Pairing Implementation Revisited

Michael Scott

MIRACL.com

February 10, 2019

Abstract

Pairing-based cryptography is now a mature science. However implementation of a pairing-based protocol can be challenging, as the efficient computation of a pairing is difficult, and the existing literature on implementation might not match with the requirements of a particular application. Furthermore developments in our understanding of the true security of pairings render much of the existing literature redundant. Here we take a fresh look and develop a simpler three-stage algorithm for calculating pairings, as they arise in real applications.
Keywords: Pairing-based Cryptography, Implementation

1 Introduction

For practical pairing-based cryptography, so-called type-3 pairings [15] on a chosen member of a family of small-discriminant parameterised pairing-friendly curves are the setting of choice [26]. For these curves to be pairing-friendly, the base field prime modulus p and the group prime order r are described by particular fixed polynomial formulae in terms of an integer parameter u [26]. These curves have a fixed "embedding degree" k, such that their security depends, in part, on the difficulty of a discrete logarithm problem defined over the extension field $\mathbb{F}_{p^k}$. Therefore a protocol implementor's first decision is to (a) choose a suitable family, and then (b) choose a u of a size and shape that supports efficient and secure implementation.

The apogee of a decade of work by many authors on efficient pairing implementation is summed up in the 2013 paper by Aranha, Barreto, Longa and Ricardini, "The Realm of Pairings" [2]. In the abstract pairings are described as "notoriously hard to implement efficiently". Nonetheless a thorough description is provided of a highly optimized algorithm for the calculation of a pairing on a particular 254-bit BN pairing friendly curve at what was then considered to be a security level very close to 128-bit AES symmetric encryption.

In that work the BN family [7] with $k = 12$ is chosen, where

$$\begin{aligned} p &= 36u^4 + 36u^3 + 24u^2 + 6u + 1 \\ r &= 36u^4 + 36u^3 + 18u^2 + 6u + 1 \end{aligned} \tag{1}$$

with $u = -(2^{62} + 2^{55} + 1)$, a particularly fortuitous choice, notable for its extremely small hamming weight of just 3, which facilitates numerous optimizations.

However as a starting point for the contemporary implementor of a pairing-based protocol this has several shortcomings:

- The BN curve used is now considered to offer only between 100 and 110 bits of AES equivalent security.
- The choice of a parameter u with extremely small Hamming weight may not be compatible with other requirements, such as sub-group security [4], [32].
- In real applications the important primitive is usually not the single stand-alone pairing, but rather the multi-pairing [29], [18], [31].

Around the time BN curves were introduced, Schirokauer [28] warned that the special form of p may reduce the difficulty of the extension field discrete logarithm problem on which the security of these pairings are based. Eventually this prediction was borne out by a series of papers, see [19, 3, 21, 23]. In response to these developments many researchers have turned instead to the BLS curves [6], which were actually the first pairing friendly curves of this type to be discovered. As an added advantage, the BLS curves are in fact a "family of families", that support members with embedding degrees of 12, 24 and 48 (hereafter referred to as BLS12, BLS24 and BLS48 curves respectively), that are suitable at different security levels.

When choosing the parameter u, it is generally accepted that a lower signed Hamming weight is preferable. However insistence on an extremely small Hamming weight greatly restricts the number of suitable curves which can be found. Unfortunately some pairing-friendly curves have already been standardised where no attempt was made to reduce the Hamming weight [13]. And only lately has the requirement for sub-group security been recognised [4], [32], and curves which fulfill this requirement are harder to find, and hence in general u with larger Hamming weight must suffice. The impact of this is mostly quite manageable, but it does have a major impact on a particular optimization [20] implemented in [2], in the context of the final exponentiation part of the pairing calculation. Basically when exponentiating to the power of u, for every set bit in u an expensive decompression operation is required, which involves a modular inversion. While this is not

a major issue for their choice of u, it would rapidly become one as the Hamming weight of u were increased, calling into question the applicability of this particular optimization. In general optimizations that depend heavily on very particular features of a particular pairing-friendly curve, cannot be generalised to other settings.

