J Cryptol (2016) 29:28–60
DOI: 10.1007/s00145-014-9188-7

Journal of
CRYPTOLOGY

Fast Cryptography in Genus 2

Joppe W. Bos and Craig Costello[*]
Microsoft Research, Redmond, USA
jbos@microsoft.com

Huseyin Hisil[†]
Yasar University, Izmir, Turkey

Kristin Lauter
Microsoft Research, Redmond, USA

Received 11 March 2013
Online publication 15 November 2014

Abstract. In this paper, we highlight the benefits of using genus 2 curves in public-key cryptography. Compared to the standardized genus 1 curves, or elliptic curves, arithmetic on genus 2 curves is typically more involved but allows us to work with moduli of half the size. We give a taxonomy of the best known techniques to realize genus 2-based cryptography, which includes fast formulas on the Kummer surface and efficient four-dimensional GLV decompositions. By studying different modular arithmetic approaches on these curves, we present a range of genus 2 implementations. On a single core of an Intel Core i7-3520M (Ivy Bridge), our implementation on the Kummer surface breaks the 125 thousand cycle barrier which sets a new software speed record at the 128-bit security level for constant-time scalar multiplications compared to all previous genus 1 and genus 2 implementations.

1. Introduction

Since its invention in the 1980s, elliptic curve cryptography [41,50] has become a popular and standardized approach to instantiate public-key cryptography. The use of elliptic curves, or *genus* 1 *curves*, has been well studied and consequently all of the speed records for fast curve-based cryptography are for elliptic curves (cf. the ECRYPT online benchmarking tool eBACS [10]). Jacobians of hyperelliptic curves of high genus have also been considered for cryptographic purposes, but for large genus there are "faster-than-generic" attacks on the discrete logarithm problem [2,20,23,27]. Such attacks are not

[*] Part of this work was done while the second author was working in the Department of Mathematics and Computer Science at the Technische Universiteit Eindhoven, Netherlands

[†] This article is the extended version of an earlier article: Fast Cryptography in Genus 2, EUROCRYPT, LNCS, Vol. 7881, pp. 194-210, ©IACR 2013, 10.1007/978-3-642-38348-9_12

known, however, for *genus* 2 *curves*. In [29], Gaudry showed that scalar multiplication on the Kummer surface associated with the Jacobian of a genus 2 curve can be more efficient than scalar multiplication on the Jacobian itself. Thus, it was proposed (cf. [6]) that hyperelliptic curve cryptography in genus 2 has the potential to be competitive with its genus 1 elliptic curve cryptography counterpart. One significant hurdle for genus 2 cryptography to overcome is the difficulty of generating secure genus 2 curves: that is, such that the Jacobian has a large prime or almost prime group order. In particular, for fast cryptographic implementations, it is advantageous to use special prime fields where the underlying arithmetic is fast and to generate curves over those fields with suitable group orders. A major catalyst for this work is that genus 2 point counting methods and complex multiplication (CM) methods for constructing genus 2 curves with a known group order have become more practical. Hence, the time is ripe to give a taxonomy and a cross-comparison of all of the best known techniques for genus 2 curves over prime fields. The focus on prime fields is motivated by the recommendations made by the United States' National Security Agency Suite B of Cryptographic Protocols [55].

In this paper, we set new performance speed records at the 128-bit security level using genus 2 hyperelliptic curves. For instance, using the Kummer surface given by Gaudry and Schost [34], we present the fastest curve-based scalar multiplication over prime fields to date—this improves on the recent prime field record for elliptic curves from Longa and Sica which was presented at Asiacrypt 2012 [48]. Furthermore, our implementations on the Kummer surface inherently run in constant time, which is one of the major steps toward achieving a side-channel resistant implementation [42]. Thus, we present the fastest constant-time software for curve-based cryptography compared to *all* prior implementations.

Another advantage for genus 2 curves is that the endomorphism ring is larger than it is in the case of genus 1, so it is possible to achieve higher dimensional GLV scalar decompositions [26] (without passing to an extension field to make use of GLS [25]). For prime fields, we implement four-dimensional GLV decompositions on Buhler–Koblitz (BK) curves [16] and on Furukawa–Kawazoe–Takahashi (FKT) curves [24], both of which are faster than all prior eBACS-documented implementations. To optimize overall performance, we present implementations based on two different methods that allow fast modular arithmetic: one based on the special form of the prime using "NIST-like" reduction [65] and another based on the special form of the prime when using Montgomery multiplication [51].

In addition, we put forward a multifaceted case for (a special class of) Buhler–Koblitz curves of the form $y^2 = x^5 + b$. The curves we propose are particularly flexible in applications because they facilitate both a Kummer surface implementation and a GLV decomposition. Thus, a simple Diffie–Hellman style key exchange can be instantiated using the fast formulas on an associated Kummer surface, but if a more complicated protocol requires further group operations, one has the option to instead exploit a four-dimensional GLV implementation using the Buhler–Koblitz form.

The paper is organized as follows. In Sect. 2, we recall the necessary background for this work. Section 3 outlines the two different approaches for the modular arithmetic. Section 4, 5 and 6 summarize the state-of-the-art in "generic," Kummer surface and GLV implementations, respectively, together with the specific choices and optimizations we made in each scenario. Section 7 presents our performance results. In Sect. 8, we propose a particular family of curves that allow both Kummer surface and GLV implementations. Section 9 concludes the paper.

2. Preliminaries

We start by recalling some basic facts and notation concerning genus 2 curves in §2.1. In §2.2, we outline the CM method, which is used several times in this work to generate secure curves. In §2.3, we briefly review the main techniques used to compute scalar multiplications.

2.1. *Genus 2 Curves*

A hyperelliptic genus 2 curve over a field of odd characteristic K can be defined by an affine model $\mathcal{C} : y^2 = f(x)$, where $f(x)$ has degree 5 or 6 and has no double roots. We call $\mathcal{C}$ a *real* hyperelliptic curve if the degree of f is 6, and if such an $f(x)$ has a rational root, then we can isomorphically transform the curve over K so that f has degree 5 instead, in which case we say $\mathcal{C}$ is an *imaginary* hyperelliptic curve. Arithmetic is currently slightly faster in the imaginary case.

Unlike genus 1 elliptic curves, in genus 2, the points on the curve do not form a group. Roughly speaking, unordered pairs of points on the curve form a group, where the group operation adds two pairs of points by passing a cubic through the four points, finding the other two points of intersection with the curve, and then reflecting them over the x axis. More formally, we call this group the *divisor class group*, which consists of the degree zero divisors modulo the principal divisors on $\mathcal{C}$. There is a natural isomorphism between the divisor class group and the K-rational points on $J_{\mathcal{C}}$, the *Jacobian* of $\mathcal{C}$, which is the abelian variety we work with. For genus 2 hyperelliptic curves, each divisor class has a unique *reduced* representative consisting of at most two $\bar{K}$-rational points (which are not reflections of each other) minus the point(s) at infinity. The corresponding elements in the Jacobian can therefore be represented by encoding these two points via a pair of polynomials, where the x coordinates of the points are the roots of the first polynomial and the second polynomial is a line passing through the two points. This encoding is called the *Mumford representation*; it writes general points $D \in J_{\mathcal{C}}$ as $D = (x^2 + u_1x + u_0, v_1x + v_0)$. In order to avoid confusion when x and y are used as two of the Kummer coordinates in Sect. 5, in this paper we will often abbreviate the Mumford representation to instead write $D = (u_1, u_0, v_1, v_0)$. Following [19], we will save inversions by introducing an additional coordinate Z to write such points as $D = (U_1 : U_0 : V_1 : V_0 : Z)$, where $u_i = U_i/Z$ and $v_i = V_i/Z$ for $i \in \{0, 1\}$ and $Z \neq 0$.

2.2. *The CM Method*

There are two high-level strategies for constructing cryptographically strong genus 2 curves. The first strategy is *point counting*, which typically involves fixing a particular genus 2 curve $\mathcal{C}$ (over an underlying field) and using the classical Schoof-Pila [59,62] algorithm to compute $\# J_{\mathcal{C}}$, repeating the process for different curves until this group order is prime or almost prime. Until recently, using this technique to compute the group orders of Jacobians of curves which target the 128-bit security level was infeasible. However, in their record-breaking work, Gaudry and Schost [34] presented a fast combination of the Schoof–Pila algorithm and the baby-step giant-step method [63] that manages to compute the order of the Jacobian corresponding to any such a curve in around 1,000 CPU hours. They further integrated an early abort strategy into this extended point counting routine to find a 128-bit secure curve in over 1,000,000 CPU hours. The Kummer surface associated with the curve they found is especially attractive for fast implementations, and we use it to obtain record performance numbers in this work. Even more recently, on families of curves which have been constructed to have known real multiplication (RM), Gaudry, Kohel and Smith [33] gave an accelerated Schoof–Pila algorithm and set a record for RM point counting, computing a 128-bit secure Jacobian in about 3 hours.

The second strategy for finding cryptographically secure genus 2 curves is the CM method, which we use several times throughout this paper to find curves defined over prime fields that facilitate fast field arithmetic. The CM method works as follows. For a smooth, projective, irreducible genus 2 curve, $\mathcal{C}$, over a prime field $\mathbb{F}_p$ with ordinary Jacobian $J_{\mathcal{C}}$, the Frobenius endomorphism has a quartic characteristic polynomial $f(t) = t^4 - s_1t^3 + s_2t^2 - ps_1t + p^2$. Let K be the quartic CM field defined by the polynomial f and fix an embedding of K into the complex numbers. We denote by π a complex root of the polynomial $f(t)$. The roots of f consist of conjugate pairs $(\pi, \overline{\pi})$ and $(\pi', \overline{\pi}')$, with the property $\pi'\overline{\pi}' = \pi\overline{\pi} = p$. If a solution to $\pi\overline{\pi} = p$ exists in the field K, then the ideal $\mathfrak{p} = (\pi)$ in $\mathcal{O}_K$ has relative norm $\mathfrak{p}\overline{\mathfrak{p}} = p$ (this is the norm relative to K_0, where K_0 is the real quadratic subfield of K). Thus, given a CM field K and a prime p, the ordinary genus 2 curves over $\mathbb{F}_p$ with CM by K (i.e., with $\mathrm{End}(J_{\mathcal{C}}) \cong \mathcal{O}_K$) correspond to generators π of principal ideals with relative norm p such that $|\pi| = \sqrt{p}$. Note that a generator may have to be scaled by a unit in $\mathcal{O}_K$ to ensure that $|\pi| = \sqrt{p}$. Since $\#J_{\mathcal{C}}(\mathbb{F}_p) = (1-\pi)(1-\overline{\pi})(1-\pi')(1-\overline{\pi}')$, in order to know the possible group orders for genus 2 curves with CM by K, it suffices to find the prime ideal decomposition of p in $\mathcal{O}_K$ (which determines all possible π's). For primes which split completely into principal ideals in the reflex field of K, there are always two possible group orders when $K \neq \mathbb{Q}(\zeta_5)$ is Galois cyclic and four possible group orders when K is non-Galois (see [22, Proposition 4] for the possibilities). When $K = \mathbb{Q}(\zeta_5)$ and $p \equiv 1 \bmod 10$ (as used in Sect. 6), p always splits completely into four principal ideals and there are ten possible group orders in this case.

