On Computing Products of Pairings

R. Granger and N.P. Smart

Dept. Computer Science,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom.
{granger,nigel}@cs.bris.ac.uk

Abstract. In many pairing-based protocols often the evaluation of the product of many pairing evaluations is required. In this paper we consider methods to compute such products efficiently. Focusing on pairing-friendly fields in particular, we evaluate methods for the Weil, Tate and Ate pairing algorithms for ordinary elliptic curves at various security levels. Our operation counts indicate that the minimal cost of each additional pairing relative to the cost of one is ≈ 0.61, 0.45, and 0.43, for each of these pairings respectively at the 128-bit security level. For larger security levels the Ate pairing can have a relative additional cost of as low as 0.13 for each additional pairing.
These estimates allow implementors to make optimal algorithm choices for given scenarios, in which the number of pairings in the product, the security level, and the embedding degree are factors under consideration.

1 Introduction

Let $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ be a non-degenerate bilinear map (or simply pairing) from additive groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to a multiplicative group $\mathbb{G}_T$, all of prime order r. In some protocols the evaluation of products of the form

$$e = \prod_{i=1}^{n} \hat{t}(P_i, Q_i) \qquad (1)$$

is required. For example, in the BBG HIBE scheme [7] ones needs to compute a ratio of two pairings (such a ratio can be computed via a product of two pairings in a trivial manner), in the BBS short group signature scheme [8] one needs at one point to compute a product of three pairings, in the Water's [20] IBE scheme extended to an l-level HIBE or WIBE [1] requires an evaluation of a product of $l+1$ pairing values, in a number of ID-based key distribution protocols one needs to compute a product of two pairing values [9]. A common question asked amongst protocol designers therefore is how efficiently can such a product be computed?

A naive way to compute e is to evaluate each $\hat{t}(P_i, Q_i)$ independently, and then multiply the results. However, since only the entire product is required, and not each individual pairing evaluation, one is free to apply more efficient methods.

All pairing evaluation algorithms currently employed in cryptography are based on elliptic curves, and so make use of Miller's algorithm [18], which can be used to compute both the Weil and Tate pairings and their variants [5, 13, 4]. The algorithm consists of a sequence of elementary operations, or 'Miller operations', which are function-evaluation steps very similar to a point doubling or addition on the Jacobian under consideration. Depending on the pairing being computed, after either one or two such Miller operations are performed, and possibly a final powering depending on the type of pairing computed.

For example, to compute the reduced Tate pairing of points P and Q on an ordinary elliptic curve with even embedding degree, the function-evaluation step for doubling is simply

$$f \leftarrow f^2 \cdot l(Q),$$

where l is the tangent line to the curve at a point, depending on P and the loop iteration in Miller's algorithm. In (1) there are n such terms f_i, and by using a single accumulator f for their product, initialised to 1, each doubling function-evaluation step becomes

$$f \leftarrow f^2 \cdot \prod_{i=1}^{n} l_i(Q_i). \tag{2}$$

Therefore one needs only compute a single squaring in the extension field per doubling, rather than n squarings as with the naive method. By the same argument, in this example one can also trivially combine the final powerings required in each pairing evaluation in (1). Thus the basic idea to improve efficiency is to compute only once those operations common to each pairing evaluation.

In the general scenario, the main question to consider in computing products of a particular pairing is to what extent can one improve upon the naive method? In this note we make some simple observations that attempt to answer this question, for the Weil and Tate pairings, and also the recently defined Tate pairing variant, the Ate [15] pairing.

Following their introduction by Koblitz and Menezes [16], we focus throughout on pairing-friendly fields, since these allow for many optimisations, and have various practical benefits [14]. Furthermore, in order to gain a real idea of how these improvements perform, we also consider several practical security levels.

The paper is organised as follows. In Section 2 we provide a brief description of pairing-friendly fields and their arithmetic. In Section 3 we detail the methods and operation counts for computing products of pairings for our selection of pairing algorithms. Lastly, in Section 4 we give some efficiency comparisons for our choice of finite fields for various security levels.

