Computing the Tate Pairing*

Michael Scott

School of Computing,
Dublin City University,
Ballymun, Dublin 9, Ireland
mike@computing.dcu.ie

Abstract. We describe, in detail sufficient for easy implementation, a fast method for calculation of the Tate pairing, as required for pairing-based cryptographic protocols. We point out various optimisations and tricks, and compare timings of a pairing-based Identity Based Encryption scheme with an optimised RSA implementation.

Keywords: Elliptic curves, pairing-based cryptosystems.

1 Introduction

In the fast growing world of pairing-based cryptography (for background see [1]) there are many protocols, many pairings (Tate, Weil, modified Weil etc.) and many choices for the *embedding degree* k, as well as a choice of super- or non-supersingular curves over fields of large or small characteristic. The range of protocols is impressive, many with novel properties [6, 7, 28]. For a recent review see [11]. However so far there are not many reported implementations of the fast algorithms for pairings that have been developed in [2, 4, 13].

Here for the sake of being concrete we will focus exclusively on the Tate Pairing on non-supersingular curves over a field of large prime characteristic. We will also focus on the case $k = 2$ for the following reasons:

- It simplifies the description
- Choosing $k = 2$ makes it easy to pick a group order of the lowest possible Hamming weight which is very efficient.
- Choosing $k = 2$ allows us to implement the Tate pairing based protocols using only $E(\mathbb{F}_p)$ elliptic curves as supported by many cryptographic libraries.
- $k = 2$ permits the important *denominator elimination* optimisation [2].
- It allows for easy times-2 compression of the Tate pairing value [25].
- In protocols elliptic curve point multiplication can often be replaced with faster exponentiation using the identity $e_r(wP, Q) = e_r(P, Q)^w$.
- Elliptic curves suitable for pairing based cryptosystems are, by design, in flagrant breach of the MOV condition, as required for "ordinary" elliptic curves [20]. The ECC community recently got a scare when Semaev [27] suggested that a new index calculus type attack on normal elliptic curves may

* Research supported by Enterprise Ireland grant IF/2002/0312/N.

be possible. In the context considered here an index calculus attack is already possible [20], and therefore we need not be too concerned. Nevertheless a choice of a small value of k reduces the impact of any such new attack.
– For a given level of security it is our experience that $k = 2$ is fastest.
– In many protocols it is required to do a point multiplication prior to application of the Tate pairing. Using $k = 2$ this implies a point multiplication only on an $E(\mathbb{F}_p)$ curve, rather than a point multiplication on a curve defined over a higher extension field, which would be computationally more expensive.
– $\mathbb{F}_{p^2}$ arithmetic is particularly easy to implement. This is sometimes called the *quadratic extension field.* If it is assumed in this paper that the prime modulus p is 3 mod 4, then an element in $\mathbb{F}_{p^2}$ can be considered as a "complex number", $a+bi$, a, $b \in \mathbb{F}_p$, where i is $\sqrt{-1}$. Note that -1 is always a quadratic non-residue for a 3 mod 4 prime. There are exactly $(p-1)(p+1)$ elements in the field $\mathbb{F}_{p^2}$. Note that $(a+ib)^p = (a-ib)$, where $a-bi$ is the *conjugate* of $a+ib$. Also an element $\in \mathbb{F}_{p^2}$ can be squared (or multiplied) using just two (or three) $\mathbb{F}_p$ modular multiplications using the identity $(a+bi)^2 = (a+b)(a-b)+2abi$ and Karatsuba's method respectively. Sometimes we use the notation $[a, b]$ to denote the $\mathbb{F}_{p^2}$ number $a+bi$.
– Using $k = 2$ the time-critical function is 512-bit modular multiplication. This is the same operation as required for 1024-bit RSA decryption using the Chinese Remainder theorem and therefore it is likely to be supported by hardware accelerators and co-processors. Highly optimized code for this common operation may be already supported by cryptographic software libraries.