When a CM field K gives rise to a suitable group order over $\mathbb{F}_p$, the next problem is to construct a genus 2 curve with the desired number of points. We use Shimura's theory which shows that CM abelian varieties correspond to ideal classes in $\mathcal{O}_K$, and their invariants are values of the genus 2 Siegel modular functions defined by Igusa; these invariants can be computed modulo p as roots of the *Igusa class polynomials*. These Igusa class polynomials have coefficients in $\mathbb{Q}$ and are computationally expensive to compute. There are three general methods of approaching this computation: the complex analytic method [72], the Chinese remainder theorem (CRT) method [22], and the p-adic method [32]. All of the class polynomials we used in this work were taken from Kohel's comprehensive Echidna database [43]. Upon computing the Igusa invariants, we can then reconstruct the curve $\mathcal{C}/\mathbb{F}_p$ using the Mestre-Cardona-Quer algorithm [49].

Depending on the scenario, we use the CM method in one of two ways. We either start by fixing a prime field $\mathbb{F}_p$ before searching through many CM fields until we find a curve whose Jacobian has prime or almost prime group order, or conversely, we start with a fixed CM field K and then search over many prime fields until we find a suitable curve. The first approach is used when we do not require curves corresponding to a particular

CM field or when the defining equation for $\mathcal{C}$ is not important, which is the case when searching for "generic" curves (see §4.2) and for curves facilitating arithmetic on the Kummer surface (see §5.5). Alternatively, we use the second approach when we need either a certain defining equation for $\mathcal{C}$ (e.g., the GLV curves in §6.2), or if we need to fix a particular CM field (e.g., the van Wamelen curves in §8.5). Roughly speaking, if we can afford flexibility in the curves we search for, then this allows us to be picky with the underlying fields we choose. Conversely, being picky with the curves we seek usually means we have to be more flexible with the primes we search with.

2.3. *Scalar Multiplication*

There are many different ways to compute the scalar multiplication. Most approaches, like the double-and-add algorithm, are based on *addition chains* [61] and a typical optimization to lower the number of point additions is using *windows* [14] of a certain width $w > 1$. Given the input point P, we compute a lookup table consisting of the multiples $[c]P$ such that $0 \leq c < 2^w$ and perform a point addition once every w bits (instead of at most once per bit). After adding a precomputed multiple, we can "slide" to the next set-bit in the binary representation of the scalar; such *sliding windows* [68] lower the number of point additions required and halve the size of the lookup table since only the odd multiples of P are required. When computing the negation of a group element is inexpensive, which is the case for both elliptic and genus 2 curves, we can either add or subtract the precomputed point, reducing the total number of group operations even further; this is called the *signed windows* approach [54]. See [9] for a summary of these techniques.

Adding an affine point to a projective point to obtain another projective point, often referred to as mixed addition, is usually faster than adding two projective points. In order to use these faster formulas, a common approach is to convert the precomputed projective points into their affine form. This requires an inversion for each point in the table. Using Montgomery's *simultaneous inversion* method [52], I independent inversions can be replaced by $3(I-1)$ multiplications and a single inversion, which is typically much faster.

3. Fast Modular Arithmetic Using Special Primes

When performing arithmetic modulo a prime p in practice, it is common to use primes of a special form since this may allow fast reduction. For instance, in the FIPS 186-3 standard [69], NIST recommends the use of five prime fields when using the elliptic curve digital signature algorithm (but see also [4]). A study of a software implementation of the NIST-recommended elliptic curves over prime fields on the x86 architecture is given by Brown et al. [15], and in [11], a comparison is made between the performance when using Montgomery multiplication [51] and specialized multiplication using the NIST primes. In this section, we describe two different approaches to obtain fast modular arithmetic. We use the prime $p_{1271} = 2^{127} - 1$ to illustrate both methods, since this prime is used in some of our implementations (cf. Sects. 4 and 5).

3.1. *Generalized Mersenne Primes*

Primes that enable fast reduction techniques are usually of the form $2^s \pm \delta$, where $s, \delta \in \mathbb{Z}^+$, and $\delta \ll 2^s$. The constant δ is small compared to the word size of the target architecture, which is typically 32 or 64 bits. Another popular choice is using a generalized Mersenne prime of the form $2^s + \sum_{i \in S} i$, where S is a set of integers $\pm 2^j$ such that $|2^j| < 2^s$ and the cardinality of S is small. For example, fast reduction modulo $p = 2^s - \delta$ can be done as follows. For integers $0 \leq a, b, c_h, c_\ell, \delta < 2^s$, write $c = a \cdot b = c_h \cdot 2^s + c_\ell \equiv c_\ell + \delta c_h \pmod{2^s - \delta}$ where $0 \leq c_\ell + \delta c_h < (\delta + 1)2^s$. At the cost of a multiplication by δ (which might be a shift depending on the form of δ) and an addition, compute $c' \equiv c \pmod{p}$ where c' is (much) smaller than c, depending on the size of δ. This is the basic idea behind Solinas' reduction scheme [65], which is used to implement fast arithmetic modulo the NIST primes [69]. We refer to this type of reduction as *NIST-like reduction*. When computing $a \cdot b \bmod p_{1271}$ with $0 \leq a, b < p_{1271}$, one can first compute the multiplication $c = a \cdot b = c_1 \cdot 2^{128} + c_0$, where $0 \leq c_1, c_0 < 2^{128}$. A first reduction step can be computed as $c' = (c_0 \bmod 2^{127}) + 2 \cdot c_1 + \lfloor c_0 / 2^{127} \rfloor \equiv c \pmod{p_{1271}}$, such that $0 \leq c' < 2^{128}$. One can then reduce c' further using conditional subtractions. Modular reduction in the case of p_{1271} can therefore be computed without using any multiplications.

3.2. *Montgomery-Friendly Primes*

Montgomery multiplication [51] involves transforming each of the operands into their Montgomery representations and replacing the conventional modular multiplications by Montgomery multiplications. One of the advantages of this method is that the computational complexity is usually better than the classical method by a constant factor.

Let $r = 2^b$ be the radix of the system and $b > 2$ be the bit length of a word. Let p be an n-word odd prime such that $r^{n-1} \leq p < r^n$, and suppose we have an integer $0 \leq X < p$. The Montgomery radix $R = r^n$ is a fixed integer such that $\gcd(R, p) = 1$. The Montgomery residue of X is defined as $\widetilde{X} = X \cdot R \bmod p$. The Montgomery product of two integers is defined as $M(\widetilde{X}, \widetilde{Y}) = \widetilde{X} \cdot \widetilde{Y} \cdot R^{-1} \bmod p$. Practical instances of Montgomery multiplication use the precomputed value $\mu = -p^{-1} \bmod r$. The interleaved Montgomery multiplication algorithm, in which multiplication and reduction are combined, computes $C = M(A, B)$ for $0 \leq A, B < p$. Let $A = \sum_{i=0}^{n-1} a_i \cdot r^i$, where $0 \leq a_i < r$, and start with $C = 0$. For $i = 0$ to $n - 1$, the result C is updated as

$$C \leftarrow C + a_i \cdot B, \quad C \leftarrow \Big(C + ((\mu \cdot C) \bmod r) \cdot p\Big) \Big/ r.$$

The division by r can be implemented by a shift, since the precomputed value μ ensures that the least significant digit (b bits) of $(C + ((\mu \cdot C) \bmod r) \cdot p)$ is zero. It can be shown that the final Montgomery product C is $0 \leq C < 2 \cdot p$, and therefore, a final conditional subtraction is needed when complete reduction is required. In order to avoid handling additional carries in the Montgomery multiplication, which requires more instructions, our implementations prefer 127-bit moduli over 128-bit moduli. In [45], it is noticed that fixing part of the modulus can have advantages for Montgomery multiplication. For instance, the precomputation of μ can be avoided when $-p^{-1} \equiv \pm 1 \pmod{r}$, which also avoids computing a multiplication by μ for every iteration inside the Montgomery multiplication routine. This technique has been suggested in [1,36,40] as well. When μ is small, e.g., $\mu = \pm 1$, one could lower the cost of the multiplication of p with $(\mu \cdot c_0) \bmod r$ by choosing the $n - 1$ most significant words of p in a similar fashion as for the generalized Mersenne primes: $\lfloor p/2^b \rfloor = 2^s + \sum_{i \in S} i$.

Consider the prime p_{1271} on 64-bit architectures: $r = 2^{64}$ and we have $\mu = -p_{1271}^{-1} \bmod 2^{64} = 1$, so that the multiplication by μ can be avoided. Write $C = c_2 \cdot 2^{128} + c_1 \cdot 2^{64} + c_0$ with $0 \leq c_2, c_1, c_0 < 2^{64}$. Due to the shape of the most-significant word of $p_{1271} = (2^{63} - 1) \cdot 2^{64} + (2^{64} - 1)$, the result of $\frac{C+((\mu \cdot C) \bmod r) \cdot p}{r}$ can be obtained using only two shift and two 64-bit addition instructions by computing $c_2 \cdot 2^{64} + c_0 \cdot 2^{63} + c_1$. Similar to the NIST-like reduction, Montgomery reduction in the setting of p_{1271} can be computed without using any multiplications.

3.3. *Other Arithmetic Operations*

Besides fast multiplication and reduction, the whole spectrum of modular operations is required to implement curve arithmetic. Here, we outline the different approaches we use.

