Chapter 7

The state-of-the-art

This chapter summarises the evolution of pairing computation over the last decade. We illustrate the landmark achievements that accelerated early implementations of pairings from "a few minutes" [Men93][1] into current implementations that take less than a millisecond [AKL+11].

Initial improvements in pairing computations were spearheaded by evidence that computing the Tate pairing $f_{r,P}(D_Q)^{(q^k-1)/r}$ is more efficient than computing the Weil pairing $f_{r,P}(D_Q)/f_{r,Q}(D_P)$. At first glance it seems that comparing the two computations amounts to comparing an exponentiation by $(q^k-1)/r$ to a (second) run of Miller's algorithm $f_{r,Q}(D_P)$, and indeed, at levels of security up to 128 bits, this comparison does favour the Tate pairing (cf. [SCA06, Tab. 1-5], [Sco07c]). However, as we will see in Section 7.1, exponentiating by $(q^k-1)/r$ actually facilitates many "Tate-specific" optimisations within the associated Miller loop. It is these enhancements that gave the field of pairing computation its first big boost.

7.1 Irrelevant factors (a.k.a. denominator elimination)

In this section we will work our way to a refined version of Miller's algorithm for pairings over large prime fields, which is mostly due to improvements sug-

[1] As Scott says however, this comparison is unfair – in 1993 there was no incentive to try and optimise the computation past what was needed to apply the MOV attack [MOV93].

gested by Barreto, Kim, Lynn and Scott [BKLS02], and also partly due to Galbraith, Harrison and Soldera [GHS02]. Thus, it is often referred to as the BKLS algorithm [Sco05a, WS07], or sometimes as the BKLS-GHS algorithm [Sco05b, BGOS07]. Our exposition will make use of twisted curves, which we discussed in Section 4.3 and the employment of which is originally due to Barreto, Lynn and Scott [BLS03]. The early works that included Barreto, Lynn and Scott are also culminated in [BLS04].

We start with an observation that allows us to conveniently replace the divisor D_Q with the point Q in the Tate pairing definition. Namely, so long as $k > 1$ and P and Q are linearly independent, then $f_{r,P}(D_Q)^{(q^k-1)/r} = f_{r,P}(Q)^{(q^k-1)/r}$ [BKLS02, Th. 1]. This saves the hassle of defining a divisor equivalent to $D_Q = (Q) - (\mathcal{O})$ with support disjoint to $(f_{r,P})$, but more importantly allows us to simply evaluate the intermediate Miller function at the point Q (rather than two points) in each iteration of Algorithm 5.1.

Example 7.1.1 (Magma script). We reuse the parameters from Example 5.3.1 so a comparison between intermediate values is possible. Thus, let $q = 47$, $E/\mathbb{F}_q : y^2 = x^3 + 21x + 15$, $\#E(\mathbb{F}_q) = 51$, $r = 17$, $k = 4$, $\mathbb{F}_{q^4} = \mathbb{F}_q(u)$ with $u^4 - 4u^2 + 5 = 0$, $P = (45, 23) \in \mathbb{G}_1$ and $Q = (31u^2 + 29, 35u^3 + 11u) \in \mathbb{G}_2$. Thus, the Tate pairing is $e(P, Q) = f_{r,P}(Q)^{(q^k-1)/r} = (32u^3 + 17u^2 + 43u + 12)^{287040} = 33u^3 + 43u^2 + 45u + 39$, which is the same value we got when instead computing $f_{r,P}(D_Q) = f_{r,P}([2]Q)/f_{r,P}(Q)$ in Example 5.3.1. When comparing the fifth columns of both tables, one should keep in mind that the numerator and denominator of the fractions in Example 5.3.1 were themselves both computed as fractions. Indeed, updates in this example are just the denominator of the updates in Example 5.3.1, which gives an indication of how advantageous it is to evaluate the pairing functions at one point (e.g. Q), rather than at a divisor consisting of multiple points (e.g. $([2]Q) - (Q)$). Notice that the values $f_{r,P}(D_Q)$ and $f_{r,P}(Q)$ output after the Miller loops in both examples are not the same, but

i/r_i	steps of Alg. 5.1	point R	update ℓ/v	update at Q $\ell(Q)/v(Q)$	paired value f
	1	$(45, 23)$			1
3/0	3-5	$(12, 16)$	$\frac{y+33x+43}{x+35}$	$\frac{35u^3+36u^2+11u+13}{31u^2+17} = 6u^3 + 19u^2 + 36u + 33$	$6u^3 + 19u^2 + 36u + 33$
2/0	3-5	$(27, 14)$	$\frac{y+2x+7}{x+20}$	$\frac{35u^3+15u^2+11u+18}{31u^2+2} = 39u^3 + 8u^2 + 20u + 18$	$11u^3 + 17u^2 + 24u + 4$
1/0	3-5	$(18, 31)$	$\frac{y+42x+27}{x+29}$	$\frac{35u^3+33u^2+11u+23}{31u^2+11} = 18u^3 + 32u^2 + 41u + 30$	$22u^3 + 34u^2 + 5u + 10$
0/1	3-5	$(45, 24)$	$\frac{y+9x+42}{x+2}$	$\frac{35u^3+44u^2+11u+21}{31u^2+31} = 21u^3 + 26u^2 + 25u + 20$	$8u^3 + 22u^2 + 5u + 27$
	6-10	$\mathcal{O}$	$x + 2$	$= 31u^2 + 31$	$32u^3 + 17u^2 + 43u + 12$
	12	$f_{r,P}(Q) \leftarrow 32u^3 + 17u^2 + 43u + 12$			

the *final exponentiation* maps them to the same element in μ_{17}. This is because $f_{r,P}(D_Q)$ and $f_{r,P}(Q)$ lie in the same coset of $(\mathbb{F}_{q^k}^*)^r$ in $\mathbb{F}_{q^k}^*$, i.e. they are the same element in the quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$.

We are now in a position to describe the important *denominator elimination* optimisation. Barreto *et al.* were the first to notice that $q-1 \mid (q^k-1)/r$ [BKLS02, Lemma 1], since if $r \mid q-1$ then the embedding degree would be $k=1$. This allows us to write the final exponent as $(q^k-1)/r = (q-1)\cdot c$, which gives $f_{r,P}(Q)^{(q^k-1)/r} = (f_{r,P}(Q)^{q-1})^c$, meaning that any elements of $\mathbb{F}_q$ contributing to $f_{r,P}(Q)$ will be mapped to one under the final exponentiation. Thus, one can freely multiply or divide $f_{r,P}(Q)$ by an element of $\mathbb{F}_q$ without affecting the pairing value [BKLS02, Corr. 1]. When working over supersingular curves with $k=2$, the x-coordinate of Q is defined over $\mathbb{F}_q$ (see any of Examples 4.1.4, 4.1.5 ,4.3.1, 5.2.1). Therefore, the vertical lines appearing on the denominators of Miller's algorithm for the Tate pairing are entirely defined over $\mathbb{F}_q$: the line is a function $x - x_R$ that depends on $P \in E(\mathbb{F}_q)[r]$, which is evaluated at $x_Q \in \mathbb{F}_q$. Thus, in this case the contribution of (each of) the denominators to $f_{r,P}(Q)$ ends up being mapped to 1 under the final exponentiation, so these denominators (the v's in the ℓ/v's – see Steps 5 and 9 in Algorithm 5.1) can be removed from the Miller loop.

For ordinary curves with $k > 2$ however, the x-coordinate of Q will no longer be in the base field $\mathbb{F}_q$, but in some proper subfield $\mathbb{F}_{q^e}$ of $\mathbb{F}_{q^k}$, where $e = k/d$ and d is the degree of the twist employed[2] – see Section 4.3. Here it helps to assume that k is even, i.e. $k = 2\ell$, so that (at the very least) we can take $Q = (x_Q, y_Q)$ where $x_Q \in \mathbb{F}_{q^\ell}$ is such that $y_Q \in \mathbb{F}_{q^k} \setminus \mathbb{F}_{q^\ell}$. Thus, when advancing beyond $k=2$ supersingular curves, Barreto *et al.* generalised the original statement to facilitate the same trick. Namely, that $q^e - 1 \mid (q^k-1)/r$ for any proper factor $e \mid k$ [BLS03, Lemma 5, Corr. 2], so denominators can be omitted from computations in general.