1.1 Acknowledgements

The authors would like to thank Eike Kiltz for bringing the issue of products of multiple pairings in various applications to our attention, and Frederik Vercauteren who provided comments on an earlier version of this paper.

2 Pairing-Friendly Fields

Koblitz and Menezes [16] introduced the concept of pairing-friendly fields which are particularly well suited to pairing implementations (see also [14]). They are Kummer extensions of $\mathbb{F}_p$ defined by the polynomial

$$f(X) = X^k + f_0 \tag{3}$$

for values of $p \equiv 1 \pmod{12}$ and $k = 2^i 3^j$. Generally one assumes that k is even, which aids in efficiency due to the well-known denominator elimination trick in pairing computations. Following [14, 16], we focus in particular on the cases $k = 6$, 12 and 24. We give here a brief description of the various operations together with their arithmetic costs, and refer the reader to [14] for further details.

Define $\mathbb{F}_{p^k} = \mathbb{F}_p[\theta]/(f(\theta))$, where $f(\theta) = 0$, and for any $k = 2^i 3^j$, let M_k, S_k and I_k denote the time for a multiplication, squaring and inversion respectively in the field $\mathbb{F}_{p^k}$. Here we assume that addition operations take a negligible amount of time.

Using the Karatsuba and Toom-Cook methods for multiplication and squaring, to compute products (resp. squares) of polynomials of degree $2^i 3^j - 1$ over $\mathbb{F}_p$ requires $v(k)$ multiplications (resp. squarings) in the field $\mathbb{F}_p$, where $v(k) = 3^i 5^j$, i.e., $M_k = v(k) M_1$.

Furthermore, we can assume that f_0 in (3) has been chosen so that multiplication by f_0 can be performed quickly by simple additions rather than a full multiplication. With this representation, the Frobenius automorphism can be computed with just a few additions, and inversions can be reduced to a single inversion in $\mathbb{F}_p$, together with a few multiplications in $\mathbb{F}_{p^k}$, see [14] for more details. We also refer to [14] for a discussion of the relative costs of inversion in the fields under consideration.

In the Tate and Ate pairings, one must perform an exponentiation, also known as 'the final powering'. Essentially this need only be performed (see Section 3) in the cyclotomic subgroup of $\mathbb{F}_{p^k}^{\times}$ of order $\Phi_k(p)$, which we denote by $G_{\Phi_k(p)}$. Inversion and the Frobenius operation in $G_{\Phi_k(p)}$ are essentially free, and using special properties of this subgroup, one can improve upon the squaring cost in $\mathbb{F}_{p^k}$, again see [14] for details of this.

For the exponentiation step, using Lucas sequences [19] one can exponentiate in $G_{\Phi_k(p)}$ by an exponent l for a cost of

$$C_{\mathrm{Luc}}(l) = (M_{k/2} + S_{k/2}) \log_2 l.$$

Also, using well known methods (see [14]), one can perform exponentiation via standard signed sliding window methods [6] since inversion is cheap in $G_{\Phi_k(p)}$. If $l \leq p$ then the best way to perform the exponentiation, using windows of width at least i, will take time

$$C_{\mathrm{SSW}}(l) = \overline{S}(1 + \log_2 l) + M_k \left(\frac{\log_2 l}{i+2} + (2^{i-2} - 1) \right)$$

where $\overline{S}$ denotes the time needed to perform a squaring operation in $G_{\Phi_k(p)}$. We also need to store 2^{i-2} elements during the exponentiation algorithm.

When $l \geq p$, as is the case in the final powering of the algorithm to compute the Tate pairing, one uses the fact that we can perform the Frobenius operation on $G_{\Phi_k(p)}$ for free. Thus we write l in base p, and perform a simultaneous exponentiation. Using the techniques of Avanzi [2], we can estimate the time needed to perform such a multi-exponentiation by

$$C_{\mathrm{bigSSW}}(l) = (d + \log_2 p)\overline{S} + \left(d(2^{i-1} - 1) + \frac{\log_2 l}{i+2} - 1 \right) M_k$$

using windows of width i, where $d = \lceil \log_2 l / \log_2 p \rceil$. The precomputation and storage can be reduced using techniques described in [3].

