Avoiding Full Extension Field Arithmetic in Pairing Computations

Craig Costello, Colin Boyd, Juan Manuel González Nieto, and Kenneth Koon-Ho Wong.

Information Security Institute
Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{craig.costello,c.boyd,j.gonzaleznieto,kk.wong}@qut.edu.au

Abstract. The most costly operations encountered in pairing computations are those that take place in the full extension field $\mathbb{F}_{p^k}$. At high levels of security, the complexity of operations in $\mathbb{F}_{p^k}$ dominates the complexity of the operations that occur in the lower degree subfields. Consequently, full extension field operations have the greatest effect on the runtime of Miller's algorithm. Many recent optimizations in the literature have focussed on improving the overall operation count by presenting new explicit formulas that reduce the number of subfield operations encountered throughout an iteration of Miller's algorithm. Unfortunately, almost all of these improvements tend to suffer for larger embedding degrees where the expensive extension field operations far outweigh the operations in the smaller subfields. In this paper, we propose a new way of carrying out Miller's algorithm that involves new explicit formulas which reduce the number of full extension field operations that occur in an iteration of the Miller loop, resulting in significant speed ups in most practical situations of between 5 and 30 percent.
Keywords: Pairings, Miller's algorithm, Tate pairing, ate pairing.

1 Introduction

At the beginning of this century, pairing-based cryptography became extremely popular after the first practical identity-based encryption scheme was constructed using the powerful bilinearity property of pairings [13]. Accompanied by many other exciting breakthroughs that resulted from pairings, the discovery of ID-based encryption heightened the demand for practical pairings which can be computed efficiently. Since then, much research has been invested towards achieving faster pairings and consequently the speed of computing Miller's algorithm [34] for calculating pairings has significantly increased. Initial improvements in pairing computations were spearheaded by evidence that the Tate pairing was much more efficient than the Weil pairing, since the final exponentiation in the Tate pairing facilitates several clever simplifications in the Miller iterations [4, 6, 7, 35]. The continual evolution of security requirements and standards has lead to a large emphasis being placed on obtaining secure curve constructions for a range of embedding degrees. As a result, the construction of pairing-friendly curves has become an active field of research in itself [5, 14, 36, 20, 8, 24, 10, 21, 30], so that cryptographers can now choose from an array of flexible curve options that offer high levels of efficiency in pairing computations [22]. More recently, Hess, Smart and Vercauteren [27] generalized prior work by Duursma and Lee [19] and Barreto *et al.* [3] to develop the ate pairing which benefits from a truncated loop length and is usually much faster than the Tate pairing. The ate pairing has since enjoyed its own improvements [33, 32], to the point where ate pairing variants can now be computed with optimal loop lengths [37, 26].

In very recent times, researchers have achieved further speedups by deriving fast explicit formulas for specific stages of a Miller iteration [15, 18, 28, 1, 16, 17], so that each

* The first author acknowledges funding from the Queensland Government Smart State PhD Scholarship. This work has been supported in part by the Australian Research Council through Discovery Project DP0666065.

iteration requires less subfield operations, resulting in a faster pairing. Unfortunately, such improvements are less effective when applied to the Tate pairing because the operations that are saved occur in the base field $\mathbb{F}_p$, and as the embedding degree k gets large, the complexity of the operations occurring in the full extension field $\mathbb{F}_{p^k}$ dominates the complexity of those operations occurring in $\mathbb{F}_p$, so that the relative speedup resulting from savings in the base field becomes much less. In the ate pairing with a twist of degree d, faster explicit formulas save operations in the subfield $\mathbb{F}_{p^{k/d}}$, the complexity of which grows at the same rate as the complexity of operations in $\mathbb{F}_{p^k}$, so that an increased embedding degree will not drastically effect the relative speedup. Nevertheless, optimized implementations of the ate pairing make use of the highest available twist for a given k, so that the complexity of operations in $\mathbb{F}_{p^{k/d}}$ is much less than those in $\mathbb{F}_{p^k}$. For example, the ate pairing computed on a BN curve [8] where $k = 12$ uses a sextic twist ($d = 6$), so that any computations saved through faster explicit formulas are those in the much smaller field $\mathbb{F}_{p^2}$. An optimized construction of the extension field [31, 9] results in the complexity of operations in $\mathbb{F}_{p^{12}}$ being no less than 15 times greater than the analogous operations in $\mathbb{F}_{p^2}$, so that any speedups that result from faster explicit formulas are still greatly overshadowed by the expensive operations in $\mathbb{F}_{p^{12}}$. At any level, full extension field operations greatly outweigh subfield operations for both Tate-and ate-like pairings.

Eisenträger, Lauter and Montgomery [29] managed to avoid full extension field arithmetic in pairing computations by combining two linear Miller functions into a single function of degree 2, which they call a parabola, and achieving a speedup by replacing two multiplications by the two linear functions with a single multiplication by the parabola. However, the algorithm in [29] has limited application in state-of-the-art pairing implementations because it only applies to stages of the algorithm that require point addition, and optimized implementations will choose loop parameters with low Hamming weight that minimize the occurrence of these additions. Blake, Murty and Xu [12] extended the observations in [29] to form combinations of Miller lines that apply to every iteration of the Miller loop, proposing a version of Miller's algorithm that is somewhat analogous to the 2^n-ary windowing methods for general exponentiation (cf. [2, §9]), using a window of size $n = 2$. Again, the techniques proposed in [12] are not optimized for modern implementations of Miller's algorithm because the main benefit of the combined linear functions in their case was to avoid field divisions, a problem that became obsolete after the introduction of denominator elimination in [4]. In this paper, we extend the notion of combining Miller lines into higher degree polynomials and present a more general approach, which we call Miller 2^n-tuple-and-add. Specifically, we show how to combine explicit formulas from n consecutive Miller double-and-add iterations into more complicated explicit formulas for one Miller 2^n-tuple-and-add iteration. The price we pay for spending more subfield operations to evaluate these more complicated formulas is greatly rewarded by the large savings that result from avoiding costly arithmetic in the full extension field. For both Tate and ate-like pairings, we show that the Miller 2^n-tuple-and-add algorithm achieves significant speedups over the standard Miller double-and-add routine for the majority of pairing-friendly embedding degrees. Our method offers (among others) the following important advantages over the prior work in [12]:

- Our method works for general $n \geq 1$. All prior work (except for $n = 2$ in [12]) has used $n = 1$.
- Our method handles any addition steps encountered in Miller's 2^n-tuple-and-add algorithm in exactly the same way, regardless of the 2^n-ary representation of the loop

parameter. The method in [12] for $n = 2$ uses formulas that differ depending on the quarternary representation of the loop parameter. An important consequence of this is that higher values of n do not result in more complex additions, as they do for $n = 2$ in [12].
- The techniques and analyses in [12] focus on reducing the number of field divisions (inversions) that occur in the affine representation of the Miller lines. Field inversions are extremely costly in pairing implementations and have been phased out thanks to denominator elimination and the application of non-affine (projective) coordinate systems to pairing computations that eliminate field inversions altogether. The explicit formulas herein are derived using projective coordinates, and these formulas are reduced to give much faster operation counts.

The rest of this paper is organized as follows. Section 2 provides a brief background on pairings and Miller's algorithm. In Section 3 we describe the general Miller 2^n-tuple-and-add algorithm, before discussing a general strategy to obtain explicit formulas for 2^n-tuple-and-add in Section 4. In Section 5, we derive explicit formulas for the cases of Miller quadruple-and-add ($n = 2$) and Miller octuple-and-add ($n = 3$), and obtain operation counts for a typical iteration in each scenario. In Section 6, we compare the operation counts for the quadruple-and-add and octuple-and-add algorithm with the standard double-and-add algorithm. We draw conclusions in the same section.

2 Background

Let E be an elliptic curve over $\mathbb{F}_p$. Assume E is given by the short Weierstrass equation $y^2 = x^3 + ax + b$ and let $\mathcal{O}$ be the neutral element on E. For the points $R, S \in E$, let $l_{R,S}$ and $v_{R,S}$ respectively be the sloped and vertical lines in the standard chord-and-tangent addition of R and S, the divisors of which are $\mathrm{div}(l_{R,S}) = (R) + (S) + (-(R + S)) - 3(\mathcal{O})$ and $\mathrm{div}(v_{R,S}) = (-(R + S)) + (R + S) - 2(\mathcal{O})$. When $R = S$, we have $l_{R,R}$ and $v_{R,R}$ as the sloped and vertical lines in the point doubling of R. Herein we let $g_{R,S}$ represent the quotient $g_{R,S} = l_{R,S}/v_{R,S}$, with associated divisor $\mathrm{div}(g_{R,S}) = (R) + (S) - (R + S) - (\mathcal{O})$. For $v \in \mathbb{Z}$, let $f_{v,R}$ be a function with divisor

$$f_{v,R} = v(R) - ([v]R) - (v - 1)(\mathcal{O}).$$

Let k be the embedding degree of E with respect to some large prime r and let $E[r]$ denote the group of r-torsion points on E. We use π_p to denote the p-power Frobenius endormorphism on E and we define two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ using the two eigenspaces of π_p as $\mathbb{G}_1 = E[r] \cap \ker(\pi_p - [1])$ and $\mathbb{G}_2 = E[r] \cap \ker(\pi_p - [p])$.

For two points $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, the Tate pairing $e_r : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ is computed as $e_r(P, Q) = f_{r,P}(Q)^{(p^k-1)/r}$. Let $T = t - 1$, where t is the trace of the Frobenius on E. The ate pairing $a_T : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_3$ is computed as $a_T(Q, P) = f_{T,Q}(P)^{(p^k-1)/r}$. In the coming sections, we treat both pairings simultaneously by letting the required pairing be computed as $f_{m,R}(S)^{(p^k-1)/r}$, where it is understood that in the Tate pairing we have $m = r$, $R \in \mathbb{G}_1$ and $S \in \mathbb{G}_2$, whilst in the ate pairing we have $m = T$, $R \in \mathbb{G}_2$ and $S \in \mathbb{G}_1$.

When counting field operations, we use $\mathbf{M}$ and $\mathbf{S}$ to denote the respective costs of a multiplication and a squaring in the field $\mathbb{F}_{p^k}$, and we use $\mathbf{m}$ and $\mathbf{s}$ to represent the costs of a multiplication and a squaring in the subfield $\mathbb{F}_{p^e}$, where $e = 1$ for Tate-like pairings and

$e = k/d$ for ate-like pairings with twists of degree d. In some instances it is necessary to count operations in more than two fields, in which case we avoid ambiguities by letting $\mathbf{m}_i$ and $\mathbf{s}_i$ denote the costs of a multiplication and a squaring in the field $\mathbb{F}_{p^i}$. Lastly, we report the cost of a multiplication by a curve constant (or a small power of a curve constant) as $\mathbf{d}$.

Since the introduction of the original ate pairing [27], several variants with even shorter loop lengths have emerged [33], including the R-ate pairing [32] which often achieves the optimal loop length [37, 26]. All of these variants also take $R \in \mathbb{G}_2$ and $S \in \mathbb{G}_1$ and compute $f_{m,R}(S)$, the only difference being the construction (and size) of the loop parameter m. We refer to all such pairings collectively as ate-like pairings ($a : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_3$), and hereafter we make no specifications regarding the loop length, since it plays no role in the results of this paper. Identically, we put the twisted ate pairing [27] under the umbrella of Tate-like pairings ($e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$), since the twisted ate pairing takes its respective inputs from the same groups as the Tate pairing.

Using $f_{i+j,R} = f_{i,R} \cdot f_{j,R} \cdot g_{[i]R,[j]R}$, the usual version of Miller's algorithm computes the required function in $\lfloor \log_2(m) \rfloor$ iterations by initializing $f_{1,R}(S) = 1$ and progressively building the functions $f_{v,R}(S)$ (for $v < m$) to approach $f_{m,R}(S)$ in a double-and-add-like fashion, as summarized in Algorithm 1.