Example 7.1.2 (Magma script). Again, we will continue on from Example 7.1.1 for the sake of a convenient comparison. We simply give an updated table that details the intermediate Miller functions and pairing values subject to denominator elimination. Therefore, $e(P,Q) = f_{r,P}(Q)^{(q^k-1)/r} = (9u^3+10u^2+32u+36)^{(q^k-1)/r} = 33u^3+43u^2+45u+39$, which agrees with the Tate pairing value

[2]When $d=3$ cubic twists are able to be employed for odd k, it is the y-coordinate of Q that is in the subfield; we will treat this in Chapter 4.

i/ r_i	steps of Alg. 5.1	point R	update ℓ	update at Q $\ell(Q)$	paired value f
	1	$(45, 23)$			1
3/0	3-5	$(12, 16)$	$y + 33x + 43$	$35u^3 + 36u^2 + 11u + 13$	$35u^3 + 36u^2 + 11u + 13$
2/0	3-5	$(27, 14)$	$y + 2x + 7$	$35u^3 + 15u^2 + 11u + 18$	$44u^3 + 34u^2 + 3u + 44$
1/0	3-5	$(18, 31)$	$y + 42x + 27$	$35u^3 + 33u^2 + 11u + 23$	$5u^3 + 24u^2 + 21u + 24$
0/1	3-5	$(45, 24)$	$y + 9x + 42$	$35u^3 + 44u^2 + 11u + 21$	$21u^3 + 36u^2 + 9u + 25$
	6-10	$\mathcal{O}$	$x + 2$	$31u^2 + 31$	$9u^3 + 10u^2 + 32u + 36$
	12	$f_{r,P}(Q) \leftarrow 9u^3 + 10u^2 + 32u + 36$			

in Examples 5.3.1 and 7.1.1. Notice again that the value output from the Miller loop is not equal to either of the values output in 5.3.1 or 7.1.1, but rather that all three are equivalent under the relation $a = b$ if $a/b \in (\mathbb{F}_{q^k}^*)^r$.

We now refine Miller's algorithm for the Tate pairing computation subject to the BKLS-GHS improvements. Specifically, notice that the denominators that were on lines 5 and 9 have now gone (under the assumption that k is even), and that the second input is now the point Q, rather than a divisor equivalent to D_Q. Further notice that we have necessarily include the final exponentiation in Algorithm 7.1 since this is what facilitates the modifications. We have also assumed a Type 3 pairing so the coordinates of P and Q lie in fields that allow for denominator elimination. Recall from the discussion at the end of Example 5.3.1, or from Example 7.1.2, that the vertical line joining $(r-1)P = -P$ and P in the last iteration can also be omitted. Thus, an optimised Tate pairing computation will execute the main loop from $i = n-2$ to $i = 1$ before performing a "doubling-only" iteration to finish; we left the main loop to $i = 0$ for simplicity.

7.2 Projective coordinates

Although the optimisations described in the previous section removed the denominators in Step 5 and Step 9 of Algorithm 7.1, $\mathbb{F}_q$-inversions are still apparent in the routine since the affine explicit formulas for the elliptic curve group operations (see Eq. (2.4) and (2.5)) require them. The penalty for performing field inversions in PBC is not as bad as it is in ECC (more on this later), but in any case inversions are still much more costly than field multiplications. In this section we employ the same techniques to avoid field inversions as we did in the context of ECC in Example 2.1.9. Namely, we show how Algorithm 7.1 can become inversion-free if we adopt projective coordinates. In the early days the situation for projective coordinates in the context of pairings was perhaps a little

Algorithm 7.1 The BKLS-GHS version of Miller's algorithm for the Tate pairing.

Input: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ (Type 3 pairing) and $r = (r_{n-1} \dots r_1 r_0)_2$ with $r_{n-1} = 1$.
Output: $f_{r,P}(Q)^{(q^k-1)/r} \leftarrow f$.

1: $R \leftarrow P$, $f \leftarrow 1$.
2: **for** $i = n - 2$ down to 0 **do**
3: Compute the sloped line function $\ell_{R,R}$ for doubling R.
4: $R \leftarrow [2]R$.
5: $f \leftarrow f^2 \cdot \ell_{R,R}(Q)$.
6: **if** $r_i = 1$ **then**
7: Compute the sloped line function $\ell_{R,P}$ for adding R and P.
8: $R \leftarrow R + P$.
9: $f \leftarrow f \cdot \ell_{R,P}(Q)$.
10: **end if**
11: **end for**
12: **return** $f \leftarrow f^{(q^k-1)/r}$.

unclear [Gal05, IX.14], but nowadays all of the record-breaking implementations (at least up to the 128-bit security level) have exploited the savings offered by working in projective space.

The potential of projective coordinates was mentioned in passing in the early landmark papers [BKLS02, §3.2], [GHS02], but the first detailed investigation was by Izu and Tagaki [IT02]. As Galbraith mentions [Gal05, IX.14], the analysis in [IT02] is misleading, however projective coordinates did not wait too long before more accurate expositions that also endorsed their usefulness surfaced [CSB04, Sco05a]. The following example shows how projective coordinates can be used to achieve an inversion-free version of Miller's algorithm.

Example 7.2.1 (Magma script). In the context of standard ECC operations, we gave the (homogeneous) projective point addition formulas in Example 2.1.9. Thus, here we will give the homogeneous doubling formulas for computing $(X_{[2]R} : Y_{[2]R} : Z_{[2]R}) = [2](X_R : Y_R : Z_R)$ on $E/\mathbb{F}_q : Y^2Z = X^3 + aXZ^2 + bZ^3$ in Step 4 of Algorithm 7.1, together with the formulas for computing the line function $\ell_{R,R}(Q)$ in Step 3. The affine doubling formulas in Equation (2.5) are moved into homogeneous projective space via the substitution $x = X/Z$ and $y = Y/Z$,

which gives:

$$\lambda = \frac{3X_R^2 + Z_R^2}{2Y_R Z_R}; \qquad \nu = -\frac{3X_R^3 + X_R Z_R^2 - 2Y_R^2 Z_R}{2Y_R Z_R^2};$$
$$\frac{X_{[2]R}}{Z_{[2]R}} = \frac{-8X_R Y_R^2 Z_R + 6X_R^2 Z_R^2 + 9X_R^4 + Z_R^4}{4Y_R^2 Z_R^2};$$
$$\frac{Y_{[2]R}}{Z_{[2]R}} = -\frac{8Y_R^4 Z_R^2 + Z_R^6 - 12X_R Z_R^3 Y_R^2 - 36X_R^3 Z_R Y_R^2 + 27Z_R^2 X_R^4 + 9Z_R^4 X_R^2 + 27X_R^6}{8Y_R^3 Z_R^3},$$

where $\ell : y - (\lambda x + \nu)$ is still an affine line tangent to E at the point R. It is again the ability to multiply by factors in proper subfields of $\mathbb{F}_{q^k}$ that allows us to arrive at an inversion-free routine. Namely, we clear the denominators of λ and ν through multiplication by $2Y_R Z_R^2$, so the line ℓ becomes

$$\ell : (2Y_R Z_R^2) \cdot y - ((3X_R^2 Z_R + Z_R^3) \cdot x - (3X_R^3 + X_R Z_R^2 - 2Y_R^2 Z_R)),$$

which will be evaluted at $y = y_Q$ and $x = x_Q$. Note that since Q remains fixed throughout the routine, there is no need to cast it into projective space. Finally, setting $Z_{[2]R} = 8Y_R^3 Z_R^3$ and updating the numerator of $X_{[2]R}$ above allows us to compute $(X_{[2]R} : Y_{[2]R} : Z_{[2]R})$ from $(X_R : Y_R : Z_R)$ without any $\mathbb{F}_q$-inversions. Thus, we have an inversion-free way to proceed through the Miller doubling stage (Steps 3-5 of Algorithm 7.1), and performing the analogous procedure for the Miller addition stage (Steps 7-9) will give an inversion-free Miller loop.