3.3.1. *Modular Inversion*

When using the regular representation of integers, one can either use the (binary) extended GCD algorithm to compute the modular inversion or use the special form of the modulus to compute the inverse by using modular exponentiations. For instance, in the case of p_{1271}, one can exploit the congruence $a^{2^{127}-2} \equiv a^{-1} \pmod{p_{1271}}$. The situation when working in Montgomery form is slightly different. Given the Montgomery form $\tilde{a} = a2^{bn} \bmod p$ of an integer a, we want to compute the Montgomery inverse $\tilde{a}^{-1}2^{2bn} \equiv a^{-1}2^{bn} \pmod{p}$. This would require a classical inversion and modular multiplication; however, we found that the approach presented in [13] (which uses the binary version of the Euclidean algorithm from [38]) is faster in practice. The first step of this approach computes a value $\tilde{a}^{-1}2^{k} \equiv a^{-1}2^{k-bn} \pmod{p}$, for some $0 \leq k < 2bn$. This value is then corrected via a Montgomery multiplication with 2^{3bn-k}. This last multiplication typically requires a lookup table with the different precomputed values $2^{3rn-k} \bmod p$. In the case of $p = 2^{127} - 1$, one can avoid this lookup table since $2^{t} \bmod 2^{127} - 1 = 2^{t \bmod 127}$.

3.3.2. *Modular Addition/Subtraction*

Let $0 \leq a, b < 2^k - c$. We compute $(a + b) \bmod (2^k - c)$ as $((((a + c) + b) \bmod 2^k) - c \cdot (1 - \texttt{carry}((a + c) + b, 2^k))) \bmod 2^k$. The carry function $\texttt{carry}(x, y)$ returns either zero or one if $x < y$ or $x \geq y$, respectively. The output is correct and bounded by $2^k - c$, since if $a + b + c < 2^k$, then $a + b < 2^k - c$, while if $a + b + c \geq 2^k$, then $(a + b + c) \bmod 2^k = a + b - (2^k - c) < 2^k - c$. Note that since $a + c < 2^k$, the addition requires no carry propagation. Furthermore, c is multiplied with either one or zero such that this multiplication amounts to data movement.

The modular subtraction $(a - b) \bmod (2^k - c)$ is performed by computing $(((a - b) \bmod 2^k) - c \cdot \texttt{borrow}(a - b)) \bmod 2^k$. Analogous to the carry function, the borrow function $\texttt{borrow}(x)$ returns zero or one if $x \geq 0$ or $x < 0$, respectively. If $a < b$, then $0 \leq (a - b) \bmod 2^k - c = a - b + (2^k - c) < 2^k - c$, and if $a \geq b$, then $0 \leq a - b < 2^k - c$. In some scenarios, one can compute additions as $(((a + b) \bmod 2^k) + c \cdot \texttt{carry}((a + b), 2^k)) \bmod 2^k$, but we note that here the output may not be completely reduced and can be greater than or equal to $2^k - c$.

4. "Generic" Genus 2 Curves and Their Arithmetic

To give a concrete idea of the advantage gained when working on the Kummer surface or when exploiting GLV endomorphisms, we also consider the generic scenario that employs neither of these techniques.

4.1. *Explicit Formulas*

We make use of the fast formulas for arithmetic on imaginary quadratic curves from [19], which focus on reducing the total number of multiplications in projective point doublings, point additions, and mixed additions.[1] Due to the small size of our fields, the cost of modular addition and subtraction compared to modular multiplication is relatively high. Hence, we optimized the formulas from [19] for 128-bit fields by trading some addition and subtractions for multiplications (see Algorithms 1, 2, and 3).

We assume that our curves are of the form $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$, and count multiplications by the f_i as full multiplications, unless they are zero.[2] Letting **m**, **s**, and **a** be the cost of $\mathbb{F}_p$-multiplications, $\mathbb{F}_p$-squarings, and $\mathbb{F}_p$-additions or subtractions, respectively, we summarize the modified counts as follows. For $D = (U_1 : U_0 : V_1 : V_0 : Z)$, one can compute $[2]D$ in $34\mathbf{m} + 6\mathbf{s} + 34\mathbf{a}$—see Algorithm 1. For the special GLV curves in Sect. 6, which have $f_2 = f_3 = 0$, the projective doubling can be computed using $32\mathbf{m} + 6\mathbf{s} + 32\mathbf{a}$. For $D = (U_1 : U_0 : V_1 : V_0 : Z)$ and $D' = (U_1' : U_0' : V_1' : V_0' : Z')$, one can compute the projective addition $D + D'$ in $44\mathbf{m} + 4\mathbf{s} + 29\mathbf{a}$—see Algorithm 2. For the mixed addition between the projective point $D = (U_1 : U_0 : V_1 : V_0 : Z)$ and the affine point $D' = (u_1' : u_0' : v_1' : v_0')$, one can compute the projective result $D + D'$ in $37\mathbf{m} + 5\mathbf{s} + 29\mathbf{a}$—see Algorithm 3. Full and mixed additions cost the same on the special GLV curves. Given these operation counts, our "generic" implementations performed fastest when using 4-bit signed sliding windows (see §2.3).

4.2. *Curves*

To find "generic" curves for comparison against the GLV and Kummer techniques, we searched Kohel's Echidna database [43] with two fixed primes that facilitate our chosen techniques for field arithmetic. We terminated the search when we found curves with Jacobians of prime order. While these curves are not general in the sense that their CM field is chosen in advance, there is no reason that the corresponding timings obtained would differ from taking any other generic curve over the same prime fields[3], unless such curves are real (degree-6) curves which cannot be transformed into imaginary (degree-5) curves.

[1] Note that the formulas to compute the projective doubling from [19] can be sped up since the first multiplication to compute UU is redundant.

[2] Over prime fields, it is standard to zero the coefficient of the x^4 term via an appropriate substitution.

[3] This is assuming that such generic curves will also have *full-sized* coefficients.

Algorithm 1 Projective doubling for general points in the Jacobian of $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$ and f_2, f_3 (curve constants)
Output: $[2]P = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

```
1. $U_0'' \leftarrow U_0 \cdot Z$
2. $t_1 \leftarrow Z^2$
3. $t_2 \leftarrow U_1^2$
4. $t_3 \leftarrow 2 \cdot t_2$
5. $t_4 \leftarrow 2 \cdot U_0''$
6. $t_5 \leftarrow t_3 + t_4$
7. $t_5 \leftarrow t_5 \cdot U_1$
8. $t_6 \leftarrow V_1^2$
9. $t_7 \leftarrow f_2 \cdot t_1$
10. $t_6 \leftarrow t_7 - t_6$
11. $t_6 \leftarrow t_6 \cdot Z$
12. $t_6 \leftarrow t_6 + t_5$
13. $t_1 \leftarrow f_3 \cdot t_1$
14. $t_1 \leftarrow t_1 + t_2$
15. $t_4 \leftarrow t_1 - t_4$
16. $t_4 \leftarrow t_4 + t_3$
17. $V_0'' \leftarrow V_0 \cdot Z$
18. $t_1 \leftarrow U_1 \cdot V_1$
19. $t_2 \leftarrow 2 \cdot t_1$
20. $t_1 \leftarrow t_1 + V_0''$
21. $t_2 \leftarrow t_2 - V_0''$
22. $t_3 \leftarrow t_3 + U_0''$
23. $t_3 \leftarrow V_1 \cdot t_3$
24. $t_5 \leftarrow t_3 \cdot t_4$
25. $t_7 \leftarrow t_6 \cdot t_2$
26. $t_5 \leftarrow t_5 - t_7$
27. $t_6 \leftarrow t_6 \cdot V_1$
28. $t_4 \leftarrow t_4 \cdot t_1$
29. $t_4 \leftarrow t_4 - t_6$
30. $t_3 \leftarrow t_3 \cdot V_1$
31. $t_1 \leftarrow t_1 \cdot t_2$
32. $t_3 \leftarrow t_3 - t_1$
33. $t_1 \leftarrow t_5 \cdot t_4$
34. $t_2 \leftarrow t_3 \cdot t_4$
35. $t_4 \leftarrow t_4^2$
36. $t_6 \leftarrow U_0'' \cdot t_4$
37. $t_7 \leftarrow t_4 \cdot Z$
38. $t_4 \leftarrow t_4 \cdot U_1$
39. $t_3 \leftarrow 2 \cdot t_3$
40. $t_3 \leftarrow t_3^2$
41. $t_3 \leftarrow t_3 \cdot Z$
42. $t_2 \leftarrow 2 \cdot t_2$
43. $U_0'' \leftarrow t_2 \cdot Z$
44. $V_1'' \leftarrow V_1 \cdot U_0''$
45. $V_0'' \leftarrow V_0'' \cdot t_2$
46. $t_2 \leftarrow t_1 - t_4$
47. $t_5 \leftarrow t_5^2$
48. $t_8 \leftarrow 2 \cdot t_3$
49. $t_8 \leftarrow t_8 - t_2$
50. $t_8 \leftarrow t_8 - t_1$
51. $t_8 \leftarrow t_8 \cdot U_1$
52. $t_8 \leftarrow t_8 + t_5$
53. $t_5 \leftarrow 2 \cdot V_1''$
54. $t_8 \leftarrow t_8 + t_5$
55. $V_1'' \leftarrow t_6 + V_1''$
56. $t_6 \leftarrow t_6 \cdot t_2$
57. $U_1'' \leftarrow 2 \cdot t_2$
58. $U_1'' \leftarrow U_1'' - t_3$
59. $t_2 \leftarrow U_1'' - t_2$
60. $t_4 \leftarrow t_4 - U_1''$
61. $t_4 \leftarrow t_4 \cdot t_2$
62. $t_4 \leftarrow t_4 \cdot Z$
63. $Z'' \leftarrow U_0'' \cdot Z$
64. $t_1 \leftarrow t_1 - U_1''$
65. $U_1'' \leftarrow U_1'' \cdot Z''$
66. $U_0'' \leftarrow t_8 \cdot U_0''$
67. $V_1'' \leftarrow V_1'' - t_8$
68. $V_1'' \leftarrow V_1'' \cdot t_7$
69. $V_1'' \leftarrow t_4 - V_1''$
70. $V_0'' \leftarrow V_0'' \cdot t_7$
71. $t_1 \leftarrow t_1 \cdot t_8$
72. $t_1 \leftarrow t_1 - t_6$
73. $V_0'' \leftarrow t_1 - V_0''$
74. $Z'' \leftarrow Z'' \cdot t_7$
```

4.2.1. *Generic Curve Over $\mathbb{F}_p$ with $p = 2^{127} - 1$.*

The CM field $K = \mathbb{Q}[x]/(x^4 + 137x + 4429)$ has class number 6 [43] and gives rise to a curve $\mathcal{C}$ over $\mathbb{F}_p$ whose Jacobian has prime order

$$r{=}28948022309329048848169239995659025138451177973091551374101475732892580332259,$$

which is 254 bits. A possible degree 5 model is $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$, where

$$f_3{=}34744234758245218589390329770704207149,\ \ f_2{=}132713617209345335075125059444256188021,$$
$$f_1{=}90907655901711006083734360528442376758,\ \ f_0{=}6667986622173728337823560857179992816.$$

