On the Efficient Implementation of Pairing-Based Protocols

Michael Scott *

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

Abstract. The advent of Pairing-based protocols has had a major impact on the applicability of cryptography to the solution of more complex real-world problems. However there has always been a question mark over the performance of such protocols. In response much work has been done to optimize pairing implementation, and now it is generally accepted that being pairing-based does not preclude a protocol from consideration as a practical proposition. However although a lot of effort has gone into the optimization of the stand-alone pairing, in many protocols the pairing calculation appears in a particular context within which further optimizations may be possible. It is the purpose of this paper to bridge the gap between theory and practise, and to show that even complex protocols may have a surprisingly efficient implementation. We also point out that in some cases the usually recommended pairing friendly curves may not in fact be optimal. We claim a new record with our implementation of a pairing at the AES-256 bit level.
Keywords: Pairing implementation.

1 Introduction

Pairings, with their richer structure, allow solution to problems for which traditional public key methods could offer no solution. The classic example would be Identity-based Encryption, long known as a concept, stubbornly resistant to realisation using standard methods, yet easy to construct using pairings.

Other problems of interest, like short signature schemes [13], non-interactive key exchange [33], attribute-based cryptography [38] (which has particular significance for security in the context of cloud computing [2]), and public key encryption with key-word search [12], also succumbed. These protocols, although often proven secure under reasonable assumptions, were becoming ever more complex, outstripping the ability of implementers to keep up. Indeed in parallel with these developments optimal methods for calculating bilinear pairings and ancillary operations were also being developed. To give an idea of the current

* Research supported by the Claude Shannon Institute, Science Foundation Ireland Grant 06/MI/006

state-of-the-art, see [9] where a pairing at the equivalent of AES-128 bit security is calculated on the standard contemporary computing platform in less than a millisecond. See also [3].

However depending on the context in which the pairing is needed within a particular protocol further optimizations may also be possible, and it is the purpose of this paper to highlight these possibilities and measure their impact. We also give some performance statistics for practical implementation of some protocols.

2 Pairings

There are, it is generally accepted, 4 varieties of pairing, of which the Type-1 and Type-3 are the most common, and the most practical for implementation purposes [18]. Here we restrict ourselves to Type-1 and Type-3 pairings implemented on elliptic curves. Note that while pairing-based protocols are commonly described in the context of a Type-1 pairing, most can be ported to the Type-3 setting. This will of course impact on the security assumptions, but that is not a concern here.

The notation for the pairing is $e(P, Q)$. We will refer to P as the left-hand argument and Q as the right-hand argument. The pairing itself evaluates as an element in the k-th extension of the underlying finite field $\mathbb{F}_q$, where k is called the embedding degree, which is a fixed parameter associated with the chosen curve. Note that only elliptic curves $E(\mathbb{F}_q)$ with reasonably small values of embedding degrees are of interest here – such curves are called pairing-friendly.

For sensible choices, at least one of P or Q is a point in $E(\mathbb{F}_q)$, and the other may be represented as a point in $E(\mathbb{F}_{q^d})$, where d is an exact divisor of k. For security it is important the discrete logarithm problems in both the elliptic curve fields (ECDLP) and the k-th extension of the finite field (DLP) should be of equal complexity. Indeed as pointed out some years ago by Menezes, Okamoto and Vanstone [26] (see also [17]), the former problem can be easily converted to the latter on pairing-friendly curves. Unfortunately index calculus methods apply in the finite field setting, so parameter sizes must be adjusted upwards to take this into account. We must also be wary of a direct Pollard-rho attack on the ECDLP problem. For a survey of the discrete logarithm problem in all of its settings, see [29].

Therefore in an ideal world, for AES-128 security, one would like the elliptic curve group size to be 256-bits embedded in a field of the same size (to defend against Pollard-rho), and an extension degree of 12 to raise the size of the finite field DLP problem to 3072 bits (to defend against index calculus). In fact using a member of the well known Barreto-Naehrig family of pairing friendly curves [7], we can achieve exactly that.