7.3 Towered extension fields

This section discusses efficient methods of constructing the full extension field $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$, where the ultimate goal is to minimise the cost of the arithmetic in $\mathbb{F}_{q^k}$. Indeed, the majority of operations within the pairing algorithm take place in the full extension field, which is far more expensive to work in than its proper subfields, so the complexity of Miller's algorithm heavily depends on the complexity of the associated $\mathbb{F}_{q^k}$-arithmetic.

So far we have been using one irreducible degree k polynomial to construct $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. This has been satisfactory, since our small examples have mostly had embedding degrees $k = 2$ or $k = 3$, where we have no other option but to use polynomials of degree two and three to respectively construct $\mathbb{F}_{q^k}$. However, for large values of k, which will be composite in all cases of interest to us, there is an a natural alternative which turns out to be much faster. This idea was first

put forward by Koblitz and Menezes [KM05], who proposed using embedding degrees of the form $k = 2^i 3^j$ and building up to $\mathbb{F}_{q^k}$ using a series of quadratic and cubic extensions that successively *tower* up the intermediate fields. For such k, they show that if $q \equiv 1 \bmod 12$ and if α is neither a square or cube in $\mathbb{F}_q$, then the polynomial $x^k - \alpha$ is irreducible in $\mathbb{F}_q[x]$ [KM05, Th. 1]. This means that the tower can be constructed by a sequence of Kummer extensions: this involves successively adjoining the square root or cube root α, then the square root or cube root of that, and so on.

Example 7.3.1 (Magma script). Let $q = 97$, and consider constructing $\mathbb{F}_{q^{12}}$ using $\alpha = 5$ which is a non-square and non-cube in $\mathbb{F}_q$, so that $\mathbb{F}_{q^{12}}$ can be constructed directly as $\mathbb{F}_{q^{12}} = \mathbb{F}_q[X]/(X^{12} - \alpha)$. Choosing instead a tower of quadratic and cubic extensions, we could construct $\mathbb{F}_{q^{12}}$ as

$$\mathbb{F}_q \xrightarrow{\beta^2-\alpha} \mathbb{F}_{q^2} \xrightarrow{\gamma^3-\beta} \mathbb{F}_{q^6} \xrightarrow{\delta^2-\gamma} \mathbb{F}_{q^{12}}.$$

We show a random element in $\mathbb{F}_{q^{12}}$:

$$\big((79\beta + 63)\gamma^2 + (29\beta + 63)\gamma + (38\beta + 27)\big)\,\delta + (63\beta + 22)\gamma^2 + (93\beta + 10)\gamma + 75\beta + 10.$$

Observe what happens if, instead of performing multiplications in $\mathbb{F}_{q^{12}}$ over $\mathbb{F}_q$, we start by performing multiplications over $\mathbb{F}_{q^6}$. Writing $a, b \in \mathbb{F}_{q^{12}}$ over $\mathbb{F}_{q^6}$ gives $a = a_0 + a_1\delta$ and $b = b_0 + b_1\delta$, with $a_0, a_1, b_0, b_1 \in \mathbb{F}_{q^6}$. Thus, $a \cdot b = (a_0 b_0 - a_1 b_1 \gamma) + (a_0 b_1 + a_1 b_0)\delta$, where each of the components inside the parentheses are in $\mathbb{F}_{q^6}$. To perform each of the multiplications in $\mathbb{F}_{q^6}$, we then work over $\mathbb{F}_{q^2}$, so for example we would need to compute a multiplication between $a_0 = a_{0,0} + a_{0,1}\gamma + a_{0,2}\gamma^2$ and $b_0 = b_{0,0} + b_{0,1}\gamma + b_{0,2}\gamma^2$, where each component $a_{0,i}$ and $b_{0,i}$ is in $\mathbb{F}_{q^2}$. In this way the operations filter down the tower until we are performing multiplications in $\mathbb{F}_q$.

The computational advantage of adopting a tower of extensions may not be immediately evident. Namely, suppose we were to analyse the complexity of the $\mathbb{F}_{q^{12}}$ multiplication in Example 7.3.1. If we were to employ the naive "schoolbook" method of multiplying two extension field elements, which operates component-wise, then an $\mathbb{F}_{q^{12}}$ multiplication computed directly over $\mathbb{F}_q$ would cost 144 $\mathbb{F}_q$ multiplications. If we instead descend down the tower employing schoolbook multiplication, then an $\mathbb{F}_{q^{12}}$ multiplication would cost 4 $\mathbb{F}_{q^6}$ multiplications, each of which would cost 9 $\mathbb{F}_{q^2}$ multiplications, with each of these costing 4 mul-

tiplications in $\mathbb{F}_q$, giving $4 \cdot 9 \cdot 4 = 144$ base field multiplications in this case too. However, one of the reasons that the towered approach betters a direct extension to $\mathbb{F}_{q^k}$ is because there exist much better (than schoolbook) methods of performing arithmetic in quadratic and cubic extensions. Specifically, the Karatsuba method [KO63] for quadratic extensions allows us to compute multiplications in $\mathbb{F}_{q^{2u}}$ using 3 multiplications in $\mathbb{F}_{q^u}$, or to compute a squaring in $\mathbb{F}_{q^{2u}}$ using only 2 multiplications in $\mathbb{F}_{q^u}$. The same method applied to cubic extensions allows us to compute multiplications in $\mathbb{F}_{q^{3u}}$ using only 6 multiplications in $\mathbb{F}_{q^u}$ (rather than 9), and squarings in $\mathbb{F}_{q^{3u}}$ using 6 $\mathbb{F}_{q^u}$-squarings (which are faster than $\mathbb{F}_{q^u}$-multiplications in general). There are also other methods and variations which are competitive for these small extensions, such as the Toom-Cook method [Too63, CA66], which computes an $\mathbb{F}_{q^{3u}}$ multiplication using only 5 $\mathbb{F}_{q^u}$ multiplications, but this requires a substantially higher number of additions. A helpful report that compares all of these methods in the contexts of pairings is given by Devegili *et al.* [DOSD06]. Referring back to the examples above, and this time descending down the tower using Karatsuba multiplications for the quadratic and cubic extensions gives that $\mathbb{F}_{q^{12}}$ multiplications now cost $3 \cdot 6 \cdot 3 = 54$ $\mathbb{F}_q$ multiplications; a huge improvement over the schoolbook method. We note that a different ordering of the quadratic and cubic towers from $\mathbb{F}_q$ to $\mathbb{F}_{q^{12}}$ could be chosen, and that this would give the same number of $\mathbb{F}_q$ multiplications for a multiplication in $\mathbb{F}_{q^{12}}$, but that there are certainly reasons (other than the twisted curve) that we would prefer one tower over another.

It could potentially be misleading however, to argue that the low number of $\mathbb{F}_q$ multiplications offered by degree 2 and 3 Karatsuba-like methods is what makes the towered extensions preferable to a direct extension. Indeed, the Karatsuba and Toom-Cook algorithms generalise to extensions of any degree [WP06], [Ber01, §6]. In fact, generalised Toom-Cook theoretically guarantees that we will be able to perform the $\mathbb{F}_{q^{12}}$ multiplication from the above example (via a direct extension) using only 23 $\mathbb{F}_q$ multiplications, which is less than half the number of $\mathbb{F}_q$ multiplications used in our towered Karatsuba approach. However, such high-degree generalisations require an enormous number of $\mathbb{F}_q$ additions, and the theoretical number of multiplications they save is nowhere near enough to offset this deficit. Thus, technically speaking, it is in the saving of $\mathbb{F}_q$-additions that the towered approach gains its advantage. Indeed, the additions encountered when performing the highest level multiplications at the

top most sub-extension of the tower filter down linearly to $\mathbb{F}_q$, whilst performing $\mathbb{F}_{q^k}$-arithmetic via a direct extension blows the number of additions out (at the very least) quadratically.