3 Methods for Computing Products of Pairings

In this section we extend the best known algorithms for the computation of the Weil, Tate, and Ate pairings for ordinary elliptic curves defined over pairing-friendly fields, to consider the efficient computation of a product of n pairings in each case.

Let E be an ordinary elliptic curve described by the equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$, for $p \equiv 1 \bmod 12$, and let P be a basepoint of prime order r dividing $\#E(\mathbb{F}_p)$. Assuming E is not anomalous, let k be the multiplicative order of q modulo r, i.e., the embedding degree of E under the MOV attack [17]. For the purposes of this paper we assume k is 6, 12 or 24.

The main ingredient in Miller's algorithm for computing a pairing is the evaluation of certain line functions at certain divisors. For each pairing we consider, the basic step is the evalution of $F_P(Q)$, where F_P is a function whose divisor is equivalent to $r(P) - r(\infty)$, for points P and Q defined over either $\mathbb{F}_{p^k}$ or a proper subfield. It is usual to assume that $P \in E(\mathbb{F}_p)$ and Q is the image of a point in $\overline{E}(\mathbb{F}_{p^e})$, where $\overline{E}$ is either the quadratic or sextic twist of E. In particular it is common to take either the curve $y^2 = x^3 - 3x + b$, or the curve $y^2 = x^3 + b$ which admit twists of degree $d = 2$ and $d = 6$ respectively. Thus we define e via $e = k/d$, since we have assumed that k is a multiple of six.

We follow Koblitz and Menezes' analysis [16] but we do not assume the group order to be a Solinas prime, i.e., one of negligible Hamming weight. Hence in addition to the line computations that arise from a doubling for the function-evaluation step in each loop iteration, we must also consider the line computation arising from an addition, for every set bit in the binary expansion of r. We extended the analysis in this way as we are interested in the cost per addition pairing in the product, which will depend on the heavily on the Hamming weight of r. We now treat each pairing in turn.

3.1 The Weil Pairing

Let F_P and F_Q be functions whose divisors are $r(P) - r(\infty)$ and $r(Q) - r(\infty)$ respectively, which are normalised so that $F_P(\infty)/F_Q(\infty) = 1$. The Weil pairing $\hat{t}$ (see [18]) of $P \neq Q$ is given by

$$(-1)^r \frac{F_P(Q)}{F_Q(P)}. \tag{4}$$

As pointed out by Koblitz and Menezes [16], for even k one can replace $\hat{t}$ by its $(1 - p^{k/2})$-th power to eliminate all subfield terms, as with the Tate pairing. This can be accomplised with just one squaring in $\mathbb{F}_{p^k}$. The function-evaluation steps - also known as 'Miller operarions' - for a single pairing are then

$$f \leftarrow f^2 \cdot l(Q) \quad \text{and} \quad f \leftarrow f \cdot l'(Q), \tag{5}$$

for lines l, l' corresponding to point doubling and addition respectively. Clearly, if one of the points being paired lies in $E(\mathbb{F}_p)$, then the Miller operations become more efficient. In particular, if $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^k})$, then the computation of $F_P(Q)$ is cheaper than that of computing $F_Q(P)$. The computation of the former has thus been dubbed 'Miller-Lite' by Solinas, while the latter we refer to as Miller-Full. By (4), the total cost of a single Weil pairing computation is

$$C_{\text{Weil}} = C_{\text{Lite}} + C_{\text{Full}},$$

where C_{Lite} and C_{Full} represent the respective costs of a Miller-Lite and a Miller-Full operation. Note we have dropped the final squaring and division costs since these are negligible.

To compute the product of n terms $F_{P_i}(Q_i)$, for each bit of r we simply combine the squarings in $\mathbb{F}_{p^k}$ as in (2), and if the bit is set also, multiply in n extra terms arising from the addition line function-evaluation.