Type-1 pairings, denoted $G_1 \times G_1 \rightarrow G_T$, are implemented on supersingular elliptic curves over $\mathbb{F}_p$, $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$, with maximum embedding degrees of 2, 4 and 6 respectively. In a Type-1 pairing both P and Q are points over the base field G_1, and the pairing supports the property of symmetry, that is $e(P, Q) =$

$e(Q, P)$, so left and right hand arguments can be swapped at will. A problem with Type-1 pairings is that none are known which are as efficient at the AES-192 and AES-256 levels of security as the Type-3 equivalent, primarily as none support embedding degrees greater than 6 (on elliptic curves).

Type-3 pairings, here denoted $G_2 \times G_1 \rightarrow G_T$, are much more numerous and support any embedding degree. They are only known on elliptic curves over fields of large prime characteristic $E(\mathbb{F}_p)$. Usually we prefer those that support the maximal twist (minimizing d above), and those which support the required size of group (for the chosen level of security) in the smallest possible field. This feature is captured in the ρ parameter of the pairing-friendly curve – see the Freeman-Scott-Teske taxomony of pairing-friendly curves for more details [16]. There such curves are referred to as curves which support "efficient arithmetic". For a Type-3 pairing one of P or Q is in the larger field G_2, a point on $E(\mathbb{F}_{q^d})$, and the symmetry property does not hold.

Another matter of choice is that of which pairing function to implement. The Tate pairing was initially preferred, until the discovery of truncated loop variants like the η_T pairing [6], which was more efficient over fields of small characteristic. In the Type-3 setting the situation is a little more complex. In the Miller algorithm the left-hand argument, in the course of the calculation, undergoes a point multiplication by a fixed system parameter. The best truncated loop variants of the Tate pairing in the Type-3 setting are the ate pairing [20], or the R-ate pairing [23]. These can achieve the maximum loop reduction, by a factor of $\phi(k)$, where $\phi(.)$ is the Euler totient function [37]. In both cases the left-hand parameter must, unfortunately, be the element in the larger field $E(\mathbb{F}_{p^d})$. However this only partially offsets the advantage of optimal loop reduction.

Algorithm 1 Computation of basic ate pairing $e(P, Q)$ using Miller's algorithm

INPUT: $P \in E(\mathbb{F}_{p^d}), Q \in E(\mathbb{F}_p)$, trace t, even k, where P has order r
OUTPUT: $e(P, Q)$
1: $T \leftarrow P$, $m \leftarrow 1$
2: $n \leftarrow t - 1$
3: **for** $i \leftarrow \lfloor \lg(n) \rfloor - 1$ **downto** 0 **do**
4: $\quad m \leftarrow m^2.l_{T,T}(Q)$
5: $\quad T \leftarrow 2T$
6: $\quad$ **if** $n_i = 1$ **then**
7: $\quad\quad m \leftarrow m.l_{T,P}(Q)$
8: $\quad\quad T \leftarrow T + P$
9: $\quad$ **end if**
10: **end for**
11: $m \leftarrow m^{(p^k-1)/r}$
12: **return** m

In the Miller algorithm the line function $l_{A,B}(Q)$ is a distance calculated between the fixed point Q and the lines that arise when adding B to A on the elliptic curve in the standard way.

A final choice is projective or affine coordinates for the representation of elliptic curve points. Recall that the latter representation requires a single base field modular inversion for each point addition or doubling. However over $\mathbb{F}_{p^d}$, for larger d, this becomes less significant than multiplications over the full extension field. Also the line functions required for Miller's algorithm are simpler to calculate using affine coordinates. Recent work indicates that for embedding degrees less than 12 projective coordinates may be optimal, but that above that affine may be preferred [22]. However this decision depends to an extent on the chosen platform [1]. For BN curves we certainly lean towards projective coordinates [3].

3 Optimizations

The pairing itself is open to some interesting optimisations, depending on its context. These often involve precomputation, and so in some contexts storage availability may become an issue. As we will see these optimizations can interact in interesting ways.