Papers on pairings often start with a literature review of pairing-based cryptography applications. However a closer look at that literature reveals an interesting fact: The primitive of interest is rarely the stand-alone pairing, but rather the multi-pairing, that is the product of multiple individual pairings. An examination of 16 protocols mentioned in [2] show that the great majority require a multi-pairing computation, and indeed often the number of pairings involved is a variable parameter of the scheme. This point is also made by Fouotsa et al. [14] (who also propose some novel pairing types that are actually best implemented as pairing products). Adapting a single pairing implementation to work efficiently in a multi-pairing setting is by no means a trivial matter, as some important optimizations become available, so for example the product of two pairings can be calculated jointly much faster than as the product of two individual pairings [29], [18], [31].

The typical description of a pairing implementation consists of two phases, the Miller loop followed by the final exponentiation. Here we suggest further dividing the Miller loop into two distinct phases.

2 Security

There is as yet no apparent consensus on how exactly to hit the standard security levels of 128, 192 and 256 bits using pairings. As pointed out by Aranha there are the optimistic and pessimistic schools of thought [1]. However there is general agreement that the 256-bit BN curve no longer provides 128-bits of security, with estimates of its actual security varying from 100 to 110-bits. The problem is that the finite extension field of size $3072 = 12 \times 256$ bits is not big enough. Therefore a plausible response is to move to a family of pairing-friendly curves, which while retaining the same group size, increase the extension field size. And this is precisely what a BLS12 curve provides. Furthermore a BLS24 curve has been proposed for the 192-bit level of security [3], and a BLS48 curve for 256-bits of security [22]. Despite the lack of consensus and standardisation, some implementors have already "jumped the gun" by switching from the BN to the BLS12 curve [10].

As a concrete working example we propose a $k = 12$ BLS pairing-friendly curve for 128-bit security. For this curve

$$p = (u-1)^2(u^4 - u^2 + 1)/3 + u$$
$$r = u^4 - u^2 + 1$$

A BLS curve has a CM discriminant of $D = -3$, and hence supports a sextic twist which we will assume to be a D-type twist [30], and has the defining equation

$$y^2 = x^3 + b$$

where b is small. For this curve the basic ate pairing is also optimal [33], which somewhat simplifies our task. Extension of our idea to BN curves and BLS curves of higher embedding degree, and to M-type twists, is straightforward. We find that $u = 10008000001001200_{16}$ (65 bits, Hamming weight of 5) generates a 383-bit prime modulus and 256-bit prime group size, with an extension field of size 4596 bits, and is GT-Strong [32] as regards sub-group security, and with $b = 15$. According to at least some authorities [23], a curve like this should be sufficient for 128-bits of security.

3 Line functions

The optimal ate pairing calculates $w = e(Q, P)$, where $Q \in \mathbb{G}_2$, $P \in \mathbb{G}_1$ are points in elliptic curve groups of order r, and $w \in \mathbb{G}_T$ is an element in the k-th degree extension, in this case $\mathbb{F}_{p^{12}}$, also of order r. The elliptic curve groups $\mathbb{G}_1$ and $\mathbb{G}_2$ must be distinct groups of order r, taken over the extension $\mathbb{F}_{p^{12}}$. However a common optimization is to choose $\mathbb{G}_1$ as the group on the curve $E(\mathbb{F}_p)$ taken over the base field $\mathbb{F}_p$, and $\mathbb{G}_2$ is represented on $E'(\mathbb{F}_{p^2})$ as the sextic twist of the trace-zero group, which can be instantly untwisted when needed to a point on $E(\mathbb{F}_{p^{12}})$.

Points on the twisted curve $E'(\mathbb{F}_{p^2})$ obey

$$y^2 = x^3 + b'$$

where for a D-type twist $b' = b/\xi$ and $\xi \in \mathbb{F}_{p^2}$ is neither a square or a cube. Under the not very restrictive condition that $p = 3 \bmod 8$, then an excellent choice is $\xi = 1 + \sqrt{-1}$.