Algorithm 1 Miller double-and-add Algorithm

Input: R, S, $m = (m_{l-1}...m_1, m_0)_2$.
Output: $f_{m,R}(S)$.

1: $T \leftarrow R$, $f \leftarrow 1$.
2: **for** $i = l - 2$ to 0 **do**
3: Compute $g = g_{T,T}(S)$
4: $T \leftarrow [2]T$.
5: $f \leftarrow f^2 \cdot g$.
6: **if** $m_i \neq 0$ **then**
7: Compute $g = g_{T,R}(S)$
8: $T \leftarrow T + R$.
9: $f \leftarrow f \cdot g$.
10: **end if**
11: **end for**
12: **return** f.

At the beginning of an iteration of Algorithm 1, let the intermediate multiple of the point R be $T = [v]R$, so that the current Miller function f relating to the point T has divisor

$$\mathrm{div}(f_{v,R}) = v(R) - ([v]R) - (v-1)(\mathcal{O}).$$

Miller's double-and-add algorithm forms the function $f_{2v,R}$ relating to the point $[2]T = [2v]R$ as $f_{2v,R} = f_{v,R}^2 \cdot g_{[2]T}$, where $\mathrm{div}(g_{[2]T}) = 2(T) - ([2]T) - (\mathcal{O})$, so that $f_{2v,R}$ has divisor

$$\begin{aligned} \mathrm{div}(f_{2v,R}) &= \mathrm{div}(f_{v,R}^2 \cdot g_{[2]T}) = 2 \cdot \mathrm{div}(f_{v,R}) + \mathrm{div}(g_{[2]T}) \\ &= 2 \cdot \big(v(R) - ([v]R) - (v-1)(\mathcal{O})\big) + (2(T) - ([2]T) - (\mathcal{O})) \\ &= 2v(R) - ([2v]R) - (2v-1)(\mathcal{O}). \end{aligned}$$

We obtain the Miller function $f_{2v,R}$ by squaring the Miller function $f_{v,R}$ and multiplying this result by the "line" function(s) involved in the point doubling of T. In a standard

implementation of Miller's algorithm, the functions $f_{v,R}$ and $g_{T,T}$ are contained in the full extension field, so that the function update (step 5 of Algorithm 1) comes at a cost of $1\mathbf{M}+1\mathbf{S}$. Assuming (for now) that no intermediate addition operations are required (i.e. $m_i = 0$ for n consecutive i's in Algorithm 1), n consecutive iterations of Miller's double-and-add algorithm above transform the function $f_{v,R}$ into the function $f_{2^n v,R}$. The cost of the n function updates that occur in n such iterations is then $n\mathbf{M} + n\mathbf{S}$.

3 2^n-ary pairings: Miller 2^n-tuple-and-add

In this section we generalize the above (double-and-add) method by combining n consecutive doubling steps into one 2^n-tupling step and we show that this reduces the number of expensive function updates that occur in $\mathbb{F}_{p^k}$. For any n, we naturally refer to this process as the Miller 2^n-tuple-and-add algorithm. Consider n consecutive squarings on the function $f_{v,R}$, which equates to raising $f_{v,R}$ to the power 2^n. The divisor of the resulting function is given as

$$\mathrm{div}\big((f_{v,R})^{2^n}\big) \quad = \quad 2^n \cdot \mathrm{div}(f_{v,R}) \quad = \quad 2^n v(R) - 2^n([v]R) - 2^n(v-1)(\mathcal{O}). \tag{1}$$

To obtain the desired Miller function $f_{2^n v,R}$ from $f_{v,R}$, we must now find a function f^* such that $\mathrm{div}((f_{v,R})^{2^n}) + \mathrm{div}(f^*) = \mathrm{div}(f_{2^n v,R}) = 2^n v(R) - ([2^n v]R) - (2^n v - 1)(\mathcal{O})$. We construct f^* as

$$f^* = \prod_{i=1}^{n} (g_{[2^{i-1}]T,[2^{i-1}]T})^{2^{n-i}}, \tag{2}$$

the divisor of which is

$$\begin{aligned} \mathrm{div}(f^*) \quad &= \quad \sum_{i=1}^{n} 2^{n-i} \cdot \mathrm{div}(g_{[2^{i-1}]T,[2^{i-1}]T}) \quad = \quad \sum_{i=1}^{n} 2^{n-i} \cdot (2([2^{i-1}]T) - ([2^i]T) - (\mathcal{O})) \\ &= \quad 2^n(T) - ([2^n]T) - (2^n - 1)(\mathcal{O}). \end{aligned} \tag{3}$$

Substituting $T = [v]R$ into (3) and combining this with (1) reveals that $\mathrm{div}((f_{v,R})^{2^n}) + \mathrm{div}(f^*) = \mathrm{div}(f_{2^n v,R})$, so that f^* is indeed the required function. We note that the construction of f^* is intuitive. Namely, f^* is simply the product of the n different g's that are formed throughout each of the n equivalent double-and-add iterations, each of which accumulates a different exponent depending on how many squarings it encounters in the iterations that follow. In this light, Miller 2^n-tuple-and-add is much the same as Miller double-and-add; the major difference is that in Miller 2^n-tuple-and-add we do not multiply the Miller function by its update g immediately after it is squared. Rather, we form a product f^* of n powers of such g's and we delay the multiplication of f^* by f so that it occurs only once in what is the equivalent of n double-and-add iterations.

For the addition step in the Miller 2^n-tuple-and-add algorithm, we now have to consider adding some multiple $[w]R$ of R ($w < 2^n$) to the intermediate point and updating the Miller function accordingly. Suppose the intermediate point is $T = [v]R$ and the related Miller function prior to the addition has divisor $\mathrm{div}(f_{v,R}) = v(R) - ([v]R) - (v-1)(\mathcal{O})$ as before. We require a function f^+ such that $\mathrm{div}(f_{v,R}) + \mathrm{div}(f^+) = \mathrm{div}(f_{(v+w),R}) = (v+w)(R) - ([v+w]R) - (v+w-1)(\mathcal{O})$. The straightforward way to construct such a function is

$$f^+ = \prod_{i=0}^{w-1} g_{T+[i]R,R}, \tag{4}$$

the divisor of which is

$$\begin{aligned}\operatorname{div}(f^+) &= \sum_{i=0}^{w-1} \operatorname{div}(g_{T+[i]R,R}) = \sum_{i=0}^{w-1} \big[(R) + (T+[i]R) - (T+[i+1]R) - (\mathcal{O})\big] \\ &= w(R) + (T) - (T+[w]R) - w(\mathcal{O}).\end{aligned}$$

Again, substituting $T = [v]R$ gives $\operatorname{div}(f^+) = w(R) + ([v]R) - ([v+w]R) - w(\mathcal{O})$, so that $\operatorname{div}(f_{v,R}) + \operatorname{div}(f^+) = \operatorname{div}(f_{(v+w),R})$, and we see that f^+ is clearly the desired function. However, if we compute f^+ in the above fashion, we have to compute the product of w different addition lines, and since w can take any value between 1 and $2^n - 1$, computing the addition step with the explicit formulas that result from the product in (4) can become quite costly. Instead, consider an alternative method of computing the addition line as follows. Let f^+_{alt} be such that $\operatorname{div}(f^+_{\text{alt}}) = \operatorname{div}(f^+)$ and take

$$f^+_{\text{alt}} = f_{w,R} \cdot g_{[v]R,[w]R}, \tag{5}$$

so that $\operatorname{div}(f^+_{\text{alt}}) = \operatorname{div}(f_{w,R}) + \operatorname{div}(g_{[v]R,[w]R}) = w(R) + ([v]R) - ([v+w]R) - w(\mathcal{O})$. The advantage of the computation of f^+_{alt} over the computation of f^+ is that f^+_{alt} is comprised of only two functions, regardless of the size of w. Moreover, the function $f_{w,R}$ is the same function throughout the entire Miller 2^n-tupling loop and does not change depending on where the addition/s occurs. Thus, the $f_{w,R}$'s can be precomputed (for all necessary values of w) prior to entering the Miller 2^n-tupling loop so that we must only construct one new line function ($g_{[v]R,[w]R}$) at each addition stage. Importantly, this addition line is computed by applying the standard addition formulas to the coordinates of the point $[v]R$, which changes in each iteration, and the point $[w]R$ whose coordinates can be cached initially. From here on, the construction of f^+ refers to the construction of f^+_{alt} described in (5). We summarize in Algorithm 2, where we note that the first value in the base 2^n representation of m will not be $m_{l-1} = 1$ in general, so that we begin with an addition before entering the loop when $m_{l-1} \neq 1$.

Algorithm 2 Miller 2^n-tuple-and-add Algorithm

Input: R, S, $m = (m_{l-1}...m_1, m_0)_{2^n}$, and the necessary precomputed values of $w[R]$ where $w < 2^n$.
Output: $f_{m,R}(S)$.

1: $T \leftarrow R$, $f \leftarrow 1$.
2: Compute function f^+ as the product described in (5) with $w = m_{l-1}$.
3: $f \leftarrow f \cdot f^+$.
4: $T \leftarrow T + [m_{l-1}]R$.
5: **for** $i = l-2$ to 0 **do**
6: Compute function f^* in the 2^n-tupling of T.
7: $T \leftarrow [2^n]T$.
8: $f \leftarrow f^{2^n} \cdot f^*$.
9: **if** $m_i \neq 0$ **then**
10: Compute function f^+ as the product described in (5) with $w = m_i$.
11: $T \leftarrow T + [m_i]R$.
12: $f \leftarrow f \cdot f^+$.
13: **end if**
14: **end for**
15: **return** f.

In regards to full extension field arithmetic only, one standard iteration of Algorithm 2 (which usually has $m_i = 0$) requires $1\mathbf{M} + n\mathbf{S}$. When $n = 1$, we recover the usual Miller double-and-add algorithm which requires $\lfloor \log_2(m) \rfloor$ iterations, each incurring $1\mathbf{M} + 1\mathbf{S}$. For $n = 2$, the algorithm requires half as many iterations ($\lfloor \log_4(m) \rfloor$) that each incur a cost of $1\mathbf{M} + 2\mathbf{S}$, offering a $1\mathbf{M}$ saving over two equivalent standard double-and-add iterations. For general n, we save $(n-1)\mathbf{M}$ for each of the $\lfloor \log_{2^n}(m) \rfloor$ iterations of the Miller 2^n-tuple-and-add algorithm, giving a relative saving of $\frac{n-1}{n}\mathbf{M}$ over each equivalent standard double-and-add iteration. Therefore the larger we allow n to become, the more full extension field arithmetic we can avoid in the pairing computation.

The price we pay for increasing n is an increase in the complexity of the formulas required to compute the function f^*. As n grows, the size of f^* (in its explicit form) grows rapidly so that many more operations are required to compute it. However, these operations are performed in substantially smaller subfields of the full extension field, where the computations are much cheaper. We can achieve significant speedups in the pairing computation if the price we pay for computing the more complex product of line functions f^* in the smaller subfields of $\mathbb{F}_{p^k}$ is less than the savings we obtain in $\mathbb{F}_{p^k}$ itself.

In the following section we shed light on the details concerning the combination of steps 6 and 7 and the combination of steps 10 and 11 that are summarized in Algorithm 2.

4 A Strategy for Obtaining Explicit Formulas

This section provides the details for deriving explicit formulas for Miller 2^n-tuple-and-add implementations. We pay close attention to the steps in Algorithm 2 that require deeper explanations.

Line 6 of Algorithm 2: Algorithm 3 (below) uses the standard doubling formulas to construct the affine line product f^* for Miller 2^n-tupling in accordance with (2).

Algorithm 3 Constructing explicit formulas for f^*

Input: $R = (x_1, y_1)$ and $S = (x_S, y_S)$.
Output: f^*.

