Scaling security in pairing-based protocols

Michael Scott

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

Abstract. In number theoretic cryptography there is always the problem of scaling-up security to a higher level. This usually means increasing the size of the modulus, from, say 1024 bits to 2048 bits. In pairing-based cryptography however another option is available, keeping the modulus constant and increasing instead the embedding degree. This has a big potential advantage in smart-card and embedded applications – security can be scaled up while continuing to use the same sized calculations. For example a cryptographic co-processor which does 512-bit modular multiplications can be directly re-used in the higher security setting. Here we investigate the scaling-up issue in the context of prime characteristic non-supersingular elliptic curves. We also confirm the observation that at higher levels of security a slightly modified Weil pairing becomes more efficient than the Tate pairing.
Keywords: Cryptographic key sizes, pairing-based cryptosystems.

1 Introduction

In the majority of number-theoretic cryptographic protocols, both over the finite field and on elliptic and higher genus curves, the problem of increased security implies only one obvious solution – increase the size of the modulus, traditionally by doubling its size. Interestingly in the case of the RSA scheme, based as it is not on the difficulty of a discrete logarithm problem, but rather on the problem of integer factorisation, there is an alternative solution. Multi-prime RSA has been frequently suggested by many people as an alternative to simply doubling the length of the prime factors of the public key. If $n = pq$, and p and q are originally 512 bits, then an increased security implementation might make p and q both 1024-bits. The multi-prime alternative is to use $n = pqrs$ where p, q, r and s all remain 512-bit primes. This does not apparently introduce any serious new weaknesses [14], and represents a completely viable solution, with some significant performance advantages. And yet multi-prime RSA is not widely used, and has only relatively recently appeared in standardisation documents [18]. The reason for this may be that the optimal way of scaling RSA security was never properly addressed early on by the research community. And it is very simple to just double the bit length of p and q – for one thing it requires no further security analysis.

For a particular discrete logarithm based scheme, there is no such choice. For increased security, one must increase the size of the modulus. There is of course the possibility of switching to a different scheme for which the same-sized modulus provides more security – for example one can switch from a standard finite field setting to a scheme such as LUC [30], or XTR [19]. But as a way of scaling security this seems very clumsy. And there is something else to be considered – when attacking a discrete logarithm based system an attacker can choose to exploit the small size of the group (using the Pollard Lambda algorithm), or the small size of the field (using index calculus methods, if they apply, otherwise the Pollard Rho algorithm) [24]. For balanced security in a finite field one might choose a modulus of 1024-bits (to resist index calculus attacks) and a group size of 160-bits (to resist Pollard Lambda attacks).

The issue of how to scale up security in pairing-based cryptosystems has also recently and independently been addressed by Koblitz and Menezes [17]. As will be seen our experimental evidence largely supports their conclusions.

In pairing based cryptography there are, as for RSA, two distinct strategies to obtain increased levels of security; double the size of the prime modulus, or double the embedding degree. In both cases the group size must also be increased as appropriate.

However field size is a particularly significant parameter for smart-card and embedded implementations that use co-processor support, as it is field size that determines the hardware requirement.

Here we analyse the approach of simply doubling the embedding degree. This has some immediate implications. Firstly supersinglar elliptic curves have a maximum embedding degree of 6. While it is true that higher embedding degrees can be obtained on hyper-elliptic curves [25], as a method of scaling security switching to a different characteristic or to a higher genus curve again seems very clumsy. Therefore we consider here only non-supersingular curves, while acknowledging that small characteristic supersingular elliptic and hyperelliptic curves can be a very efficient vehicle for pairing-based cryptography. See for example [2] for some recent results, in particular the discovery of the new η_T pairing.

The aim is to provide a mechanism for scaling security which requires minimal changes to an existing implementation, in terms of algorithm, software and hardware.

In most protocols it is the time taken to calculate the pairing that is most significant. However the time taken for field exponentiation and elliptic curve point multiplication must also be taken into account when evaluating in detail the perfomance of a particular protocol. In many cases precomputation can be exploited to eliminate much of the computational cost, and this must also be taken into account. Such in-depth analysis must be done on a protocol by protocol basis, and is outside the scope of this paper, where we concentrate solely on the cost of the pairing.

Generating suitable non-supersingular elliptic curves can be quite difficult. Many algorithms have been suggested, but the current state-of-the-art has its own constraints. To summarise:-

Let F be the number of bits in the field modulus p, and G the size of the group order r in bits. The embedding degree is k. Then it is relatively easy to generate suitable curves with $G < F/2$ for any k. In some specific cases it is possible to generate curves with $F/2 < G < F$. For $k = \{3, 4, 6\}$ it is possible to find curves with $G = F$. Very recently Barreto and Naehrig [6] have discovered a remarkable formula for easily generating curves with $G = F$ and $k = 12$.