We do concede that $k = 2$ may not be optimal in some settings such as a short signature scheme, like for example the BLS scheme [7].

In this paper we draw heavily from the theoretical results described by Barreto et al. [4] and [2]. Our results improve a little on those described there using ideas from [25].

2 The Curve

There are many ways proposed to find non-supersingular curves of low embedding degree suitable for pairing-based protocols. See for example [3, 5, 8, 10, 21] and [26]. Using these methods the existance of a suitable elliptic curve is first determined, and then the actual parameters of the curve are found using the method of Complex Multiplication as described in [14] and implemented in [23].

The particular curve we will use (found using the "folklore" method described by Galbraith in Chapter 9 of [5]), is described in the Weierstrass form

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

with $B \in \mathbb{F}_p$. If $x, y \in \mathbb{F}_p$, the curve has $\#E(\mathbb{F}_p)$ points on it, where $\#E(\mathbb{F}_p) = p+1-t$ and t is the *trace of the Frobenius* [20]. If $x, y \in \mathbb{F}_{p^2}$, it has $\#E(\mathbb{F}_{p^2}) = (p+1-t)(p+1+t)$ points. A related *twisted* curve $E'(\mathbb{F}_p)$ is

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

If $x, y \in \mathbb{F}_p$, this curve will have $p + 1 + t$ points on it. For our chosen curve:-

$$\begin{aligned}
B &= 68061659825436829401585865346840003227868864824516292145748121298848783826612 \\
&\quad 1706017410197802303768179517423581649948460652150151762287285211627769549950 \\
p &= 11711338024714009669995700965425239711927177698599625717955894184681899877662 \\
&\quad 8279774412183568462075735094723078737566623007544372323984528307791007809703 03 \\
\#E &= 11711338024714009669995700965425239711927177698599625717955894184681899877662 \\
&\quad 6115395699969459692937084044003442082738128503993513036518753780985035340756 38 \\
t &= 2164378712214108769138651050719636654828494503550859287465774526805972468946 66
\end{aligned}$$

Note that p is 512-bits long and is congruent to 3 mod 4. $\#E(\mathbb{F}_p)$ has (by design) a 160-bit prime factor r of low Hamming weight, where $r = 2^{159} + 2^{17} + 1$, a Solinas prime. The group of points on $E(\mathbb{F}_{p^2})$ of order r exhibit the required $k = 2$ embedding degree behaviour [24] – observe that $r \mid p + 1$ – and therefore form a suitable setting for calculation of the Tate pairing. Note that the discrete logarithm problem over $\mathbb{F}_{p^2}$ where p is 512 bits is regarded as being approximately as hard as a discrete logarithm problem over $\mathbb{F}_p$ where p is 1024 bits. Therefore this curve satisfies contemporary security requirements. Roughly speaking it will be as difficult to "break" as 1024–bit RSA.

Note that although we have chosen to use a non-supersingular curve here, the method described will also work without modification for a large prime characteristic $k = 2$ supersingular curve, as originally suggested by Boneh & Franklin [6].

3 The Tate Pairing Algorithm

The notation for the Tate pairing is $e_r(P, Q)$, where P and Q are points on the elliptic curve $E(\mathbb{F}_{p^k})$, with P a point of order r. Q may also be of order r, but not necessarily (see below). The Tate pairing evaluates as an element in $\mathbb{F}_{p^k}$.

The algorithm itself consists of two parts; an application of Miller's algorithm, followed by a final exponentiation. For an easy-to-read introduction to the Tate pairing, Miller's algorithm, and its derivation from divisor theory see [18]. See also [29] for a nice discussion on pairings.