If both left-hand and right-hand arguments are fixed, then clearly the pairing itself can be precalculated and stored. If the left-hand parameter is fixed, then its multiples that arise in its multiplication by the fixed loop variable can be precalculated offline and stored in affine coordinates. We will call this the "fixed argument" optimization, first pointed out by Scott [34] and recently analysed in more detail by Costello and Stebila [15]. No advantage can be taken of a fixed right-hand parameter, but for a Type-1 pairing only, symmetry can be exploited to move it to the left-hand side and precalculate its multiples as before. For a protocol on a Type-3 pairing it might well be worthwhile to consider reversing the roles of the left-hand and right-hand parameters in the protocol, in order to get the precomputable variable (if there is one) on the left side of the pairing.

Note that if storage is not an issue, and the left-hand parameter is fixed, then the size of d in $E(\mathbb{F}_{q^d})$ doesn't really matter, and so it is no longer important to use a pairing-friendly curve with the maximal twist. However it will be advantageous to use the pairing which gives us the maximal loop reduction, and the extra storage required for precomputation is often offset by the degree of loop reduction that can be achieved [15]. Therefore the possibility of precomputation may impact our choice of pairing-friendly curve. We will return to this point later.

A second raft of optimisations is associated with the calculation of products of pairings, as arise in many protocols. As first pointed out by Solinas [36] and Scott [34] and expanded upon by Granger and Smart [19] three major optimisations are possible in this setting. All of the pairings can share the same Miller variable m, and all can share the final exponentiation. Also Montgomery's trick can be used to carry out all of the point additions/doublings for all of the pairings simultaneously in affine coordinates, while sharing just one modular inversion. If products of pairings play a major part in a protocol this may be a further incentive to use affine coordinates through-out. However if most or all of the left-hand parameters can be precomputed on, this optimization becomes less interesting.

In this case the inclusion of one more pairing to a product of pairings typically carries only the extra cost of n sparse k-th extension field multiplications, if there are n iterations required in the main Miller loop.

Well-known precomputation optimizations apply to powering fixed elements in finite fields, and multiplying fixed points on elliptic curves. These may be applicable in some steps of a pairing-based protocol, and should not be overlooked.

It is interesting to observe however that precomputation optimizations do not really apply to Type-1 pairings over fields of small characteristic. This is related to the fact on a supersingular curve $E(\mathbb{F}_{2^m})$ (for example) point doublings require only field squarings, and these are very fast (asymptotically and in practise) compared to field multiplications. Therefore cheap multiples of a point may be calculated efficiently on-the-fly rather than needing to be precalculated. Also in our experience such Type-1 pairings tend to result in smaller implementations, and so they may be considered a better choice for the space-constrained environment where precomputation would not be an option anyway [30]. In the sequel we will concentrate on the Type-3 pairing scenario.

4 Pairing friendly curves

We will use a variety of pairing-friendly curves to cover all common security levels, corresponding to common AES-like security levels of 80, 128, 192 and 256 bits. All are associated with an even embedding degree k, as the denominator-elimination optimization is important [5]. The curves used are given in Table 1. These curves have been obtained from a number of sources. For a review of the available pairing friendly curves see the Freeman-Scott-Teske taxonomy [16]. The curve CP-80 is a Cocks-Pinch curve [10], MNT-80 is a Miyaji-Nakabayashi-Takano curve [27], BN-128 is a Barreto-Naehrig curve [7], KSS-192 is a Kachisa-Schaefer-Scott curve [21], and finally BLS-256 is a Barreto-Lynn-Scott curve [4]. These have been chosen to conservatively satisfy the above levels of security in terms of the difficulty of the various discrete logarithm problems which defend pairing-based cryptography from attack. In all cases the size of $k.\lg p$ is within the limits given in [16], Table 1.1. Note that the choice of curve for AES-192 security is not straightforward as there is not an obvious best choice amongst the available pairing-friendly curves. A KSS curve with $k = 16$ would be a contender, as would a BN curve with a group size of 640-bits [32]. Some tests indicate that our choice is slightly better than the BN alternative, and requires a smaller G_1. Our choice also supports a sextic twist on G_2. For an alternative point of view, see [28].