In some applications, like a short signature scheme [9], it is very important that $G = F$. And at first glance this also appears to be most efficient. However in the majority of applications it is not important, and so we will restrict ourselves (mostly) to the condition $G < F/2$ where curves are plentiful and easy to find. An added advantage of choosing $G < F/2$ is that in this case the group order r can be chosen to have the lowest possibly Hamming weight, with performance benefits [3].

For our purposes we define 3 levels of security, which are roughly within the limits laid down by Lenstra and Verheul [20]. These are referred to here as (1024/160) security, (2048/192) security, and (4096/224) security, where the first figure refers to the effective field size kF, and the second refers to the group size. Here we advocate fixing $F = 512$ and using $k = 2$, $k = 4$ and $k = 8$ to achieve the higher levels of security. Of course much greater granualarity could be achieved by using intermediate values of k, and different values for F. However we justify our approach by pointing out that in practice the "algorithm" used by cryptographers is that if using an N bit "RSA equivalent" level of security, and cryptanalysts threaten at the $N/2$ level, then switch to $2N$ bit security.

Note that we could easily continue on to achieve yet higher levels of security.

2 The Tate pairing

In this section we will review the Tate pairing algorithm on non-supersingular curves over fields of large prime characteristic p. We choose a prime r of size G which has a low Hamming weight. Next we find an elliptic curve over $\mathbb{F}_p$ where p is of size F and whose order is divisible by r, and where $r|p^k - 1$. Assuming that r does not divide $p^i - 1$ for any value of $i < k$, then this is a curve suitable for use in pairing-based cryptography, and has an embedding degree of k. Here we will always assume that k is of the form 2^n. Observe the well-known fact that the cyclotomic polynomial

$$\Phi_{2^n}(x) = x^{2^{n-1}} + 1$$

is irreducible, and

$$p^{2^n} - 1 = (p^{2^{n-1}} - 1).\Phi_{2^n}(p)$$

Since we know that r does not divide $(p^{2^{n-1}} - 1)$, then it must divide $\Phi_{2^n}(p)$.

The general Tate pairing is written as $e_r(P,Q)$, where P is a point on $E(\mathbb{F}_{p^k})$ in a subgroup of order r and Q is also on $E(\mathbb{F}_{p^k})$, a representative of the coset of points which includes a member of a distinct subgroup, also of order r. The Tate pairing evaluates as a non-trivial element of the extension field F_{p^k} of order r.

The Tate pairing has the following relevant properties.

1. $e_r(aP, bQ) = e_r(P,Q)^{ab}$ for all $a, b \in \mathbb{F}_r$ (Bilinearity)
2. $e_r(P,P) = 1$

The bilinearity property is the one that enables the implementation of many novel protocols.

In practise it is common for efficiency reasons to choose $P \in E(\mathbb{F}_p)$. It is also possible to manipulate Q as a point on the twisted curve defined over $\mathbb{F}_{p^{k/2}}$, before mapping it to a point on $E(\mathbb{F}_{p^k})$ prior to calculation of the pairing [5]. The coordinates of this point will exhibit some redundancy which can be exploited to speed up the calculation.

The Tate pairing algorithm consists of an application of Miller's algorithm followed by a final exponentiation. The purpose of the final exponentiation is to yield a unique result of order r. In Miller's algorithm the point P is implicitly multiplied by r using the standard double and add method, and at each iteration a distance relationship between the current point and the fixed point Q is calculated and accumulated in a $\mathbb{F}_{p^k}$ variable.

Now the field $\mathbb{F}_{p^k}$ has $p^k - 1$ elements in it, and by definition $r|p^k - 1$. Assuming that $k = 2^n$ the final exponentiation is to the power of $(p^{2^n} - 1)/r$. This can be written as $(p^{2^{n-1}} - 1)((\Phi_{2^n}(p)/r)$. The exponentiation by $(p^{2^{n-1}} - 1)$ is cheap, using the Frobenius action. The hard work here is the exponentiation to the power of $\Phi_{2^n}(p)/r$.

As demonstrated in [28] the output of the Tate pairing can be compressed to half its size for even k. The compressed pairing returns an element of $\mathbb{F}_{k/2}$. The final exponentiation in this case is replaced with the somewhat simpler calculation of a Lucas sequence.