An $\mathbb{F}_{p^{12}}$ value can be represented as $\mathbb{F}_{p^2}[\omega]/(\omega^6 - \xi)$, that is as an array of six $\mathbb{F}_{p^2}$ elements $A + C\omega + E\omega^2 + B\omega^3 + D\omega^4 + F\omega^5$ which can be written as $(A + B\omega^3) + (C + D\omega^3)\omega + (E + F\omega^3)\omega^2$ if we consider an $\mathbb{F}_{p^{12}}$ as a triplet of $\mathbb{F}_{p^4}$ elements [8]. For convenience we will represent such a value as

$$[[AB], [CD], [EF]]$$

The Miller loop needs to evaluate so-called line functions. These are elements in $\mathbb{F}_{p^{12}}$ that arise during the multiplication of Q by u using a standard double-and-add algorithm, and can be considered as a distance metric between these multiples of Q and the second stationary point P.

Assume Q is represented using homogeneous projective coordinates, and that its current multiple has coordinates (X, Y, Z), and assume that the

coordinates of the fixed point P are provided in affine form as (x_P, y_P). Then the line function that arises as the result of a point doubling, is given by

$$[[AB], [C0], [00]]$$

for a D-type twist, where $A = -2YZy_P$, $B = 3b'Z^2 - Y^2$ and $D = 3X^2x_P$. For a point addition of (X_2, Y_2, Z_2) to (X, Y, Z) then $A = (X - ZX_2)y_P$, $B = (Y - ZY_2)X_2 - (X - ZX_2)Y_2$, and $C = -(Y - ZY_2)x_P$ [2].

Note the sparsity of the $\mathbb{F}_{p^{12}}$ line function. Exploitation of this sparsity has been a major target of optimization efforts.

The Miller loop calculates what is essentially the product of many line function evaluations. Costing an $\mathbb{F}_{p^2}$ multiplication as m it has been observed that the product of two such sparse values costs only $6m$ using a standard Karatsuba-like formula. This compares with the product of two fully dense values which costs $18m$, and the product of a dense value by a sparse value which costs $13m$ [2].

In fact it is easily confirmed that the product of two sparse values results in a product which looks like

$$[[AB], [CD], [E0]]$$

which still retains some exploitable sparsity (we shall refer to such a value as "somewhat sparse"). Therefore an important first task for the implementor is to write a function which can compute the product of two elements in $\mathbb{F}_{p^{12}}$, fully exploiting any sparsity which may exist in either multiplicand. Note that exploitable sparsity arises in the context in which the multiplication happens, not by on-the-fly examination of individual components – which might result in unwanted side-channel leakage.

If, as is sometimes suggested, the point Q is instead processed in affine coordinates as (X, Y) then the line function is somewhat simpler, with $A = y_P$, $B = \lambda X - Y$ and $C = -\lambda x_P$, where λ is the line slope calculated in the standard way for point doubling or addition (and requiring a potentially expensive and non-constant time inversion in $\mathbb{F}_p$).

In all cases the values for A, B and C can be jointly multiplied at will by any suitable element in $\mathbb{F}_p$, as such a contribution will be wiped out by the final exponentiation [5]. In the affine case a nice optimization [25] is to multiply across by $1/y_P$, in which case $A = 1$, and since multiplication by 1 has no cost, a dense-sparse multiplication now reduces to $10m$.

4 The classic Miller loop

The main **for** loop in algorithm 1 is here referred to as the Miller loop, and the last line 8 is called the final exponentiation. The implementation of

Algorithm 1: Ate pairing on BLS12 curve

Input: $Q \in \mathbb{G}_2$, $P \in \mathbb{G}_1$, curve parameter u

Output: $f \in \mathbb{F}_{p^{12}}$

1 $f \leftarrow 1$
2 $T \leftarrow Q$
3 **for** $i \leftarrow \lfloor \log_2(u) \rfloor - 1$ **to** 0 **do**
4 $\quad f \leftarrow f^2$
5 $\quad f \leftarrow f.l_{T,T}(P)$, $T \leftarrow 2T$
6 $\quad$ **if** $u_i = 1$ **then**
7 $\quad\quad f \leftarrow f.l_{T,Q}(P)$, $T \leftarrow T + Q$
8 $f \leftarrow f^{(p^6-1)(p^2+1)(p^4-p^2+1)/r}$
9 **return** f

the final exponentiation is specific to each different pairing-friendly curve. For the BLS12 curve a fast algorithm is given in [16]. However the final exponentiation is not the focus of this work.