Extension fields of degree k, in which the bulk of the pairing computation takes place, are built using a tower of extensions - see [8] for details.

5 A simple example - Boneh-Boyen IBE

To illustrate these points let us first look at a relatively simple protocol and see which optimizations apply. Note that in a pairing based protocol we should first

Table 1. Pairing-friendly curves

Curve	G_2 type	k	Modulus (bits)	ρ	Towering	Pairing
CP-80	$\mathbb{F}_p$ (projective)	2	512	3.2	1-2	Tate
MNT-80	$\mathbb{F}_{p^3}$ (affine)	6	160	1	1-2-6/1-3-6	ate
BN-128	$\mathbb{F}_{p^2}$ (projective)	12	256	1	1-2-4-12	R-ate
KSS-192	$\mathbb{F}_{p^3}$ (affine)	18	512	1.33	1-3-6-18	R-ate
BLS-256	$\mathbb{F}_{p^4}$ (affine)	24	640	1.25	1-2-4-8-24	ate

look at fixed system parameters, that are often introduced in the Setup part of the protocol. These are obvious candidates for precomputation. A second good place to look is at the private keys of the users, created often in the Keygen or Extract step of the protocol. Clearly a particular individual is free to precompute on their own fixed private key.

Our first example is the BB_1 Identity Based Encryption scheme from Boneh and Boyen, as described in section 4.3 of [11] (the full version).

Setup Select random points $P \in G_2$ and $Q \in G_1$. If the pairing friendly group is of size r, then pick random group elements α, β and $\delta \in \mathbb{Z}_r$. Set $Q_a = \alpha Q$ and $Q_d = \delta Q$. Compute $v = e(P,Q)^{\alpha\beta}$. Finally choose a hash function $H_1(.)$ which hashes an element in G_T to a string of length m bits, and a second hash function $H_2(.)$ which hashes an identity string to a group element. The public parameters are $\{Q, Q_a, Q_d, v\}$, and the master key is $\{P, \alpha, \beta, \delta\}$.

Extract Given the master key and an identity ID, generate a random w and extract the private key as $D_0 = (\alpha\beta + w(\alpha H_2(ID) + \delta))P$ and $D_1 = wP$.

Encrypt Given ID and a message M of length m, pick a random s and output $C = \{M \oplus H_1(v^s), sQ, sQ_d + sH_2(ID)Q_a\}$

Decrypt To decrypt the ciphertext $C = \{C_0, C_1, C_2\}$ using the private key $\{D_0, D_1\}$, output $M = C_0 \oplus H_1(e(D_0, C_1)/e(D_1, C_2))$

As pointed out by the authors, this protocol is remarkable to the extent that it can benefit from multiple precomputation optimizations. However the fixed-argument optimization was overlooked. To benefit from this on a Type-3 pairing, the precomputable variable (in this case the private key) must appear on the left side of the pairing. Indeed the applicability of this optimization helps fix the optimal role assignment in many pairing-based protocols on Type-3 curves. Costello and Stebila did consider this optimization for this protocol (Table 5 [15]), but not in conjunction with the multi-pairing optimization which also applies in the decryption step. Granger and Smart [19] estimated the gain from using a multi-pairing, but not in the context where the fixed-argument optimization applied. Finally Lauter, Montgomery and Naehrig [22] did consider both optimizations in combination (see also [1]), but not in the context of a particular protocol.

To summarise for this protocol, all point multiplications and extension field exponentiations in the Extract and Encrypt steps can benefit from "fixed-point"

optimizations. In the Decrypt step the quotient of pairings can be trivially replaced with a product, and so both the multi-pairing and fixed-argument optimizations apply.

To measure performance we present timings for an actual implementation [35], for each step of this protocol. See table 2. All of the code and implementations described here can be found in [35], along with details of the pairing-friendly curves used. Our hardware is a 64-bit Intel i5 520M, clocked at 2.4GHz. Our implementation runs on a single core, and uses a mixture of C++, C and some automatically generated assembly language. We acknowledge that hand-crafted assembly will always do better, sometimes a lot better [3], especially at lower levels of security.