2.1 Non-supersingular Vs Supersingular

There is a view (unsupported by any hard evidence) that non-supersingular curves are intrinsically "more secure" than supersingular curves. We will not comment on this. There is also a view that it is faster to use a supersingular curve. For prime field characteristic one is restricted to the case $k = 2$. However in this case as demonstrated in [27] the optimal pairing algorithm for non-supersingular curves is just as efficient as for supersingular curves. Recently it has been suggested [17] that the ease of domain generation for supersingular curves makes it easier to generate a modulus p with a low Hamming weight, which in turn leads to faster implementation. But as they also point out a low Hamming weight p raises security concerns. Furthermore it is also quite possible to generate low Hamming weight moduli for non-supersingular curves, using for

example the methods of [7] or [29]). See section 5 below for more details on curve generation.

Nevertheless the vast majority of pairing based protocols have been described in the context of super-singular curves. Most can be transferred directly onto a non-supersingular curve and will work fine under the so-called co-Bilinear Diffie-Hellman assumption, co-BDH, rather than under the original BDH assumption [8]. However there are subtle differences. Recall that when using supersingular curves a distortion map exists, and so $\hat{e}(P,Q) = e_r(P,\phi(Q))$, where $P,Q \in E(\mathbb{F}_p)$. Then this distorted Tate pairing has the properties

1. $\hat{e}_r(aP,bQ) = \hat{e}_r(P,Q)^{ab}$ for all $a,b \in \mathbb{F}_r$ (Bilinearity)
2. $\hat{e}_r(P,P) \neq 1$
3. $\hat{e}_r(P,Q) = \hat{e}_r(Q,P)$

Note that property 2 is different and property 3 is new. However the two pairings do share the important property of bilinearity which is the property required for the new pairing-based protocols. Sometimes these different properties can affect the behaviour of a protocol in interesting ways. However the main implication is that when working on non-supersingular curves we need to remember to treat the two parameters $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^k})$ quite differently. They cannot be interchanged or moved from side to side of the pairing calculation. Another potentially significant difference is that manipulation of the Q parameter prior to the pairing calculation is much simpler using super-singular curves, as Q will be a point on the base curve rather than on the curve taken over a larger extension field when $k > 2$, even using the "twist" idea (see below). Finally as pointed out in [27] it is possible to do useful precomputation on the first parameter if it should be a constant, but not on the second. Since the parameters can be switched from side to side with supersingular curves (using property 2 above) then this feature is easier to exploit than in the non-supersingular case.

3 A Tower of extensions

We will require a scalable implementation of finite field arithmetic over the field $\mathbb{F}_{p^{2^n}}$. The simplest way to do this will be to use a tower of extensions [23]. That is an element of $\mathbb{F}_{p^{2^n}}$ will be represented as a pair of elements from $\mathbb{F}_{p^{2^{n-1}}}$, and so on recursively, starting with an efficient implementation of $\mathbb{F}_p$.

For a suitable extension field arithmetic representation we first need an irreducible polynomial. For example if $p = 3 \mod 4$, then $x^2 + 1$ is a suitable irreducible polynomial for representation of the extension field $\mathbb{F}_{p^2}$. Elements of the field can be represented as a polynomial $ax+b$ where $a,b \in \mathbb{F}_p$. Alternatively "solve" the irreducible polynomial and set $x = \sqrt{-1}$, and use as a representation $a + b.\sqrt{-1}$. Numbers in this form can be manipulated directly without explicit reference to an irreducible poynomial. Note that -1 is a quadratic non-residue with respect to p, iff $p = 3 \mod 4$. However unfortunately this representation with its simple irreducible polynomial does not permit a simple tower of extensions to be built on top of it, as $p^2 = 1 \mod 4$.

An alternative is to choose $p = 1 \mod 4$. In this case the irreducible polynomial $x^2 + 2$ can be chosen, and it does support an infinite tower of similar extensions. In practise we choose $p = 5 \mod 8$, as it is important to have a simple formula for modular square roots, which does exist for this case, but not in general for $p = 1 \mod 8$ (the calculation of square roots is required to generate points on the elliptic curve). So –

An element of $\mathbb{F}_{p^2}$ is $a + b.(-2)^{1/2}$, or $[a, b]$ a pair of elements from $\mathbb{F}_p$.
An element of $\mathbb{F}_{p^4}$ is $c + d.(-2)^{1/4}$, or $[c, d]$ a pair of elements from $\mathbb{F}_{p^2}$.
An element of $\mathbb{F}_{p^8}$ is $e + f.(-2)^{1/8}$, or $[e, f]$ a pair of elements from $\mathbb{F}_{p^4}$.