4.2.2. *Generic Curve Over $\mathbb{F}_p$ with $p = 2^{128} - 173$.*

The CM field $K = \mathbb{Q}[x]/(x^4 + 41x + 389)$ has class number 1 [43] and gives rise to a curve $\mathcal{C}$ over $\mathbb{F}_p$ whose Jacobian has prime order

$$r{=}115792089237316195429342203801033554170931615651881657307308068079702089951781,$$

which is 257 bits. A possible degree 5 model is $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$, where

$$f_3{=}318258242717201709453901384328569236653,\ \ f_2{=}75380722035796344355219475510170298006,$$
$$f_1{=}129416082603460579272847694630998099237,\ \ f_0{=}143864072772599444046778416709082679388.$$

Algorithm 2 Projective addition between general points in the Jacobian of $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$, $Q = (U_1' : U_0' : V_1' : V_0' : Z')$.
Output: $P + Q = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

```
1. $U_1'' \leftarrow U_1 \cdot Z'$
2. $U_0'' \leftarrow U_0 \cdot Z'$
3. $t_1 \leftarrow V_0 \cdot Z'$
4. $t_2 \leftarrow V_0' \cdot Z$
5. $t_1 \leftarrow t_1 - t_2$
6. $t_2 \leftarrow U_0' \cdot Z$
7. $t_3 \leftarrow U_1' \cdot Z$
8. $t_4 \leftarrow t_3 \cdot t_2$
9. $t_2 \leftarrow t_2 - U_0''$
10. $t_5 \leftarrow U_1'' - t_3$
11. $t_6 \leftarrow U_1'' \cdot U_0''$
12. $t_4 \leftarrow t_4 - t_6$
13. $t_6 \leftarrow V_1' \cdot Z$
14. $Z'' \leftarrow Z \cdot Z'$
15. $t_7 \leftarrow V_1 \cdot Z'$
16. $t_8 \leftarrow t_7 - t_6$
17. $t_6 \leftarrow t_7 + t_6$
18. $t_9 \leftarrow U_1''^2$
19. $t_{10} \leftarrow Z'' \cdot t_2$
20. $t_{10} \leftarrow t_9 + t_{10}$
21. $t_{11} \leftarrow t_3^2$
22. $t_3 \leftarrow U_1'' + t_3$
23. $t_{12} \leftarrow t_{10} - t_{11}$
24. $t_{11} \leftarrow t_9 + t_{11}$
25. $t_9 \leftarrow t_4 \cdot t_8$
26. $t_4 \leftarrow t_4 \cdot t_5$
27. $t_5 \leftarrow t_1 \cdot t_5$
28. $t_1 \leftarrow t_1 \cdot t_{12}$
29. $t_8 \leftarrow t_2 \cdot t_8$
30. $t_2 \leftarrow t_2 \cdot t_{12}$
31. $t_1 \leftarrow t_9 + t_1$
32. $t_5 \leftarrow t_5 + t_8$
33. $t_2 \leftarrow t_2 - t_4$
34. $t_4 \leftarrow t_5 \cdot Z''$
35. $t_8 \leftarrow t_2 \cdot t_4$
36. $t_2 \leftarrow t_2^2$
37. $t_5 \leftarrow t_5 \cdot t_4$
38. $t_4 \leftarrow t_1 \cdot t_4$
39. $U_1'' \leftarrow U_1'' \cdot t_5$
40. $t_9 \leftarrow 2 \cdot t_4$
41. $t_9 \leftarrow t_9 - t_2$
42. $t_{12} \leftarrow t_5 \cdot t_3$
43. $t_9 \leftarrow t_9 - t_{12}$
44. $t_2 \leftarrow t_9 - t_2$
45. $t_2 \leftarrow t_2 \cdot t_3$
46. $t_{11} \leftarrow t_5 \cdot t_{11}$
47. $t_2 \leftarrow t_2 + t_{11}$
48. $t_2 \leftarrow t_2/2$
49. $t_{12} \leftarrow Z'' \cdot t_5$
50. $U_0'' \leftarrow U_0'' \cdot t_{12}$
51. $t_{12} \leftarrow t_8 \cdot t_{12}$
52. $t_{11} \leftarrow Z' \cdot t_{12}$
53. $V_0'' \leftarrow t_{11} \cdot V_0$
54. $V_1'' \leftarrow t_{11} \cdot V_1$
55. $t_{11} \leftarrow t_4 - t_9$
56. $t_4 \leftarrow U_1'' - t_4$
57. $t_1 \leftarrow t_1^2$
58. $t_6 \leftarrow t_8 \cdot t_6$
59. $t_1 \leftarrow t_1 \cdot Z''$
60. $t_1 \leftarrow t_1 + t_6$
61. $t_1 \leftarrow t_1 - t_2$
62. $t_2 \leftarrow t_1 - U_0''$
63. $t_5 \leftarrow t_2 \cdot t_5$
64. $t_2 \leftarrow t_9 \cdot t_{11}$
65. $t_{11} \leftarrow t_1 \cdot t_{11}$
66. $t_6 \leftarrow U_1'' \cdot t_4$
67. $t_6 \leftarrow t_6 + t_2$
68. $t_5 \leftarrow t_6 + t_5$
69. $t_4 \leftarrow U_0'' \cdot t_4$
70. $t_{11} \leftarrow t_4 + t_{11}$
71. $t_9 \leftarrow t_9 \cdot t_8$
72. $U_1'' \leftarrow t_9 \cdot Z''$
73. $U_0'' \leftarrow t_1 \cdot t_8$
74. $t_5 \leftarrow t_5 \cdot Z''$
75. $V_1'' \leftarrow t_5 - V_1''$
76. $V_0'' \leftarrow t_{11} - V_0''$
77. $Z'' \leftarrow Z'' \cdot t_{12}$
```

5. The Kummer Surface

Gaudry [29] built on earlier observations by Chudnovsky and Chudnovsky [17] to show that scalar multiplication in genus 2 can be greatly accelerated by working on the Kummer surface associated with a Jacobian, rather than by working on the Jacobian itself. Although the Kummer surface is not technically a group, it is close enough to a group to be able to define scalar multiplications on it and is therefore an attractive setting for Diffie–Hellman like protocols that do not require any further group operations [64].

5.1. *The Squares-only Kummer Routine*

The Kummer surface that was originally proposed for cryptography in [29] is a surface whose constants are parameterized by the four *fundamental theta constants* ($\vartheta_1(0)$, $\vartheta_2(0)$, $\vartheta_3(0)$, $\vartheta_4(0)$), and whose coordinates come from the four *fundamental theta functions* ($\vartheta_1(\mathbf{z})$, $\vartheta_2(\mathbf{z})$, $\vartheta_3(\mathbf{z})$, $\vartheta_4(\mathbf{z})$), all of which are values of the classical genus 2 *Riemann theta function*. Bernstein [6] pointed out that one can work entirely with the squares of the fundamental theta constants without any loss of efficiency. This provides more flexibility when transforming a given genus 2 curve into an associated Kummer surface and makes it easier to control the size of squared fundamental theta constants, for which small values can give worthwhile speedups. For example, it might be the case

Algorithm 3 Mixed addition between general points in the Jacobian of $\mathcal{C} : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

```
Input: $P = (U_1 : U_0 : V_1 : V_0 : Z)$, $Q = (u_1, u_0, v_1, v_0)$.
Output: $P + Q = (U_1'' : U_0'' : V_1'' : V_0'' : Z'')$.

1. $t_1 \leftarrow v_0 \cdot Z$
2. $V_0'' \leftarrow V_0 - t_1$
3. $t_1 \leftarrow v_1 \cdot Z$
4. $t_2 \leftarrow t_1 + V_1$
5. $t_1 \leftarrow t_1 - V_1$
6. $V_1'' \leftarrow u_1 \cdot Z$
7. $t_3 \leftarrow V_1'' + U_1$
8. $t_4 \leftarrow u_0 \cdot Z$
9. $t_5 \leftarrow V_1'' \cdot t_4$
10. $t_6 \leftarrow U_1 \cdot U_0$
11. $t_6 \leftarrow t_6 - t_5$
12. $U_0'' \leftarrow U_0 - t_4$
13. $t_5 \leftarrow V_1''^2$
14. $t_7 \leftarrow U_1^2$
15. $U_1'' \leftarrow V_1'' - U_1$
16. $t_8 \leftarrow t_5 - t_7$
17. $t_5 \leftarrow t_5 + t_7$
18. $t_7 \leftarrow Z \cdot U_0''$
19. $t_8 \leftarrow t_7 + t_8$
20. $t_7 \leftarrow t_6 \cdot t_1$
21. $t_1 \leftarrow U_0'' \cdot t_1$
22. $U_0'' \leftarrow U_0'' \cdot t_8$
23. $t_6 \leftarrow t_6 \cdot U_1''$
24. $U_1'' \leftarrow V_0'' \cdot U_1''$
25. $t_8 \leftarrow V_0'' \cdot t_8$
26. $t_7 \leftarrow t_7 - t_8$
27. $t_1 \leftarrow t_1 - U_1''$
28. $U_0'' \leftarrow U_0'' - t_6$
29. $t_8 \leftarrow U_0''^2$
30. $t_6 \leftarrow t_1 \cdot Z$
31. $U_0'' \leftarrow U_0'' \cdot t_6$
32. $t_1 \leftarrow t_1 \cdot t_6$
33. $V_1'' \leftarrow t_1 \cdot V_1''$
34. $t_5 \leftarrow t_1 \cdot t_5$
35. $V_0'' \leftarrow t_7 \cdot t_6$
36. $t_6 \leftarrow t_6^2$
37. $t_7 \leftarrow t_7^2$
38. $t_4 \leftarrow t_4 \cdot t_6$
39. $t_6 \leftarrow U_0'' \cdot t_6$
40. $U_1'' \leftarrow 2 \cdot V_0''$
41. $U_1'' \leftarrow U_1'' - t_8$
42. $t_2 \leftarrow U_0'' \cdot t_2$
43. $t_7 \leftarrow t_7 \cdot Z$
44. $t_7 \leftarrow t_7 + t_2$
45. $t_2 \leftarrow t_1 \cdot t_3$
46. $U_1'' \leftarrow U_1'' - t_2$
47. $t_8 \leftarrow U_1'' - t_8$
48. $t_3 \leftarrow t_3 \cdot t_8$
49. $t_3 \leftarrow t_3 + t_5$
50. $t_3 \leftarrow t_3/2$
51. $t_7 \leftarrow t_7 - t_3$
52. $t_8 \leftarrow V_1'' - V_0''$
53. $V_0'' \leftarrow V_0'' - U_1''$
54. $t_5 \leftarrow t_7 - t_4$
55. $V_1'' \leftarrow V_1'' \cdot t_8$
56. $t_1 \leftarrow t_1 \cdot t_5$
57. $t_1 \leftarrow t_1 + V_1''$
58. $V_1'' \leftarrow U_1'' \cdot V_0''$
59. $V_1'' \leftarrow V_1'' + t_1$
60. $t_4 \leftarrow t_4 \cdot t_8$
61. $V_0'' \leftarrow V_0'' \cdot t_7$
62. $V_0'' \leftarrow t_4 + V_0''$
63. $t_4 \leftarrow t_6 \cdot v_1$
64. $V_1'' \leftarrow V_1'' - t_4$
65. $U_1'' \leftarrow U_1'' \cdot Z$
66. $U_1'' \leftarrow U_1'' \cdot U_0''$
67. $U_0'' \leftarrow t_7 \cdot U_0''$
68. $V_1'' \leftarrow Z \cdot V_1''$
69. $Z'' \leftarrow Z \cdot t_6$
70. $t_7 \leftarrow Z'' \cdot v_0$
71. $V_0'' \leftarrow V_0'' - t_7$
```