We use an 8-bit window size for fixed-point optimizations. For the fixed-argument optimization we do not exploit the merging optimization as described in [15]. However exploiting this idea (at the cost of doubling the storage requirement) would give a further small but useful speed-up.

Table 2. BB_1 protocol - timings in milliseconds

Curve	CP-80		MNT-80		BN-128		KSS-192		BLS-256	
with/without precomp	w	w/o	w	w/o	w	w/o	w	w/o	w	w/o
Extract	0.207	1.020	0.663	2.239	0.363	0.854	5.265	10.237	10.360	30.598
Encrypt	0.366	1.695	0.194	0.767	0.653	1.646	3.625	8.596	8.110	29.460
Decrypt (2 pairings)	1.213	2.360	1.392	3.788	4.097	4.680	36.438	41.090	65.846	73.469
Decrypt (multi-pairing)	0.834	1.991	1.043	3.383	2.533	3.106	20.614	25.221	36.900	44.035

The most striking conclusion to be drawn from this table is that by exploiting both the fixed-argument and multi-pairing precomputation optimizations, for this protocol we basically get the two pairings for the price of one (or less). In the case of the MNT curve, the gain is even more striking. For these curves only a quadratic twist is possible for G_2, where point multiplication is therefore particularly expensive.

Note that the latency of both the encryption and extract stages can be further optimized by exploiting the fact that a lot of the calculation can be carried out offline, prior to the presentation of the inputs. For example the encrypting entity can compute and maintain offline a pool of tuplets $\{s, v^s, sQ, sQ_d\}$ prior to receiving the the message and ID. In this way the online part of the calculation reduces to a single fixed-point multiplication. The same applies to the extract phase by precomputing tuplets $\{w, wP\}$.

6 Attribute based cryptography

Next we consider a much more elaborate protocol, a ciphertext-policy attribute-based encryption scheme due to Waters [38]. Our first problem is that the protocol ([38] section 3) is described in the context of a Type-1 pairing. However this protocol has already been implemented on an MNT curve by Akinyele et al., [2] so we follow the example of their conversion to a Type-3 setting.

Each participant in this scheme is equipped with a set of attributes. When a message is encrypted, it is encrypted in such a way that it is only acessible to recipients who possess a particular combination of attributes. This combination of attributes is described in a logical fashion, which can be put into the form of a tree-like structure. The attributes form the leaf-nodes of the tree. Each attribute might appear at several leaf nodes. Each node evaluates as true or false, and these results eventually arrive at the root of the tree, which itself evaluates as true or false. If a recipient's attributes satisfy this access structure, then they can decrypt the ciphertext.

This tree structure can in turn be converted into an LSSS matrix, which is required by the protocol. For this we use the method of Liu and Cao [25]. The number of rows in the matrix is the number of leaf nodes of the access tree. Each row of the matrix is associated with an (not necessarily distinct) attribute by a function $f(.)$. If attribute j is associated with row i of the LSSS matrix, then $j = f(i)$. A secret value s will be hidden using a secret sharing operation using the LSSS matrix, and can only be recovered by a valid recipient. The satisfiability of the access structure is reflected in the ability to find the inverse of the subset of the LSSS matrix associated with the attributes available to a recipient, who can then regenerate s. The protocol as described here includes an optimized Decrypt step found in the implementation associated with [2].

Setup Select random points $P \in G_2$ and $Q \in G_1$. If the pairing friendly group is of size r, then pick random group elements α and $\delta \in \mathbb{Z}_r$. Set $P_d = \delta P$, $Q_d = \delta Q$, $Q_\alpha = \alpha Q$ and $v = e(P,Q)^\alpha$. For each of the U attributes in the system, generate a random $H_i \in G_2$. The public parameters are $\{P, Q, v, P_d, Q_d, H_1 \ldots H_U\}$. The master key is Q_α.