Now the implementation of each new layer of the tower can be built in an identical fashion on top of the previous layer. In the terminology of the programming language C++, a templated class which supports the operations of field addition, subtraction, multiplication and division (plus square rooting and Lucas powering) can be written just once which will support all these instantiations. We omit implementation details as they are quite straightforward.

There is now an extra constraint on the curve generation process, that $p = 5 \mod 8$. However we found that it was still easy to find a multitude of suitable curves.

3.1 The Twist Idea

Let i be the k-th root of -2. Then if the point $([x, 0], [0, y])$ is a point on the curve $E : y^2 = x^3 + Ax + B$ over the field $\mathbb{F}_{p^k}$, then it is easy to verify by simple substitution that the point (i^2x, i^4y) is a point on the twisted curve $E' : y^2 = x^3 + i^4Ax + i^6B$ over the field $\mathbb{F}_{p^{k/2}}$. It is also a simple matter to map points back from the twisted representation to the original curve.

3.2 Frobenius action

In the field F_{p^k} the Frobenius action is defined as the well-known identity

$$(x + iy)^p = x^p + i^p y^p$$

This means that exponentiation by p is almost for free. Using the Tower of extensions this formula can be applied recursively to x^p and y^p. However the term i^p needs to be handled carefully. For example if $i = (-2)^{1/8}$, and $p = 5 \mod 8$, then $i^p = (-2)^{(p-5)/8} i^5$, where the term $(-2)^{(p-5)/8}$ can be precalculated.

4 The Algorithm

In this section we describe the compressed BKLS version of the Tate Pairing algorithm (which in turn is based on the original Miller algorithm), closely following the treatment in [27], which is based on earlier work by Barreto et al [3]

and Galraith et al [13]. Compressed pairings are described in [28]. For efficiency we use a standard projective coordinate system, as described in [15], for the implicit point multiplication on $E(\mathbb{F}_p)$. First we need a function to execute a point addition, obtaining the line slope and finally calculating the contribution of the current iteration of the algorithm. Capital letters denote curve points, lower case letters and symbols represent elements of $\mathbb{F}_p$ or simple integers, and boldface letters represent elements of $\mathbb{F}_{p^{k/2}}$.

$\boldsymbol{g}(R, P, \mathbf{x_q}, \mathbf{y_q})$
1. $x, y, z \leftarrow R$
2. $\lambda_n, \lambda_d = R.add(P)$
3. **return** $y\lambda_d - \lambda_n(xz - \mathbf{x_q}z^3) - \mathbf{y_q}z^3\lambda_d \cdot i$

The function $A.add(B)$ performs the standard projective point addition, but also returns the line slope as the rational λ_n/λ_d. Note that the line slope is needed anyway to perform the point addition, so no extra work is involved. Assume that the point P is of prime order r.

Tate(r, p, P, Q)
1. $\boldsymbol{m} = 1$
2. $R = P$
3. $n = r - 1$
4. $\mathbf{x_q}, \mathbf{y_q} \leftarrow untwist(Q)$
5. **for** $i \leftarrow \lfloor \lg(r) \rfloor - 2$ **downto** 0 **do**
6. $\quad \boldsymbol{m} = \boldsymbol{m}^2 \cdot \boldsymbol{g}(R, R, \mathbf{x_q}, \mathbf{y_q})$
7. $\quad$ **if** $n_i = 1$ **then** $\boldsymbol{m} = \boldsymbol{m} \cdot \boldsymbol{g}(R, P, \mathbf{x_q}, \mathbf{y_q})$
8. **end for**
9. $\boldsymbol{m} = \boldsymbol{m}^{(p^{k/2}-1)}$
10. **return** $V_{(p^{k/2}+1)/r}(2\mathbf{m_a})$

The variable $\boldsymbol{m}$ is the *Miller variable*, and is the only variable that is an element of the full extension field $\mathbb{F}_{p^k}$. Note that $\boldsymbol{m} = [\mathbf{m_a}, \mathbf{m_b}]$. The notation n_i refers to the i-th bit of n. Note that the choice of a low Hamming weight r means that this bit may be 1 only once in the entire calculation. The function *untwist* converts the point Q on the twisted curve to the point $([\mathbf{x_q}, 0], [0, \mathbf{y_q}])$ on the original curve over the full extension field $\mathbb{F}_{p^k}$, as described in section 3.1. The calculation in line 9 is greatly simplified by repeated application of the Frobenius action, as described in section 3.2. The value of $(p^{k/2} + 1)/r$ can be precomputed and stored. The function V_n is the well-known Lucas function.

In certain circumstances the first parameter to the pairing algorithm, the point P, may be a constant. If it is, then we can benefit significantly from a precomputation. All the points and slopes in the implicit multiplication of P by r can be precomputed and stored, and a much simpler affine version of the function $g(.)$ can be used [27].