It turns out that in computing products of pairings one can achieve extra advantages by assuming that the points P and Q, and all intermediate points within Miller's algorithm, are held in affine coordinates. At first sight this seems to require n inversions in $\mathbb{F}_p$, for the Miller-Lite algorithm, and n inversions in $\mathbb{F}_{p^e}$ for the Miller-Full algorithm. However, using Montgomery's trick [11, Algorithm 10.3.4], one can trade the n inversions in $\mathbb{F}_p$ (resp. $\mathbb{F}_{p^e}$) for $3n-3$ multiplications in $\mathbb{F}_p$ (resp. $\mathbb{F}_{p^e}$) plus one inversion in $\mathbb{F}_p$ (resp. $\mathbb{F}_{p^e}$). For products of a larger number of pairings using affine coordinates will always be more efficient, however for smaller values the exact cross over point depends on the ratio between the cost of an inversion in $\mathbb{F}_p$ to that of a multiplication.

In Table 1, we list the cost of all relevant operations in all cases.

Table 1. Cost of Miller operations per bit in product of n pairings

Operation	d	Cost
Jacobian Projective Coordinates		
Miller-Lite double	2	$(4S_1 + (2e+7)M_1 + M_k)n + S_k$
	6	$(5S_1 + (2e+6)M_1 + M_k)n + S_k$
Miller-Lite add	2/6	$(3S_1 + (2e+10)M_1 + M_k)n$
Miller-Full double	2	$(2eM_1 + 4S_e + 6M_e + M_k)n + S_k$
	6	$(2eM_1 + 5S_e + 6M_e + M_k)n + S_k$
Miller-Full add	2/6	$(2eM_1 + 3S_e + 10M_e + M_k)n$
Affine Coordinates		
Miller-Lite double	2/6	$(2S_1 + (e+6)M_1 + M_k)n - 3M_1 + I_1 + S_k$
Miller-Lite add	2/6	$(S_1 + (e+5)M_1 + M_k)n - 3M_1 + I_1$
Miller-Full double	2/6	$(2S_e + 6M_e + eM_1 + M_k)n - 3M_e + I_eS_k$
Miller-Full add	2/6	$(S_e + 5M_e + eM_1 + M_k)n - 3M_e + I_e$

For sake of a comparison we assume that the signed Hamming weight for the relevant loops is on average 1/3 the length of the loop, using the non-adjacent form [10] of a binary expansion to reduce the number of addition line function evaluations in the loop. Therefore, the total cost for computing the product of n Weil pairings, for group order r, using affine coordinates and $d = 2$, is,

$$((2S_1 + 2S_e + (2e+6)M_1 + 6M_e + 2M_k)n - 3M_1 - 3M_e + I_1 + I_e + 2S_k)\log_2 r$$
$$+ ((S_1 + S_e + (4e+5)M_1 + 6M_e + 2M_k)n - 3M_1 - 3M_e + I_1 + I_e)(\log_2 r)/3.$$

3.2 The Tate Pairing

In computing the Tate pairing one executes one Miller-Lite operation and then performs an exponentiation by $(p^k - 1)/r$ in order to eliminate r-th powers and obtain an r-th root of unity. As observed by Koblitz and Menezes, the exponentiation can be sped up by first exponentiating by $(p^k - 1)/\Phi_k(p)$, which is very cheap since the Frobenius is also, and then exponentiating by $\Phi_k(p)/r$. Once raised to the power $(p^k - 1)/\Phi_k(p)$, the pairing output is an element of $G_{\Phi_k(p)}$ and so the available techniques for fast arithmetic can be utilised. Hence, we can make use of the fast squaring described in Section 2.

By the same argument as for the Weil pairing, the computation of the line functions reduces to (5), and so we can use the same arithmetic for computing the Miller-Lite part of the algorithm.