that the fundamental theta constants associated with a genus 2 curve cannot be defined over $\mathbb{F}_p$, but all of their squares can be.

Cosset [18] formally presented the "squares-only" setting, in which the Kummer surface $\mathcal{K}$ is completely defined by the *squared fundamentals* $(a^2, b^2, c^2, d^2) = (\vartheta_1(0)^2, \vartheta_2(0)^2, \vartheta_3(0)^2, \vartheta_4(0)^2)$ as

$$\begin{aligned}
\mathcal{K}: \quad & E'xyzt = ((x^2 + y^2 + z^2 + t^2) - F(xt + yz) - G(xz + yt) - H(xy + zt))^2, \\
& \text{where} \quad E' = 4E^2a^2b^2c^2d^2, \quad E = \frac{ABCD}{(a^2d^2 - b^2c^2)(a^2c^2 - b^2d^2)(a^2b^2 - c^2d^2)}, \\
& \qquad F = \frac{a^4 - b^4 - c^4 + d^4}{a^2d^2 - b^2c^2}, \quad G = \frac{a^4 - b^4 + c^4 - d^4}{a^2c^2 - b^2d^2}, \\
& \qquad\qquad H = \frac{a^4 + b^4 - c^4 - d^4}{a^2b^2 - c^2d^2}, \\
& \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} a^2 \\ b^2 \\ c^2 \\ d^2 \end{bmatrix}.
\end{aligned} \qquad (1)$$

We write $(x : y : z : t) = (\vartheta_1(\mathbf{z})^2 : \vartheta_2(\mathbf{z})^2 : \vartheta_3(\mathbf{z})^2 : \vartheta_4(\mathbf{z})^2)$ for the coordinates of a projective point on $\mathcal{K}$. We present here the four algorithms needed to achieve scalar multiplication on a Kummer surface using the squared coordinates. Algorithm 4, the Hadamard transform (H), is a building block used to improve efficiency throughout the entire routine: The linear algebra involved in computing A, B, C, D from (a^2, b^2, c^2, d^2) in (1) appears numerous times in the formulas for arithmetic on $\mathcal{K}$, and this is an optimized way to do those operations [6]. Algorithm 5, $\mathcal{K}(\texttt{DBL})$, computes the doubling $[2]P \in \mathcal{K}$ of a point $P \in \mathcal{K}$, while Algorithm 6, $\mathcal{K}(\texttt{DBLADD})$, computes the *pseudo-addition* of the distinct points $P, Q \in \mathcal{K}$ with known difference $P - Q \in \mathcal{K}$. Both of these algorithms are the squares-only formulas from [18]. Algorithm 7 computes the scalar multiple $[k]P \in \mathcal{K}$ of $P \in \mathcal{K}$ using a genus 2 version [29] of the Montgomery ladder [52]. The six surface constants that appear in the algorithms are defined as

$$y_0 = \frac{a^2}{b^2}, \quad z_0 = \frac{a^2}{c^2}, \quad t_0 = \frac{a^2}{d^2}, \quad y_0' = \frac{A}{B}, \quad z_0' = \frac{A}{C}, \quad t_0' = \frac{A}{D}. \tag{2}$$

Algorithm 4 The Hadamard transform ($\texttt{H}$).

```
Input: $(x, y, z, t)$.
Output: $\texttt{H}(x, y, z, t)$.
$t_1 \leftarrow x + y,\ t_2 \leftarrow z + t$
$t_3 \leftarrow x - y,\ t_4 \leftarrow z - t$
$x \leftarrow t_1 + t_2,\ y \leftarrow t_1 - t_2$
$z \leftarrow t_3 + t_4,\ t \leftarrow t_3 - t_4$
return $(x, y, z, t)$.
```

Algorithm 5 Doubling, $\mathcal{K}(\texttt{DBL})$.

```
Input: $P = (x : y : z : t)$ and constants $y_0, z_0, t_0, y_0', z_0', t_0'$.
Output: $[2]P = \texttt{DBL}(P)$.
$x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$.
$x \leftarrow x^2,\ y \leftarrow y^2,\ z \leftarrow z^2,\ t \leftarrow t^2$.
$y \leftarrow y \cdot y_0',\ z \leftarrow z \cdot z_0',\ t \leftarrow t \cdot t_0'$.
$x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$.
$x \leftarrow x^2,\ y \leftarrow y^2,\ z \leftarrow z^2,\ t \leftarrow t^2$.
$y \leftarrow y \cdot y_0,\ z \leftarrow z \cdot z_0, t \leftarrow t \cdot t_0$.
return $(x : y : z : t)$.
```

Although the formulas in Algorithm 6 are presented for general inputs P, Q, and $P - Q$, the inputs to $\mathcal{K}(\texttt{DBLADD})$ in the laddering algorithm are always of the form $[m]P$ and $[m + 1]P$, so their difference is always the initial point P (see lines 4 and 6 of Algorithm 7). Thus, the inversions in Lines 15 and 16 of Algorithm 6 can all be precomputed. In fact, since $\mathcal{K}$ is projective we can multiply each coordinate in this line by any scalar, say $\bar{x}$, such that Lines 15 and 16 are modified to compute three multiplications: $y' \leftarrow Y \cdot (\bar{x}/\bar{y})$, $z' \leftarrow Z \cdot (\bar{x}/\bar{z})$, and $t' \leftarrow T \cdot (\bar{t}/\bar{y})$, where the quotients in the parentheses are precomputed and stay fixed throughout the scalar multiplication [6,29].

5.2. *Extracting the Squared Kummer Surface Parameters from $\mathcal{C}$*

In [29], Gaudry showed the relationship between the Kummer surface and the isomorphic Rosenhain model of the genus 2 curve $\mathcal{C}$, given as

$$\mathcal{C}_{\mathrm{Ros}}\colon\ y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu), \tag{3}$$

where the Rosenhain invariants λ, μ and ν are linked to the squared fundamentals by

$$\lambda = \frac{a^2c^2}{b^2d^2},\ \mu = \frac{c^2(1+\sqrt{CD/AB})}{d^2(1-\sqrt{CD/AB})},\ \nu = \frac{a^2(1+\sqrt{CD/AB})}{b^2(1-\sqrt{CD/AB})},$$

with A, B, C, D as in (1). Since the three Rosenhain invariants are functions of the four squared fundamentals, there is a degree of freedom when inverting the equations to compute (a^2, b^2, c^2, d^2) from (λ, μ, ν). Thus, we can set $d^2 = 1$ [31] and compute one set of the squared fundamentals as

$$c^2 = \sqrt{\frac{\lambda\mu}{\nu}}, \qquad b^2 = \sqrt{\frac{\mu(\mu-1)(\lambda-\nu)}{\nu(\nu-1)(\lambda-\mu)}}, \qquad a^2 = b^2c^2\frac{\nu}{\mu}.$$

Given a hyperelliptic curve $\mathcal{C}$ of genus 2, there are up to 120 unique Rosenhain triples λ, μ, ν that give an isomorphic representation $\mathcal{C}_{\mathrm{Ros}} \cong \mathcal{C}$ over the algebraic closure [28, §2.2]. So for a given curve with rational 2-torsion, we can hope that there may be at least one Rosenhain triple for which the square roots above lie in the same field as λ, μ, and ν, such that the Kummer surface is also defined over the same field (but see §8.3). If the 2-torsion is rational, then 16 must divide the cardinality of $J_{\mathcal{C}}$ [29].

Algorithm 6 Combined doubling and pseudo-addition, $\mathcal{K}$(DBLADD).

```
Input: $P = (x: y: z: t)$, $Q = (x': y': z': t')$,
    $P - Q = (\bar{x}: \bar{y}: \bar{z}: \bar{t})$, and $y_0, z_0, t_0, y'_0, z'_0, t'_0$
Output: $([2]P, P+Q) = \mathtt{DBLADD}(P, Q, P-Q)$
$x, y, z, t \leftarrow \mathtt{H}(x, y, z, t)$
$x', y', z', t' \leftarrow \mathtt{H}(x', y', z', t')$
$X \leftarrow x \cdot x',\ Y \leftarrow y \cdot y'_0$
$Z \leftarrow z \cdot z'_0,\ T \leftarrow t \cdot t'_0$
$x \leftarrow x^2,\ y \leftarrow y \cdot Y$
$z \leftarrow z \cdot Z,\ t \leftarrow t \cdot T$
$Y \leftarrow Y \cdot y', Z \leftarrow Z \cdot z', T \leftarrow T \cdot t'$
$x, y, z, t \leftarrow \mathtt{H}(x, y, z, t)$
$X, Y, Z, T \leftarrow \mathtt{H}(X, Y, Z, T)$
$x \leftarrow x^2,\ y \leftarrow y^2$
$z \leftarrow z^2,\ t \leftarrow t^2$
$X \leftarrow X^2,\ Y \leftarrow Y^2$
$Z \leftarrow Z^2,\ T \leftarrow T^2$
$y \leftarrow y \cdot y_0,\ z \leftarrow z \cdot z_0,\ t \leftarrow t \cdot t_0$
$x' \leftarrow X/\bar{x},\ y' \leftarrow Y/\bar{y}$
$z' \leftarrow Z/\bar{z},\ t' \leftarrow T/\bar{t}$
return $((x: y: z: t), (x': y': z': t'))$
```