Using a tower of extensions this same algorithm can be used for any positive value of n, where the extension degree is $k = 2^n$. Although implementations

details might vary slightly (in particular in the *untwist* function and in the Frobenius action), by and large it is fair to say that the same basic algorithm works for any n. But how do we find suitable non-supersingular curves for $k = 2$, $k = 4$, $k = 8$ etc?

5 Curve Generation

It was long known that non-supersingular curves might exhibit a low embedding degree – hence the famous MOV condition [21] that was recommended to avoid the use of such curves in classic elliptic curve cryptography. However since they were generally regarded as a "bad thing", there was not much interest in deliberately generating them. However fortunately at around the time that pairings became cryptographically interesting Miyaji et al. published their paper on the construction of so-called MNT curves [22]. The original paper described constructions for non-supersingular elliptic curves over fields of prime characteristic with embedding degrees of 3, 4 and 6 with $G = F$ (Recall that F is the number of bits in the prime modulus p, and that G is the number of bits in the prime group order r). The condition $G = F$ implies that the curve $E(\mathbb{F}_p)$ is of prime order r, and such curves are very rare, although subsequent developments extended the idea to produce many more curves where the curve order was hr, for small values of h [29]. The term MNT curve is now commonly used to refer to any non-singular curve with a useful embedding degree.

A major development was an algorithm of Cocks and Pinch (unfortunately unpublished, but essentially the same algorithm is described in [7]). This method makes it quite easy to find a curve with any desired embedding degree, but with the restriction $G < F/2$.

Subsequently a series of papers [4], [10], [11], [29] discovered alternative strategies which in special cases could find curves with $F/2 < G < F$. Although at first glance it appears preferable to choose curves with $G = F$, unfortunately until very recently there was no known method for doing so with $k > 6$. This situation changed with the surprising discovery by Barreto and Naehrig [6] that solutions with $G = F$ for the particular case of $k = 12$ were indeed possible. However it seems to be generally true that for $G > F/2$ the number of possible curves starts to become constrained, and in particular one no longer has control of the choice of r. This is a pity because it is clearly more efficient if r is chosen to have a low Hamming weight.

Since it is our ideal to develop a method which can be scaled without limit, the Cocks and Pinch algorithm seems to be the only option, although we will consider one candidate curve generated using the method described by Brezing and Weng [10].

These algorithms give us enough information to know that a suitable curve exists, with the desired embedding degree and with a curve order divisible by the chosen r. However to find the actual curve parameters we must use the method of Complex Multiplication as described in [15], and implemented in, for example, [26].

The actual elliptic curves $E_n(\mathbb{F}_p)$ that we used are described using the standard Weierstrass representation [15] for curves of prime characteristic, $y^2 = x^3 + Ax + B$.

$\mathbf{E_{160}}$

$F = 512, G = 160, k = 2$

$p = 8d19a10497f9cc35bc026c6a7651da9ce4d794d4d67a4ab6e77d3b322e462b899$
$1a31e8fbfcb60b411f0e5afd85977d7eee3dd09cd197dafe4f377c3e23af2af$

$A = -3$

$B = 4d438977ff9360a5df4294efe4bd7465351b7d19e99c17463f3c7f72ab95ca3c95$
$c82339f2c8b9f8801bc29328ae2d73969493d719fbbe3d34f689778ebd85ae$

$r = 800000000000000000000000000000000020001$

$\mathbf{E_{192}}$

$F = 512, G = 192, k = 4$

$p = b7fd9899de545d9d6b3644da15e9b662fd08ca211da537978b5556299c6b02f77$
$d5e776d48c14b14c65e905646de2318391b0ac3820ab2be9eb54b70927ee9b5$

$A = -3$

$B = 865b15c872e80609aC5a6b397438a020d9c1cfbccd2d7e57480b1c80f6325461f$
$b1bf32b28d689412c6875e267676d926dd0232dbcc049efa670cfe2cb971a65$

$r = 800000000000000000000000000000000000000000000005$

$\mathbf{E_{192bw}}$

$F = 256, G = 192, k = 8$

$p = cc485d26177a1a5fcc9d53ba93da298fd7f2f23d8fc02a8123bf24f9548a5f15$

$A = 0$

$B = 2$

$r = 9d0261dd89cf83d5d20198162c22c942ef68622a6df25621$

$\mathbf{E_{224}}$

$F = 512, G = 224, k = 8$

$p = bc1e1bf222c03dfd1bd55f2f220cf3b3f7f185834d6db68cdcc11d413e77ea4ee$
$cf8ffa5038e46c303a407da24b15aa57cc8bd6df69c5d9806742fca59f8604d$