1: $(x, y) \leftarrow (x_1, y_1)$, $f^* \leftarrow 1$.
2: **for** $i = 1$ to n **do**
3: $\quad \lambda \leftarrow (3x^2 + a)/(2y)$.
4: $\quad x' \leftarrow \lambda^2 - 2x$.
5: $\quad y' \leftarrow \lambda(x - x') - y$.
6: $\quad g \leftarrow \lambda(x - x_S) + y_S - y$.
7: $\quad f^* \leftarrow f^* \cdot g^{2^{n-i}}$.
8: $\quad (x, y) \leftarrow (x', y')$.
9: **end for**
10: **return** f^*.

We note that Algorithm 3 computes the product g under the assumption of an even embedding degree, so that the denominator v_i of the i-th product update $g_i = l_i/v_i$ can be eliminated and the g_i's simply become the l_i's described at the beginning of Section 2. In the following sections we use different projections on the affine form of f^* depending on the curve model.

Line 7 of Algorithm 2: Depending on the formulas derived for f^*, there are two possibilities that need to be considered for computing the point multiplication $[2^n]T$. The first option would be to output the explicit formulas for x' and y' in Algorithm 3. These compounded formulas would obviously be much more complicated than the standard point doubling formulas (i.e. computing $[2]T$), however the more complicated explicit formulas for computing $[2^n]T = (x', y')$ may end up sharing many common subexpressions with the explicit formula for f^* so that the overall count would be less. The second option simply involves repeating n consecutive doublings on the point T. The heuristic argument would suggest that optimized formulas for computing $[2^n]T$ directly should require no more operations than those required in the repetitive doublings, suggesting that the first option should always take preference. However, our experiments indicated that attempts to optimize 2^n-tupling formulas always tend to reduce to the same formulas that arise from n repeated doublings. For the sake of simplicity, we therefore opt for the latter suggestion and perform n repetitive doublings to compute $[2^n]T$. Furthermore, it also tends to be the case that the higher degree subexpressions obtained in the explicit formulas for computing $[2^n]T$ directly do not appear in the simplified expressions for f^*. However, many operations used in the very first doubling of T also appear readily in the components of f^* and we make use of these common subexpressions. Namely, the doubling formulas used to compute $[2]T$ are chosen so that the simultaneous computation of f^* and $[2]T$ comes at minimal cost. Therefore, it is often the case that the formulas used to compute $[2]T$ may not be the same formulas as those used to compute the $n-1$ doublings that follow.

Lines 10 and 11 of Algorithm 2: In the addition stage of Miller 2^n-tuple-and-add, we are required to add some multiple $w[R]$ of R ($w < 2^n$) to the intermediate point T. Here we simply cache the value $[w]R$ before the iterations start and perform a standard point addition. The Miller function update f^+ required in line 7 of Algorithm 2 requires the computation of the product $f^+ = f_{w,R}(S) \cdot g_{T,[w]R}(S)$. By definition, $g_{T,[w]R}(S)$ is the line function corresponding to the addition of T to $[w]R$, evaluated at the point S. Therefore, the combination of lines 11 and 12 of Algorithm 2 can simply be viewed as a standard point addition between T and $[w]R$, as well as the extra multiplication of $g_{T,[w]R}(S)$ by the cached value $f_{w,R}(S)$.

5 Miller Quadrupling and Octupling

In this section we focus on applying the generalized algorithm in Section 3 to the cases $n = 2$ and $n = 3$. We present reduced explicit formulas that arise for the Miller quadruple-and-add and Miller octuple-and-add algorithms on curves of the form $E : y^2 = x^3 + b$ ($j(E) = 0$) and $E : y^2 = x^3 + ax$ ($j(E) = 1728$), since these are the most efficient curve shapes used in practice [22]. We focus solely on the 2^n-tupling stage of the algorithm (i.e. steps 6 and 7 in Algorithm 2), since optimized loop parameters will result in very few additions. We therefore delay any discussion of the additions until the following section.

5.1 Miller Quadruple-and-add

We begin by setting $n = 2$ in (3) to obtain the Miller update f^* corresponding to the quadrupling of T as

$$f^* = \prod_{i=1}^{2} (g_{[2^{i-1}]T,[2^{i-1}]T})^{2^{2-i}} = \big(g_{T,T}\big)^2 \cdot \big(g_{[2]T,[2]T}\big),$$

which has divisor $4(T) - ([4]T) - 3(\mathcal{O})$.

Quadruple-and-add on $y^2 = x^3 + b$. We obtain f^* as the affine output of Algorithm 3 with $n = 2$. For curves of this form, the fastest explicit formulas for the $n = 1$ case were derived using homogeneous projective coordinates [16, 17]. Our experiments[1] indicated that these coordinates also give the fastest results for $n \geq 1$, so we substitute $x_1 = X_1/Z_1$ and $y_1 = Y_1/Z_1$ into f^* to obtain the projectified version, F^*, as

$$F^* = \alpha \cdot (L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),$$

where $\alpha = -Z_1^3(X_1(X_1^3 - 8bZ_1^3) - 4Z_1(X_1^3 + bZ_1^3) \cdot x_S)^2/(64Z_1^7Y_1^5(27X_1^6 - 36X_1^3Y_1^2Z_1 + 8Y_1^4Z_1^2))$ can be eliminated to give $\hat{F}^* = F^*/\alpha$, where the $L_{i,j}$ coefficients are

$$L_{2,0} = -6X_1^2Z_1(5Y_1^4 + 54bY_1^2Z_1^2 - 27b^2Z_1^4), \quad L_{0,1} = 8X_1Y_1Z_1(5Y_1^4 + 27b^2Z_1^4),$$
$$L_{1,1} = 8Y_1Z_1^2(Y_1^4 + 18bY_1^2Z_1^2 - 27b^2Z_1^4), \quad L_{0,0} = 2X_1(Y_1^6 - 75bY_1^4Z_1^2 + 27b^2Y_1^2Z_1^4 - 81b^3Z_1^6).$$
$$L_{1,0} = -4Z_1(5Y_1^6 - 75bZ_1^2Y_1^4 + 135Y_1^2b^2Z_1^4 - 81b^3Z_1^6).$$

We let $(X_{D^n} : Y_{D^n} : Z_{D^n}) = [2^n](X_1 : Y_1 : Z_1)$ and compute the first doubling with small extra computation as

$$X_{D^1} = 4X_1Y_1(Y_1^2 - 9bZ_1^2), \quad Y_{D^1} = 2Y_1^4 + 36bY_1^2Z_1^2 - 54b^2Z_1^4, \quad Z_{D^1} = 16Y_1^3Z_1$$

The calculation of the $L_{i,j}$ coefficients and the intermediate point $(X_{D^1} : Y_{D^1} : Z_{D^1}) = [2](X_1, Y_1, Z_1)$ requires $11\mathbf{m}_e + 11\mathbf{s}_e + 3\mathbf{d}$. To calculate $(X_{D^2} : Y_{D^2} : Z_{D^2}) = [4](X_1, Y_1, Z_1)$, we double the point $(X_{D^1} : Y_{D^1} : Z_{D^1})$ using the doubling formulas in [17] which cost $3\mathbf{m}_e + 5\mathbf{s}_e + 1\mathbf{d}$. The multiplication of each of the four $L_{i,j} \neq L_{0,0}$ by $x_S^i y_S^j$ costs $e\mathbf{m}_1$ (cf. [17]). As discussed in Section 3, the extension field arithmetic required in line 8 of Algorithm 2 costs $1\mathbf{M} + 2\mathbf{S}$. Thus, the total cost for the quadrupling stage is $14\mathbf{m}_e + 16\mathbf{s}_e + 4e\mathbf{m}_1 + 4\mathbf{d} + 1\mathbf{M} + 2\mathbf{S}$ (see Appendix A.1 for the sequence of operations, and see Appendix B for a Magma script that computes the Miller quadruple-and-add algorithm using the formulas in A.1).

Quadruple-and-add on $y^2 = x^3 + ax$. For curves of this shape, the fastest formulas for the standard double-and-add case were derived in weight-$(1, 2)$ coordinates in [17]. Again, our experiments agree with these coordinates for such curves for $n \geq 1$, so we subsitute $x_1 = X_1/Z_1$ and $y_1 = Y_1/Z_1^2$ into f^* (the output of Algorithm 3) to obtain F^* as

$$F^* = \alpha \cdot (L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),$$

where $\alpha = -Z_1^6(-4X_1Z_1(X_1^2 + aZ_1^2)x_S + (X_1^2 - aZ_1^2)^2)^2$ can be eliminated to give $\hat{F}^* = F^*/\alpha$, where the $L_{i,j}$ coefficients are

$$\begin{aligned}
L_{1,0} &= -2X_1Z_1(5X_1^8 + 4aX_1^6Z_1^2 + 38a^2X_1^4Z_1^4 + 20a^3X_1^2Z_1^6 - 3a^4Z_1^8),\\
L_{2,0} &= -Z_1^2(15X_1^8 + 68aX_1^6Z_1^2 + 10a^2X_1^4Z_1^4 - 28a^3X_1^2Z_1^6 - a^4Z_1^8),\\
L_{0,1} &= 4Y_1X_1Z_1(5X_1^6 + 13aX_1^4Z_1^2 + 15a^2X_1^2Z_1^4 - a^3Z_1^6),\\
L_{1,1} &= 4Y_1Z_1^2(X_1^2 - aZ_1^2)(X_1^4 + 6aX_1^2Z_1^2 + a^2Z_1^4),\\
L_{0,0} &= X_1^2(X_1^8 - 20aX_1^6Z_1^2 - 26a^2X_1^4Z_1^4 - 20a^3X_1^2Z_1^6 + a^4Z_1^8).
\end{aligned}$$

[1] We searched through a range of different coordinate systems (cf. [11]) to find the coordinate system which gave the most simple projectified line coefficients.

Again, we compute the first doubling with small extra computation as

$$X_{D^1} = (X_1^2 - aZ_1^2)^2, \quad Y_{D^1} = 2Y_1(X_1^2 - aZ_1^2)(X_1^4 + 6X_1^2aZ_1^2 + a^2Z_1^4), \quad Z_{D^1} = 4Y_1^2.$$

The calculation of the $L_{i,j}$ coefficients and the intermediate point $(X_{D^1} : Y_{D^1} : Z_{D^1}) = [2](X_1, Y_1, Z_1)$ requires $10\mathbf{m} + 14\mathbf{s} + 2\mathbf{d}$. To calculate $(X_{D^2} : Y_{D^2} : Z_{D^2}) = [4](X_1, Y_1, Z_1)$, we double the point $(X_{D^1} : Y_{D^1} : Z_{D^1})$ using the doubling formulas in [17] which cost $1\mathbf{m}+6\mathbf{s}+1\mathbf{d}$. Thus, the total cost for the quadrupling stage is $11\mathbf{m}_e + 20\mathbf{s}_e + 4e\mathbf{m}_1 + 3\mathbf{d} + 1\mathbf{M} + 2\mathbf{S}$ (see Appendix A.2).

5.2 Miller Octuple-and-add

We begin by setting $n = 3$ in (3) to obtain the Miller update f^* corresponding the octupling of T as

$$f^* = \prod_{i=1}^{3} (g_{[2^{i-1}]T,[2^{i-1}]T})^{2^{3-i}} = \big(g_{T,T}\big)^4 \cdot \big(g_{[2]T,[2]T}\big)^2 \cdot \big(g_{[4]T,[4]T}\big),$$

which has divisor $8(T) - ([8]T) - 7(\mathcal{O})$.