For the exponentiation we ignore the small cost to power by $(p^k - 1)/\Phi_k(p)$, as this is negligible. Thus a Tate pairing computation requires time

$$C_{\text{Tate}} = C_{\text{Luc}}(\Phi_k(p)/r) + C_{\text{Lite}}$$

or

$$C_{\text{Tate}} = C_{\text{bigSSW}}(\Phi_k(p)/r) + C_{\text{Lite}},$$

whichever is the fastest. Therefore the total cost for computing the product of n Tate pairings, in affine coordinates, is

$$\begin{aligned}&((2S_1 + (e+6)M_1 + M_k)n - 3M_1 + I_1 + S_k)\log_2 r\\ &+ ((S_1 + (e+5)M_1 + M_k)n - 3M_1 + I_1)(\log_2 r)/3\\ &+ \min\{C_{\mathrm{Luc}}(\Phi_k(p)/r), C_{\mathrm{bigSSW}}(\Phi_k(p)/r)\}.\end{aligned}$$

3.3 The Ate Pairing

The Ate [1] pairing was introduced by Hess, Smart and Vercauteren [15] as an extension of the Eta pairing [4] from supersingular, to ordinary elliptic curves. It is particularly suited to the case where $d = 6$, and hence e is smaller than for other curves.

The central idea behind the Ate pairing is rather than compute a function with divisor $r(P) - r(\infty)$ at Q, with $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^k})$, one instead computes a function with divisor

$$(t-1)(Q) - ((t-1)Q) - (t-2)(\infty)$$

evaluated at P, where t is the trace of frobenius of E over $\mathbb{F}_p$. Since the trace of E is $O(\sqrt{p})$, the computational overheads can be smaller than the Tate pairing depending on the security level and embedding degree. The computational overhead is more pronounced when t is small compared to $\sqrt{p}$. For convenience we let $T = t - 1$.

For the standard Ate pairing one computes

$$f_{T,Q}(P)^{(p^k-1)/r},$$

whilst for the twisted Ate pairing one computes

$$f_{T^e,P}(Q)^{(p^k-1)/r}.$$

Note that the length of the loop in computing the Miller function depends on the size of the trace. Hence, one can exploit techniques that allow the construction of curves with trace as small as $\log_2(r)/\phi(k)$ bits, for $k \geq 12$ [12]. We refer the interested reader to [15] for further details regarding the derivation of the Ate pairing.

For this note, we consider both the standard and the twisted Ate pairing, the degree of the relevant twist, and the size of the trace. We assume the signed Hamming weight of T is around $\log_2(t)/3$, and that of T^e is around $e\log_2(t)/3$. Then the cost for computing the product of n Ate pairings can be easily read from Table 1. For example, the cost of computing the product of n Standard Ate pairings with average trace, in affine coordinates, is

$$\begin{aligned}&((2S_e + 6M_e + eM_1 + M_k)n - 3M_e + I_eS_k)\log_2 t\\ &+ ((S_e + 5M_e + eM_1 + M_k)n - 3M_e + I_e)(\log_2 t)/3\\ &+ \min\{C_{\mathrm{Luc}}(\Phi_k(p)/r), C_{\mathrm{bigSSW}}(\Phi_k(p)/r)\},\end{aligned}$$

[1] Pronounced in the natural way, the Ate pairing is called as such because it is effectively the Eta pairing but with the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ reversed.

whilst, the cost of computing the product of n twisted Ate pairings with small trace, i.e. $\log_2 t \approx \log_2 r/\phi(k)$, in affine coordinates, is

$$\begin{aligned} &((2S_1 + (e+6)M_1 + M_k)n - 3M_1 + I_1 + S_k)\,(e \log_2 r)/\phi(k) \\ &+ ((S_1 + (e+5)M_1 + M_k)n - 3M_1 + I_1)\,(e \log_2 r)/(3\phi(k)) \\ &+ \min\{C_{\mathrm{Luc}}(\Phi_k(p)/r), C_{\mathrm{bigSSW}}(\Phi_k(p)/r)\}, \end{aligned}$$

4 Results

Assuming that $S_1 \approx M_1$ and that $I_1/M_1 \approx 10$, in Tables 2 and 3 we list the number of multiplications/squarings in $\mathbb{F}_p$ required to compute the product of n pairings, for each of the AES security levels; that is 80-bits, 128-bits, 192-bits and 256-bits, for embedding degrees $6, 12$ and 24. We also list the ratio of the cost per additional pairing compared to the cost of a single pairing. The table present the affine coordinate versions of the algorithms only. With a ratio of $I_1/M_1 \approx 10$ the affine coordinate versions are always faster as long as $n \geq 2$.