Given the simple test to determine irreducibility of the binomial $x^k - \alpha$ when $q \equiv 1 \bmod 12$ and $k = 2^i 3^j$ above, Koblitz and Menezes defined a *pairing-friendly field* to be a prime field with characteristic q of this form. However, given the number of conditions already imposed on the search for pairing-friendly curves, Benger and Scott argue that this extra restriction is unnecessary [BS10]. They relax this constraint and introduce the notion of *towering-friendly fields*: a field $\mathbb{F}_{q^m}$ is called towering-friendly if all prime divisors of m also divide $q - 1$. For such fields, they invoke Euler's conjectures to give an irreducibility theorem that facilitates all intermediate subextensions to be constructed via a binomial.

Loop shortening has played a major role in the evolution of pairing computation. Indeed, the series of landmark works that are summarised in this section have an impressive evolution of their own. Duursma and Lee [DL03] were the first to show that, in special cases, a bilinear pairing can be obtained without iterating Miller's algorithm as far as the large prime group order r. Barreto *et al.* [BGOS07] generalised this observation to introduce the η_T pairing (the *eta pairing*); a pairing which achieves a much shorter loop length (than r) on any supersingular curve. Hess, Smart and Vercauteren [HSV06] simplified and extended the η_T pairing to ordinary curves, introducing the *ate pairing*, whose loop length is $T = t - 1$, where t is the trace of the Frobenius endomorphism (see Eq. (2.6)), which is much smaller than r in general cases of interest. A number of authors followed this work with observations that in many cases we can do even better than the ate pairing. This included the introduction of the *R-ate pairing* [LLP09], as well as other optimised variants of the ate pairing [MKHO07]. Vercauteren [Ver10] culminated all of these works and introduced the notion of *optimal pairings*, conjecturing a lower bound on the loop length required to obtain a bilinear pairing on any given curve, and showing how to achieve it in many cases of interest. His conjecture was proven soon after by Hess, who drew a line under all the loop-shortening work to date, putting forward a general framework that encompasses all elliptic curve pairings [Hes08].

Our intention in this section is to bring the reader up to speed with optimal pairings, by picking a few examples that illustrate key concepts. For the sake of simplicity, we are forced to skip past some of the key works mentioned in

the last paragraph; in particular, we will not present the η_T pairing that targets supersingular curves, since it is most suited to curves over fields of characteristic 2 and 3. We will also not be giving examples of the works that came between the ate and optimal ate pairing papers (e.g. [MKHO07, LLP09]), in hope that the reader will not have too much trouble following an immediate generalisation.

At a high level, the notion of loop shortening makes use of two observations. Firstly, recall from Chapter 2 (in particular Example 2.2.11), that appropriate endomorphisms on E compute some multiple $[\lambda]P$ from P, which essentially allow us to "skip ahead" in the fundamental computation of $[m]P$ from P. Just as they can be used to shorten the double-and-add loop for scalar multiplications in ECC, efficient endomorphisms can be used to shorten the Miller loop in PBC. The second observation is that, given any two bilinear pairings on E, their product or quotient will also give a bilinear pairing. More generally, we can say that if $e_1, ..., e_n$ are bilinear pairings on E, then $\prod_i e_i^{j_i}$ $(j_i \in \mathbb{Z})$ will also be a bilinear pairing [ZZH08a, Corr. 1].

We start with an example of Scott's idea [Sco05b], which came from the first paper to look at loop shortening on any type of ordinary curve. He looked at a special case of ordinary curves called *not supersingular curves* (NSS). These should not be confused with the more general term non-supersingular, which (by definition) means all ordinary curves. NSS curves are a special type of ordinary curve, but they cover the cases that are most useful in the context of pairings. In fact, we have already seen NSS curves, as they are precisely the curves described in Table 6.2. Essentially, the modularity conditions imposed on the curves $y^2 = x^3 + b$ and $y^2 = x^3 + ax$ in Table 6.1 is what makes them supersingular, because these conditions force the maps ϕ described in that table to be defined over the extension field – i.e. these congruences make ϕ a distortion map. On the other hand, the alternative modularities on the same curves in Table 6.2 mean that the associated ϕ's are defined over $\mathbb{F}_q$. Thus, Scott starts with the motivating question: *under these circumstances, what becomes of these distortion maps?* The rest of his paper responds by showing that they are useful, not as distortion maps, but rather as efficient endomorphisms on E. The following example does not give the details of Scott's algorithm; it merely hints towards it by showing the potential of the endomorphisms ϕ on an NSS curve.

Example 7.3.2 (Magma script). Taking $x = -1$ generates the smallest BN curve (see Example 6.3.3 for the polynomials) with $q = 19$, $E/\mathbb{F}_q : y^2 = x^3 + 2$

and $r = 13$ as the group order. It is clearly an NSS curve (see Table 6.2 or [Sco05b, Eq. 4]). The non-trivial cube roots of unity are defined over $\mathbb{F}_q$, and are $\zeta_3 = 7$ and $\zeta_3^2 = 11$. They both define a different endomorphism on E (e.g. $\zeta_3 : (x, y) \mapsto (\zeta_3 x, y)$) which corresponds to a different scalar multiplication λ, i.e. $(\zeta_3 x, y) = [\lambda](x, y)$. The two different λ's are the solutions of $\lambda^2 + \lambda + 1 = 0 \bmod r$, which comes from $\lambda^3 \equiv 1 \bmod r$ matching $\zeta_3^3 = [1]$ in $\mathrm{End}(E)$, so $\lambda_1 = 9$ and $\lambda_2 = 3$ correspond to ζ_3 and ζ_3^2 respectively. Miller's algorithm would usually double-and-add to compute $[r]P = [\lambda^2 + \lambda + 1]P = [\lambda]([\lambda]P + P) + P$. However, for $P = (x, y)$, the endomorphism allows us to easily calculate the point $[\lambda]P + P = (-(\lambda + 1)x, -y)$. Thus, if we store the values of the points in the $n = \lfloor \log_2 \lambda \rfloor$ doublings that build up to $[\lambda]P$, the values of the points in the second n doublings can be found at the cost of a single multiplication. This is already more efficient, but Scott notices that since the points are related, the lines they contribute in the point doubling phase of Miller's algorithm are similarly related. Namely, the contribution to the pairing value in the first n iterations is $(y_Q - y_i) - m_i(x_Q - x_i)$, where (x_i, y_i) is the point $[2^i]P$, and m_i is the line slope resulting for the point doubling (we use m in this example because λ is already taken). It follows (see [Sco05b, §5]) that the contribution to the pairing value from the final n doublings will be $(-y_Q - y_i) - m_i(\lambda x_Q - x_i)$. This means we only need to loop as far as $n = \lfloor \log_2 \lambda \rfloor$ (rather than $2n = \lfloor \log_2 \lambda^2 \rfloor$) to get all the information we need. See Scott's paper for the algorithm description that ties all this together, where he deals with cases where $\lambda = 2^a + 2^b$. Thus, to finish our example with the algorithm write $\lambda_1 = 2^{a_1} + 2^{b_1}$ and $\lambda_2 = 2^{a_2} + 2^{b_2}$ with $a_1 = 3$, $b_1 = 0$, $a_2 = 1$, $b_2 = 0$.