Now we describe the optimised BKLS algorithm for the particular case of $k = 2$, with denominator elimination applied [2]. Basically (and very loosely) Miller's algorithm first carries out an implicit multiplication of P by r, using the standard line-and-tangent double-and-add algorithm for elliptic curve point multiplication [20]. The result of this multiplication will (of course) be O, the point at infinity, as P is of order r. If a line or tangent should ever pass through Q then the pairing algorithm will fail, but it can be arranged that this won't ever happen (see below). At each step in the process an $\mathbb{F}_{p^2}$ value is calculated from a distance relationship between the current line or tangent and the point Q. This consists of a numerator (derived from the line or tangent associated with the addition or doubling of a point), and a denominator (derived from a vertical line through the destination point).

This value is multiplicatively accumulated, and its final value is the output of Miller's algorithm.

However this value, an element in $\mathbb{F}_{p^2}$ may not be of order r. To ensure a unique answer of order r the output of Miller's algorithm must be exponentiated to the power of $(p-1)(p+1)/r$. This is then the final result of the Tate pairing.

Observe that this final exponentiation itself can be considered in two parts – an exponentiation to the power of $(p-1)$ followed by an exponentiation to the power of $(p+1)/r$. The first exponentiation to $(p-1)$ ensures that any $\mathbb{F}_p$ component of the output of Miller's algorithm is reduced to 1 (by Fermat's little theorem), and hence can be ignored. An important observation in [2] is that under certain circumstances (that will pertain here) the "denominator" component is always in $\mathbb{F}_p$ and hence can be discarded. So we only deal here with the numerator.

In contrast to P, Q is not involved in any point addition or multiplication. Only the coordinates of Q are actually required.

P and Q could be chosen as linearly-independent points from $E(\mathbb{F}_{p^2})[r]$, and represented in standard (x, y) affine coordinates with $x, y \in \mathbb{F}_{p^2}$. However from an implementation view-point the requirement for point multiplication on $E(\mathbb{F}_{p^2})$ is a little difficult, as most existing cryptographic libraries would not support this. However this problem can be neatly side-stepped by modifying the algorithm to accept P as a point on $E(\mathbb{F}_p)$. If it helps, think of P as an point on $E(\mathbb{F}_{p^2})$ whose coordinates have an imaginary part of zero. Solinas [29] calls this rather nicely *Miller light.* It solves another problem - if Q is truly on $E(\mathbb{F}_{p^2})$ then P and Q can never be linearly dependant, and no line generated in the implicit multiplication of P will ever pass through Q. This *Miller light* algorithm will obviously be much faster.

As already stated Q need not be of order r. In fact it can be any point on the curve. Whatever its value it will be a member of a *coset* which does include exactly one point of order r (or the point at infinity). All points in the same coset are equivalent as far as the Tate pairing is concerned [24]. Our only problem is to ensure that Q is not in a coset associated with a point of order r, which is in the same subgroup as P (otherwise P and Q will be linearly dependent). This condition is met by points of the form $Q([a, 0], [0, d])$. It is not difficult to see that if there are $p + 1 - t$ points of the form $Q([a, 0], [c, 0])$ on $E(\mathbb{F}_p)$ then there will be $p + 1 + t$ points of the form $Q([a, 0], [0, d])$ on $E(\mathbb{F}_{p^2})$. (Simply substitute all possible $a < p$ for x in the curve equation, not forgetting the point at infinity. If the right hand side is a quadratic residue the points are $Q([a, 0], [\pm c, 0])$, otherwise they are $Q([a, 0], [0, \pm d])$). There will always be a subgroup of order r consisting of points of this latter form. Such points stay in this form under point multiplication. Furthermore there is a simple relationship between such points on $E(\mathbb{F}_{p^2})$ and points on the twisted curve $E'(\mathbb{F}_p)$. In fact for every point $Q([-a, 0], [0, d])$ on the curve defined over the quadratic extension field $\mathbb{F}_{p^2}$, there is a point $Q(a, d)$ on the twisted curve defined over $\mathbb{F}_p$ [3]. This *isomorphism* is very convenient, as it means that Q can be treated as a point on $E'(\mathbb{F}_p)$

In many protocols, for example [6], there is a need to hash and map an arbitrary string to a curve point of order r. For a general point on $E(\mathbb{F}_{p^2})$ this requires point multiplication by the large co-factor $(p+1-t)(p+1+t)/r$. However if the string is hashed instead to a point on the twisted curve, then the cofactor is the much smaller $(p+1+t)/r$.

