Instruction set extensions for fast arithmetic in finite fields GF(p) and GF(2^{m}). In *CHES*, pages 133–147, 2004.
20. J. Hennessy and D. Patterson. *Computer Architecture - a Qualitative Approach (third edition)*. Morgan Kaufmann, 2003.
21. F. Hess, N. Smart, and F. Vercauteren. The eta pairing revisited. Cryptology ePrint Archive, Report 2006/110, 2006. `http://eprint.iacr.org/2006/110`.
22. IEEE Computer Society, New York, USA. *IEEE Standard Specifications for Public-Key Cryptography – IEEE Std 1363:2000*, 2000.
23. A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Algorithm Number Theory Symposium – ANTS IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2000.
24. A. K. Lenstra. Unbelievable security. Matching AES security using public key systems. In *Advances in Cryptology – Asiacrypt 2001*, volume 2248, pages 67–86. Springer-Verlag, 2001.
25. R. Lercier. Discrete logarithms in GF(p). Posting to NMBRTHRY List, 2001.
26. N. McCullagh and P. S. .L. M. Barreto. Efficient and forward-secure identity-based signcryption. Cryptology ePrint Archive, Report 2004/117, 2004. `http://eprint.iacr.org/2004/117`.
27. A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
28. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton, Florida, 1996. URL: `http://cacr.math.uwaterloo.ca/hac`.
29. V. Miller. Short programs for functions on curves. unpublished manuscript, 1986. `http://crypto.stanford.edu/miller/miller.pdf`.
30. Y. Nogami and Y. Morikawa. A fast implementation of elliptic curve cryptosystem with prime order defined over f_{p^8}, 1998. `http://www.trans.cne.okayama-u.ac.jp/nogami-group/papers/kiyou(2).pdf`.
31. D. Page, N. P. Smart, and F. Vercauteren. A comparison of MNT curves and supersingular curves. Cryptology ePrint Archive, 2004. `http://eprint.iacr.org/2004/165`.
32. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptography ePrint Archive, Report 2003/054, 2003. `http://eprint.iacr.org/2003/054`.
33. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.
34. M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
35. M. Scott, 2006. `http://ftp.computing.dcu.ie/pub/crypto/miracl.zip`.

36. M. Scott and P. Barreto. Compressed pairings. In *Advances in Cryptology – Crypto' 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, 2004. Also available from `http://eprint.iacr.org/2004/032/`.

37. A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of Crypto 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer-Verlag, 1984.

38. E. Thomé. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 107–124. Springer-Verlag, 2001.

A The Ate pairing non-supersingular $k = 4$ curve

The curve over $\mathbb{F}_p$ is described in the Weierstrass form as

$$E : y^2 = x^3 - 3x + B$$

The curve has $p + 1 - t$ points on it, which is a number divisible by a large prime r. For our chosen curve

$$\begin{aligned}
B &= 47757104637654076446719767983734023399018465557799879638758483211193582773900 \\
p &= 73190453176371233031922874717260488242507261313747586254294463297030724930453 \\
r &= 7039968169563831716203361508047454068025613140101 \\
t &= 122310802304306476153797569
\end{aligned}$$

B Point addition on the supersingular curve

For the supersingular curves over $\mathbb{F}_{2^m}$ of the form $y^2 + y = x^3 + x + b$ considered in this paper, formulae for affine and projective point doubling and addition are given here. Using affine coordinates the points (x_1, y_1) and (x_2, y_2) are added to get (x_3, y_3). Using Jacobian coordinates, where $(x, y) = (X/Z^2, Y/Z^3)$, we add (X_1, Y_1, Z_1) to (X_2, Y_2, Z_2) to get (X_3, Y_3, Z_3), without any inversions. Recall that field squarings are much faster than field multiplications, which in turn are much faster than field inversions. We ignore the cost of field additions. In all cases we attempt to minimize the number of intermediate variables. In the projective case some savings can be made if it is known that $Z = 1$.

B.1 Affine coordinates - point doubling

Note that Affine point doubling is inversion and multiplication free.

$$
\begin{aligned}
x_3 &= x_1^2 \\
x_3 &= x_3^2 \\
y_3 &= y_1^2 \\
y_3 &= y_3^2 \\
y_3 &= y_3 + x_3 \\
x_3 &= x_3 + 1
\end{aligned}
$$

This requires just 4 squarings.

B.2 Affine coordinates - point addition

$$
\begin{aligned}
t_1 &= y_1 + y_2 \\
t_2 &= x_1 + x_2 \\
t_1 &= t_1/t_2 \\
t_2 &= t_1^2 + x_1 + x_2 \\
y_3 &= y_1 + 1 + t_1(t_2 + x_1) \\
x_3 &= t_2
\end{aligned}
$$

This requires 1 inversion, 2 multiplications and 1 squaring.

B.3 Projective coordinates - point doubling

$$
\begin{aligned}
X_3 &= X_1^2 \\
X_3 &= X_3^2 \\
Y_3 &= Y_1^2 \\
Y_3 &= Y_3^2 \\
Z_3 &= Z_1^2 \\
Z_3 &= Z_3^2 \\
T_1 &= Z_3.X_3 \\
Y_3 &= Y_3 + T_1 \\
T_1 &= Z_3^2 \\
X_3 &= X_3 + T_1
\end{aligned}
$$

This requires 1 multiplication and 7 squarings

B.4 Projective coordinates - point addition

$$
\begin{aligned}
T_1 &= Z_1^2 \\
T_2 &= T_1.Z_1 \\
T_3 &= Z_2^2 \\
T_4 &= T_3.Z_2 \\
T_5 &= T_4.T_2 \\
T_6 &= Z_1.Z_2 \\
T_1 &= T_1.X_2 \\
T_3 &= T_3.X_1 + T_1 \\
T_4 &= T_4.Y_1 \\
T_2 &= T_2.Y_2 + T_4 \\
Z_3 &= T_6.T_3 \\
T_6 &= T_3^2 \\
T_3 &= T_6.T_3 \\
T_6 &= T_2.T_6 \\
T_7 &= T_2^2 \\
X_3 &= T_7 + T_3 \\
T_2 &= T_2.T_7 \\
T_6 &= T_1.T_6 \\
T_3 &= (T_5 + T_4).T_3 \\
Y_3 &= T_2 + T_6 + T_3
\end{aligned}
$$

This requires 14 multiplications and 4 squarings.