The ϕ maps on NSS curves clearly offer an advantage, but there is another endomorphism we have already seen that turns out to be much more powerful. Namely, the ate pairing makes use of the Frobenius endomorphism π on E. A key observation is that the Frobenius endomorphism acts trivially on elements in the base field, i.e. $\pi(P) = P$ in $\mathbb{G}_1$, so we instead look at using the trace-zero subgroup $\mathbb{G}_2$ where π acts non-trivially. Here $\pi(Q) = [q](Q)$, but since $[q](Q) = [t - 1](Q)$, we have $\pi(Q) = [T](Q)$ (recall that $T = t - 1$). Hess, Smart and Vercauteren [HSV06] use this endomorphism to derive the ate pairing a_T, which is a map

$$a_T : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T,$$

defined as

$$a_T(Q, P) = f_{T,Q}(P)^{(q^k-1)/r}.$$

It helps to see a brief sketch of their proof as follows. We show that a_T is bilinear by relating its value $f_{T,Q}(P)^{(q^k-1)/r}$ to the Tate pairing (with Q as the first argument), which we already know is bilinear. Since $q \equiv T \bmod r$, $T^k \equiv 1 \bmod r$ (because k is the embedding degree), so write $mr = T^k - 1$ for some m. Recall the Tate pairing (with Q as the first argument) as $e(Q, P) = f_{r,Q}(P)^{(q^k-1)/r}$, which (under simple properties of divisors) means $e(Q, P)^m = f_{mr,Q}(P)^{(q^k-1)/r} = f_{T^k-1,Q}(P)^{(q^k-1)/r}$. We can then (again using simple properties of divisors) split this into a product of $f_{T,[T^i]Q}(P)$, each of which is raised to an appropriate exponent. Since $Q \in \mathbb{G}_2$, each of these $[T^i]Q$'s is the same as $\pi^i(Q)$, and since π is purely inseparable of degree q, all of the values $f_{T,[T^i]Q}(P)$ in the product become $f^{q^i}_{T,Q}(P)$, so we can clean up the exponent to get $e(Q, P) = a_T(Q, P)^v$. The exponent v does not divide r in general, so the bilinearity of the ate pairing follows from that of the Tate pairing (see [HSV06, Th. 1] for the full details).

Since there is a final exponentiation, the optimisations that transformed Miller's algorithm into the BKLS version still apply, so we only need to update the input definitions in Algorithm 7.1. Namely, r becomes T, P and Q (from $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively) switch roles. For no other reason than for ease of future reference, we write these updates in an ate-specific version below. Note that if $T = t - 1 < 0$, then it is fine to take $T = |T|$ [Ver10, §C]. There is only one trick that was used in the Tate pairing that does not carry across to the ate setting. Namely, we can no longer ignore the last bit in the final iteration like we did in Section 7.1, because if an addition occurs in the final iteration it will now be a sloped line, whilst in the Tate pairing the last addition line joined P and $[r - 1]P = -P$ and was therefore vertical.

Example 7.3.3 (Magma script). It helps to immediately see the difference between the ate and Tate pairing, so we will continue on from Example 7.1.2: $q = 47$, $E/\mathbb{F}_q : y^2 = x^3 + 21x + 15$, $\#E(\mathbb{F}_q) = 51$, $r = 17$, $k = 4$, $\mathbb{F}_{q^4} = \mathbb{F}_q(u)$, $u^4 - 4u^2 + 5 = 0$, $P = (45, 23) \in \mathbb{G}_1$ and $Q = (31u^2 + 29, 35u^3 + 11u) \in \mathbb{G}_2$. The trace of Frobenius is $t = -3$, so take $T = 4$. Thus, we will compute the ate pairing via Algorithm 7.2 with only two doublings. We have combined the indeterminate function ℓ and its evaluation $\ell(P)$ at P into the same column to fit the table in. Thus, the ate pairing a_T is computed as $a_T(Q, P) = f_{r,Q}(P)^{(q^k-1)/r} =$

Algorithm 7.2 The BKLS-GHS version of Miller's algorithm for the ate pairing.

Input: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ (Type 3 pairing) and $T = (T_{n-1} \dots T_1 T_0)_2$ with $T_{n-1} = 1$.
Output: $f_{T,Q}(P)^{(q^k-1)/r} \leftarrow f$.

1: $R \leftarrow Q$, $f \leftarrow 1$.
2: **for** $i = n-2$ down to 0 **do**
3: Compute the sloped line function $\ell_{R,R}$ for doubling R.
4: $R \leftarrow [2]R$.
5: $f \leftarrow f^2 \cdot \ell_{R,R}(P)$.
6: **if** $r_i = 1$ **then**
7: Compute the sloped line function $\ell_{R,Q}$ for adding R and Q.
8: $R \leftarrow R + Q$.
9: $f \leftarrow f \cdot \ell_{R,Q}(P)$.
10: **end if**
11: **end for**
12: **return** $f \leftarrow f^{(q^k-1)/r}$.

i/ R_i	steps of Alg. 5.1	point R	update (ℓ); update at P ($\ell(P)$)	paired value f
	1	$(31u^2 + 29, 35u^3 + 11u)$		1
1/0	3-5	$(7u^2 + 25, 37u^3 + 28u)$	$y + (u^3 + 32u)x + 42u^3 + 15u$; $40u^3 + 45u + 23$	$40u^3 + 45u + 23$
0/0	3-5	$(16u^2 + 12, 6u^3 + 24u)$	$y + (28u^3 + 22u)x + 17u^3 + 26u$; $8u^3 + 29u + 23$	$44u^3 + 24u^2 + 41u + 31$
	12		$f_{r,Q}(P) \leftarrow 44u^3 + 24u^2 + 41u + 31$	

$$(44u^3 + 24u^2 + 41u + 31)^{287040} = 21u^3 + 37u^2 + 25u + 25.$$

Notice the price we pay for the much shorter loop in the ate pairing, in that it is now the first argument of the pairing (Q) that is defined over the larger field, so the elliptic curve operations (doublings/additions) and line function computations are now taking place in $\mathbb{F}_{q^k}$. For example, compare the second and third columns of the table in Example 7.3.3 to the table in Example 7.1.2. It is here that the power of a high-degree twist really aids our cause. Namely, utilising the twisting isomorphism allows us to move the points in $\mathbb{G}_2$, which is defined over $\mathbb{F}_{q^k}$, to points in $\mathbb{G}_2'$, which is defined over the smaller field $\mathbb{F}_{q^{k/d}}$. In Example 7.3.3 above where $k = 4$, the maximum degree twist permitted by E is $d = 2$, so we could have performed the point operation and line computations in $\mathbb{F}_{q^{k/2}} = \mathbb{F}_{q^2}$. However, if the curve had have been of the form $y^2 = x^3 + ax$, we could have utilised a $d = 4$ quartic twist (see Section 4.3) and performed these operations all the way down in the base field $\mathbb{F}_q$; i.e. in this case we would pay no price for a much smaller loop. In general though, provided we make use of high-degree twists in the ate pairing, then the price we pay in doing more work

(per iteration) in the larger field is nowhere near enough to offset the savings we gain through having a much shorter loop, meaning that the ate pairing (or one of its variants) is much faster than the Tate pairing. We now turn to describing optimal pairings. Vercauteren [Ver10] begins with the observation that the ate pairing a_T corresponding to $T \equiv q \bmod r$ is a special case of the pairing a_{λ_i} that is obtained by taking any power $\lambda_i \equiv q^i \bmod r$; some specific consequences of this observation were previously considered in [MKHO07, ZZH08b]. Since λ_i corresponds to the loop length of the pairing a_{λ_i}, we would like it to be as small as possible. Thus, we would like to find the smallest value of $q^i \bmod r$ ($i \in \mathbb{Z}$), and since $q^k \equiv 1 \bmod r$, finding the smallest a_{λ_i} would only require testing the possibilities up to $k-1$ ($i = k$ clearly gives the trivial degenerate pairing). However, Vercauteren actually does much better than this by observing that since $q^i \bmod r$ induces a bilinear pairing a_{λ_i}, then any linear combination of $\sum_{i=0}^{l} c_i q^i \equiv 0 \bmod r$ gives rise to a bilinear pairing

$$(Q, P) \mapsto \left(\prod_{i=0}^{l} f_{c_i,Q}^{q^i}(P) \cdot \prod_{i=0}^{l-1} \ell_i \right)^{(q^k-1)/r}, \tag{7.1}$$