In algorithm 1, T represents the multiples of Q that arise, and $l_{T,S}$ represents the line function that arises when adding point S to point T. It is coordinates of T and P that are used to calculate these line functions. Now consider line 6 of the algorithm when the i-th bit of u equals 1, that is $u_i = 1$. In this loop iteration after squaring f, the value of f is updated as $f \leftarrow f.l_{T,T}(P).l_{2T,Q}(P)$. If calculated as described in this standard algorithm this would cost two dense-sparse multiplications, that is $26m$. However by calculating $l_{T,T}(P).l_{2T,Q}(P)$ separately, and only then multiplying f by this product, the cost will be only $23m$, as the sparse-sparse multiplication costs $6m$ and the multiplication of the dense f by their "somewhat sparse" product costs $17m$.

This observation will not in most cases lead to significant savings, as u should have a small Hamming weight, and the case $u_i = 1$ should therefore be rare. But it does point to an alternate way of structuring the Miller loop, such that the sparseness of the line functions can be fully exploited in every case, and not lost by premature multiplication by a fully dense value.

5 An alternate construction

First let us break the classic algorithm for the Miller loop into two parts, the calculation and storage of the line functions in algorithm 2, and the Miller loop in algorithm 3.

Note that the Miller loop part in algorithm 3 is now very simple, and is dependent only on the number of bits in u, and not on its Hamming weight. Observe that the value of f becomes dense after a couple of iterations of the loop, due to the iterative squaring action. The complex computation and

Algorithm 2: Calculate and store line functions for BLS12 curve

Input: $Q \in \mathbb{G}_2$, $P \in \mathbb{G}_1$, curve parameter u
Output: An array g of $\lfloor \log_2(u) \rfloor$ line functions $\in \mathbb{F}_{p^{12}}$

$T \leftarrow Q$
for $i \leftarrow \lfloor \log_2(u) \rfloor - 1$ **to** 0 **do**
    $g[i] \leftarrow l_{T,T}(P)$, $T \leftarrow 2T$
    if $u_i = 1$ **then**
        $g[i] \leftarrow g[i].l_{T,Q}(P)$, $T \leftarrow T + Q$
return g

Algorithm 3: Miller loop for BLS12 curve

Input: An array g of $\lfloor \log_2(u) \rfloor$ line functions $\in \mathbb{F}_{p^{12}}$
Output: $f \in \mathbb{F}_{p^{12}}$

$f \leftarrow 1$
for $i \leftarrow \lfloor \log_2(u) \rfloor - 1$ **to** 0 **do**
    $f \leftarrow f^2.g[i]$
return f

combination of line functions is now handled in the separate line function algorithm 2, where the sparsity that arises can naturally be exploited to the full. However it is only when we consider the multi-pairing that we will fully appreciate the advantage of this approach.

Before proceeding it would be appropriate to stop and consider the amount of storage required for the array g. For a BLS12 curve with a 384-bit modulus p, then a total of at least $12 \times 384 \times \lfloor \log_2(u) \rfloor$ bits would be required. If targetting the 128-bit security level then a typical u might be 65 bits in length, and so the total storage requirement would be about $40k$ bytes. We consider this to be quite acceptable for a moderately complex multi-pairing-based protocol, although it may admittedly cause a problem for a very small embedded application.

6 Efficient and flexible multi-pairings

A multi-pairing is the product of more than one pairing, $e(Q_1, P_1).e(Q_2, P_2)...e(Q_n, P_n)$. It requires just one shared final exponentiation, irrespective of the number of pairings involved. And it also requires just one invocation of our new simplified Miller loop as described above. The only modification required is to the line function algorithm, where now more line functions associated with the other pairings must be accumulated into the array g. See algorithm 4. A plausible way to proceed would be to calculate the line functions for the first pairing using algorithm

2, and then accumulate the line functions associated with the other $n-1$ pairings one at a time, using algorithm 4.