For the parameter sizes we follow the work of [14, 16]. The first set of parameters in each table corresponds to the 80-bit security level, the next two sets of parameters refer to the 128-bit security level, whilst the next two correspond to the 192-bit level, the final three are related to the 256-bit level.

As an example consider the 128-bit security level and the parameter set $k = 12$, $p \approx r \approx 2^{256}$ and $d = 6$. This parameter choice is generally considered to be the most efficient choice at this security level. We see that each extra pairing costs ranges roughly between 32 percent and 61 percent of the cost of one pairing.

At the highest security level and most efficient implementation choice of $k = 24$, $p \approx 2^{640}$, $r \approx 2^{512}$ and $d = 6$, we find that the cost per extra pairing ranges from 13 percent to 63 percent. For protocols in which one needs to compute the product of three pairing values one finds that the Ate pairing with a small value of t can be over three times faster than using the Tate pairing. It is also over five times faster than naively computing the three Tate pairings independently and then multplying the result.

References

1. M. Abdalla, D. Catalano, A.W. Dent, J. Malone-Lee, G. Neven and N.P. Smart. Identity-based encryption gone wild. In *ICALP 2006*, Springer-Verlag LNCS XXXX, XXXX–XXXX, 2006.
2. R.M. Avanzi. On Multi-exponentiation in cryptography. Preprint, Cryptology ePrint Archive, Report 2002/154, 2002.
3. R. M. Avanzi and P. Mihailescu. Generic efficient arithmetic algorithms for PAFFs (Processor Adequate Finite Fields) and related algebraic structures. In *Selected Areas in Cryptology – SAC 2003*, Springer-Verlag LNCS 3006, 320–334, 2004.
4. P. Barreto, S. Galbraith, C. Ó hÉigearaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. Preprint, Cryptology ePrint Archive, Report 2004/375.

5. P. Barreto, H. Kim, B. Lynn and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In *Advances in Cryptology – CRYPTO 2002*, Springer-Verlag LNCS 2442, 354–368, 2002.
6. I.F. Blake, G. Seroussi and N.P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.
7. D. Boneh, X. Boyen and E.-G. Goh. Hierarchichical identity based encryption with constant size ciphertext. In *Advances in Cryptology – EUROCRYPT 2005*, Springer-Verlag LNCS 3494, 440–456, 2005.
8. D. Boneh, X. Boyen and H. Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, Springer-Verlag LNCS 3152, 41–55, 2004.
9. M.Z. Cheng, L. Chen and N.P. Smart. A built-in decisional function and security proof of ID-based key agreement protocols from pairings. Preprint, 2006.
10. W. Clark and J. Liang. On arithmetic weight for a general radix representation. *IEEE Trans. Info. Theory*, **19**, 823–826, 1973.
11. H. Cohen. *A course in computational algebraic number theory.* Springer-Verlag, GTM 139, 1993.
12. P. Duan, S. Cui and C.W. Chan. Special polynomial families for generating more suitable elliptic curves for pairing-based cryptosystems. Preprint, Cryptology ePrint Archive, Report 2005/342, 2005.
13. S. Galbraith, K. Harrison and D. Soldera. Implementing the Tate pairing. In *Algorithmic Number Theory Symposium – ANTS V*, Springer-Verlag LNCS 2369, 324–337, 2002.
14. R. Granger, D. Page and N.P. Smart. *High security pairing-based cryptography revisited.* In *Algorithmic Number Theory Symposium – ANTS VII*, Springer-Verlag LNCS XXXX, XXXX–XXXX, 2006.
15. F. Hess, N.P. Smart and F. Vercauteren. The Eta pairing revisited. Preprint, Cryptology ePrint Archive, Report 2006/110, 2005
16. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In *Cryptography and Coding*, Springer-Verlag LNCS 3796, 13–36, 2005.
17. A. J. Menezes, T. Okamoto and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Info. Theory*, **39**, 1639–1646, 1993.
18. V.S. Miller The Weil Pairing, and its efficient calculation. *Journal of Cryptology*, **17**, 235–261, 2004.
19. M. Scott and P.S.L.M. Barreto. Compressed pairings. In *Advances in Cryptology – CRYPTO 2004*, Springer-Verlag LNCS 3152, 140–156, 2004.
20. B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT 2005*, Springer-Verlag LNCS 3494, 114–127, 2005.