Encrypt The inputs are a message M, and an $m \times n$ LSSS matrix S. Generate a random vector $\bar{u} = (s, y_2, \ldots y_n) \in \mathbb{Z}_r$. Then calculate the m vector $\bar{\lambda} = S.\bar{u}$ and generate another random m vector $\bar{x} \in \mathbb{Z}_r$. Calculate the ciphertext as $C_t = Mv^s$, $C_d = sP$, and for i equal 1 to m calculate $C_i = \lambda_i P_d - x_i H_{f(i)}$ and $D_i = x_i Q$. Note that the same attribute may be associated with different indices i.

Keygen The inputs are the master key and a set of ℓ attributes A assigned to an individual. Pick a random group element $t \in \mathbb{Z}_r$ and create a private key as $K = Q_\alpha + tQ_d$, $L = tQ$ and $K_i = tH_i$ for each possessed attribute $i \in A$.

Decrypt First reduce the matrix S by removing rows associated with attributes that are not in A and remove redundant all-zero columns from the matrix. Next calculate the vector $\bar{\omega}$ which in the first row of S^{-1}. For a reasonable

number of attributes, the ω_i will be very small integers. (The shared secret $s = \bar{\omega}.\bar{\lambda}$.) Set all $C_j \leftarrow \omega_j C_j$ and $D_j \leftarrow \omega_j D_j$. Where the same attribute is associated with more than one row of the S matrix, combine the associated C_j and D_j values by simply adding them. (We exploit bilinearity as $e(K_i, D_j).e(K_i, D_k) = e(K_i, D_j + D_k)$, and rewrite $D_i = D_j + D_k$). Finally recover the message as

$$M = C_t.e(C_d, -K)e(\sum_{i \in A} C_i, L) \prod_{i \in A} e(K_i, D_i)$$

Casting an implementer's eye over the above we observe that

1. All point multiplications and extension field exponentiations in the Encrypt and Extract steps benefit from the fixed-point optimization.
2. The latency of both Encrypt and Keygen steps can benefit from the off-line maintainance of precomputed pools of data, in the former case tuples of $\{s, sP, v^s, x_1, x_1Q, x_1H_1, \ldots x_1H_U, x_2, \ldots x_UH_U\}$ and in the latter tuples of $\{t, tQ, tQ_d, tH_1 \ldots tH_U\}$. Now Keygen requires essentially no work at all other than plucking values from the pool and a single point addition, and encryption requires an on-line calculation of just m fixed-point multiplications, if there are m leaf nodes in the access tree.
3. We have assigned roles to G_1 and G_2 to facilitate the fixed-argument precomputation for the potentially expensive Decrypt step. As can be seen decryption boils down primarily to a multi-pairing of $\ell + 2$ pairings. For the first two of these the precomputable parameter (K and L) unfortunately falls on the "wrong side" of the pairing. Interestingly on a Type-1 pairing over a field of prime characteristic, symmetry could be expoited to move these over to the other side. However the ℓ pairings associated with the potentially large number of attributes can all profit from precomputation.

We provide timings for an implementation in the context of a fairly elaborate but arbitrarily chosen access structure with a total of $U = 12$ attributes. The LSSS matrix has $m = 14$ rows (two of the attributes appear twice), and the recipient has $\ell = 6$ attributes, which are sufficient to satisfy the access tree. We also show decryption times where an extra attribute is required, increasing the number of pairings in the multi-pairing by 1. Note the Encryption and Keygen timings are for the complete operation, without exploiting the possibility of off-line maintainance of precomputed pools. One surprising observation is that the Encrypt step for this implementation of this protocol is actually more time-consuming than the pairing-heavy Decrypt step. This goes counter to the received wisdom.

For a contrived system with 20 attributes, all of which are required for decryption, for the BN-128 curve it takes 9.93ms for encryption and 13.05ms for decryption. This is more than 30 times faster (and at a higher level of security) than the timings reported in [2] on the same processor, and significantly faster than the results reported recently by Acar et al. ([1], Table 3).