Octuple-and-add on $y^2 = x^3 + b$. For the octupling line product, we use homogeneous projective coordinates to give F^* as

$$\begin{aligned} F^* = \alpha \cdot (&L_{4,0} \cdot x_S^4 + L_{3,0} \cdot x_S^3 + L_{2,0} \cdot x_S^2 + L_{1,0} \cdot x_S \\ &+ L_{3,1} \cdot x_S^3 y_S + L_{2,1} \cdot x_S^2 y_S + L_{1,1} \cdot x_S y_S + L_{0,0}), \end{aligned}$$

where α is again contained in a proper subfield of $\mathbb{F}_{p^k}$ and can be eliminated to give $\hat{F}^* = F^*/\alpha$. The $L_{i,j}$ coefficients are

$$\begin{aligned} &L_{4,0} = (-9X_1^2Z_1^2) \cdot \mathcal{S}_{4,0}, \quad L_{3,0} = (-12Z_1^2Y_1^2) \cdot \mathcal{S}_{3,0}, \quad L_{2,0} = (-54X_1Y_1^2Z_1) \cdot \mathcal{S}_{2,0} \\ &L_{1,0} = (-36X_1^2Y_1^2) \cdot \mathcal{S}_{1,0}, \quad L_{0,0} = ((Y_1^2 + 3bZ_1^2)Y_1^2) \cdot \mathcal{S}_{0,0}, \quad L_{3,1} = (8Y_1Z_1^3) \cdot \mathcal{S}_{3,1} \\ &L_{2,1} = (216X_1Y_1Z_1^2) \cdot \mathcal{S}_{2,1}, \quad L_{1,1} = (72X_1^2Y_1Z_1) \cdot \mathcal{S}_{1,1}, \quad L_{0,1} = (8Y_1^3Z_1) \cdot \mathcal{S}_{0,1} \end{aligned}$$

with

$$\mathcal{S}_{i,j} = \sum_{k=0}^{11} c_{i,j,k} \cdot (Y_1^2)^{11-k}(bZ_1^2)^k,$$

where $c_{i,j,k}$ is the coefficient of $(Y_1^2)^{11-k}(bZ_1^2)^k$ belonging to $L_{i,j}$ (see Appendix A.3). As an example, we have

$$\begin{aligned} L_{0,0} = (Y_1^2(Y_1^2 + 3bZ_1^2)) \cdot \big(&Y_1^{22} - 3375bY_1^{20}Z_1^2 - 262449b^2Y_1^{18}Z_1^4 - 2583657b^3Y_1^{16}Z_1^6 \\ &+ 47678058b^4Y_1^{14}Z_1^8 - 40968342b^5Y_1^{12}Z_1^{10} - 272740770b^6Y_1^{10}Z_1^{12} \\ &+ 738702990b^7Y_1^8Z_1^{14} - 669084219b^8Y_1^6Z_1^{16} + 206730549b^9Y_1^4Z_1^{18} \\ &- 23914845b^{10}Y_1^2Z_1^{20} + 14348907b^{11}Z_1^{22}\big). \end{aligned}$$

We describe a general method to compute each of the terms of the form $(Y_1^2)^{11-k}(bZ_1^2)^k$ that are required to compute the $L_{i,j}$ coefficients, where $0 \leq k \leq 11$. In general, it is best

to compute each one of these products rather than attempting to factorize, particular when each of these terms is present in every $L_{i,j}$. We compute every required even power of Y_1 by first repetitively squaring Y_1 until we have all necessary terms of the form $Y_1^{2^t}$ that are less than the largest power of Y_1 occuring in the summations of the $L_{i,j}$. That is, we compute $Y_1^{2^t}$ for $t = 1, 2, 3, 4$ since Y_1^{22} is the largest power of Y_1 occuring in the $L_{i,j}$ summations. Using $\{Y_1^2, Y_1^4, Y_1^8, Y_1^{16}\}$, we can compute all other $(Y_1^2)^z < (Y_1^2)^{16}$, $z \neq 2^t$ using one squaring each for each z. For example, we can compute Y_1^{12} as $Y_1^{12} = Y_1^8 \cdot Y_1^4 = ((Y_1^8 + Y_1^4)^2 - Y_1^{16} - Y_1^8)/2$, although in practice we compute $2Y_1^{12}$ to avoid the division by 2. To compute the remaining $(Y_1^2)^t > Y_1^{16}$, we use a field multiplication[2]. We do the same for each of the $(bZ_1^2)^k$ terms.

We do not count multiplications by the $c_{i,j,k}$, although we make no attempt to disguise the extra cost that is incurred as their sizes grow. We do however, point out that it is often the case that the $c_{i,j,k}$'s for a fixed k (but different i,j's) share large common factors so that we need not multiply $(Y_1^2)^{11-k}(bZ_1^2)^k$ by each of the $c_{i,j,k}$'s, but rather we combine previous products to obtain most of these multiplications at a much smaller (mostly negligible) cost.

The total operation count for the point octupling and the computation of the octupling line product is $40\mathbf{m}_e + 31\mathbf{s}_e + 8e\mathbf{m}_1 + 2\mathbf{d} + 1\mathbf{M} + 3\mathbf{S}$ (see Appendix A.3).

Octuple-and-add on $y^2 = x^3 + ax$. Following the trend of the fastest formulas for the $n = 1$ and $n = 2$ cases for curves of this shape, we again projectify f^* using weight-$(1, 2)$ coordinates to give

$$\begin{aligned} F^* = \alpha \cdot (&L_{4,0} \cdot x_S^4 + L_{3,0} \cdot x_S^3 + L_{2,0} \cdot x_S^2 + L_{1,0} \cdot x_S \\ &+ L_{3,1} \cdot x_S^3 y_S + L_{2,1} \cdot x_S^2 y_S + L_{1,1} \cdot x_S y_S + L_{0,0}), \end{aligned}$$

where we ignore the subfield cofactor α to give $\hat{F}^* = F^*/\alpha$. The $L_{i,j}$ coefficients are given as

$$\begin{aligned} &L_{4,0} = (-4X_1^2 Z_1^4) \cdot \mathcal{S}_{4,0}, \quad L_{3,0} = (-16X_1^3 Z_1^3) \cdot \mathcal{S}_{3,0}, \quad L_{2,0} = (-8X_1^4 Z_1^2) \cdot \mathcal{S}_{2,0} \\ &L_{1,0} = (16X_1^5 Z_1) \cdot \mathcal{S}_{1,0}, \quad L_{0,0} = (4X_1^6) \cdot \mathcal{S}_{0,0}, \quad L_{3,1} = (4Y_1 Z_1^4) \cdot \mathcal{S}_{3,1} \\ &L_{2,1} = (4X_1 Y_1 Z_1^3) \cdot \mathcal{S}_{2,1}, \quad L_{1,1} = (4X_1^2 Y_1 Z_1^2) \cdot \mathcal{S}_{1,1}, \quad L_{0,1} = (4X_1^3 Y_1 Z_1) \cdot \mathcal{S}_{0,1}, \end{aligned}$$

with

$$\mathcal{S}_{i,j} = \sum_{k=0}^{16} c_{i,j,k} \cdot (X_1^2)^{16-k} (bZ_1^2)^k,$$

where $c_{i,j,k}$ is the coefficient of $(X_1^2)^{16-k}(bZ_1^2)^k$ belonging to $L_{i,j}$ (see Appendix A.4). As an example, we have

$$\begin{aligned} L_{2,0} = -8X1^4 Z1^2 \cdot (&189X_1^{32} + 882bX_1^{30}Z_1^2 + 6174b^2X_1^{28}Z_1^4 - 26274b^3X_1^{26}Z_1^6 - 1052730b^4X_1^{24}Z_1^8 \\ &- 449598b^5X_1^{22}Z_1^{10} - 1280286b^6X_1^{20}Z_1^{12} - 1838850b^7X_1^{18}Z_1^{14} - 23063794b^8X_1^{16}Z_1^{16} \\ &- 1543290b^9X_1^{14}Z_1^{18} + 539634b^{10}X_1^{12}Z_1^{20} + 646922b^{11}X_1^{10}Z_1^{22} + 1386918b^{12}X_1^8Z_1^{24} \\ &+ 75846b^{13}X_1^6Z_1^{26} + 17262b^{14}X_1^4Z_1^{28} + 922b^{15}X_1^2Z_1^{30} - 35b^{16}Z_1^{32}). \end{aligned}$$

The total operation count for the point octupling and the computation of the octupling line product is $31\mathbf{m}_e + 57\mathbf{s}_e + 8e\mathbf{m}_1 + 5\mathbf{d} + 1\mathbf{M} + 3\mathbf{S}$ (see Appendix A.4).

[2] We point out that if higher degree terms also required computation it may be advantageous to compute Y_1^{32} so that each of the terms $(Y_1^2)^t > Y_1^{16}$ can be computed using field squarings instead of multiplications. This advantage would depend on the platform (the $\mathbf{s}$:$\mathbf{m}$ ratio) and the number of $(Y_1^2)^t > Y_1^{16}$ terms required.

6 Comparisons

We draw comparisons between 6 standard loops of Miller double-and-add, 3 standard loops of Miller quadruple-and-add and 2 standard loops of Miller octuple-and-add, since each of these equates to one 64-tuple-and-add loop, and this is the most primitive level at which a fair comparison can be made. We note that the estimated percentage speedups in Table 1 are for the computation of the Miller loop only and do not take into account the significant fixed cost of final exponentiation. We neglect additions since low hamming-weight loop parameters used in pairing implementations will result in a similar amount of additions regardless of n, and we saw in sections 3 and 4 that additions come at approximately the same cost for different n. The counts for $n = 1$ are due to the fastest formulas given for curves with $j(E) = 0$ and $j(E) = 1728$ in [17]. We multiply these counts and those obtained for $n = 2$ and $n = 3$ in Section 5 accordingly.

$j(E)$	Doubling: $n = 1$ (6 loops)	Quadrupling: $n = 2$ (3 loops)	Octupling: $n = 3$ (2 loops)
0	$12\mathbf{m}_e + 42\mathbf{s}_e + 12e\mathbf{m}_1 + 6\mathbf{M} + 6\mathbf{S}$	$42\mathbf{m}_e + 48\mathbf{s}_e + 12e\mathbf{m}_1 + 3\mathbf{M} + 6\mathbf{S}$	$80\mathbf{m}_e + 64\mathbf{s}_e + 16e\mathbf{m}_1 + 2\mathbf{M} + 6\mathbf{S}$
1728	$12\mathbf{m}_e + 48\mathbf{s}_e + 12e\mathbf{m}_1 + 6\mathbf{M} + 6\mathbf{S}$	$33\mathbf{m}_e + 60\mathbf{s}_e + 12e\mathbf{m}_1 + 3\mathbf{M} + 6\mathbf{S}$	$64\mathbf{m}_e + 114\mathbf{s}_e + 16e\mathbf{m}_1 + 2\mathbf{M} + 6\mathbf{S}$

Table 1. Operation counts for the equivalent number of iterations of 2^n-tuple and add for $n = 1, 2, 3$.

Table 1 shows that the number of subfield operations increases when n gets larger, whilst the number of full extension field multiplications decreases. To determine whether these trade-offs become favorable for $n = 2$ or $n = 3$, we adopt the standard procedure of estimating the equivalent number of base field operations for each operation count [27, 17]. We assume that the higher degree fields are constructed as a tower of extensions, so that for pairing-friendly fields of extension degree $z = 2^i \cdot 3^j$, we can assume that $\mathbf{m}_z = 3^i \cdot 5^j \mathbf{m}_1$ [31]. We split the comparison between pairings on $\mathbb{G}_1 \times \mathbb{G}_2$ (the Tate pairing, the twisted ate pairing) and pairings on $\mathbb{G}_2 \times \mathbb{G}_1$ (the ate pairing, R-ate pairing, etc). For each pairing-friendly embedding degree reported, we assume that the highest degree twist is utilized in both settings; the curves with $j(E) = 0$ utilize degree 6 twists whilst the curves with $j(E) = 1728$ utilize degree 4 twists. To compare across operations, we follow the EFD [11] and present two counts in each scenario: the top count assumes that $\mathbf{s}_z = \mathbf{m}_z$, whilst the bottom count assumes that $\mathbf{s}_z = 0.8\mathbf{m}_z$. When quadrupling or octupling gives a faster operation count, we provide an approximate percentage speedup for the computation of the Miller loop, ignoring any additions that occur.