Algorithm 4: Accumulate another set of line functions into g

Input: The array g, $Q_j \in \mathbb{G}_2$, $P_j \in \mathbb{G}_1$, curve parameter u
Output: Updated array g of $\lfloor \log_2(u) \rfloor$ line functions $\in \mathbb{F}_{p^{12}}$

```
1 T ← Q_j
2 for i ← ⌊log_2(u)⌋ − 1 to 0 do
3     t ← l_{T,T}(P_j), T ← 2T
4     if u_i = 1 then
5         t ← t.l_{T,Q_j}(P_j), T ← T + Q_j
6     g[i] ← g[i].t
7 return g
```

So to calculate a multi-pairing consisting of the product of n distinct pairings, proceed as follows

- Execute algorithm 2 for the first pairing
- Call algorithm 4 $n-1$ times, for each additional j-th pairing involved in the multi-pairing.
- Invoke algorithm 3, to trigger the combined Miller loop calculation.
- Carry out the final exponentiation.

In practise it may be appropriate to use a more flexible accumulation policy, depending on the provenance of the Q_j. It may be the case that a particular Q_j is a constant, or it may be an important secret, or it may be a publicly known value. In the case where it is a constant, the multiples of Q_j can be calculated in affine form, and used to precalculate the affine form of the line functions from $(\lambda X - Y)$ and $-\lambda$ as described above, requiring only a single $\mathbb{F}_{p^2}$ multiplication per line function. This will be much faster. See also [12] and [31]. If Q_j is an important secret it may be wise to process it in constant time using homogeneous projective coordinates using the exception-free formulae proposed in [27]. If the Q_j is publicly known, then processing them using non-constant time affine coordinates might be more efficient. Each of these scenarios will require its own variant of algorithm 4. In this way the pairing line functions can be calculated using an optimal technique for each pairing involved.

For example consider the well-known Boneh-Lynn-Shacham short signature scheme [9], for which signature verification requires the product of two pairings. In this case the first Q_1 is a constant fixed generator, the second Q_2 is a public key. Therefore is makes sense to process the line functions that arise for each pairing differently.

7 Fully exploiting sparsity

In general sparse elements in $\mathbb{F}_{p^{12}}$ should be multiplied by other sparse elements where possible, to preserve the benefits of sparsity for as long as possible. To this end it would be advantageous to accumulate the product of two pairings at a time, where possible. See algorithm 5, where we add $e(Q_j, P_j).e(Q_k, P_k)$ to an existing accumulation of products in g.

Algorithm 5: Accumulate two sets of line functions into g

Input: The array g, $Q_j, Q_k \in \mathbb{G}_2$, $P_j, P_k \in \mathbb{G}_1$, curve parameter u
Output: Updated array g of $\lfloor \log_2(u) \rfloor$ line functions $\in \mathbb{F}_{p^{12}}$

$T \leftarrow Q_j$
$S \leftarrow Q_k$
for $i \leftarrow \lfloor \log_2(u) \rfloor - 1$ **to** 0 **do**
    $t \leftarrow l_{T,T}(P_j)$, $T \leftarrow 2T$
    $t \leftarrow t.l_{S,S}(P_k)$, $S \leftarrow 2S$
    if $u_i = 1$ **then**
        $v \leftarrow l_{T,Q_j}(P_j)$, $T \leftarrow T + Q_j$
        $v \leftarrow v.l_{S,Q_k}(P_k)$, $S \leftarrow S + Q_k$
        $t \leftarrow t.v$
    $g[i] \leftarrow g[i].t$
return g

In the case of affine coordinates this would also facilitate Montgomery's simultaneous inversion trick [24] which can be used to combine the modular inverses required when adding or doubling points in affine coordinates.

8 Implementation

The method described has been implemented in the AMCL multi-lingual crypto library [1]. Here we derive indicative costs in terms of $\mathbb{F}_{p^2}$ multiplications for each stage of a multi-pairing calculation using our 383-bit BLS12 curve as an example.

We start with the final exponentiation, where we will assume the use of the Granger-Scott Cyclotomic squaring formulae [17] which costs $6m$, rather than the Karabina method [20], which is significantly faster only if u has an extremely small Hamming weight, and we don't want to make that assumption here. Using the formula derived in [16] the cost will be close to $2499m$ plus one inversion in $\mathbb{F}_p$.

Our simpler Miller loop will iterate 65 times (with the first iteration "for free", as $f = 1$), at a cost of one $\mathbb{F}_{p^{12}}$ squaring using the Chung-Hasan SQR3

[1] `https://github.com/apache/incubator-milagro-crypto`

method [11] (which we cost as $11m$) and one multiplication per iteration. This multiplication can be assumed to be dense-sparse for a single pairing, and dense-somewhat-sparse for every set bit in u. So for a single pairing the cost will be $1556m$. For a double pairing the multiplication can be assumed to be dense-somewhat-sparse, and dense-dense for every bit set in u, so in this case the cost will be $1797m$. For the product of more than two pairings, all multiplications will be dense-dense, and the cost will be $1856m$.