Next we describe the BKLS algorithm in detail. If the current point in the implicit multiplication is $A(x,y)$, and if the next point doubling or addition generates a line of slope λ, then we will require the function

```
$f(A, \lambda, Q)$
1. $x, y \leftarrow A$
2. $a, d \leftarrow Q$
3. **return** $y - \lambda(a + x) - di$
```

which calculates and returns an $\mathbb{F}_{p^2}$ value. We also need a function which adds two points (or doubles a point) on the elliptic curve $E(\mathbb{F}_p)$. Assume therefore the existance of a function $A.add(B)$ which adds B to A and returns the line slope λ. Next we need a function to calculate the contribution of the most recent point addition/doubling to the $\mathbb{F}_{p^2}$ *Miller variable m*.

```
$g(A, B, Q)$
1. $T = A$
2. $\lambda = A.add(B)$
3. **return** $f(T, \lambda, Q)$
```

Now we are ready for the full BKLS algorithm.

Capital letters indicate an elliptic curve point. The variables r, n, p, and i are all simple integers. The notation n_i refers to the i-th bit of n. The variable λ and the coordinates of A, P and Q are all in $\mathbb{F}_p$. Only the Miller variable m requires support for elementary $\mathbb{F}_{p^2}$ arithmetic. The standard elliptic curve point addition/doubling formula involves the calculation of the line slope so there is no extra work involved in obtaining λ [20]. The conjugate of m is denoted $\bar{m}$.

```
**BKLS**$(r, p, P, Q)$
1. $m = 1$
2. $A = P$
3. $n = r - 1$
4. **for** $i \leftarrow \lfloor \lg(r) \rfloor - 2$ **downto** 0 **do**
5.     $m = m^2 \cdot g(A, A, Q)$
6.     **if** $n_i = 1$ **then** $m = m \cdot g(A, P, Q)$
7. **end for**
8. $m = \bar{m}/m$
9. $m = m^{(p+1)/r}$
10. **return** $m$
```

As noted by Duursma and Lee [12] the very last step of Miller's algorithm can be ignored, as it does not contribute to the pairing value, so we do not need to process the last bit of r. As a result, with our choice of r as a Solinas prime with a Hamming weight of 3, the condition $n_i = 1$ is met precisely once.

As pointed out in [25], the value of m after line 8 of this algorithm is already *unitary.* Unitary values like $m = u + vi$ have the following useful properties:

– $u^2 + v^2 = 1$
– $(u + vi)^{-1} = (u - vi)$
– $(u + vi)^n = V_n(2u)/2 + U_n(2u)vi$

where V_n and U_n are the well-known Lucas sequences. The first property tells us that given u, then v can be uniquely determined from its sign. If it is clear from the protocol that this sign is not important, then v can be dropped.

This compressed pairing $\varepsilon_r(P, Q)$ as defined in [25] can be calculated in line 9 of the algorithm as $V_{(p+1)/r}(2u)$ and this single $\mathbb{F}_p$ value can be returned in line 10. Fortunately there is a well-known fast and efficient laddering algorithm which calculates $V_n(\cdot)$, and requires very little memory [25]. Note that unitary values remain unitary under any subsequent exponentiation. Therefore any subsequent exponentiation of the compressed pairing value can also be computed using the fast Lucas $V_n(\cdot)$ function.

3.1 Resistance to Simple Power Attack Analysis

We can make the following generalisations about pairing-based cryptographic protocols based on the BKLS tate pairing algorithm described above

– r and p are fixed and public
– secrets may be introduced as unknown points P and/or Q
– secrets may be introduced as exponents of pairings.