Unsurprisingly, Table 2 illustrates that the relative speed up for pairings on $\mathbb{G}_1 \times \mathbb{G}_2$ grows as the embedding degree grows. This is due to the increasing gap between the complexity of operations in $\mathbb{G}_1$ (which is defined over $\mathbb{F}_q$) and $\mathbb{G}_2$ (which is defined over $\mathbb{F}_{q^k}$). In this case we see that $6 \leq k \leq 16$ favor Miller quadruple-and-add, whilst Miller octuple-and-add takes over for $k > 16$, where it is clear that it is worthwhile spending many more operations in the base field in order to avoid costly arithmetic in $\mathbb{F}_{q^k}$. For pairings on $\mathbb{G}_2 \times \mathbb{G}_1$, we have a consistent speed up across all embedding degrees that utilize sextic twists. This is due to the complexity of the subfield operations in $\mathbb{F}_{q^e}$ growing at the same rate as the complexity of operations in $\mathbb{F}_{q^k}$. Table 2 indicates that Miller double-and-add is still preferred for ate-like pairings using

		Pairings on $\mathbb{G}_1 \times \mathbb{G}_2$ (Tate, twisted ate)			Fastest (% speed up)	Pairings on $\mathbb{G}_2 \times \mathbb{G}_1$ (ate, R-ate)			Fastest (% speed up)
k	$j(E)$	Doubling (6 loops)	Quadrupling (3 loops)	Octupling (2 loops)		Doubling (6 loops)	Quadrupling (3 loops)	Octupling (2 loops)	
4	1728	180 159.6	186 163.2	266 232.4	Doubling -	180 159.6	186 163.2	266 232.4	Doubling -
6	0	246 219.6	237 209.4	280 249.2	Quadrupling 5%	246 219.6	237 209.4	280 249.2	Quadrupling 5%
8	1728	408 366	360 315.6	426 370.8	Quadrupling 14%	528 466.8	546 477.6	782 681.2	Doubling -
12	0	618 555.6	519 455.4	536 469.2	Quadrupling 18%	726 646.8	699 616.2	824 731.6	Quadrupling 5%
16	1728	1080 973.2	870 760.8	890 770	Quadrupling 22%	1560 1376.4	1614 1408.8	2314 2011.6	Doubling -
18	0	990 891.6	801 701.4	792 689.2	Octupling 22%	1206 1074	1161 1023	1368 1214	Quadrupling 5%
24	0	1722 1551.6	1353 1181.4	1288 1113.2	Octupling 28%	2154 1916.4	2073 1824.6	2440 2162.8	Quadrupling 5%
32	1728	3072 2770.8	2376 2072.4	2250 1935.6	Octupling 30%	4632 4081.2	4794 4178.4	6878 5970.8	Doubling -
36	0	2826 2547.6	2187 1907.4	2040 1757.6	Octupling 31%	3582 3186	3447 3033	4056 3594	Quadrupling 5%
48	0	5010 4515.6	3831 3335.4	3512 3013.2	Octupling 33%	6414 5701.2	6171 5425.8	7256 6424.4	Quadrupling 5%

Table 2. Comparisons for Miller double-and-add, Miller quadruple-and-add and Miller octuple-and-add at various embedding degrees.

quartic twists, where we could conclude that the gap between operations in $\mathbb{F}_{q^{k/4}}$ and those in $\mathbb{F}_{q^k}$ isn't large enough to favor higher Miller tupling.

The large improvements in Table 2 certainly present a case for the investigation of higher degree Miller tupling ($n \geq 4$). At these levels however, the formulas become quite complex and we have not reported any discoveries from these degrees due to space considerations. Namely, the size of the 2^n-tupling line in (2) grows exponentially as n increases (i.e. the degree of the affine 2^n-tupling line formula is twice that of the 2^{n-1}-tupling line). The fact that quadrupling was still preferred over octupling in most cases seems to suggest that larger n might not result in significant savings, at least for embedding degrees of this size.

We conclude by acknowledging that (in optimal implementations) the speedups in Table 2 may not be as large as we have claimed. In generating the comparisons, we reported the multiplication of the intermediate Miller value f by the Miller update g as a full extension field multiplication in $\mathbb{F}_{p^k}$, with complexity $\mathbf{M} = \mathbf{m}_k = 3^i \cdot 5^j$ for $k = 2^i \cdot 3^j$. Although the value f is a general full extension field element, g tends to be sparse, especially when sextic twists are employed. For even degree twists, g takes the form $g = g_1\alpha + g_2\beta + g_0$, where $g \in F_{p^k}$, $g_0, g_1, g_2 \in \mathbb{F}_{p^{k/d}}$ and α and β are algebraic elements that do not affect multiplication costs (cf. [17]). For sextic twists, a general element of $\mathbb{F}_{p^k}$ would be written as a polynomial over $\mathbb{F}_{p^e}$ with six (rather than three) different coefficients belonging to $\mathbb{F}_{p^{k/6}}$. In this case, multiplying two general elements of $\mathbb{F}_{p^k}$ would clearly require more multiplications than performing a multiplication between a general element (like f) and a sparse element (like g). Since the techniques in this paper gain advantage by avoiding multiplications between f and g, reporting a lesser complexity for this multiplication would decrease the relative

speedup. Nevertheless, Miller quadruple-and-add and Miller octuple-and-add will still strongly outperform the standard Miller double-and-add routine if we take $\mathbf{m}_k \ll 3^i \cdot 5^j$, particularly for pairings on $\mathbb{G}_1 \times \mathbb{G}_2$ with large embedding degrees.

7 Acknowledgements

The authors wish to thank Huseyin Hisil, Douglas Stebila, Nicole Verna, and the anonymous referees for helpful comments and suggestions on earlier versions of this paper.

References

1. Christophe Arene, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. Faster pairing computation. Cryptology ePrint Archive, Report 2009/155, 2009. http://eprint.iacr.org/.
2. Roberto M. Avanzi, Henri Cohen, Christophe Doche, Gerhard Frey, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *The Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC, 2005.
3. Paulo S. L. M. Barreto, Steven D. Galbraith, Colm O'Eigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007.
4. Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.
5. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.
6. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 2003.
7. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Efficient implementation of pairing-based cryptosystems. *J. Cryptology*, 17(4):321–334, 2004.
8. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.
9. Naomi Benger and Michael Scott. Constructing tower extensions for the implementation of pairing-based cryptography. Cryptology ePrint Archive, Report 2009/556, 2009. http://eprint.iacr.org/.
10. Waldyr D. Benits Junior and Steven D. Galbraith. Constructing pairing-friendly elliptic curves using Gröbner basis reduction. In Galbraith [23], pages 336–345.
11. Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. http://www.hyperelliptic.org/EFD.
12. Ian F. Blake, V. Kumar Murty, and Guangwu Xu. Refinements of miller's algorithm for computing the weil/tate pairing. *J. Algorithms*, 58(2):134–149, 2006.
13. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
14. Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Des. Codes Cryptography*, 37(1):133–141, 2005.
15. Sanjit Chatterjee, Palash Sarkar, and Rana Barua. Efficient computation of Tate pairing in projective coordinate over general characteristic fields. In Choonsik Park and Seongtaek Chee, editors, *ICISC*, volume 3506 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2004.
16. Craig Costello, Huseyin Hisil, Colin Boyd, Juan Manuel González Nieto, and Kenneth Koon-Ho Wong. Faster pairings on special Weierstrass curves. In Hovav Shacham and Brent Waters, editors, *Pairing*, volume 5671 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2009.
17. Craig Costello, Tanja Lange, and Michael Naehrig. Faster pairing computations on curves with high-degree twists. In *PKC 2010*, Lecture Notes in Computer Science. Springer, 2010. To appear.
18. M. Prem Laxman Das and Palash Sarkar. Pairing computation on twisted Edwards form elliptic curves. In Galbraith and Paterson [25], pages 192–210.
19. Iwan M. Duursma and Hyang-Sook Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2003.

20. David Freeman. Constructing pairing-friendly elliptic curves with embedding degree 10. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 452–465. Springer, 2006.
21. David Freeman. A generalized Brezing-Weng algorithm for constructing pairing-friendly ordinary abelian varieties. In Galbraith and Paterson [25], pages 146–163.
22. David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *J. Cryptology*, 23(2):224–280, 2010.
23. Steven D. Galbraith, editor. *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*. Springer, 2007.
24. Steven D. Galbraith, James F. McKee, and Paula C. Valença. Ordinary abelian varieties having small embedding degree. *Finite Fields and their Applications*, 13:800–814, 2007.
25. Steven D. Galbraith and Kenneth G. Paterson, editors. *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, volume 5209 of *Lecture Notes in Computer Science*. Springer, 2008.
26. Florian Hess. Pairing lattices. In Galbraith and Paterson [25], pages 18–38.
27. Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.
28. Sorina Ionica and Antoine Joux. Another approach to pairing computation in Edwards coordinates. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 400–413. Springer, 2008. http://eprint.iacr.org/2008/292.
29. Marc Joye, editor. *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*. Springer, 2003.
30. Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In Galbraith and Paterson [25], pages 126–135.
31. Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.
32. Eunjeong Lee, Hyang-Sook Lee, and Cheol-Min Park. Efficient and generalized pairing computation on abelian varieties. *IEEE Transactions on Information Theory*, 55(4):1793–1803, 2009.
33. Seiichi Matsuda, Naoki Kanayama, Florian Hess, and Eiji Okamoto. Optimised versions of the ate and twisted ate pairings. In Galbraith [23], pages 302–312.
34. Victor S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17:235–261, 2004.
35. Michael Scott. Computing the Tate pairing. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2005.
36. Michael Scott and Paulo S. L. M. Barreto. Generating more MNT elliptic curves. *Des. Codes Cryptography*, 38(2):209–217, 2006.
37. Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

A Explicit Formulas

In each of the following four scenarios, we provide the sequence of operations required to compute the first point doubling and the 2^n-tupling line function, followed by the additional formulae required to compute the subsequent point doublings.

A.1 Quadrupling formulas for $y^2 = x^3 + b$

$A = Y_1^2$, $B = Z_1^2$, $C = A^2$, $D = B^2$, $E = (Y_1 + Z_1)^2 - A - B$, $F = E^2$, $G = X_1^2$, $H = (X_1 + Y_1)^2 - A - G$,
$I = (X_1 + E)^2 - F - G$, $J = (A + E)^2 - C - F$, $K = (Y_1 + B)^2 - A - D$, $L = 27b^2D$, $M = 9bF$, $N = A \cdot C$, $R = A \cdot L$,
$S = bB$, $T = S \cdot L$, $U = S \cdot C$, $X_{D^1} = 2H \cdot (A - 9S)$, $Y_{D^1} = 2C + M - 2L$, $Z_{D^1} = 4J$,
$L_{1,0} = -4Z_1 \cdot (5N + 5R - 3T - 75U)$, $L_{2,0} = -3G \cdot Z_1 \cdot (10C + 3M - 2L)$, $L_{0,1} = 2I \cdot (5C + L)$, $L_{1,1} = 2K \cdot Y_{D^1}$,
$L_{0,0} = 2X_1 \cdot (N + R - 3T - 75U)$. $F^* = L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}, A_2 = Y_{D^1}^2$, $B_2 = Z_{D^1}^2$, $C_2 = 3bB_2$,
$D_2 = 2X_{D^1} \cdot Y_{D^1}$, $E_2 = (Y_{D^1} + Z_{D^1})^2 - A_2 - B_2$, $F_2 = 3C_2$, $X_{D^2} = D_2 \cdot (A_2 - F_2)$, $Y_{D^2} = (A_2 + F_2)^2 - 12C_2^2$,
$Z_{D^2} = 4A_2 \cdot E_2$.

The above sequence of operations costs $14\mathbf{m}_e + 16\mathbf{s}_e + 4e\mathbf{m}_1$.