The line function calculation is a little more complex, as it depends on how the line functions are calculated. If precomputation is possible, then for the first pairing the cost is just $1m$ per bit of u plus $7m$ for every set bit for a total of $100m$. For the second pairing the cost will be $540m$, and for the third and subsequent pairings the cost will be $965m$. If the optimization of [25] is used, further savings are possible.

Without precomputation, we must include the cost of the implicit elliptic curve $E(\mathbb{F}_{p^2})$ point multiplication of uQ_j. This can be done using either affine or homogenous projective coordinates. Assuming the latter and following [2], the costs rise from $550m$ for the first pairing, to $995m$ for the second, and $1415m$ for all subsequent pairings (assuming an $\mathbb{F}_{p^2}$ squaring is roughly equivalent to two-thirds the cost of m).

Finally if two more pairings are to be accumulated together to an existing product of two or more pairings for example using non-precomputed projective coordinates, then using algorithm 5 the additional cost will be $2650m$, a useful improvement over two single pairing accumulations which would cost $2 \times 1415m = 2830m$.

9 Higher Security

We briefly consider the attainment of 192 and 256-bits of security. For the former a BLS24 curve, with a 50-bit u that results in a 480-bit modulus p, would appear to be a possible candidate. For the latter we suggest a BLS48 curve with a 32-bit u that results in a 560-bit modulus p. Note that rather counter-intuitively u gets smaller for higher levels of security, which suggests that the size of g grows only slowly. For these suggested parameters, g would be approximately $72k$ bytes and $108k$ bytes respectively.

10 Conclusion

We suggest a novel approach to the implementation of type-3 multi-pairings, which acknowledges that the type-3 multi-pairing is in fact the primitive required by the majority of pairing-based protocols that are of practical interest. In return for a memory resource required to accumulate line functions, we derive a more flexible three stage algorithm where the contribution of each pairing in a multi-pairing can be accumulated independently into just

the first stage. Furthermore we suggest that this approach allows the sparseness that arises in the representation of the line functions to be exploited to the maximum.

References

[1] D. Aranha. Pairings are not dead, just resting. ECC 2017, 2017. https://ecc2017.cs.ru.nl/slides/ecc2017-aranha.pdf.

[2] D. Aranha, P. Barreto, P. Aranha, and J. Ricardini. The realm of pairings. In *Selected Areas in Cryptography – SAC'2013*, volume 8282 of *LNCS*, pages 3–25. Springer-Verlag, 2013. https://eprint.iacr.org/2013/722.

[3] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, Jan 2018. https://doi.org/10.1007/s00145-018-9280-5, http://eprint.iacr.org/2017/334.

[4] P. Barreto, C. Costello, R. Miscoczki, M. Naehrig, G. Pereira, and G. Zanon. Subgroup security in pairing-based cryptography. In *Latincrypt 2015*, volume 9230 of *LNCS*, pages 245–265. Springer-Verlag, 2015.

[5] 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 *LNCS*, pages 354–368. Springer-Verlag, 2002. https://www.iacr.org/archive/crypto2002/24420355/24420355.pdf, https://eprint.iacr.org/2002/008.

[6] 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 *LNCS*, pages 263–273. Springer-Verlag, 2002. https://eprint.iacr.org/2002/088.

[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 *LNCS*, pages 319–331, Kingston, 2006. Springer-Verlag. https://eprint.iacr.org/2005/133.

[8] N. Benger and M. Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In *WAIFI 2010*, volume 6087 of *LNCS*, pages 180–195. Springer-Verlag, 2010. https://eprint.iacr.org/2009/556.

[9] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *LNCS*, pages 514–532. Springer-Verlag, 2002.

[10] S. Bowe. BLS12-381: New zk-SNARK elliptic curve construction, 2018. `https://z.cash/blog/new-snark-curve/`.