Table 2. Cost of the various algorithms for a product of n pairings, $d = 2$

Security level	Algorithm	Cost for n pairings	Cost per extra pairing
$k = 6$	Weil	17256 n+ 8208	0.678
$r \approx 2^{160}$	Tate	5432 n+ 5491	0.497
$p \approx 2^{160}$	Ate (Average t)	5920 n+ 3760	0.612
	Ate (Small t)	5920 n+ 3760	0.612
	Twisted Ate (Average t)	8160 n+ 7440	0.523
	Twisted Ate (Small t)	8160 n+ 7440	0.523
$k = 6$	Weil	27624 n+ 13136	0.678
$r \approx 2^{256}$	Tate	8696 n+ 12817	0.404
$p \approx 2^{512}$	Ate (Average t)	18944 n+ 13502	0.584
	Ate (Small t)	9472 n+ 10046	0.485
	Twisted Ate (Average t)	26112 n+ 25278	0.508
	Twisted Ate (Small t)	13056 n+ 15934	0.450
$k = 12$	Weil	75710 n+ 25086	0.751
$r \approx 2^{256}$	Tate	19949 n+ 22899	0.466
$p \approx 2^{256}$	Ate (Average t)	27904 n+ 14582	0.657
	Ate (Small t)	13952 n+ 11787	0.542
	Twisted Ate (Average t)	59904 n+ 50720	0.542
	Twisted Ate (Small t)	29952 n+ 29856	0.501
$k = 6$	Weil	41472 n+ 19712	0.678
$r \approx 2^{384}$	Tate	13056 n+ 26827	0.327
$p \approx 2^{1365}$	Ate (Average t)	50468 n+ 35897	0.584
	Ate (Small t)	14208 n+ 22667	0.385
	Twisted Ate (Average t)	69598 n+ 67293	0.508
	Twisted Ate (Small t)	19584 n+ 31499	0.383
$k = 12$	Weil	113664 n+ 37632	0.751
$r \approx 2^{384}$	Tate	29952 n+ 44412	0.403
$p \approx 2^{683}$	Ate (Average t)	74338 n+ 38439	0.659
	Ate (Small t)	20928 n+ 27740	0.430
	Twisted Ate (Average t)	159822 n+ 134877	0.542
	Twisted Ate (Small t)	44928 n+ 54844	0.450
$k = 6$	Weil	55248 n+ 26272	0.678
$r \approx 2^{512}$	Tate	17392 n+ 44876	0.279
$p \approx 2^{2560}$	Ate (Average t)	94720 n+ 66982	0.586
	Ate (Small t)	18944 n+ 39334	0.325
	Twisted Ate (Average t)	130560 n+ 125862	0.509
	Twisted Ate (Small t)	26112 n+ 51110	0.338
$k = 12$	Weil	151420 n+ 50172	0.751
$r \approx 2^{512}$	Tate	39898 n+ 70264	0.362
$p \approx 2^{1280}$	Ate (Average t)	139520 n+ 70397	0.665
	Ate (Small t)	27904 n+ 48040	0.367
	Twisted Ate (Average t)	299520 n+ 251090	0.544
	Twisted Ate (Small t)	59904 n+ 84178	0.416
$k = 24$	Weil	435844 n+ 121872	0.781
$r \approx 2^{512}$	Tate	105370 n+ 183035	0.365
$p \approx 2^{640}$	Ate (Average t)	206720 n+ 139115	0.598
	Ate (Small t)	41344 n+ 115136	0.264
	Twisted Ate (Average t)	791040 n+ 663381	0.544
	Twisted Ate (Small t)	158208 n+ 219989	0.418