where the ℓ_i are simple "one-off" line functions (chords) that are needed to make the bilinearity hold – see [Ver10, Eq. 7] for details. Also, the exponentiations of each of the (at most $\ell + 1$) line functions to the power of q^i should not concern us, as these are just repeated applications of the Frobenius endomorphism in $\mathbb{G}_T$, which is essentially cost-free (more on this in Section 7.5). The main point to note is that the loop lengths of the Miller functions $f_{c_i,Q}$ are the c_i. Thus, we would like to find a multiple mr of r with a base-q expansion $mr = \sum_{i=0}^{l} c_i q^i$ that has the smallest c_i coefficients possible. Vercauteren proceeds naturally by posing this search as a lattice problem, i.e. that such small c_i are obtained by solving for short vectors in the following lattice

$$L = \begin{pmatrix} r & 0 & 0 & \dots & 0 \\ -q & 1 & 0 & \dots & 0 \\ -q^2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ -q^{\varphi(k)-1} & 0 & \dots & 0 & 1 \end{pmatrix}, \tag{7.2}$$

which is spanned by the rows, and where $\varphi(k)$ is the Euler phi function of k.

He then invokes Minkowski's theorem [Min10] to show that there exists a short vector $(v_1, ..., v_{\varphi(k)-1})$ in L such that $\max_i |v_i| \leq r^{1/\varphi(k)}$. Thus, we have an upper bound on the largest Miller loop length that will be encountered when computing the pairing in (7.1). Vercauteren uses this bound to define an optimal pairing [Ver10, Def. 3]: $e(\cdot, \cdot)$ is called an *optimal pairing* if it can be computed in $\log_2 r/\varphi(k) + \epsilon$ Miller iterations, with $\epsilon \leq \log_2 k$. He subsequently conjectures that any bilinear pairing on an elliptic curve requires at least $\log_2 r/\varphi(k)$ Miller iterations. Following [Ver10, Def. 3], Vercauteren also notes that the reason that the dimension of L is $\varphi(k)$ is because we really only need to consider q^i up to $q^{\varphi(k)-1}$. This is due to that fact that $\Phi_k(q) \equiv 0 \bmod r$ implies that q^j with $j > \varphi(k)$ can be written as linear combinations of the q^i $(i \leq \varphi(k) - 1)$ with small coefficients, which means only these q^i should be considered linearly independent.

Before giving examples, we mention a caveat. Observe that $\max_i |c_i| \leq r^{1/\varphi(k)}$ does not imply that the lower bound is met, since the number of Miller iterations required is given by $\sum_i \log_2 c_i$. However, we will be searching for small vectors in the lattice L, where q and r come from families and are therefore given as polynomials $q(x)$ and $r(x)$. Therefore, the c_i in the short vectors will themselves be polynomial expressions $c_i(x)$, meaning that the Miller functions $f_{c_i(x),Q}$ in (7.1) will typically follow from $f_{x,Q}$.

We will illustrate with three families that were used as examples in Section 6. Vercauteren gives more examples. Magma has a built in algorithm `ShortestVectors`() that serves our purpose, but the code we use in the following three examples was written by Paulo Barreto, and passed on to us by Luis Dominguez Perez.

Example 7.3.4 (Magma script). Recall the parameterisations for $k = 12$ BN curves from Example 6.3.3: $t(x) = 6x^2 + 1$, $q(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$ and $r(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$. These were actually used to generate the curve in Example 6.1.2, with $x = 9453956337761452438$ being 67 bits, which generated a 271-bit q and r. Observe that Miller's algorithm to compute $f_{r,P}(Q)$ in the Tate pairing would therefore require around 270 iterations. Alternatively, $t = t(x)$ is 137 bits, so computing the ate pairing $a_T(Q, P) = f_{T,Q}(P)^{(q^k-1)/r}$ would require around 136 iterations. However, Vercauteren's bound suggests we can do even better: since $\varphi(12) = 4$, our loop can be reduced by a factor of 4, i.e. we should require $\log_2 r/4 \approx 68$ iterations. Following (7.2) then, we seek short

vectors in the lattice

$$L = \begin{pmatrix} 36x^4+36x^3+18x^2+6x+1 & 0 & 0 & 0 \\ -6x^2 & 1 & 0 & 0 \\ 36x^3+18x^2+6x+1 & 0 & 1 & 0 \\ 36x^3+24x^2+12x+3 & 0 & 0 & 1 \end{pmatrix},$$

where the $-q(x)^i$ down the first column were immediately reduced modulo $r(x)$. Some short vectors in L are $V_1(x) = (6x+2, 1, -1, 1)$, $V_2(x) = (6x+1, 6x+3, 1, 0)$, $V_3(x) = (-5x-1, -3x-2, x, 0)$, $V_4(x) = (2x, x+1, -x, x)$. In reference to the point we made before this example, we prefer the short vectors with the minimum number of coefficients of size x, so choosing $V_1(x)$ and computing the optimal ate pairing $a_{V_1(x)}$ following (7.1) gives

$$\begin{aligned} a_{V_1(x)} &= (f_{6x+2,Q}(P) \cdot f_{1,Q}(P) \cdot f_{-1,Q}(P) \cdot f_{1,Q}(P) \cdot M)^{(q^k-1)/r}, \\ &= (f_{6x+2,Q}(P) \cdot M)^{(q^k-1)/r}, \end{aligned}$$

where $f_{1,Q} = 1$ and $f_{-1,Q} = 1/f_{1,Q}v_Q$ (which disappears in the final exponentation) can be discarded, and M is a product of 3 simple line functions that are computed easily – this example is in [Ver10, IV.A], where M is defined. The only Miller loop we need to compute is $f_{6x+2,Q}(P)$, which for our x-value, is 69 bits, meaning the optimal pairing indeed requires $\log_2 r/4 \approx 68$ iterations. Notice then, the difference between the ease of using $V_1(x)$ compared to any of the other short vectors above, which all suggest more than one Miller loop.

Example 7.3.5 (Magma script). Recall the parameterisations for $k = 16$ KSS curves from Example 6.3.4 as $t(x) = (2x^5 + 41x + 35)/35$, $q(x) = (x^{10} + 2x^9 + 5x^8 + 48x^6 + 152x^5 + 240x^4 + 625x^2 + 2398x + 3125)/980$ and $r(x) = (x^8 + 48x^4 + 625)/61250$. For any x-value, the Tate pairing requires computing the function $f_{x^8+48x^4+625,P}(Q)$, whilst the ate pairing computes the function $f_{(2x^5+41x+35)/35,Q}(P)$. Since $\varphi(k) = 8$, the ate pairing is not optimal, i.e. $\log_2 r/8$ should have an optimal pairing loop length of order $O(x)$, not $O(x^5)$. Thus, we

look for short vectors in the lattice

$$L = \begin{pmatrix} x^8+48x^4+625 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2x^5-41x & 35 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4x^6+117x^2 & 0 & 175 & 0 & 0 & 0 & 0 & 0 \\ 2x^7-29x^3 & 0 & 0 & 875 & 0 & 0 & 0 & 0 \\ 1x^4+24 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\ -1x^5-38x & 0 & 0 & 0 & 0 & 35 & 0 & 0 \\ -3x^6-44x^2 & 0 & 0 & 0 & 0 & 0 & 175 & 0 \\ 11x^7+278x^3 & 0 & 0 & 0 & 0 & 0 & 0 & 875 \end{pmatrix}.$$

A nice short vector is $V(x) = (x, 1, 0, 0, 0, -2, 0, 0)$, so indeed an optimal pairing is

$$a_{V(x)} = (f_{x,Q}(P) \cdot f_{-2,Q}(P) \cdot M)^{(q^k-1)/r},$$

where M is again a product of simple one-off lines, and we can compute $f_{-2,Q}(P)$ as $1/f_{2,Q}(P)$, since the vertical line that makes two equal evaporates in the final exponentiation. Note that $f_{2,Q}(P)$ is simply the first doubling of Q at P, and that $f_{x,Q}(P)$ is the only Miller loop required.