In the light of these observations it is of interest to consider the resistance of pairing-based protocols to so-called SPA attacks [17]. Observe first that the path taken through the code in the course of the execution of the BKLS algorithm is independent of any possible secrets. Furthermore, using the compressed pairing, any subsequent exponentiation of a pairing value can be carried out using a laddering algorithm, which is also known to be resistant to SPA [16]. Therefore one might with reasonable confidence expect that the power consumption profile of (and execution time for) such protocols will be constant and independent of any secret values.

To increase resistance to more sophisticated SPA and DPA attacks we suggest the following simple counter-measures

– Exploit bilinearity and calculate $e(P, Q) = e(sP, tQ)^{1/st}$, where s and t are random variables.
– Multiply the Miller variable at any time prior to line 8 in the BKLS algorithm by a random element of F_p. This does not effect the result, as all such contributions are eliminated by the final exponentiation.

4 Optimisations

In this section we will describe various speed-ups and tricks.

4.1 Projective Coordinates

It has been the experience of many that elliptic curve point addition and doubling over $E(\mathbb{F}_p)$ is faster if the points are represented in (x, y, z) *projective* coordinates rather than in (x, y) *affine* coordinates [14]. This is because affine point addition or point doubling requires an expensive modular inversion mod p. This operation is hard to optimise. If a single pairing value is being calculated and if pre-computation is not possible, then our experience is that projective coordinates, which do not require a modular inversion, are faster in practice. In the context of pairings the use of projective coordinates is also recommended by Izu and Takagi [15].

The modification to the algorithm above is simple - we just need a projective version of the $A.add(B)$ which will return the line slope as a rational $\lambda = \lambda_n/\lambda_d$, and a new $f(\cdot)$ function. It is assumed that initially P and Q are presented in affine coordinates (with z=1).

```
$f(A, \lambda, Q)$
1.  $x, y, z \leftarrow A$
2.  $a, d \leftarrow Q$
3.  $\lambda_n, \lambda_d \leftarrow \lambda$
4.  **return** $y\lambda_d - \lambda_n(az^3 + xz) - dz^3\lambda_d \cdot i$
```

4.2 Precomputation

For the calculation of $e_r(P, Q)$ in the context of a particular protocol, the parameter P may be fixed. It may for example be an individual's fixed private key, or it may be a system global. Whatever the reason, if P is fixed then it makes sense to calculate the points and slopes that arise in the implicit multiplication of rP just once, and store them, as suggested in [2] and [13]. In this case it makes sense to revert to affine coordinates, as no point addition or doubling will be needed.

The modification to the BKLS algorithm is straightforward. Assume the prior storage of $\{x_j, y_j, \lambda_j\}$ for each point A_j that arises in the point multiplication. Then in line 2 of the $g(\cdot)$ function instead of calculating the next point and slope, simply extract them from this precomputed store. The size of the store will be $3 \cdot (512/8) \cdot 160 = 30720$ bytes.

For supersingular curves we like to use the pairing $\hat{e}_r(P, Q) = e_r(P, \phi(Q))$, where $\phi(\cdot)$ is a distortion map. In this case $\hat{e}_r(P, Q) = \hat{e}_r(Q, P)$, and so a fixed parameter can always be exploited for precomputation. However this is not true for non-supersingular curves as $e_r(P, Q) \neq e_r(Q, P)$.

4.3 Products of Pairings

In some protocols, for example [28], it is necessary to calculate the product of two or more pairings. Consider the calculation of $e_r(P, Q) \cdot e_r(R, S)$. Each pairing requires an implicit point multiplication by r, an application of Miller's algorithm, and a final exponentiation. We suggest three optimisations which apply if the two pairing are calculated simultaneously rather than separately:

- Since the implicit multiplications of P and R occur in lock-step with one another, it makes sense to use affine coordinates in conjunction with Montgomery's trick. This means that just one modular inversion will be required instead of two. Montgomery's trick is based on the simple observation that $1/x = y/xy$ and $1/y = x/xy$.
- Both pairings can share the same *Miller variable* m. This means only a single squaring of m in line 5 will be required.
- Both pairings can share the final exponentiation (as pointed out by Solinas [29]).