Table 3. Waters CP-ABE protocol - timings in milliseconds

Curve	CP-80	MNT-80	BN-128	KSS-192	BLS-256
Encrypt	4.57	10.61	6.96	84.16	164.88
Keygen	1.25	2.24	1.42	17.06	33.03
Decrypt (6 attributes)	3.26	5.63	6.18	39.28	66.91
Decrypt (7 attributes)	3.41	6.03	6.68	41.54	71.04

7 Inner-Product Predicate Encryption

Finally we consider a complex Inner-Product Predicate Encryption scheme based on bilinear maps with prime-order groups, recently proposed by Park [31]. We omit the details of the protocol (see page 251 of [31]), which although described in the setting of a Type-1 pairing can easily be converted to use a Type-3 pairing. Our implementation exploits all of the precomputation possibilities. Fixed point optimizations apply to all stages of the Encrypt step, and the fixed-argument optimization applies to the Decrypt step. The Encrypt step calculations take place in G_1 (and are therefore unaffected by the size of G_2). At first glance this protocol appears to be potentially quite impractical. For an implementation supporting 10 attributes the decryption step requires the computation of the product of 42 pairings. However as can be seen from table 4 the timings achieved are surprisingly reasonable.

Table 4. Park's IPE protocol (10 attributes) - timings in milliseconds

Curve	CP-80	MNT-80	BN-128	KSS-192	BLS-256
Encrypt	13.02	4.72	9.05	32.37	61.84
Decrypt	9.49	16.23	22.22	112.76	188.47

8 Discussion

The applicability of these optimizations potentially has an impact on the optimal choice of pairing. In particular if the fixed-argument optimization applies to the pairing, then all required multiples of points on $E(\mathbb{F}_{q^d})$ can be precalculated, and thus there is no longer any major benefit in d being minimal. However while this is true in the context of a protocol step involving pairings, if G_2 becomes larger this may have a negative impact on the performance of other phases of a particular protocol.

As a concrete example compare a Brezing and Weng curve [14] with $k = 8$, $\rho = 1.25$ and $d = 4$, with the alternative efficient-arithmetic curve recommended

in FST [16], for which $k = 8$, $\rho = 1.5$ and $d = 2$. If the fixed argument optimization applies then the smaller d value for the latter is largely irrelevant to the pairing calculation, and with a superior ρ value the Brezing and Weng curve may well make a better choice at an AES-112 level of security.

In the context of a multi-pairing in conjunction with fixed arguments, then the impact of "one more" pairing will be minimized if we choose a curve which supports the minimum Miller loop length for a given level of security.

In the paper [28] the authors consider curves with $k = 15$. Despite the loss of full denominator elimination the authors contend that this might be a contender with BN curves at the AES-128 bit level. One of the problems with the $k = 15$ curve is that only a cubic twist is possible, and arithmetic on $E(\mathbb{F}_{p^5})$ is expensive compared with arithmetic on $E(\mathbb{F}_{p^2})$ as required for the BN curve which supports a sextic twist. However if the fixed argument optimization applies, this advantage for BN curves disappears as all these values required by the pairing can be precomputed. And a $k = 15$ curve will have an optimal pairing with a half-sized Miller loop compared with the BN equivalent ($\phi(15) = 8$, $\phi(12) = 4$). In [24] the authors make a similar case for a $k = 9$ curve. At the AES-256 level the KSS $k = 32$ curve [16] might turn out to be a better choice than our BLS $k = 24$ curve. The latter has a better twist (6 vs 4), but the former has twice the loop-reduction and a better ρ value of 1.125.

So in the scenario of a protocol where a multi-pairing is required, and where the great majority of the left-hand parameters are fixed – a scenario which a quick glance through the literature would seem to indicate is quite common – efficient implementation might benefit from a reappraisal of which pairing-friendly curve is optimal to use for the standard levels of security.

9 Conclusion

We have pointed out that in pairing-based protocols many precomputation optimizations are possible. By combining these the performance can be improved significantly. In particular we show that relatively complex protocols are completely practical on standard hardware at all levels of security. We give for the first time realistic timings for pairings at the AES-256 bit level. Whereas one pairing at the AES-256 level takes 37 milliseconds on a contemporary PC, we observe that "one-more" pairing in the context of a multi-pairing can take less than 5 milliseconds. For future work we ask whether the currently accepted pairing-friendly curves are actually the optimal choice when deployed in real-world protocols.