$A = -3$

$B = 62dc6331021838f1b0557dd1b016b4898fc16c084b8f2a4b9368d0ce1c4e744c6$
$e4319e1bbdd27ad060439ba50f0eb6cbc79e74404cd371263f024aecabfa9c7$

$r = 8000000000000000000000000000000000000000000000000002001$

The curves E_{160}, E_{192} and E_{224} have a field size of 512 bits, and embedding degrees of 2, 4 and 8 respectively. Observe the low Hamming weight group orders r, which are of sizes 160, 192 and 224 bits respectively. These were all generated using variations of the Cocks and Pinch algorithm. For the curve E_{160}, $p = 3 \mod 4$ and we can use the irreducible polynomial $x^2 + 1$. This will be a little faster. For all the other curves $p = 5 \mod 8$ to facilitate the building of the tower of extensions using the irreducible polynomial $x^2 + 2$ as described above. It was particularly easy to find curves using the Cocks and Pinch algorithm, and we would expect that it would be as easy to find curves with higher embedding degrees $k > 8$. The curve E_{192bw} was generated using the Brezing-Weng

algorithm [10], and has a field size of 256 bits, a group size of 192 bits, and an embedding degree of 8. Therefore in terms of security it should be similar to E_{192}. The Brezing and Weng curve was much harder to find, given the constraints imposed.

6 Results

Before presenting actual timings, it will be helpful to carry out some analysis of the expected runtime of the algorithm. First consider the situation for a fixed (kF/G) level of security. Here we often have a choice between a large k and small F, or visa versa. To be concrete for (2048/192) security, should we use $k = 2$, $F = 1024$, or $k = 4$, $F = 512$, or $k = 8$, $F = 256$? G will remain the same irrespective.

In line 6 of the main algorithm we have the multiplication of elements from $\mathbb{F}_{p^k}$, which are represented as a pair of elements from $\mathbb{F}_{p^{k/2}}$. Let x be the time required for the multiplication of elements from $\mathbb{F}_{p^{k/2}}$. Then using the Karatsuba method, the time required for the multiplication of elements from $\mathbb{F}_{p^k}$, will be $3x$.

Using our concrete example, the question is – what is x? That is how long does it take to multiply a pair of polynomials of fixed absolute size of 1024 bits, which could be organised as a degree 0 polynomial with a single 1024-bit coefficient, or as a degree 1 polynomial with 512-bit coefficients, or as a degree 3 polynomial with 256 bit coefficients. In all cases the use of the Karatsuba algorithm for polynomial multiplication can be assumed to be optimal for coefficients of these sizes. In general when using the Karatsuba method polynomial multiplication will be more efficient than integer multiplication (as will be required in the first case), as no carries are required. On the other hand for large values of k polynomial multiplication becomes quite inefficient as more small coefficients need to be processed. Somewhere in between there will be an optimal choice, which will depend largely on implementational details. Here we assume that x is a "almost" a constant, an observation that is supported by our experimental experience.

With this assumption, and letting y be the time required for a base field multiplication over $\mathbb{F}_p$, then the total time for the algorithm can be calculated as

$$G(6x + yk + 7y + 8y) + 2x.(kF/2 - G) + m.(3x + yk + 7y + 11y)$$

assuming that a point doubling on $E(\mathbb{F}_p)$ requires 8 multiplications in $\mathbb{F}_p$, and point addition requires 11 such multiplications, using standard algorithms [15]. Each iteration in the calculation of the Lucas sequence requires 2 multiplications in $\mathbb{F}_{p^{k/2}}$ [16]. We neglect the time taken to "untwist" the point Q in line 4 of the algorithm (as it is negligible), and for the Frobenius calculations in line 9. The first term accounts for the main loop of the algorithm, and the second term is the the time required for the calculation of the Lucas sequence – the final

exponentiation. The third term is the time taken when the "if" statement on line 7 is executed assuming it is executed m times. Rearranging this expression gives

$$x(4G + kF + 3m) + y((G + m)k + 15G + 18m)$$

Note that precomputation (for a constant first pairing parameter P) reduces the second term contribution to $y((G + m)k/2)$.

For a given level of security, and as a consequence of the discussion above, the first term is "almost" a constant, independent of the actual relationship between F and G. The second term clearly benefits from a smaller field size, but as kF becomes much greater than G the second term contribution to the overall cost becomes much less significant, and precomputation also becomes less relevant. Note that the kF term originated with the final exponentiation, therefore for higher levels of security we can conclude that the final exponentiation will eventually become the dominant part of the calculation.