Algorithm 7 Scalar multiplication, $\mathcal{K}(\texttt{SMUL})$.

```
Input: $P = (x : y : z : t)$ and integer
    $n = \sum_{i=0}^{\ell-1} n_i 2^i$ with $n > 2$.
Output: $[n]P \in \mathcal{K}$.
1. $P_m \leftarrow P, \quad P_p = \texttt{DBL}(P)$.
2. for $i = \ell - 2$ down to 0 do
3.     if $n_i = 1$ then
4.         $(P_p, P_m) \leftarrow \mathcal{K}(\texttt{DBLADD})(P_p, P_m, P)$
5.     else
6.         $(P_m, P_p) \leftarrow \mathcal{K}(\texttt{DBLADD})(P_m, P_p, P)$
7. $(x : y : z : t) \leftarrow P_m$.
8. return $(x : y : z : t)$.
```

5.3. *Mapping from $\mathcal{K}$ to $J_{\mathcal{C}}$*

The maps from $\mathcal{K}$ to $J_{\mathcal{C}}$ were originally given by Gaudry [29] and tweaked for the squares-only case by Cosset [18]. We reproduce them here for completeness, correcting a sign mistake introduced in the computation of v_0 in [18]. It should be noted that the map below is not directly to the Jacobian of $\mathcal{C}$, but rather to the Jacobian of the isomorphic curve $\mathcal{C}_{\text{Ros}}$ in Rosenhain form. The map takes $P = (x : y : z : t) \in \mathcal{K}$ to $D = (u_1, u_0, v_1, v_0)$ or $D = (u_1, u_0, -v_1, -v_0)$, where the choice between these two possibilities is made when we choose the square root in the computation of v_0 in (5).

We expand the first part of the map (to the u-polynomial of D), to write it as

$$u_0 = \frac{u_x x + u_y y + u_z z + u_t t}{d_x x + d_y y + d_z z + d_t t} \quad \text{and} \quad u_1 = \frac{u'_x x + u'_y y + u'_z z + u'_t t}{d_x x + d_y y + d_z z + d_t t} - u_0 - 1,$$

where

$$\begin{aligned}
u_x &= -\vartheta_1^2\vartheta_3^2\vartheta_8^2\vartheta_5^2\vartheta_9^2, & u'_x &= -\vartheta_7^2\vartheta_9^4\vartheta_5^2\vartheta_8^2, & d_x &= -\vartheta_2^2\vartheta_4^2\vartheta_{10}^2\vartheta_6^2\vartheta_7^2,\\
u_y &= -\vartheta_1^2\vartheta_3^2\vartheta_8^2\vartheta_6^2\vartheta_7^2, & u'_y &= \vartheta_7^2\vartheta_9^4\vartheta_5^2\vartheta_{10}^2, & d_y &= -\vartheta_2^2\vartheta_4^2\vartheta_{10}^2\vartheta_5^2\vartheta_9^2,\\
u_z &= \vartheta_1^2\vartheta_3^2\vartheta_8^2\vartheta_5^2\vartheta_7^2, & u'_z &= \vartheta_7^4\vartheta_9^2\vartheta_5^2\vartheta_8^2, & d_z &= \vartheta_2^2\vartheta_4^2\vartheta_{10}^2\vartheta_6^2\vartheta_9^2,\\
u_t &= \vartheta_1^2\vartheta_3^2\vartheta_8^2\vartheta_6^2\vartheta_9^2, & u'_t &= -\vartheta_7^4\vartheta_9^2\vartheta_5^2\vartheta_{10}^2, & d_t &= \vartheta_2^2\vartheta_4^2\vartheta_{10}^2\vartheta_5^2\vartheta_7^2.
\end{aligned} \tag{4}$$

For the computation of v_0 and v_1, we have

$$\begin{aligned}
\ell = -\Big(&\vartheta_{12}^2(\mathbf{z})\vartheta_7(\mathbf{z})^2 b^2c^2\vartheta_9^4 + \vartheta_{11}^2(\mathbf{z})\vartheta_9^2(\mathbf{z})a^2d^2\vartheta_7^4 + 2a^2b^2c^2d^2(xz + yt)\\
&+ \Big(x^2 + y^2 + z^2 + t^2 - F(xt + yz) - G(xz + yt) - H(xy + zt)\Big)\frac{a^2c^2 + b^2d^2}{E}\Big),
\end{aligned}$$

$$v_0 = \sqrt{\ell \cdot \frac{\vartheta_8^2 \vartheta_3^4 \vartheta_1^4 \vartheta_{14}^2(\mathbf{z})}{(\vartheta_{16}^2(\mathbf{z}) b^2 d^2 \vartheta_{10}^2)^3}},$$
$$v_1 = \frac{u_0^3 - u_0^2(u_1^2 + u_1 + (u_1+1)(\lambda+\mu+\nu) + \lambda\mu + \nu\lambda + \nu\mu) + u_0\lambda\mu\nu + u_1 v_0^2}{2 v_0 u_0}, \tag{5}$$

where the λ, μ, and ν are the particular choice of Rosenhain invariants corresponding to $\mathcal{C}_{\text{Ros}}$ in (3). The six theta constants ϑ_i^2 with $i = 5, \ldots, 10$ and the six theta functions $\vartheta_j^2(\mathbf{z})$ with $j \in \{7, 9, 11, 12, 14, 16\}$ are all exactly as in [29, §7.3-7.4].

5.4. *Twist Security*

There is an additional security consideration when working on the Kummer surface because a random point on $\mathcal{K}$ can map to either the curve $\mathcal{C}_{\text{Ros}} \cong \mathcal{C}$ or its twist $\mathcal{C}'_{\text{Ros}} \cong \mathcal{C}'$ [29, §5.2]. As long as the public generator $P \in \mathcal{K}$ is chosen so that it maps back to $J_{\mathcal{C}_{\text{Ros}}}$, then any honest party participating in a Diffie–Hellman style protocol computes with multiples of P that also map back to $J_{\mathcal{C}_{\text{Ros}}}$. However, an attacker could feed a party another point $P' \in \mathcal{K}$ that (unbeknownst to the party) maps back to $\mathcal{C}'_{\text{Ros}}$, and on return of $[s]P'$, attack the discrete logarithm problem on the twist instead. It is undesirable to include a check of which curve the Kummer points map to, because the maps above are overly involved. The best solution is to compute curves where both $J_{\mathcal{C}}$ and $J_{\mathcal{C}'}$ have large prime order subgroups. The ideal situation is to have $J_{\mathcal{C}} = 16 \cdot r$ and $J_{\mathcal{C}'} = 16 \cdot r'$, where r and r' are large primes (or almost primes) of the same size. Such curves and their associated Kummer surfaces are called *twist-secure* [33,34].

5.5. *Curves and Their Kummers*

Our implementations use two different Kummer surfaces defined over the prime fields with $p = 2^{127} - 1$ and $p = 2^{128} - 34827$. In the case of $p = 2^{127} - 1$, we use the twist-secure curve found by Gaudry and Schost [34]. For the prime $p = 2^{128} - 34827$, we used the CM method to generate a twist-secure genus 2 curve.

5.5.1. *Kummer Surface over $p = 2^{127} - 1$.*

Gaudry and Schost [34] label the curve as $\mathcal{C}_{11,-22,-19,-3}$, since the squared fundamental theta constants are $(a^2, b^2, c^2, d^2) = (11, -22, -19, -3)$. A corresponding degree 5 isomorphic Rosenhain model is given by the constants

$$\lambda{=}28356863910078205288614550619314017618, \quad \mu{=}154040945529144206406682019582013187910,$$
$$\nu{=}113206060534360680770189432771018826227.$$

The group orders of $J_{\mathcal{C}} \cong J_{\mathcal{C}_{\text{Ros}}}$ and $J_{\mathcal{C}'} \cong J_{\mathcal{C}'_{\text{Ros}}}$ are given by $2^4 \cdot r$ and $2^4 \cdot r'$, respectively, where

$$r{=}1809251394333065553414675955050290598923508843635941313077767297801179626051,$$
$$r'{=}1809251394333065553571917326471206521441306174399683558571672623546356726339,$$

which are 250- and 251-bit primes, respectively. The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'=3729914622627959090638987406589505673, \quad F=145242473685766417331928186098925456110,$$
$$G=816677680610252312312099057836243707490, \quad H=540582355476407258010377720836421071700.$$

Since the curve is twist-secure, we are free to choose any generator, for example the generator

$$P=(P_x\colon P_y\colon P_z\colon P_t)=[2](1\colon 1\colon 1\colon 78525529738642755703105688163803666634)$$

has order r on $\mathcal{K}$. The identity element is $\mathcal{O} = (a^2\colon b^2\colon c^2\colon d^2) \in \mathcal{K}$.

5.5.2. *Kummer Surface over* $p = 2^{128} - 34827$.

For this prime, we found a twist-secure Kummer surface with CM by the quartic field $K = \mathbb{Q}[x]/(x^4+25x+155)$ which has class number 4 [43]. One choice of the Rosenhain model is given by the constants

$$\lambda=45778738964487293478077907344653244 21, \quad \mu=234789861994364729479821884660190521407,$$
$$\nu=174333573523192164016359058694895260480.$$

The group orders of $J_{\mathcal{C}} \cong J_{\mathcal{C}_{\text{Ros}}}$ and $J_{\mathcal{C}'} \cong J_{\mathcal{C}'_{\text{Ros}}}$ are given by $2^4 \cdot r$ and $2^4 \cdot r'$, respectively, where

$$r=7237005577332262213873777499831869959603008537304907265194947995580039622121,$$
$$r'=7237005577332262214072595626254115559239653709785542361513021741721255316601,$$

which are 252- and 253-bit primes, respectively. One choice of the squared fundamentals corresponding to the above Rosenhain triple is

$$a^2=201243713144713214956272800789965999200, \quad b^2=146836762876902436263761475043196343 60,$$
$$c^2=337904041799211257424383908244663970063, \quad d^2=1.$$

The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'=253880210280671006989033320516440357350, \quad F=159016999959358912503454506705451672908,$$
$$G=72998263011580475773814426394752329 07, \quad H=137930629160012836756188734308287568 06.$$

A generator on $\mathcal{K}$ with order r that maps back to $J_{\mathcal{C}_{\text{Ros}}}$ is $P = (P_x\colon P_y\colon P_z\colon P_t)$ where

$$P_x=1, \quad P_y=295122894880835761537997219301486683608,$$
$$P_z=116829357115721232420761146513526735912, \quad P_t=99251552912154476320478841520348830750.$$

The identity element $\mathcal{O} = (a^2\colon b^2\colon c^2\colon d^2) \in \mathcal{K}$.