The second and third optimisations depend on the observation that the final result will be the product of each pairing's Miller variable. These optimisations apply equally well to the product of multiple pairings. For the product of two pairings it will be about 50% faster than computing two separate pairings.

4.4 Protocol Optimisations

In pairing based protocols the most important property of the pairing is *bilinearity.*

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

It is more efficient to calculate $e_r(P, wQ)$ as $e_r(P, Q)^w$, as exponentiation by w in F_{p^2} will be much faster that point multiplication by w on $E(F_p)$ (Note however that this may not be true for values of $k > 2$.) A protocol like the Boneh and Franklin IBE scheme [6] may demand that an arbitrary string be hashed and mapped to the second Tate pairing parameter Q of order r. Since Q is on the twisted curve, the hashing would result in a random point S of order $p+1+t$, and the mapping would require a point multiplication by a large constant cofactor $c = (p+1+t)/r$, so $Q = cS$. However bilinearity applies *even though S is not of order r*. So rather than calculate $e_r(P, cS)$ again its faster to calculate $e_r(P, S)^c$. Depending on the protocol it may in fact be valid to dispense with the co-factor altogether. Alternatively, as for example in the Boneh and Franklin scheme, it may be possible to calculate $e(P, cS) = e(cP, S)$, and the constant c can be permanently combined with a constant P, and thus eliminated completely from the calculation [19].

On occasion a protocol (such as Boneh and Franklin IBE decryption) requires us to check that a curve point is in fact of order r, and then to calculate a pairing. Since the pairing carries out an implicit point multiplication of its first parameter by r, these functions can be combined. First re-organise the protocol if necessary so that the point whose order is to be tested is the first parameter of the pairing. Then in the BKLS algorithm insert the line

```
7a. if $A \neq -P$ then return Wrong_Order
```

5 Case Study – The Sakai and Kasahara Identity Based Encryption Scheme

Here we describe the simplest variant of the Sakai and Kasahara Identity Based Encryption scheme [22], and deploy some of our optimisations, and make use of the compressed pairing. This IBE method is not as well-known as that of Boneh and Franklin [6], but it has its advantages, for example no pairing calculation is required for encryption. It is not secure against a chosen-ciphertext attack, but then neither is unadorned RSA with which we will be comparing it. As for all IBE schemes, it can be described in four stages, Setup, Extract, Encrypt and Decrypt.

– **Setup:** Global parameters are generated by the trusted key-issuing authority. It generates a suitable curve (like ours) and generates random points $P \in E(\mathbb{F}_p)[r]$ and $Q \in E'(\mathbb{F}_p)[r]$, and then generates its own master secret $s \in \mathbb{F}_r$. The authority makes public the values Q, sQ, and $g = \varepsilon_r(P, Q)$. Also made public are two hash functions $h_1 : \{0,1\}^* \rightarrow \mathbb{F}_r$ and $h_2 : \mathbb{F}_p \rightarrow \{0,1\}^c$
– **Extract:** Each user approaches the trusted authority and is issued with a private key. In the case of Alice, her identifying string is hashed to a value $a = h_1(Alice's\ Identity)$ and she is issued with the private key $D = \frac{1}{s+a}P$
– **Encrypt:** To encrypt a secret session key k and send it to Alice, first find $a = h_1(Alice's\ Identity)$ and then generate a random $w \in \mathbb{F}_r$ and create the ciphertext

$$C_1 = w(sQ + aQ)$$
$$C_2 = k \oplus h_2(V_w(g))$$

– **Decrypt:** Alice recovers the session key as $k = h_2(\varepsilon_r(D, C_1)) \oplus C_2$.