[11] J. Chung and M. A. Hasan. Asymmetric squaring formulae, 2006. `http://www.cacr.math.uwaterloo.ca/`.

[12] C. Costello and D. Stebila. Fixed argument pairings. In *Latincrypt – 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 92–108. Springer-Verlag, 2010.

[13] International Organization for Standardization. Information technology - security techniques – cryptographic techniques based on elliptic curves. part 5: Elliptic curve generation. ISO/IEC 15946-5, 2009. See `https://tools.ietf.org/pdf/draft-kasamatsu-bncurves-02.pdf`.

[14] E. Fouotsa, P. Pecha, and N. El Mrabet. Beta weil pairing revisited. *Afrika Mathematika*, pages 1–18, 2019.

[15] S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.

[16] L. Ghamman and E. Fouotsa. On the computation of the optimal ate pairing at the 192-bit security level. Cryptology ePrint Archive, Report 2004/058, 2016. `http://eprint.iacr.org/2016/130/`.

[17] R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *PKC 2010*, volume 6056 of *LNCS*, pages 209–223. Springer-Verlag, 2010.

[18] R. Granger and N. P. Smart. On computing products of pairings. Cryptology ePrint Archive, Report 2006/172, 2006. `http://eprint.iacr.org/2006/172`.

[19] Antoine Joux and Cécile Pierrot. The special number field sieve in $\mathbb{F}_{p^n}$ - application to pairing-friendly constructions. In Zhenfu Cao and Fangguo Zhang, editors, *Pairing 2013*, volume 8365 of *LNCS*, pages 45–61, Beijing, China, 2013. Springer. `https://eprint.iacr.org/2013/582`.

[20] K. Karabina. Squaring in cyclotomic subgroups. *Mathematics of Computation*, 82:555–579, 2013.

[21] T. Kim and R. Barbulescu. The extended tower number field sieve: A new complexity for the medium prime case. In *Crypto 2016*, volume 9814 of *LNCS*, pages 543–571. Springer-Verlag, 2016. https://eprint.iacr.org/2015/1027.

[22] Y. Kiyomura, A. Inoue, Y. Kawahara, M. Yasuda, T. Takagi, and T. Kobayashi. Secure and efficient pairing at 256-bit security level. In *ACNS 2017*, volume 10355 of *LNCS*, pages 59–79. Springer-Verlag, 2017.

[23] A. Menezes, P. Sarkar, and S. Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *Mycrypt 2016*, volume 10311 of *LNCS*, pages 83–108. Springer-Verlag, 2016. https://eprint.iacr.org/2016/1102.

[24] P. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[25] Y. Mori, S. Akagi, Y. Nogami, and M. Shirase. Pseudo 8-sparse multiplication for efficient ate-based pairing on Barreto-Naehrig curve. In *Pairing 2013*, volume 8365 of *LNCS*, pages 78–88. Springer-Verlag, 2013.

[26] N. El Mrabet and M. Joye, editors. *Guide to Pairing-Based Cryptography*. Chapman and Hall/CRC, 2016.

[27] J. Renes, C. Costello, and L. Batina. Complete addition formulas for prime order elliptic curves. In *Eurocrypt – 2016*, volume 9665 of *Lecture Notes in Computer Science*, pages 403–428. Springer-Verlag, 2016.

[28] Oliver Schirokauer. The number field sieve for integers of low weight. *Mathematics of Computation*, 79(269):583–602, January 2010. https://doi.org/10.1090/S0025-5718-09-02198-X, http://eprint.iacr.org/2006/107.

[29] M. Scott. Computing the Tate pairing. In *CT-RSA*, volume 3376 of *LNCS*, pages 293–304. Springer-Verlag, 2005.

[30] M. Scott. A note on twists for pairing-friendly curves, 2009. indigo.ie/~mscott/twists.pdf.

[31] M. Scott. On the efficient implementation of pairing-based protocols. In *IMACC 2011*, volume 7089 of *LNCS*, pages 296–308. Springer-Verlag, 2011. https://eprint.iacr.org/2011/334.

[32] M. Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Report 2013/688, 2013. Available from http://eprint.iacr.org/2013/688.

[33] F. Vercauteren. Optimal pairings. *IEEE Transactions of Information Theory*, 56:455–461, 2009. `https://eprint.iacr.org/2008/096`.