5.6. *Implementation Details and Side-channel Resistance*

From Algorithm 7, it is clear that for every bit in the scalar, except the first one, the combined double and pseudo-addition routine (Algorithm 6) is called. The main branch, i.e., checking if the bit is set (or not), can be converted into straight-line code by masking the in- and output appropriately. In this case, since no lookup tables are used, the algorithm and runtime become independent of the input. The only input-dependent value is the scalar n whose bit-size can differ, meaning that the total runtime could potentially leak the value of the most significant bits. In order to make the implementation run in constant time, either we can increase the scalar via addition of the subgroup order, or we can artificially increase the running time by computing on dummy values such that the computation of $\mathcal{K}(\texttt{DBLADD})$ occurs exactly $\lceil \log_2(r) \rceil - 1$ times after calling $\mathcal{K}(\texttt{DBL})$ once only.

We note that we incur a cost of $16\mathbf{m} + 9\mathbf{s} + 32\mathbf{a}$ each time $\mathcal{K}(\texttt{DBLADD})$ is called, where 6 of the multiplications are by surface constants. For the curve over $p = 2^{127} - 1$ found by Gaudry and Schost (see §5.5), the 6 surface constants are $y_0 = -1/2$, $z_0 = -11/19$, $t_0 = -11/3$, $y_0' = -3$, $z_0' = -33/17$, and $t_0' = -33/49$, where it is immediately clear that the multiplications by y_0 and y_0' are less expensive than full $\mathbb{F}_p$ multiplications. As we mentioned in §5.1, the projective nature of $\mathcal{K}$ allows us to simultaneously multiply the coordinates of any point on $\mathcal{K}$ by a constant factor. From Algorithm 6, we can see that this also permits us to rescale either set of the surface constants, i.e., we are free to scale those appearing on Line 14 (y_0, z_0 and t_0) and/or those appearing on Lines 3 and 4 (y_0', z_0' and t_0') of Algorithm 6 by any nonzero factor in $\mathbb{F}_p$. To determine the best scaling of the surface constants, we must first note that the expressions in (2) were already scaled so that two original constants x_0 and x_0' both became 1 (and were thus omitted), meaning that any scaling must be simultaneously applied to the four constants x_0, y_0, z_0, and t_0 or the four constants x_0', y_0', z_0', and t_0'. As it stands, multiplications by $z_0 = -11/19$ and $t_0 = -11/3$ are treated as full multiplications in $\mathbb{F}_p$, so suppose we clear the denominators of this first set of constants to instead take $(x_0, y_0, z_0, t_0) = (-114, 57, 66, 418)$. In this case, all four of the multiplications are now by "single-word" constants, which are naturally faster than full $\mathbb{F}_p$ multiplications where both operands occupy two machine words. In our implementations, however, we found that the code ran faster when the constants were essentially left unchanged, save for the scaling of $(x_0, y_0, z_0, t_0) = (1, -1/2, -11/19, -11/3)$ to $(x_0, y_0, z_0, t_0) = (2, -1, -22/19, -22/3)$, where the multiplication by 2 is slightly faster than the division by 2. We optimized all of the obvious combinations of scalings at the assembly level, such as clearing the smallest denominator only, but this always destroyed one of the constants being 1, which was not made up for by the benefit of reducing two-word constants into one-word constants.

6. GLV in Genus 2

The Gallant–Lambert–Vanstone (GLV) method [26] significantly speeds up scalar multiplication on algebraic curves that admit an efficiently computable endomorphism ϕ of degree $d > 1$, by decomposing the scalar k into d "mini-scalars," all of which have bit-lengths that are approximately $1/d$ that of k. The d scalar multiplications corresponding to each of these mini-scalars can then be computed as one multi-scalar multiplication of length $\approx \log_2(k)/d$, which effectively reduces the number of required doublings by a factor of d.

6.1. *Endomorphisms*

In general, algebraic curves over prime fields do not come equipped with a useful endomorphism ϕ, which means that we have to use special curves to take advantage of the

GLV method. For genus 1 elliptic curves, Gallant et al. suggested the curves $y^2 = x^3 + b$ and $y^2 = x^3 + ax$, which both allow a two-dimensional decompositions over prime fields. On the other hand, the genus 2 analogues of these curves, Buhler–Koblitz (BK) curves of the form $y^2 = x^5 + b$ [16] and Furukawa–Kawazoe–Takahashi (FKT) curves of the form $y^2 = x^5 + ax$ [24], have ϕ's whose minimal polynomials are of degree 4, which means that we can achieve four-dimensional scalar decompositions on genus 2 curves over prime fields. We note that the Jacobians of FKT curves are not absolutely simple, so BK curves are likely to be the preferred option in practice. Besides the two families above that offer four-dimensional GLV decompositions, families of genus 2 curves with RM facilitate two-dimensional scalar decompositions [33,44]. To give an idea of the expected performance in such scenarios, we also present timings for a two-dimensional GLV decomposition on FKT curves. However, we note that the curves in [33] are likely to be even faster, since they can be found over special prime fields (e.g., with $p = 2^{127} - 1$).

6.1.1. *Dimension-4 GLV on BK Curves*

To achieve a four-dimensional GLV on curves of the form $\mathcal{C} : y^2 = x^5 + b$, we require $p \equiv 1 \pmod{10}$, so that the non-trivial fifth roots of unity are in $\mathbb{F}_p$. Buhler and Koblitz showed how we can compute the group order of $J_{\mathcal{C}}$ efficiently in this scenario [16] (also see [24, §6]), and we note that Jacobians of these curves can have prime order. Take any $\xi_5 \neq 1$ such that $1 = \xi_5^5 \in \mathbb{F}_p$ and observe that if $(x, y) \in \mathcal{C}$, then $(\xi_5 x, y) \in \mathcal{C}$. This induces an endomorphism ϕ on the Jacobian that is defined on full degree elements as $\phi : (u_1, u_0, v_1, v_0) \mapsto (\xi_5 u_1, \xi_5^2 u_0, \xi_5^4 v_1, v_0)$, which costs only 3 multiplications in $\mathbb{F}_p$ because the ξ_i^j are all precomputed. The minimal polynomial of ϕ is $T^4 + T^3 + T^2 + T + 1 = 0$.

6.1.2. *Dimension-4 GLV on FKT Curves*

Curves of the form $\mathcal{C} : y^2 = x^5 + ax$ need to be defined over fields of characteristic $p \equiv 1 \pmod{8}$, so that the eighth roots of unity are all found in $\mathbb{F}_p$. Computing the cardinality of $J_{\mathcal{C}}$ in this scenario is also efficient [24]. Since the point $(x, y) = (0, 0) \in \mathcal{C}$ induces a point of order 2 on $J_{\mathcal{C}}$, the best we can do is to find a curve whose Jacobian is of order two times a prime. Let $\xi_8 \neq 1$ be a primitive eighth root of unity in $\mathbb{F}_p$, and observe that if $(x, y) \in \mathcal{C}$, then $(\xi_8^2 x, \xi_8 y)$. The induced endomorphism on full degree Jacobian elements is $\phi : (u_1, u_0, v_1, v_0) \mapsto (\xi_8^2 u_1, \xi_8^4 u_0, \xi_8^7 v_1, \xi_8 v_0)$, which costs 4 multiplications in $\mathbb{F}_p$ and which satisfies the minimal polynomial $T^4 + 1 = 0$.

6.1.3. *Dimension-2 GLV on FKT Curves*

The reason we chose FKT curves for the two-dimensional example is because we can take the endomorphism $\phi^2 : (u_1, u_0, v_1, v_0) \mapsto (\xi_8^4 u_1, u_0, \xi_8^6 v_1, \xi_8^2 v_0)$, which has minimal polynomial $T^2 + 1 = 0$. For the Buhler–Koblitz curves, we can still get a two-dimensional decomposition by defining $\phi : (x, y) \mapsto ((\xi_5 + \xi_5^{-1})x, y)$ on $\mathcal{C}$ and extending $\mathbb{Z}$-linearly under the canonical embedding of $\mathcal{C}$ in $J_{\mathcal{C}}$. In this case, ϕ satisfies the minimal polynomial $T^2 + T - 1$ in $J_{\mathcal{C}}$.

6.2. *Curves*

We searched for BK and FKT curves over prime fields $\mathbb{F}_p$ for 127-bit primes that are suited to Montgomery style reduction and for 128-bit primes that are suited to the NIST-style modular reduction. There are only a few isomorphism classes for both types of curves over any particular prime field, so we had to search numerous primes before we found cryptographically suitable curves. Since the definitions of both prime forms encompass a vast number of primes, we were able to find a field (in both cases) that simultaneously gave a prime order BK and an FKT curve with an optimal cofactor of 2.

6.2.1. *GLV Curves over a* 127*-bit Prime Field*

Let $p_{127m} = (2^{63} - 27443) \cdot 2^{64} + 1$. This is a Montgomery-friendly prime (see §3.2) where $\mu = -p_{127m}^{-1} \bmod 2^{64} = -1$. The Jacobians of the curves $\mathcal{C}_{\mathrm{BK}}/\mathbb{F}_{p_{127m}} : y^2 = x^5 + 17$ and $\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}} : y^2 = x^5 + 17x$ have orders $\#J_{\mathcal{C}_{\mathrm{BK}}} = r$ and $\#J_{\mathcal{C}_{\mathrm{FKT}}} = 2 \cdot r'$, where

$$r=28948022309328876595115567994214488524823328209723866335483563634241778912751,$$
$$r'=14474011154664438299023932553432254007696198466166455661883334092795880233441$$

are 254- and 253-bit primes, respectively.

6.2.2. *GLV Curves over a* 128*-bit Prime Field*

Let $p_{128n} = 2^{128} - 24935$. The Jacobians of the curves $\mathcal{C}_{\mathrm{BK}}/\mathbb{F}_{p_{128n}} : y^2 = x^5 + 3^7$ and $\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}} : y^2 = x^5 + 3^7x$ have orders $\#J_{\mathcal{C}_{\mathrm{BK}}} = r$ and $\#J_{\mathcal{C}_{\mathrm{FKT}}} = 2 \cdot r'$, respectively, where

$$r=115792089237316195401210495125503591471546519982099914586091636775415022457661,$$
$$r'=57896044618658097706542424143127279595817201688638085882569066869306899160801.$$

are 256- and 255-bit primes, respectively.