where $V_w(\cdot)$ is the Lucas function and $\varepsilon_r(\cdot)$ is the compressed pairing. The correctness of the algorithm follows immediately from bilinearity. Note that a pairing calculation is only needed for decryption. And for this pairing the first parameter is a constant – Alice's private key. Therefore the precomputation optimisation of section 4.2 is appropriate. For encryption Alice's public key can be regarded as $(sQ + aQ)$, and if multiple messages are to be sent to Alice, then this value can also be precalculated and cached. The calculation of aQ, if required, is carried out on the twisted curve $E'(\mathbb{F}_p)$, as is the subsequent point multiplication by w. If the point to be multiplied is fixed, then the precomputation method of Brickell et al. can be used here to advantage [9].

6 Results

The IBE scheme described above was implemented using a mixture of C++, C and assembly, both with and without precomputation, and compared with a similarly optimised RSA implementation (standard windowing techniques for modular exponentiation are used). We focus on the decryption operation in both cases - clearly RSA encryption will be much faster. The RSA implementation uses as a public key the product of two 512-bit prime factors, and decryption uses the

Chinese Remainder Theorem. So in both cases 512-bit modular multiplication or squaring in $\mathbb{F}_p$ is the time-critical operation. This was implemented using inline unlooped assembly language. We are not claiming that our multi-precision implementation is the fastest possible on the targeted hardware, a 1GHz Pentium III, but it is the same implementation for both IBE and RSA. For the pairing calculation without precomputation projective coordinates are used.

It is possible to calculate exactly the number of $\mathbb{F}_p$ multiplications (called henceforth a *mul*) needed for the pairing; for now we count squarings as multiplications, although squarings will be a little quicker. Using the standard algorithms for point addition and doubling as described in [14], a projective point addition requires 11 muls, and a doubling requires 8. The projective $f(\cdot)$ function needs 8 muls. Hence the $g(\cdot)$ function performs 16 muls for each point doubling and 19 muls for each addition. Line 5 of the main algorithm will require one $\mathbb{F}_{p^2}$ squaring, and one $\mathbb{F}_{p^2}$ multiplication at a cost of 5 muls, plus a call to $g(\cdot)$ for a point doubling, for a total of 21 muls. The loop is repeated 159 times, so line 5's total contribution is $159 \cdot 21 = 3339$ muls. The point addition of line 6 is only carried out once, and adds an extra 22 muls. So by the end of Miller's algorithm the cost so far is 3361 muls. Line 9 calls for a $\mathbb{F}_{p^2}$ modular inversion. Assuming that we are going to compress the pairing, this calls for 5 muls and a single inversion in $\mathbb{F}_p$. The exponent $(p+1)/r$ will be $512-160 = 352$ bits, and the Lucas laddering algorithm [25] requires exactly 2 muls per bit. So the grand total for the whole pairing is $3361 + 5 + 352 \cdot 2 = 4070$ muls plus one modular inversion.

Using precomputation the savings are substantial. The cost of each call to the $g(\cdot)$ function now falls to just 1 mul. Repeating the analysis above the total is now reduced to 1667 muls plus one inversion. By contrast a representative run of an RSA decryption program requires 1020 modular squarings and 222 modular multiplications.

Some tests show that, in the environment used, a modular squaring is equivalent to 0.89 of a modular multiplication, and that a modular inversion costs close to 28 muls. Introducing these equivalent costs, we can specify the computation required quite precisely in terms of the number of muls.

Timings are for the number theoretic parts of the algorithm; message padding and hashing are excluded. Our actual timing do not quite live up to those that could be projected from the raw mul counts. The pairing calculation, unlike RSA, involves a large number of modular additions and subtractions which have been ignored in the analysis above. In our implementation elliptic curve addition and subtraction is done in C, whereas the pairing is implemented in C++, which will be a little slower due to the tendency of C++ to create and destroy temporary variables. Finally precomputation may suffer somewhat as the precomputed data plus program variables are too large for the Pentium III 16K byte L1 data cache.