A.2 Quadrupling formulas for $y^2 = x^3 + ax$

$A = X_1^2,\quad B = Y_1^2,\quad C = Z_1^2,\quad D = aC,\quad X_{D1} = (A-D)^2,\quad E = 2(A+D)^2 - X_{D1},\quad F = ((A-D+Y_1)^2 - B - X_{D1}),$
$Y_{D1} = E \cdot F,\quad Z_{D1} = 4B,\quad G = A^2,\quad H = D^2,\quad I = G^2,\quad J = H^2,\quad K = (X_1+Z_1)^2 - A - C,\quad L = K^2,$
$M = (Y_1+K)^2 - L - B,\quad N = ((G+H)^2 - I - J),\quad R = aL,\quad S = R \cdot G,\quad T = R \cdot H,\quad L_{1,1} = 2C \cdot Y_{D1},$
$L_{0,1} = M \cdot (5A \cdot (G+3H) + D \cdot (13G - H)),\quad L_{2,0} = -C \cdot (15I + 17S + 5N - 7T - J),\quad L_{1,0} = -K \cdot (5I + S + 19N + 5T - 3J),$
$L_{0,0} = A \cdot (I - 5S - 13N - 5T + J).\quad F^* = L_{1,0} \cdot x_S + L_{2,0} \cdot x_S^2 + L_{0,1} \cdot y_S + L_{1,1} \cdot x_S y_S + L_{0,0}, A_2 = X_{D1}^2,\quad B_2 = Y_{D1}^2,$
$C_2 = Z_{D1}^2,\quad D_2 = aC_2,\quad X_{D2} = (A_2 - D_2)^2,\quad E_2 = 2(A_2+D_2)^2 - X_{D2},\quad Z_{D2} = 4B_2,$
$F_2 = ((A_2 - D_2 + Y_{D1})^2 - B_2 - X_{D2},\quad Y_{D2} = E_2 \cdot F_2.$

The above sequence of operations costs $11\mathbf{m}_e + 20\mathbf{s}_e + 4e\mathbf{m}_1$.

A.3 Octupling formulas for $y^2 = x^3 + b$

$Y_{1,2} = Y_1^2,\quad Z_{1,s} = Z_1^2,\quad Z_{1,2} = bZ_{1,s},\quad Z_{1,s^2} = Z_{1,s}^2\quad A = X_1^2,\quad B = b^2 Z_{1,s^2}\quad C = (X_1+Y_1)^2 - A - Y_{1,2},$
$D = (Y_1+Z_1)^2 - Y_{1,2} - Z_{1,s},\quad E = 9Z_{1,2},\quad X_{D1} = C \cdot (Y_{1,2} - E),\quad Y_{D1} = (Y_{1,2}+E)^2 - 108B,\quad Z_{D1} = 4Y_{1,2} \cdot D, Y_{1,4} = Y_{1,2}^2,$
$Y_{1,8} = Y_{1,4}^2,\quad Y_{1,16} = Y_{1,8}^2,\quad Y_{1,6} = (Y_{1,2}+Y_{1,4})^2 - Y_{1,4} - Y_{1,8}, Y_{1,10} = (Y_{1,8}+Y_{1,2})^2 - Y_{1,16} - Y_{1,4},$
$Y_{1,12} = (Y_{1,8}+Y_{1,4})^2 - Y_{1,16} - Y_{1,8},\quad Y_{1,14} = (Y_{1,8}+Y_{1,6})^2 - Y_{1,16} - 2Y_{1,12},\quad Y_{1,18} = Y_{1,16} \cdot Y_{1,2},\quad Y_{1,20} = Y_{1,16} \cdot Y_{1,4},$
$Y_{1,22} = Y_{1,16} \cdot Y_{1,6},\quad Z_{1,4} = B,\quad Z_{1,8} = Z_{1,4}^2,\quad Z_{1,16} = Z_{1,8}^2,\quad Z_{1,6} = (Z_{1,2}+Z_{1,4})^2 - Z_{1,4} - Z_{1,8},$
$Z_{1,10} = (Z_{1,8}+Z_{1,2})^2 - Z_{1,16} - Z_{1,4},\quad Z_{1,12} = (Z_{1,8}+Z_{1,4})^2 - Z_{1,16} - Z_{1,8},\quad Z_{1,14} = (Z_{1,8}+Z_{1,6})^2 - Z_{1,16} - 2Z_{1,12},$
$Z_{1,18} = Z_{1,16} \cdot Z_{1,2},\quad Z_{1,20} = Z_{1,16} \cdot Z_{1,4},\quad Z_{1,22} = Z_{1,16} \cdot Z_{1,6},\quad C_0^{YZ} = Y_{1,22},\quad C_1^{YZ} = Y_{1,20} \cdot Z_{1,2},\quad C_2^{YZ} = Y_{1,18} \cdot Z_{1,4},$
$C_3^{YZ} = Y_{1,16} \cdot Z_{1,6},\quad C_4^{YZ} = Y_{1,14} \cdot Z_{1,8},\quad C_5^{YZ} = Y_{1,12} \cdot Z_{1,10},\quad C_6^{YZ} = Y_{1,10} \cdot Z_{1,12},\quad C_7^{YZ} = Y_{1,8} \cdot Z_{1,14},\quad C_8^{YZ} = Y_{1,6} \cdot Z_{1,16},$
$C_9^{YZ} = Y_{1,4} \cdot Z_{1,18},\quad C_{10}^{YZ} = Y_{1,2} \cdot Z_{1,20},\quad C_{11}^{YZ} = Z_{1,22},\quad F = A \cdot Z_{1,s},\quad G = (Y_{1,2}+Z_{1,s})^2 - Y_{1,4} - Z_{1,s^2},\quad H = C \cdot D,\quad I = C^2$
$J = Y_{1,2} \cdot (Y_{1,2} + 3Z_{1,2}),\quad K = D \cdot Z_{1,s},\quad L = C \cdot Z_{1,s},\quad M = A \cdot D,\quad N = Y_{1,2} \cdot D,$

$$\begin{aligned}
L_{4,0} = -18F \cdot (&-9565938C_{10}^{YZ} + 95659380C_9^{YZ} - 101859525C_8^{YZ} + 14880348C_7^{YZ} + 57100383C_6^{YZ} - 52396146C_5^{YZ} \\
&+ 14332383C_4^{YZ} - 4578120C_3^{YZ} - 513162C_2^{YZ} + 15732C_1^{YZ} + 7C_0^{YZ}),\\
L_{3,0} = -12G \cdot (&-14348907C_{11}^{YZ} + 239148450C_{10}^{YZ} - 643043610C_9^{YZ} + 350928207C_8^{YZ} - 60407127C_7^{YZ} - 8575227C_6^{YZ} \\
&- 7841853C_5^{YZ} + 12011247C_4^{YZ} - 3847095C_3^{YZ} - 1325142C_2^{YZ} + 56238C_1^{YZ} + 35C_0^{YZ}),\\
L_{2,0} = -27H \cdot (&-54206982C_{10}^{YZ} + 157660830C_9^{YZ} - 120282813C_8^{YZ} + 50368797C_7^{YZ} - 25747551C_6^{YZ} + 10693215C_5^{YZ} \\
&- 3826845C_4^{YZ} + 777789C_3^{YZ} + 35682C_2^{YZ} + 4102C_1^{YZ} + 7C_0^{YZ} + 4782969C_{11}^{YZ}),\\
L_{1,0} = -18I \cdot (&-4782969C_{11}^{YZ} + 28697814C_{10}^{YZ} - 129317310C_9^{YZ} + 130203045C_8^{YZ} - 48479229C_7^{YZ} + 11593287C_6^{YZ} \\
&- 619407C_5^{YZ} + 1432485C_4^{YZ} - 883197C_3^{YZ} + 32814C_2^{YZ} - 1318C_1^{YZ} + C_0^{YZ}),\\
L_{0,0} = 2J \cdot (&14348907C_{11}^{YZ} - 47829690C_{10}^{YZ} + 413461098C_9^{YZ} - 669084219C_8^{YZ} + 369351495C_7^{YZ} - 136370385C_6^{YZ} \\
&- 20484171C_5^{YZ} + 23839029C_4^{YZ} - 2583657C_3^{YZ} - 524898C_2^{YZ} - 6750C_1^{YZ} + C_0^{YZ}),\\
L_{3,1} = 8K \cdot (&-28697814C_{10}^{YZ} + 95659380C_9^{YZ} - 61115715C_8^{YZ} + 6377292C_7^{YZ} + 19033461C_6^{YZ} - 14289858C_5^{YZ} \\
&+ 3307473C_4^{YZ} - 915624C_3^{YZ} - 90558C_2^{YZ} + 2484C_1^{YZ} + C_0^{YZ}),\\
L_{2,1} = 216L \cdot (&3188646C_{10}^{YZ} - 7085880C_9^{YZ} + 4546773C_8^{YZ} - 3779136C_7^{YZ} + 5084775C_6^{YZ} - 3601260C_5^{YZ} \\
&+ 1192077C_4^{YZ} - 363744C_3^{YZ} - 56610C_2^{YZ} + 1960C_1^{YZ} + C_0^{YZ}),\\
L_{1,1} = 72M \cdot (&-9565938C_{10}^{YZ} + 10628820C_9^{YZ} - 11160261C_8^{YZ} + 20549052C_7^{YZ} - 24360993C_6^{YZ} + 11674206C_5^{YZ} \\
&- 2214945C_4^{YZ} + 434808C_3^{YZ} - 112266C_2^{YZ} + 8148C_1^{YZ} + 7C_0^{YZ}),\\
L_{0,1} = 8N \cdot (&-14348907C_{11}^{YZ} + 28697814C_{10}^{YZ} - 77590386C_9^{YZ} + 208856313C_8^{YZ} - 152208639C_7^{YZ} + 87333471C_6^{YZ} \\
&- 19135521C_5^{YZ} + 543105C_4^{YZ} - 2329479C_3^{YZ} + 508302C_2^{YZ} - 4138C_1^{YZ} + 21C_0^{YZ}),
\end{aligned}$$

$F^* = \alpha \cdot (L_{4,0} \cdot x_S^4 + L_{3,0} \cdot x_S^3 + L_{2,0} \cdot x_S^2 + L_{1,0} \cdot x_S + L_{3,1} \cdot x_S^3 y_S + L_{2,1} \cdot x_S^2 y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),$
$A_2 = Y_{D1}^2,\quad B_2 = Z_{D1}^2,\quad C_2 = 3bB_2,\quad D_2 = 2X_{D1} \cdot Y_{D1},\quad E_2 = (Y_{D1}+Z_{D1})^2 - A_2 - B_2,\quad F_2 = 3C_2,\quad X_{D2} = D_2 \cdot (A_2 - F_2),$
$Y_{D2} = (A_2+F_2)^2 - 12C_2^2,\quad Z_{D2} = 4A_2 \cdot E_2.$

$A_3 = Y_{D2}^2,\quad B_3 = Z_{D2}^2,\quad C_3 = 3bB_3,\quad D_3 = 2X_{D2} \cdot Y_{D2},\quad E_3 = (Y_{D2}+Z_{D2})^2 - A_3 - B_3,\quad F_3 = 3C_3,\quad X_{D3} = D_3 \cdot (A_3 - F_3),$
$Y_{D3} = (A_3+F_3)^2 - 12C_3^2,\quad Z_{D3} = 4A_3 \cdot E_3.$

The above sequence of operations costs $40\mathbf{m}_e + 32\mathbf{s}_e + 8e\mathbf{m}_1$.