Example 7.3.6 (Magma script). Recall the parameterisations for a $k = 24$ BLS curve from Example 6.3.2 as $t(x) = x + 1$, $q(x) = (x-1)^2(x^8 - x^4 + 1)/3 + x$ and $r(x) = \Phi_{24}(x) = x^8 - x^4 + 1$. The Tate pairing requires the computation $f_{x^8-x^4+1,P}(Q)$ whilst the ate pairing computes $f_{x,Q}(P)$. Since $\varphi(k) = 8$, the ate pairing is already optimal, i.e. it has a loop length of $\log_2(r)/8$. In cases when the ate pairing is not optimal, like the previous two examples, it is common that other variants like the R-ate pairing of [LLP09] also achieve optimality. For example, Scott uses the R-ate pairing to achieve optimality for $k = 12$ and $k = 18$ implementations targeting the 128 and 192-bit security levels [Sco11, Table 1].

7.4 Low Hamming weight loops

This short section describes a more obvious optimisation to Miller's algorithm. This trick was suggested in the very early papers on pairing computation, but for reasons that will become clear in a moment, we have delayed its introduction in this section until after we described the ate and optimal ate pairings. Regardless of the pairing-based protocol, the loop length of the pairing is known publicly;

therefore, unlike ECC where we try to avoid special choices of scalars that might give attackers unnecessary advantage, in PBC there is no problem in specialising the choice of the loop length. In this light, it is advantageous to use curves where the loop length has a low Hamming weight, thus minimising the number of additions incurred in Miller's algorithm.

For supersingular curves over prime fields, where $\#E(\mathbb{F}_q) = q + 1$, finding a curve whose large prime divisor r has low Hamming weight is relatively easy. Thus, in the early days, facilitating a low Hamming weight Miller loop was not too difficult. However, once the introduction of parameterised families were needed for higher embedding degrees, the polynomial representation for $r(x)$ meant that controlling the loop length (r) of the Tate pairing was a little more difficult. The best we could do in this scenario is search for x values of low Hamming weight, in the hope that the polynomial $r(x)$ wouldn't completely destroy this. Nowadays however, the introduction of the ate and optimal ate pairings makes this optimisation very relevant. Namely, as we saw in the examples in the previous section, the loop length associated with the optimal Miller function is often some small function of x, if not x itself. Thus, choosing x to be of low Hamming weight can be very advantageous for a faster Miller loop, as we show in the following example. In fact, we will see in the next section that a faster Miller loop is only a partial consequence.

Example 7.4.1 (Magma script). Both $x = 258419657403767392$ and $x = 144115188109674496$ are 58-bit values that result in $k = 24$ BLS curves suitable for pairings at the 224-bit security level. The former was found by kick-starting the search at a random value between 2^{57} and 2^{58}, and as such, has a Hamming weight of 28, as we would expect. On the other hand, the second value is actually $2^{57}+2^{25}+2^{18}+2^{11}$, which has Hamming weight 4. Thus, we would much prefer the second value since this would result in 24 less additions through the Miller loop. Another nice alternative that gives similar parameter sizes is $x = 2^{56} + 2^{40} - 2^{20}$, which does not have a low Hamming weight, but rather a low NAF-weight (weight in the signed binary representation), for which Miller's algorithm can be easily updated to take advantage of.

7.5 The final exponentiation

Until now, our optimisations have all applied to the Miller loop. This was a natural place to look for tricks and shortcuts in the early days, since at low levels of security, the Miller loop is by far the bottle-neck of the algorithm. However, as the security level increases, the relative cost of the final exponentiation also increases [DS10]. It appears that, all known high-level optimisations considered, pairings on BN curves at the 128-bit security level is roughly the "crossover point" where the complexities of the Miller loop and the final exponentiation are similar [AKL+11, Table 4], [BGDM+10, Table 3], [NNS10, Table 2]. Thus, at higher levels of security, the final exponentiation is the most time-consuming stage of the pairing computation.

For curves belonging to families, Scott *et al.*'s algorithm [SBC+09a] is the fastest method to date. In this section we illustrate their technique by means of an example, which we take directly from our joint work with Kristin Lauter and Michael Naehrig [CLN11]. This work looked at $k = 24$ BLS curves in detail, since this family is a frontrunner for high-security pairings, particularly when targeting 256-bit security. There are several other examples looked at in [SBC+09a].

We start with a brief description of the general algorithm, before applying it to our particular case. Suppose k is even and write $d = k/2$. We start by splitting the final exponent into three components

$$(q^k - 1)/r = \underbrace{[(q^d - 1)] \cdot [(q^d + 1)/\Phi_k(q)]}_{\text{easy part}} \cdot \underbrace{[\Phi_k(q)/r]}_{\text{hard part}},$$

where the two components on the left are the "easy part" because (the second bracket reduces to powers of q and) raising elements in $\mathbb{F}_{q^k}$ to the power of q involves a simple application of the Frobenius operator π, which almost comes for free. It is the $\Phi_k(q)/r$ term that does not reduce to such a form and which is aptly named the "hard part" of the final exponentiation. Suppose we have already exponentiated through the easy part, and our intermediate value is $m \in \mathbb{F}_{q^k}$. The straightforward way to perform the hard part, i.e. $m^{\Phi_k(q)/r}$, is to write the exponent in base q as $\Phi_k(q)/r = \sum_{i=0}^{n-1} \lambda_i q^i$, and to further exploit repeated applications of π in

$$m^{\Phi_k(q)/r} = (m^{q^{n-1}})^{\lambda_{n-1}} \dots (m^q)^{\lambda_1} \cdot m^{\lambda_0},$$

so that all the m^{q^i} terms essentially come for free, and the hard part becomes the individual exponentiations to the power of the λ_i, which are performed using generic methods. These methods, however, do not take advantage of the polynomial description of q, which is where Scott *et al.*'s work advances beyond the more obvious speed-ups.

Example 7.5.1 (Magma script). Recall the $k = 24$ BLS parameterisations from Example 7.3.6: $t(x) = x + 1$, $q(x) = (x-1)^2(x^8 - x^4 + 1)/3 + x$ and $r(x) = \Phi_{24}(x) = x^8 - x^4 + 1$. To give an idea of the task we are up against, suppose we are targeting the 256-bit security level, as we did with these curves in Example 6.3.2 with $x = 9223372036854782449$. The final exponentiation in this case involves raising a 15082-bit value $f \in \mathbb{F}_{q^{24}}$, to the 14578-bit exponent $(q^{24}-1)/r$, a number far bigger than what we would like to write here (but see the corresponding script). Performing this exponentiation using a naive square-and-multiply with no optimisations would therefore involve 14578 squarings and roughly half as many multiplications in the 15082-bit field, a computation that would blow out the pairing complexity by several orders of magnitude. To take a much faster route, we start by splitting the exponent as

$$(q^{24} - 1)/r = \underbrace{[(q^{12} - 1) \cdot (q^4 + 1)]}_{\text{easy part}} \cdot \underbrace{[(q^8 - q^4 + 1)/r]}_{\text{hard part}}.$$