A final observation is that using the Cocks and Pinch construction with $G < F/2$ we can choose a low Hamming weight value for r, and hence $m = 1$. Other constructions typically result in $m \approx G/2$. This can be reduced somewhat using standard windowing methods, or using the technique described in [12].

To get a rough count of the number of base field multiplications required for $k = 2^n$, we can substitute $x = 3^{n-1}.y$ (assuming Karatsuba) into the above formulae.

Actual timings were carried out on a 3GHz Pentium IV processor, and are shown in Table 2. In particular we compare the scaling approach advocated here, with fixed size prime modulus and doubling k, with the alternative scaling method of simply doubling the size of the prime modulus, and keeping $k = 2$ fixed, using curves E_{192a} and E_{224a}. We also compare curves generated using the Cocks and Pinch algorithm with an example curve E_{192bw} generated using an alternative approach [10] for which $G > F/2$

Table 1. Timings – Pentium IV 3GHz – Tate Pairing

Curve	(kF/G)	k	F (bits)	G (bits)	Time (ms)	with precomp.
$\mathbf{E_{160}}$	(1024/160)	2	512	160	8.9	4.9
$\mathbf{E_{192}}$	(2048/192)	4	512	192	20.5	16
$\mathbf{E_{192bw}}$	(2048/192)	8	256	192	25	22
$\mathbf{E_{192a}}$	(2048/192)	2	1024	192	45	26
$\mathbf{E_{224}}$	(4096/224)	8	512	224	92	85
$\mathbf{E_{224a}}$	(4096/224)	2	2048	224	209	137

Observe that the curve generated using the simple Cocks and Pinch algorithm E_{192} is in practice a little faster than the Brezing and Weng curve E_{192bw} for the same security level. (Note that the for the Brezing and Weng curve we do

not use the algorithm as described above, but rather a standard windowing algorithm, which is more suitable in this case where the Hamming weight of r is not insignificant.) The method of doubling the prime modulus as a method of scaling suffers, as point multiplication on an elliptic curve over a large prime field is very expensive. Precomputation (if it applies) solves this problem to an extent, but only at the cost of large precomputed tables.

7 The Weil Pairing

Originally it was thought that the Tate pairing would always be a better choice than the Weil pairing, a view articulated for example in [7].

It has been suggested recently that at higher levels of security that the Weil pairing may in fact be more efficient than the Tate pairing [1], [17]. This is supported by the observation that at higher security levels the final exponentiation of the Tate pairing becomes the most time-consuming operation. On the other hand the Weil pairing requires two invocations of Miller's algorithm, but no final exponentiation. Another way to look at it is to observe that whereas the complexity of the Tate pairing is $O((kF)^3)$, the complexity of the Weil pairing is only $O((kF)^2G)$ (using naive multiplication).

Here we will try to determine the cross-over point at which the Weil pairing becomes superior. We observe that rather than using the standard Weil pairing directly it will be advantageous to consider the Weil pairing raised to the power of $p^{k/2}-1$ (using the fast Frobenius action) as this permits various optimizations similar to the denominator elimination optimization described by Barreto et al [3]. This observation was also made independently by Koblitz and Menezes [17].

The parameters P and Q are as above, although this time it is a requirement that Q should also be of order r.

The function $g(.)$ is more complex for the Weil pairing, as a pair of point additions are required as we implement two invocations of the Miller algorithm. As before we use projective coordinates for the implicit point multiplication of the point P on $E(\mathbb{F}_p)$, but affine coordinates for the much more expensive point multiplication of Q on the twisted curve $E'(\mathbb{F}_{p^{k/2}})$. The *untwisting* required of the points is merged into the formulae, whose derivation is left as an exercise for the reader. Note that the function $g(.)$ formally computes a numerator and a denominator. However the denominator can be replaced by its conjugate, and the implied division replaced by a multiplication, exploiting the exponentiation of the final result to the power of $p^{k/2}-1$.

$\boldsymbol{g}(R, P, S, Q, x_p, y_p, \mathbf{x_q}, \mathbf{y_q})$
1. $x, y, z \leftarrow R$
2. $\lambda_n, \lambda_d = R.add(P)$
3. $\boldsymbol{n} = y\lambda_d - \lambda_n(xz - \mathbf{x_q}z^3) - \mathbf{y_q}z^3\lambda_d \cdot i$
4. $\mathbf{x}, \mathbf{y} \leftarrow S$
5. $\lambda = S.add(Q)$
6. $\boldsymbol{d} = i^4 y_p + (\mathbf{y} - \lambda(\mathbf{x} - i^2 x_p)) \cdot i$
7. **return** $(\boldsymbol{n} \cdot \boldsymbol{d})$

The full compressed Weil pairing algorithm can now be given. Note the absence of the final exponentiation. Otherwise the structure is very similar to that of the Tate pairing.