A.4 Octupling formulas for $y^2 = x^3 + ax$

$X_{1,2} = X_1^2$, $B = Y_1^2$, $Z_{1,s} = Z_1^2$, $Z_{1,2} = aZ_{1,s}$, $X_{D^1} = (X_{1,2} - Z_{1,2})^2$, $E = 2(X_{1,2} + Z_{1,2})^2 - X_{D^1}$,
$F = (X_{1,2} - Z_{1,2} + Y_1)^2 - B - X_{D^1}$, $Y_{D^1} = E \cdot F$, $Z_{D^1} = 4B$, $Z_{1,s^2} = Z_{1,s}^2$, $Z_{1,s^4} = Z_{1,s^2}^2$, $X_{1,2} = X_1^2$, $X_{1,4} = X_{1,2}^2$,
$X_{1,8} = X_{1,4}^2$, $X_{1,16} = X_{1,8}^2$, $X_{1,32} = X_{1,16}^2$, $X_{1,6} = (X_{1,2} + X_{1,4})^2 - X_{1,4} - X_{1,8}$, $X_{1,10} = (X_{1,2} + X_{1,8})^2 - X_{1,4} - X_{1,16}$,
$X_{1,12} = (X_{1,4} + X_{1,8})^2 - X_{1,8} - X_{1,16}$, $X_{1,14} = (X_{1,8} + X_{1,6})^2 - X_{1,16} - 2X_{1,12}$, $X_{1,18} = (X_{1,16} + X_{1,2})^2 - X_{1,32} - X_{1,4}$,
$X_{1,20} = (X_{1,16} + X_{1,4})^2 - X_{1,32} - X_{1,8}$, $X_{1,22} = (X_{1,16} + X_{1,6})^2 - X_{1,32} - 2X_{1,12}$, $X_{1,24} = (X_{1,16} + X_{1,8})^2 - X_{1,32} - X_{1,16}$,
$X_{1,26} = (X_{1,16} + X_{1,10})^2 - X_{1,32} - 2X_{1,20}$, $X_{1,28} = (X_{1,16} + X_{1,12})^2 - X_{1,32} - 2X_{1,24}$,
$X_{1,30} = (X_{1,16} + X_{1,14})^2 - X_{1,32} - 4X_{1,28}$, $Z_{1,4} = a^2 Z_{1,s^2}$, $Z_{1,8} = a^4 Z_{1,s^4}$, $Z_{1,16} = Z_{1,8}^2$, $Z_{1,32} = Z_{1,16}^2$,
$Z_{1,6} = (Z_{1,2} + Z_{1,4})^2 - Z_{1,4} - Z_{1,8}$, $Z_{1,10} = (Z_{1,2} + Z_{1,8})^2 - Z_{1,4} - Z_{1,16}$, $Z_{1,12} = (Z_{1,4} + Z_{1,8})^2 - Z_{1,8} - Z_{1,16}$,
$Z_{1,14} = (Z_{1,8} + Z_{1,6})^2 - Z_{1,16} - 2Z_{1,12}$, $Z_{1,18} = (Z_{1,16} + Z_{1,2})^2 - Z_{1,32} - Z_{1,4}$, $Z_{1,20} = (Z_{1,16} + Z_{1,4})^2 - Z_{1,32} - Z_{1,8}$,
$Z_{1,22} = (Z_{1,16} + Z_{1,6})^2 - Z_{1,32} - 2Z_{1,12}$, $Z_{1,24} = (Z_{1,16} + Z_{1,8})^2 - Z_{1,32} - Z_{1,16}$, $Z_{1,26} = (Z_{1,16} + Z_{1,10})^2 - Z_{1,32} - 2Z_{1,20}$,
$Z_{1,28} = (Z_{1,16} + Z_{1,12})^2 - Z_{1,32} - 2Z_{1,24}$, $Z_{1,30} = (Z_{1,16} + Z_{1,14})^2 - Z_{1,32} - 4Z_{1,28}$, $C_0^{XZ} = X_{1,32}$, $C_1^{XZ} = X_{1,30} \cdot Z_{1,2}$,
$C_2^{XZ} = X_{1,28} \cdot Z_{1,4}$, $C_3^{XZ} = X_{1,26} \cdot Z_{1,6}$, $C_4^{XZ} = X_{1,24} \cdot Z_{1,8}$, $C_5^{XZ} = X_{1,22} \cdot Z_{1,10}$, $C_6^{XZ} = X_{1,20} \cdot Z_{1,12}$,
$C_7^{XZ} = X_{1,18} \cdot Z_{1,14}$, $C_8^{XZ} = X_{1,16} \cdot Z_{1,16}$, $C_9^{XZ} = X_{1,14} \cdot Z_{1,18}$, $C_{10}^{XZ} = X_{1,12} \cdot Z_{1,20}$, $C_{11}^{XZ} = X_{1,10} \cdot Z_{1,22}$,
$C_{12}^{XZ} = X_{1,8} \cdot Z_{1,24}$, $C_{13}^{XZ} = X_{1,6} \cdot Z_{1,26}$, $C_{14}^{XZ} = X_{1,4} \cdot Z_{1,28}$, $C_{15}^{XZ} = X_{1,2} \cdot Z_{1,30}$, $C_{16}^{XZ} = Z_{1,32}$,
$G = (X_{1,2} + Z_{1,s^2})^2 - X_{1,4} - Z_{1,s^4}$, $H = (X_1 + Z1)^2 - X_{1,2} - Z_{1,s}$, $II = H^2$, $J = H \cdot II$,
$K = (X_{1,4} + Z_{1,s})^2 - X_{1,8} - Z_{1,s^2}$, $L = (H + X_{1,4})^2 - II - X_{1,8}$, $M = (Y_1 + Z_{1,s^2})^2 - B - Z_{1,s^4}$,
$N = (Y_1 + Z_{1,s})^2 - B - Z_{1,s^2}$, $R = H \cdot N$, $S = II \cdot Y_1$, $T = (X_{1,2} + Y_1)^2 - X_{1,4} - B$, $U = T \cdot H$,

$$
\begin{aligned}
L_{4,0} &= -2G \cdot (63C_0^{XZ} + 546C_1^{XZ} - 17646C_2^{XZ} - 86058C_3^{XZ} - 944238C_4^{XZ} - 925278C_5^{XZ} - 4412322C_6^{XZ} - 2092730C_7^{XZ} \\
&\quad - 318342C_8^{XZ} + 1595958C_9^{XZ} + 2710846C_{10}^{XZ} + 441618C_{11}^{XZ} + 325074C_{12}^{XZ} + 21510C_{13}^{XZ} + 2930C_{14}^{XZ} - 46C_{15}^{XZ} + C_{16}^{XZ}), \\
L_{3,0} &= -2J \cdot (105C_0^{XZ} + 756C_1^{XZ} - 15990C_2^{XZ} - 84112C_3^{XZ} - 1082058C_4^{XZ} - 610644C_5^{XZ} - 2610994C_6^{XZ} - 2003688C_7^{XZ} \\
&\quad - 13594266C_8^{XZ} - 674868C_9^{XZ} + 164566C_{10}^{XZ} + 223168C_{11}^{XZ} + 232998C_{12}^{XZ} - 492C_{13}^{XZ} + 2226C_{14}^{XZ} + 56C_{15}^{XZ} - 7C_{16}^{XZ}), \\
L_{2,0} &= -4K \cdot (189C_0^{XZ} + 882C_1^{XZ} + 6174C_2^{XZ} - 26274C_3^{XZ} - 1052730C_4^{XZ} - 449598C_5^{XZ} - 1280286C_6^{XZ} - 1838850C_7^{XZ} \\
&\quad - 23063794C_8^{XZ} - 1543290C_9^{XZ} + 539634C_{10}^{XZ} + 646922C_{11}^{XZ} + 1386918C_{12}^{XZ} + 75846C_{13}^{XZ} + 17262C_{14}^{XZ} + 922C_{15}^{XZ} - 35C_{16}^{XZ}), \\
L_{1,0} &= 4L \cdot (9C_0^{XZ} - 3666C_2^{XZ} + 2580C_3^{XZ} + 263226C_4^{XZ} + 328248C_5^{XZ} + 1359882C_6^{XZ} + 1017948C_7^{XZ} + 11998650C_8^{XZ} \\
&\quad + 1661904C_9^{XZ} + 1958226C_{10}^{XZ} + 178956C_{11}^{XZ} - 315222C_{12}^{XZ} - 39560C_{13}^{XZ} - 4842C_{14}^{XZ} - 252C_{15}^{XZ} + 7C_{16}^{XZ}), \\
L_{0,0} &= 2X_{1,6} \cdot (C_0^{XZ} - 42C_1^{XZ} - 834C_2^{XZ} - 8702C_3^{XZ} - 38898C_4^{XZ} + 80886C_5^{XZ} + 654642C_6^{XZ} + 450098C_7^{XZ} \\
&\quad + 3346502C_8^{XZ} + 450098C_9^{XZ} + 654642C_{10}^{XZ} + 80886C_{11}^{XZ} - 38898C_{12}^{XZ} - 8702C_{13}^{XZ} - 834C_{14}^{XZ} - 42C_{15}^{XZ} + C_{16}^{XZ}), \\
L_{3,1} &= 2M \cdot (8C_0^{XZ} + 73C_1^{XZ} - 2718C_2^{XZ} - 12087C_3^{XZ} - 110316C_4^{XZ} - 143283C_5^{XZ} - 603830C_6^{XZ} - 159171C_7^{XZ} \\
&\quad + 1273368C_8^{XZ} + 301915C_9^{XZ} + 286566C_{10}^{XZ} + 27579C_{11}^{XZ} + 48348C_{12}^{XZ} + 1359C_{13}^{XZ} - 146C_{14}^{XZ} - C_{15}^{XZ}), \\
L_{2,1} &= R \cdot (216C_0^{XZ} + 1719C_1^{XZ} - 49530C_2^{XZ} - 225297C_3^{XZ} - 2336292C_4^{XZ} - 1899741C_5^{XZ} - 8313570C_6^{XZ} - 3992373C_7^{XZ} \\
&\quad - 6366840C_8^{XZ} + 1434309C_9^{XZ} + 2776722C_{10}^{XZ} + 427917C_{11}^{XZ} + 107508C_{12}^{XZ} + 10017C_{13}^{XZ} + 2122C_{14}^{XZ} - 7C_{15}^{XZ}), \\
L_{1,1} &= S \cdot (504C_0^{XZ} + 3055C_1^{XZ} - 38146C_2^{XZ} - 226593C_3^{XZ} - 3358356C_4^{XZ} - 982485C_5^{XZ} - 3428010C_6^{XZ} - 4734229C_7^{XZ} \\
&\quad - 46394904C_8^{XZ} - 2925939C_9^{XZ} - 560070C_{10}^{XZ} + 510845C_{11}^{XZ} + 849828C_{12}^{XZ} + 15897C_{13}^{XZ} + 3570C_{14}^{XZ} - 7C_{15}^{XZ}), \\
L_{0,1} &= U \cdot (168C_0^{XZ} + 417C_1^{XZ} + 26106C_2^{XZ} + 19449C_3^{XZ} - 808860C_4^{XZ} - 981963C_5^{XZ} - 3150686C_6^{XZ} - 1673251C_7^{XZ} \\
&\quad - 16203528C_8^{XZ} - 1636605C_9^{XZ} - 889746C_{10}^{XZ} + 58347C_{11}^{XZ} + 226252C_{12}^{XZ} + 2919C_{13}^{XZ} + 630C_{14}^{XZ} - C_{15}^{XZ}).
\end{aligned}
$$

$$F^* = \alpha \cdot (L_{4,0} \cdot x_S^4 + L_{3,0} \cdot x_S^3 + L_{2,0} \cdot x_S^2 + L_{1,0} \cdot x_S + L_{3,1} \cdot x_S^3 y_S + L_{2,1} \cdot x_S^2 y_S + L_{1,1} \cdot x_S y_S + L_{0,0}),$$

$A_2 = X_1^2$, $B_2 = Y_1^2$, $C_2 = Z_1^2$, $D_2 = aC_2$, $X_{D^2} = (A_2 - D_2)^2$, $E_2 = 2(A_2 + D_2)^2 - X_{D^2}$, $Z_{D^2} = 4B_2$,
$F_2 = ((A_2 - D_2 + Y_1)^2 - B_2 - X_{D^2}$, $Y_{D^2} = E_2 \cdot F_2$.