We compute $f^{(q^k-1)/r} = \left(f^{(q^{12}-1)\cdot(q^4+1)}\right)^{(q^8-q^4+1)/r}$. The exponentiation inside the parentheses is almost free, since $f^{q^{12}}$ is just 12 repeated applications of the Frobenius operation π, and similarly for raising to the power of q^4, so the easy part essentially incurs just a couple of multiplications and maybe an inversion. We are now left with the exponent $(q^8 - q^4 + 1)/r$, for which we can not pull out any more "easy factors". However, a very helpful observation which aids the remaining computations is that, after the first exponentiation to the power $q^{12}-1$, the value $m \in \mathbb{F}_{q^{24}}$ is now such that its norm is $N_{\mathbb{F}_{q^{24}}/\mathbb{F}_{q^{12}}}(m) = 1$. This allows any inversions in $\mathbb{F}_{q^{24}}$ to be computed for free using a simple conjugation [SB04, NBS08, SBC+09a], and any squarings in $\mathbb{F}_{q^{24}}$ to be computed more efficiently than standard $\mathbb{F}_{q^k}$ squarings [GS10, Kar10, AKL+11]. We now make use of the non-trivial part of the algorithm in [SBC+09a], and write the hard part as

$$(q(x)^8 - q(x)^4 + 1)/r(x) = \sum_{i=0}^{7} \lambda_i(x) q(x)^i.$$

In an appendix of her thesis, Benger [Ben10] computed the λ_i for a range of curve families, including BLS curves with $k = 24$, giving $\lambda_i = \nu_i/3$, where

$$\begin{aligned}
\nu_7(x) &= x^2 - 2x + 1, \\
\nu_6(x) &= x^3 - 2x^2 + x = x \cdot \nu_7(x), \\
\nu_5(x) &= x^4 - 2x^3 + x^2 = x \cdot \nu_6(x), \\
\nu_4(x) &= x^5 - 2x^4 + x^3 = x \cdot \nu_5(x), \\
\nu_3(x) &= x^6 - 2x^5 + x^4 - x^2 + 2x - 1 = x \cdot \nu_4(x) - \nu_7(x), \\
\nu_2(x) &= x^7 - 2x^6 + x^5 - x^3 + 2x^2 - x = x \cdot \nu_3(x), \\
\nu_1(x) &= x^8 - 2x^7 + x^6 - x^4 + 2x^3 - x^2 = x \cdot \nu_2(x), \\
\nu_0(x) &= x^9 - 2x^8 + x^7 - x^5 + 2x^4 - x^3 + 3 = x \cdot \nu_1(x) + 3.
\end{aligned}$$

This representation reveals another nice property exhibited by $k = 24$ BLS curves: namely, a very convenient way to compute the ν_i with essentially just multiplications by x. Letting $\mu_i = m^{\nu_i(x)}$, this structure allows us to write the hard part of the final exponentiation as

$$m^{(q^8-q^4+1)/r} = \mu_0 \cdot \mu_1^p \cdot \mu_2^{p^2} \cdot \mu_3^{p^3} \cdot \mu_4^{p^4} \cdot \mu_5^{p^5} \cdot \mu_6^{p^6} \cdot \mu_7^{p^7},$$

where the μ_i can be computed using the following sequence of operations:

$$\begin{aligned}
\mu_7 &= (m^x)^x \cdot (m^x)^{-2} \cdot m,\ \mu_6 = (\mu_7)^x,\ \mu_5 = (\mu_6)^x,\ \mu_4 = (\mu_5)^x, \\
\mu_3 &= (\mu_4)^x \cdot (\mu_7)^{-1},\ \mu_2 = (\mu_3)^x,\ \mu_1 = (\mu_2)^x, \mu_0 = (\mu_1)^x \cdot m^2 \cdot m.
\end{aligned}$$

The computation of $m^{(q^8-q^4+1)/r}$ requires 9 exponentiations by x, 12 multiplications in $\mathbb{F}_{q^{24}}$, 2 special squarings, 2 conjugations to compute the inverses and 7 q-power Frobenius operations. We detail a possible scheduling for the full exponentiation routine in Table 7.1. Note that we can simply forget about the difference between the λ_i and the ν_i; by leaving out the 3 in the denominators, we just compute the third power of the pairing.

7.6 Other optimisations

There are hundreds of papers that have helped accelerate pairing computation to the point it is at today. Of course, we could not delve into the details of

`FinalExp`	Input: $f_{r,Q}(P) \in \mathbb{F}_{q^{24}}$ and loop parameter x

`Initialize` $f \leftarrow f_{r,Q}(P)$,

$t_0 \leftarrow 1/f$, $m \leftarrow \overline{f}$, $m \leftarrow m \cdot t_0, t_0 \leftarrow \pi_q^4(m), m \leftarrow m \cdot t_0$,

$m_1 \leftarrow m^x$, $m_2 \leftarrow m_1^x$ $m_1 \leftarrow m_1^2$, $m_1 \leftarrow \overline{m_1}$, $\mu_7 \leftarrow m_2 \cdot m_1, \mu_7 \leftarrow \mu_7 \cdot m$,
$\mu_6 \leftarrow \mu_7^x$, $\mu_5 \leftarrow \mu_6^x$, $\mu_4 \leftarrow \mu_5^x$, $\mu_7' \leftarrow \overline{\mu_7}$, $\mu_3 \leftarrow \mu_4^x$, $\mu_3 \leftarrow \mu_3 \cdot \mu_7'$,
$\mu_2 \leftarrow \mu_3^x$, $\mu_1 \leftarrow \mu_2^x$, $\mu_0 \leftarrow \mu_1^x$, $m' \leftarrow m^2$, $\mu_0 \leftarrow \mu_0 \cdot m'$, $\mu_0 \leftarrow \mu_0 \cdot m$,
$f \leftarrow \pi_q(\mu_7), f \leftarrow f \cdot \mu_6$, $f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_5$, $f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_4$,
$f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_3$, $f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_2$, $f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_1$,
$f \leftarrow \pi_q(f)$, $f \leftarrow f \cdot \mu_0$,

`Return` $f_{r,Q}(P)^{(q^{24}-1)/r} \leftarrow f$.

	Output: $f_{r,Q}(P)^{(q^{24}-1)/r}$

Table 7.1: The final exponentiation for BLS curves with $k = 24$.

all the optimisations and improvements that are available. For example, since our exposition is largely concerned with computational efficiency, we have not covered the work on compressed pairings [SB04,NBS08,Nae09] which targets low bandwidth environments, or the work by Galbraith and Lin [GL09] which looks at computing pairings using x-coordinates only.

In addition, a number of papers have looked at operations in a pairing-based protocol that are not the pairing computation itself, the most important of which are point multiplications in the pairing-specific groups $\mathbb{G}_1$ and $\mathbb{G}_2$. In Section 6.3 (and Table 6.2 in particular) we saw that the pairing-friendly curves that are most useful in practice are those of the form $E : y^2 = x^3 + b$ or $E : y^2 = x^3 + ax$. In both of these cases there is a non-trivial endomorphism $\phi \in \mathrm{End}(E)$ that facilitates faster point multiplications via GLV/GLS scalar decompositions (refer to Example 2.2.11). For point multiplications in $\mathbb{G}_1$ that take place over the base field, the standard GLV decomposition can make use of ϕ to decompose the scalar. For the more expensive point multiplications in $\mathbb{G}_2$ that take place over extension fields, the GLS technique (which additionally exploits the non-trivial action of the Frobenius endomorphism π) can be used for higher dimensional decompositions. We particularly make mention of the work of Scott *et al.* [SBC+09b] and Fuentes-Castaeda *et al.* [FCKRH11], who consider fast hashing to the group $\mathbb{G}_2$, the bottleneck of which is the expensive cofactor scalar multiplication in $\mathbb{G}_2$. For pairings to become widespread in the industry, efficient off-the-shelf solutions to

all the operations involved in pairing-based protocols need to be available.

Finally, we mention that some recent work has revived the potential of the Weil pairing in practice [AKMRH11, AFCK+12]. Indeed, since the complexity of the final exponentiation in the Tate pairing (and its ate-like variants) overtakes that of the Miller loop at higher security levels, it is natural to reconsider the Weil pairing for these scenarios. Although several of the Tate-specific optimisations do not translate across, loop shortening is available in the Weil pairing. Indeed, Hess presented a general framework for loop shortening in both the Tate and Weil pairing methodologies [Hes08]. Aranha *et al.* used this idea to derive Weil pairing variants that are particularly suited to the parallel environment [AKMRH11], and actually showed that their new Weil pairing is substantially faster than the optimal ate pairing when 8 cores are used in parallel.