Table 1. Sakai and Kasahara IBE decryption – 1GHz PIII.

algorithm	$\mathbb{F}_p$ Muls	Time (ms)
IBE decryption, w/o precomp.	3992.7	20.0
IBE decryption, with precomp.	1660.1	10.2
RSA decryption	1126.8	4.5

A similarly optimised implementation of the Tate pairing, without precomputation, on a Compaq iPaq 3660 PDA powered by a 206MHz 32-bit StrongARM processor took just 355ms. This indicates that pairing-based cryptography may find application on low powered devices such as PDAs and mobile phones.

7 Conclusions

We have shown that pairing-based cryptography is perhaps not as slow or as difficult as was generally thought. The $k = 2$ case is particularly simple and accessible, and can be quickly implemented using existing crypto resources. Various optimisations have been suggested. An implementation of an IBE scheme shows that pairing-based cryptography can perform nearly as well as long established techniques such as RSA.

Acknowledgments

Thanks to Paulo Barreto for useful comments, and Noel McCullagh for drawing to my attention the Sakai and Kasahara IBE scheme [22] and for help with the implementation on the Compaq PDA.

References

1. P. S. L. M. Barreto. The pairing-based crypto lounge, 2004. `http://planeta.terra.com.br/informatica/paulobarreto/pblounge.html`.
2. 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 354–68. Springer-Verlag, 2002.
3. 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.
4. P.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. Springer-Verlag, 2003.
5. I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography, Volume 2.* Cambridge University Press, 2005.
6. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
7. 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.
8. F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. Cryptology ePrint Archive, Report 2003/143, 2003. Available from `http://eprint.iacr.org/2003/143`.
9. E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation: Algorithms and lower bounds. In *Advances in Cryptology – Eurocrypt'92*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer-Verlag, 1993.

10. R. Dupont, A. Enge, and F. Morain. Building curves with arbitrary small MOV degree over finite prime fields. Cryptology ePrint Archive, Report 2002/094, 2002. `http://eprint.iacr.org/2002/094`.
11. R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptography : A survey. Cryptology ePrint Archive, Report 2004/064, 2004. `http://eprint.iacr.org/2004/064`.
12. I. Duursma and H.-S. Lee. Tate-pairing implementations for tripartite key agreement. In *Advances in Cryptology – Asiacrypt 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer-Verlag, 2003.
13. S. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Algorithm Number Theory Symposium – ANTS V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer-Verlag, 2002.
14. IEEE Std 1363-2000. Standard specifications for public-key cryptography. IEEE P1363 Working Group, 2000.
15. T. Izu and T. Takagi. Efficient computations of the Tate pairing for the large MOV degrees. In *ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 283–297, 2003.
16. M. Joye and S. Yen. The Montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302, Berlin, Germany, 2003. Springer-Verlag.
17. P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks, 1998. `http://www.cryptography.com/dpa/technical`.
18. W. Mao and K. Harrison. Divisors, bilinear pairings, and pairing enabled cryptographic applications, 2003. `http://hplbwww.hpl.hp.com/people/wm/research/pairing.pdf`.
19. N. McCullagh. Personal Communication, 2004.
20. A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
21. A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals*, E84-A(5):1234–1243, 2001.
22. 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`.
23. M. Scott, 2002. `http://ftp.compapp.dcu.ie/pub/crypto/cm.exe`.
24. M. Scott, 2002. `http://www.computing.dcu.ie/~mike/tate.html`.
25. 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/`.
26. M. Scott and P. Barreto. Generating more MNT elliptic curves. Cryptology ePrint Archive, Report 2004/058, 2004. Available from `http://eprint.iacr.org/2004/058/`.
27. I. Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. Cryptography ePrint Archive, Report 2004/031, 2003. `http://eprint.iacr.org/2004/031/`.
28. N. P. Smart. An identity based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, 38:630–632, 2002.
29. J. Solinas. ID-based digital signature algorithms, 2003. `http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf`.