References

1. T. Acar, K. Lauter, M. Naehrig, and D. Shumow. Affine pairings on ARM. Cryptology ePrint Archive, Report 2011/243, 2011. `http://eprint.iacr.org/2011/243`.
2. Joseph A. Akinyele, Christoph U. Lehmann, Matthew D. Green, Matthew W. Pagano, Zachary N. J. Peterson, and Aviel D. Rubin. Self-protecting electronic

medical records using attribute-based encryption. Cryptology ePrint Archive, Report 2010/565, 2010. http://eprint.iacr.org/2010/565.
3. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster explicit formulas for computing pairings over ordinary curves. In *Eurocrypt 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer-Verlag, 2011.
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. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptography*, 17:321–334, 2004.
6. P.S.L.M. Barreto, S. Galbraith, C. OhEigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007. http://eprint.iacr.org/2004/375.
7. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC'2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer-Verlag, 2006.
8. N. Benger and M. Scott. Constructing tower extensions for the implementation of pairing-based cryptography. In *WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer-Verlag, 2010.
9. J-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barretto-naehrig curves. In *Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer-Verlag, 2010.
10. I. F. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography, Volume 2*. Cambridge University Press, 2005.
11. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Berlin: Springer-Verlag, 2004. Full version available at http://www.cs.stanford.edu/~xb/eurocrypt04b/.
12. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer Berlin / Heidelberg, 2004.
13. 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.
14. F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptology*, 37:133–141, 2005.
15. C. Costello and D. Stebila. Fixed argument pairings. Cryptology ePrint Archive, Report 2010/342, 2010. http://eprint.iacr.org/2010/342.
16. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing friendly elliptic curves. *Journal of Cryptography*, 23:224–280, 2010.
17. G. Frey, M. Müller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.
18. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.
19. R. Granger and N. P. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. http://eprint.iacr.org/2006/172.

20. F. Hess, N. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Trans. Information Theory*, 52:4595–4602, 2006.
21. E. Kachisa, E. Schaefer, and M. Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In *Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 126–135. Springer-Verlag, 2008.
22. K. Lauter, P. Montgomery, and M. Naehrig. An analysis of affine coordinates for pairing computation. In *Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2010.
23. E. Lee, H-S. Lee, and C-M. Park. Efficient and generalized pairing computation on abelian varieties. *IEEE Transactions on Information Theory*, 55:1793–1803, 2009.
24. X. Lin, C. Zhao, F. Zhang, and Y. Wang. Computing the ate pairing on elliptic curves with embedding degree $k = 9$. *IEICE Transactions*, 91-A(9):2387–2393, 2008.
25. Zhen Liu and Zhenfu Cao. On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption. Cryptology ePrint Archive, Report 2010/374, 2010. `http://eprint.iacr.org/2010/374`.
26. A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639 –1646, sep 1993.
27. 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.
28. N. El Mrabet, N. Guillermin, and S. Ionica. A study of pairing computation for elliptic curves with embedding degree 15. Cryptology ePrint Archive, Report 2009/370, 2009. `http://eprint.iacr.org/2009/370`.
29. A. M. Odlyzko. Discrete logarithms: the past and the future. *Design, Codes and Cryptography*, 19:129–145, 2000.
30. L. Oliveira, D. Aranha, C. Gouvêa, M. Scott, D. Câmara, J. López, and R. Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34:485–493, 2011.
31. J. H. Park. Inner-product encryption under standard assumptions. *Designs, Codes and Cryptography*, 58:235–257, 2011.
32. G. Pereira, M. Simplício Jr., M. Naehrig, and P. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84:1319–1326, 2011.
33. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.
34. M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, 2005.
35. M. Scott. Miracl library, 2011. `http://www.shamus.ie`.
36. J. Solinas. ID-based digital signature algorithms, 2003. `http://www.cacr.math.uwaterloo.ca/conferences/2003/ecc2003/solinas.pdf`.
37. F. Vercauteren. Optimal pairings. *Information Theory, IEEE Transactions on*, 56(1):455–461, Jan. 2010.
38. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer Berlin / Heidelberg, 2011.