$A_3 = X_1^2$, $B_3 = Y_1^2$, $C_3 = Z_1^2$, $D_3 = aC_3$, $X_{D^3} = (A_3 - D_3)^2$, $E_3 = 2(A_3 + D_3)^2 - X_{D^3}$, $Z_{D^3} = 4B_3$,
$F_3 = ((A_3 - D_3 + Y_1)^2 - B_3 - X_{D^3}$, $Y_{D^3} = E_3 \cdot F_3$.

The above sequence of operations costs $32\mathbf{m}_e + 57\mathbf{s}_e + 8e\mathbf{m}_1$.

B Explicit Formulas

The following MAGMA code is a simple implementation of the Miller quadruple-and-and and Miller octuple-and-add algorithms. We specify curves of the form $y^2 = x^3 + b$ and condense the code due to space considerations. The main function `Miller2nTuple` takes as inputs the two points R and S on E, the value r which is the order of R, the two curve constants a and b, the integer n (for 2^n-tupling) and the full extension field K, so that $R, S \in E(K)$. `Miller2nTuple` either calls the function `Quadruple` or the function `Octuple` for $n = 2$ and $n = 3$ respectively (the call to `Octuple` is currently commented out).

```
function Dbl(X1,Y1,Z1, xQ, yQ,b) A:=X1^2; B:=Y1^2; C:=Z1^2; D:=3*b*C; E:=(X1+Y1)^2-A-B; F:=(Y1+Z1)^2-B-C; G:=3*D; X3:=E*(B-G);
Y3:=(B+G)^2-12*D^2; Z3:=4*B*F; L10:= 3*A; LO1:=-F; LO0:=D-B; F:=L10*xQ+LO1*yQ+LO0; return X3,Y3,Z3,F; end function;

function Add(X1, Y1, Z1, X2, Y2, Z2, xQ, yQ) c1:=X2-xQ; t1:=Z1*X2; t1:=X1-t1; t2:=Z1*Y2; t2:=Y1-t2; F:=c1*t2-t1*Y2+t1*yQ; t3:=t1^2;
X3:=t3*X1; t3:=t1*t3; t4:=t2^2; t4:=t4*Z1; t4:=t3+t4; t4:=t4-X3; t4:=t4-X3; X3:=X3-t4; t2:=t2*X3; Y3:=t3*Y1; Y3:=t2-Y3; X3:=t1*t4;
Z3:=Z1*t3; return X3, Y3, Z3, F; end function;

function Quadruple(Tx, Ty, Tz, Sx, Sy, Sx2, SxSy, b)
A:=Ty^2; B:=Tz^2; C:=A^2; D:=B^2; E:=(Ty+Tz)^2-A-B; F:=E^2; G:=Tx^2; H:=(Tx+Ty)^2-A-G; I:=(Tx+E)^2-F-G; J:=(A+E)^2-C-F; K:=(Ty+B)^2-A-D;
L:=27*b^2*D; M:=9*b*F; N:=A*C; R:=A*L; S:=b*B; T:=S*L; U:=S*C; X3:=2*H*(A-9*S); Y3:=2*C+M-2*L; Z3:=4*J; L10:=-4*Tz*(5*N+5*R-3*T-75*U);
L20:=-3*G*Tz*(10*C+3*M-2*L); LO1:=2*I*(5*C+L); L11:=2*K*Y3; LO0:=2*Tx*(N+R-3*T-75*U); F:= L10*Sx+L20*Sx2+LO1*Sy+L11*SxSy+LO0; A2:=Y3^2;
B2:=Z3^2; C2:=3*b*B2; D2:= 2*X3*Y3; E2:=(Y3+Z3)^2-A2-B2; F2:=3*C2; X3:= D2*(A2-F2); Y3:=(A2+F2)^2-12*C2^2; Z3:=4*A2*E2;
return X3,Y3,Z3,F;
end function;

function Octuple(X1, Y1, Z1, Sx, Sy, Sx2, SxSy, Sx3, Sx4, Sx2Sy, Sx3Sy, b)
Y12:=Y1^2; Z1s:=Z1^2; Z12:=b*Z1s; A:=X1^2; B:=3*Z12; C:=(X1+Y1)^2-A-Y12; DD:=(Y1+Z1)^2-Y12-Z1s; E:=3*B; X3:=C*(Y12-E);
Y3:=(Y12+E)^2-12*B^2; Z3:=4*Y12*DD; Xt,Yt,Zt:=Dbl(X1,Y1,Z1,Sx,Sy,b); Z14s:=Z1s^2; Y14:=Y12^2; Y18:=Y14^2; Y116:=Y18^2;
Y16:=(Y12+Y14)^2-Y14-Y18; Y110:=(Y18+Y12)^2-Y116-Y14; Y112:=(Y18+Y14)^2-Y116-Y18; Y114:= (Y18+Y16)^2-Y116-2*Y112; Y118:=Y116*Y12;
Y120:=Y116*Y14; Y122:=Y116*Y16; Z14:=b^2*Z14s; Z18:=Z14^2; Z116:=Z18^2; Z16:=(Z12+Z14)^2-Z14-Z18; Z110:=(Z18+Z12)^2-Z116-Z14;
Z112:=(Z18+Z14)^2-Z116-Z18; Z114:= (Z18+Z16)^2-Z116-2*Z112; Z118:=Z116*Z12; Z120:=Z116*Z14; Z122:=Z116*Z16; YZ0:=Y122; YZ1:=Y120*Z12;
YZ2:=Y118*Z14; YZ3:=Y116*Z16; YZ4:=Y114*Z18; YZ5:=Y112*Z110; YZ6:=Y110*Z112; YZ7:=Y18*Z114; YZ8:=Y16*Z116; YZ9:=Y14*Z118; YZ10:=Y12*Z120;
YZ11:=Z122; FF:=A*Z1s; G:=(Y12+Z1s)^2-Y14-Z14s; H:=C*DD; II:=C^2; J:=Y12*(Y12+3*Z12); K:=DD*Z1s; L:=C*Z1s; M:=A*DD; N:=Y12*DD;
F40 := -18*FF*(-9565938*YZ10+95659380*YZ9-101859525*YZ8+14880348*YZ7+57100383*YZ6-52396146*YZ5+14332383*YZ4-4578120*YZ3-513162*YZ2
+15732*YZ1+7*YZ0); F30:=-12*G*(-14348907*YZ11+239148450*YZ10-643043610*YZ9+350928207*YZ8-60407127*YZ7-8575227*YZ6-7841853*YZ5
+12011247*YZ4 -3847095*YZ3-1325142*YZ2+56238*YZ1+35*YZ0); F20:=-27*H*(-54206982*YZ10+157660830*YZ9-120282813*YZ8+50368797*YZ7
-25747551*YZ6+10693215*YZ5 -3826845*YZ4+777789*YZ3+35682*YZ2+4102*YZ1+7*YZ0+4782969*YZ11); F10 := -18*II*(-4782969*YZ11+ 28697814*YZ10
-129317310*YZ9+130203045*YZ8-48479229*YZ7+11593287*YZ6-619407*YZ5+1432485*YZ4-883197*YZ3+32814*YZ2-1318*YZ1+YZ0);
F00 :=2*J*(YZ0-6750*YZ1-524898*YZ2-2583657*YZ3 +23839029*YZ4-20484171*YZ5-136370385*YZ6+369351495*YZ7-669084219*YZ8+413461098*YZ9
-47829690*YZ10+14348907*YZ11); F31 := 8*K*(2484*YZ1-915624*YZ3-90558*YZ2-28697814*YZ10+YZ0+95659380*YZ9- 61115715*YZ8+6377292*YZ7
+19033461*YZ6 - 14289858*YZ5+3307473*YZ4); F21 := 216*L*(YZ0+1960*YZ1-56610*YZ2-363744*YZ3+1192077*YZ4-3601260*YZ5 +5084775*YZ6
-3779136*YZ7 +4546773*YZ8 -7085880*YZ9+3188646*YZ10); F11 := 72*M*(8148*YZ1-112266*YZ2+434808*YZ3-2214945*YZ4 +11674206*YZ5-24360993*YZ6
+20549052*YZ7-11160261*YZ8+10628820*YZ9-9565938*YZ10+7*YZ0); F01 :=8*N*(-14348907*YZ11+28697814*YZ10-77590386*YZ9+208856313*YZ8
-152208639*YZ7+87333471*YZ6-19135521*YZ5+543105*YZ4-2329479*YZ3 +508302*YZ2-4138*YZ1+21*YZ0);
F:=F01*Sy+F11*SxSy+F21*Sx2Sy+F31*Sx3Sy+F00+F10*Sx+F20*Sx2+F30*Sx3+F40*Sx4; Y32:=Y3^2; Z3s:=Z3^2; Z32:=b*Z3s; A:=X3^2; B:=3*Z32;
C:=(X3+Y3)^2-A-Y32; DD:=(Y3+Z3)^2-Y32-Z3s; E:=3*B; X3:=C*(Y32-E); Y3:=(Y32+E)^2-12*B^2; Z3:=4*Y32*DD; Y32:=Y3^2; Z3s:=Z3^2; Z32:=b*Z3s;
A:=X3^2; B:=3*Z32; C:=(X3+Y3)^2-A-Y32; DD:=(Y3+Z3)^2-Y32-Z3s; E:=3*B; X3:=C*(Y32-E); Y3:=(Y32+E)^2-12*B^2; Z3:=4*Y32*DD;
return X3,Y3,Z3,F;
end function;

function Miller2nTuple(R, S, r, a, b, n, K)
Rx:=R[1]; Ry:=R[2]; Rz:=R[3];
Sx:=S[1]; Sy:=S[2]; Sx2:=Sx^2; Sx3:=Sx^3; Sx4:=Sx^4; SxSy:=Sx*Sy; Sx2Sy:=Sx2*Sy; Sx3Sy:=Sx3*Sy;
Rmultiplesmatrix:=[[Rx, Ry, Rz]];
for i:=2 to (2^n-1) by 1 do
     iR:=i*R;
     Rmultiplesmatrix:=Append(Rmultiplesmatrix, [iR[1], iR[2], iR[3]]);
end for;
fRaddvec:=[K!1]; addproduct:=fRaddvec[1];
ptx, pty, ptz, F := Dbl(Rx,Ry,Rz,Sx,Sy,b);
addproduct*:= F;
fRaddvec:=Append(fRaddvec, addproduct);
for i:=3 to (2^n-1) by 1 do
     ptx, pty, ptz, faddvalue := Add(ptx, pty, ptz, Rx, Ry, Rz, Sx, Sy);
     addproduct*:=faddvalue;
     fRaddvec:=Append(fRaddvec, addproduct);
end for;
Tx:=Rx; Ty:=Ry; Tz:=Rz;
f1 := 1; B := IntegerToSequence(r,2^n);
if B[#B] ne 1 then
     Tx, Ty, Tz, F:= Add(Tx, Ty, Tz, Rmultiplesmatrix[B[#B]][1], Rmultiplesmatrix[B[#B]][2], Rmultiplesmatrix[B[#B]][3], Sx, Sy);
     F:=F*fRaddvec[B[#B]];
     f1:=f1*F;
end if;
for i:=#B-1 to 1 by -1 do
Tx, Ty, Tz, F:=Quadruple(Tx, Ty, Tz, Sx, Sy, Sx2, SxSy, b);
     //Tx, Ty, Tz, F:=Octuple(Tx, Ty, Tz, Sx, Sy, Sx2, SxSy, Sx3, Sx4, Sx2Sy, Sx3Sy, b);
     f1:=f1^(2^n)*F;
     if B[i] ne 0 then
          Tx, Ty, Tz, F:= Add(Tx, Ty, Tz, Rmultiplesmatrix[B[i]][1], Rmultiplesmatrix[B[i]][2], Rmultiplesmatrix[B[i]][3], Sx, Sy);
          F:=F*fRaddvec[B[i]];
          f1:=f1*F;
     end if;
end for;
return f1;
end function;
```