Weil(r, p, P, Q)
1. $\boldsymbol{m} = 1$
2. $R = P$
3. $S = Q$
4. $n = r - 1$
5. $x_p, y_p \leftarrow P$
6. $\mathbf{x_q}, \mathbf{y_q} \leftarrow untwist(Q)$
7. **for** $i \leftarrow \lfloor \lg(r) \rfloor - 2$ **downto** 0 **do**
8. $\quad \boldsymbol{m} = \boldsymbol{m}^2 \cdot \boldsymbol{g}(R, R, S, S, x_p, y_p, \mathbf{x_q}, \mathbf{y_q})$
9. $\quad$ **if** $n_i = 1$ **then** $\boldsymbol{m} = \boldsymbol{m} \cdot \boldsymbol{g}(R, P, S, Q, x_p, y_p, \mathbf{x_q}, \mathbf{y_q})$
10. **end for**
11. $\boldsymbol{m} = \boldsymbol{m}^{(p^{k/2}-1)}$
12. **return** $2\mathbf{m_a}$

Using the same curves as above, we proceeded to obtain timings for the Weil pairing, for the curves using $F = 512$ and $k = 2, 4, 8$. For the Weil pairing precomputation for a constant second parameter Q is particularly rewarding, as the point multiplication on the twisted curve (for $k = 4, 8$) is very expensive – so we include timings for this case.

Table 2. Timings – Pentium IV 3GHz – Weil Pairing

Curve	(kF/G)	k	F (bits)	G (bits)	Time (ms)	with precomp.
$\mathbf{E_{160}}$	(1024/160)	2	512	160	20	–
$\mathbf{E_{192}}$	(2048/192)	4	512	192	37	18
$\mathbf{E_{192bw}}$	(2048/192)	8	256	192	40	23
$\mathbf{E_{224}}$	(4096/224)	8	512	224	90	55

Observe that the Weil pairing is slightly more efficient at the security level (4096/224), and much more efficient if precomputation is possible. It is also worth noting that at the (2048/192) security level, when using precomputation, the Weil pairing is very nearly as efficient as the Tate Pairing.

8 Conclusions

We have demonstrated that using non-supersingular MNT curves we can easily scale the security of pairing based cryptosystems, effectively without limit. The basic algorithm remains the same at all levels of security. The proposed method of doubling the embedding degree to reach the next level of security seems to

compare well with the alternative approach of simple doubling the size of the modulus.

Finally we provide experimental evidence to support the view that at higher levels of security the Weil pairing becomes more efficient than the Tate pairing, and we identify the cross-over point.

References

1. Paulo Barreto. Private communication.
2. Paulo S. L. M. Barreto, Steven Galbraith, Colm O hEigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. `http://eprint.iacr.org/`.
3. 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.
4. 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.
5. 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.
6. 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/`.
7. I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography, Volume 2*. Cambridge University Press, 2005.
8. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
9. 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.
10. 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`.
11. 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`.
12. K. Eisentrager, K. Lauter, and P. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 343–354. 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. M. J. Hinek, M. K. Low, and E. Teske. On some attacks on multi-prime RSA. In *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 385–404. Springer-Verlag, 2002.
15. IEEE Std 1363-2000. Standard specifications for public-key cryptography. IEEE P1363 Working Group, 2000.

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. Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. Cryptology ePrint Archive, Report 2005/076, 2005. `http://eprint.iacr.org/`.
18. RSA Laboratories. PKCS # 1 v2.0 Amendment 1 – Multi-prime RSA, 2000.
19. A. K. Lenstra and E. R. Verheul. The XTR public key system. In *Advances in Cryptology – Crypto'2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 1–19, Santa Barbara, USA, 2000. Springer-Verlag.
20. Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.
21. A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.
22. 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.
23. 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`.
24. J. M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.
25. K. Rubin and A. Silverberg. Supersingular abelian varieties in cryptology. In *Advances in Cryptology – Crypto 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 336–353. Springer-Verlag, 2002.
26. M. Scott, 2002. `ftp://ftp.computing.dcu.ie/pub/crypto/cm.exe`.
27. M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
28. 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/`.
29. 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/`.
30. P. J. Smith and M. J. Lennon. LUC : A new public key system. In *Proceedings of the Ninth IFIP International Symposium on Computer Security '93*, pages 97–111. Elsevier Science Publications, 1994.