Table 3. Cost of the various algorithms for a product of n pairings, $d = 6$

Security level	Algorithm	Cost for n pairings	Cost per extra pairing
$k = 6$	Weil	10012 n+ 7782	0.563
$r \approx 2^{160}$	Tate	5006 n+ 5491	0.477
$p \approx 2^{160}$	Ate (Average t)	2506 n+ 3546	0.414
	Ate (Small t)	2506 n+ 3546	0.414
	Twisted Ate (Average t)	2506 n+ 3546	0.414
	Twisted Ate (Small t)	2506 n+ 3546	0.414
$k = 6$	Weil	16028 n+ 12454	0.563
$r \approx 2^{256}$	Tate	8014 n+ 12817	0.385
$p \approx 2^{512}$	Ate (Average t)	8021 n+ 12819	0.385
	Ate (Small t)	4010 n+ 9704	0.292
	Twisted Ate (Average t)	8021 n+ 12819	0.385
	Twisted Ate (Small t)	4010 n+ 9704	0.292
$k = 12$	Weil	42286 n+ 27132	0.609
$r \approx 2^{256}$	Tate	18585 n+ 22899	0.448
$p \approx 2^{256}$	Ate (Average t)	11861 n+ 15605	0.432
	Ate (Small t)	5930 n+ 12298	0.325
	Twisted Ate (Average t)	18602 n+ 22901	0.448
	Twisted Ate (Small t)	9301 n+ 15946	0.368
$k = 6$	Weil	24064 n+ 18688	0.563
$r \approx 2^{384}$	Tate	12032 n+ 26827	0.310
$p \approx 2^{1365}$	Ate (Average t)	21369 n+ 34078	0.385
	Ate (Small t)	6016 n+ 22155	0.214
	Twisted Ate (Average t)	21369 n+ 34078	0.385
	Twisted Ate (Small t)	6016 n+ 22155	0.214
$k = 12$	Weil	63488 n+ 40704	0.609
$r \approx 2^{384}$	Tate	27904 n+ 44412	0.386
$p \approx 2^{683}$	Ate (Average t)	31599 n+ 41166	0.434
	Ate (Small t)	8896 n+ 28508	0.238
	Twisted Ate (Average t)	49631 n+ 60657	0.450
	Twisted Ate (Small t)	13952 n+ 33980	0.291
$k = 6$	Weil	32056 n+ 24908	0.563
$r \approx 2^{512}$	Tate	16028 n+ 44876	0.263
$p \approx 2^{2560}$	Ate (Average t)	40106 n+ 63568	0.387
	Ate (Small t)	8021 n+ 38651	0.172
	Twisted Ate (Average t)	40106 n+ 63568	0.387
	Twisted Ate (Small t)	8021 n+ 38651	0.172
$k = 12$	Weil	84572 n+ 54264	0.609
$r \approx 2^{512}$	Tate	37170 n+ 70264	0.346
$p \approx 2^{1280}$	Ate (Average t)	59306 n+ 75516	0.440
	Ate (Small t)	11861 n+ 49063	0.195
	Twisted Ate (Average t)	93013 n+ 111996	0.454
	Twisted Ate (Small t)	18602 n+ 56359	0.248
$k = 24$	Weil	240756 n+ 142332	0.628
$r \approx 2^{512}$	Tate	99914 n+ 183035	0.353
$p \approx 2^{640}$	Ate (Average t)	88106 n+ 151915	0.367
	Ate (Small t)	17621 n+ 117696	0.130
	Twisted Ate (Average t)	250026 n+ 293887	0.460
	Twisted Ate (Small t)	50005 n+ 146090	0.255