6.3. *Scalar Decomposition via Division*

At Eurocrypt 2002, Park, Jeong and Lim [58] gave an algorithm for performing GLV decomposition via division in the ring $\mathbb{Z}[\phi]$ generated by ϕ. This algorithm is very simple and effective in decomposing the scalar k quickly: In the four-dimensional cases (BK and FKT), it takes 20 multiplications to fully decompose k, and in the two-dimensional case, the decomposition totals just 6 multiplications. For the curves we used, this algorithm performed slightly better on average than the (conservative) numbers quoted in [58, Table 4]. Table 1 gives the statistics from $1,000,000$ decompositions of random scalars in $[0, r)$ in each scenario. Each of the columns report the percentage frequency at which k decomposed into vectors with the given maximal bit length. For example, consider the third row which reports the statistics corresponding to four-dimensional decompositions on Buhler-Koblitz curves with r being 254 bits. The third column indicates that around 21% of scalars decomposed to 4 mini-scalars where the maximum bit length was 64, while the fourth column reports that around 59% of scalars decomposed to 4 mini-scalars

Table 1. Statistics for 1,000,000 scalar decompositions in each of the GLV scenarios .

Curve/prime-GLV dimension	r (bits)	$\max\{\lvert k_\ell \rvert\}$ (bits) / freq. (%)											
$\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}}$ - 2	253	**126**	**50.05**	125	37.39	124	9.40	123	2.38	122	0.58	≤ 121	0.20
$\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}}$ - 2	255	**127**	**50.08**	126	37.47	125	9.35	124	2.33	123	0.58	≤ 122	0.20
$\mathcal{C}_{\mathrm{BK}}/\mathbb{F}_{p_{127m}}$ - 4	254	64	21.04	**63**	**59.24**	62	18.18	61	14.46	60	0.09	≤ 59	0.01
$\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}}$ - 4	253	63	1.00	**62**	**60.29**	61	35.61	60	2.92	59	0.18	≤ 58	0.01
$\mathcal{C}_{\mathrm{BK}}/\mathbb{F}_{p_{128n}}$ - 4	256	65	0.00	64	37.59	**63**	**56.16**	62	5.85	61	0.37	≤ 60	0.03
$\mathcal{C}_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}}$ - 4	255	64	23.38	**63**	**64.26**	62	11.60	61	0.72	60	0.04	≤ 59	0.00

Each row reports a different scenario and the columns across a row show the percentage frequency corresponding to decompositions with a maximum "mini-scalar" length. The final column accounts for all decompositions whose maximum "mini-scalar" length were below a particular bound

$\{k_1, k_2, k_3, k_4\}$ where the maximum bit length was 63. The most common maximum length and its percentage frequency are shown in bold for each scenario.

6.4. *Computing the Scalar Multiplication*

We describe two approaches to implement the scalar multiplication. The d-dimensional decomposition of the scalar k results in d smaller scalars k_ℓ, for $0 \leq \ell < d$. The first approach precomputes the 2^d different points $L_i = \sum_{\ell=0}^{d-1} \left(\left\lfloor \frac{i}{2^\ell} \right\rfloor \bmod 2 \right) \cdot P_\ell$ for $0 \leq i < 2^d$ and stores them in a lookup table. When processing the j^{th} bit of the scalar, the precomputed multiple L_i is added, for $i = \sum_{\ell=0}^{d-1} 2^\ell \left(\left\lfloor \frac{k_\ell}{2^j} \right\rfloor \bmod 2 \right)$. Hence, besides the minor bit-fiddling overhead to construct the lookup table index, this requires computing at most a single curve addition and a single curve doubling per bit of the maximum of the k_ℓ's. The second approach [25] is very similar to using signed windows for a single scalar (see §2.3). We start by precomputing the multiples $L_\ell(c) = [c]P_\ell$ for d different tables: one corresponding to each scalar k_ℓ. When computing the scalar multiplication, the j^{th} part (of width w bits) in the scalar k_ℓ determines which point needs to be added (or subtracted), namely $\sum_{\ell=0}^{d-1} \pm L_\ell \left(\left\lfloor \frac{k_\ell}{2^{wj}} \right\rfloor \bmod 2^w \right)$, where the addition or subtraction depends on the addition–subtraction chain used. Thus, an addition to the running value has to be made only once every w bits and combining the lookup table values takes at most $d-1$ additions, so one needs at most d additions per w bits. The optimal value for w depends on the dimension d, the bit-size of k_ℓ, and the cost of (mixed) additions and doublings. There are multiple ways to save computations in this latter approach. After computing the multiples in the first lookup table L_0, the values for the $d-1$ other tables can be computed by applying the map ϕ to the individual point in the lookup table [25]. Since the computation of the map ϕ only takes three or four multiplications (depending on the curve used), this is a significant saving compared to computing the group operation which is an order of magnitude slower. Furthermore, since the endomorphism costs the same in affine or projective space, one can convert the points in L_0 to affine coordinates using Montgomery's simultaneous inversion method see §2.3) and obtain all of the affine points in the other lookup tables very efficiently through the application of ϕ. This means the faster mixed addition formulas can be

Table 2. Performance timings in 10^3 cycles of various programs calculating a $\lceil \log_2(r) \rceil$-bit scalar multiplication, using genus g arithmetic .

Primitive	g	CT	field char p	$\lceil \log_2(r) \rceil$	#Aut	s	10^3 cycles
curve25519 [4,8]	1	✓	$2^{255}-19$	253	2	125.8	182
ecfp256e [37]	1	×	$2^{256}-587$	255	2	126.8	227
2-GLV [48]	1	×	$2^{256}-11733$	256	6	127.0	145
surf127eps [35]	2	✓	$2^{127}-735$	251	2	124.8	236
NISTp-224 [39,69]	1	✓	$2^{224}-2^{96}+1$	224	2	111.8	302
NISTp-256 [69]	1	?	p_1	256	2	127.8	658
(a) generic127	2	×	$2^{127}-1$	254	2	126.8	295
(b) generic127	2	×	$2^{127}-1$	254	2	126.8	248
(b) generic128	2	×	$2^{128}-173$	257	2	127.8	364
(a) Kummer	2	✓	$2^{127}-1$	251	2	124.8	139
(b) Kummer	2	✓	$2^{127}-1$	251	2	124.8	122
(b) Kummer	2	✓	$2^{128}-237$	253	2	125.8	174
(a) GLV-4-BK	2	×	p_2	254	10	125.7	156
(a) GLV-4-FKT	2	×	p_2	253	8	125.3	156
(a) GLV-2-FKT	2	×	p_2	253	8	125.3	220
(b) GLV-4-BK	2	×	$2^{128}-24935$	256	10	126.7	164
(b) GLV-4-FKT	2	×	$2^{128}-24935$	255	8	126.3	167
(b) GLV-2-FKT	2	×	$2^{128}-24935$	255	8	126.3	261

The curve characteristics, such as the prime p, the cardinality r, the size of the automorphism group #Aut, and the security level $s = \log_2(\sqrt{\frac{\pi r}{2\#\mathrm{Aut}}})$, are stated as well. Here, $p_1 = 2^{256} - 2^{224} + 2^{192} + 2^{96} + 1$ and $p_2 = 2^{64} \cdot (2^{63} - 27443) + 1$. If an implementation runs in constant time (CT), we indicate this with "✓," if not with "×," and if unknown with "?"

applied when adding any element in a lookup table. In our implementations, the first approach is faster in the four-dimensional case and the second approach is faster in the two-dimensional case.

7. Results and Discussion

In §7.1, we discuss our code and the benchmarking environment we used. We present the main results in §7.2 and discuss them further in §7.3. In §7.4, we report timings in the case of key-pair generation, i.e., when a fixed public generator allows for precomputation before the scalar is known.

7.1. *Benchmark Setting and Code*

All of the implementations in Table 2 were run on an Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz with hyperthreading turned off and over-clocking ("turbo boost") disabled. The implementations labeled (a) use the Montgomery-friendly primes. They have been compiled using Microsoft Visual Studio 2012 and run on 64-bit Windows, where the timings are obtained using the time stamp counter instruction `rdtsc` over several thousand scalar multiplications. The implementations labeled (b) use the NIST-like approach and have been compiled with gcc 4.6.3 to run on 64-bit Linux, where the timings are obtained using the SUPERCOP toolkit for measuring the performance of cryptographic software (see [10]). The implementations labeled (b) are publicly available through [10] [4]. Both (a) and (b) perform a final modular inversion to ensure that the output point is in affine form: This is the standard setting when computing a Diffie–Hellman key exchange.

7.2. *Results*

Table 2 summarizes the performance and characteristics of various genus g curve implementations. For the security estimate, we assume that the fastest attacks possible are the "generic algorithms," where we specifically use the complexity of the Pollard rho [60] algorithm that exploits additional automorphisms [21,73]. If r is the largest prime factor of a group with #Aut automorphisms, we compute the security level s as $s = \log_2(\sqrt{\frac{\pi r}{2\#\mathrm{Aut}}})$[5]. We also indicate whether the implementation runs in constant time, an important step toward achieving side-channel resistance [42].

The implementations in the top part of the table are obtained from eBACS, except for [69] and [48]. The standardized NIST curves [69], one of which is at a lower security level, are both obtained from the benchmark program included in OpenSSL 1.0.1.[6] The implementation from [48] is not publicly available, but the authors gave us a precompiled binary which reported its own cycle count so that we could report numbers obtained in our test environment. All of these implementations were run on our hardware.

7.3. *Discussion*

The first thing to observe from Table 2 is that the standard NISTp-256 curve and the genus 2 curve "generic128" (see Sect. 4) offer the highest level of security. This "generic" genus 2 implementation is our slowest performing implementation, yet is it still 1.80 times faster than the NIST curve at the same security level. Interestingly, all our Kummer and four-dimensional GLV implementations manage to outperform the previous fastest genus 2 implementation [35]. Prior to this work, the fastest curve arithmetic reported on eBACS was due to Bernstein [4], while Longa and Sica [48] held the overall software speed record over prime fields. We note that the former implementation runs in constant time, while the latter does not. Even though our GLV implementations do not currently run in constant time, we note that they can be transformed into constant time implementations following, for instance, the techniques from [48]. Our approach (b) on the Kummer surface sets a new software speed record by break-

[4] The EBAT is available through http://hhisil.yasar.edu.tr/files/hisil20140312genus2.tar.gz, and a set of Magma files implementing scalar multiplications on Jacobians of genus 2 curves or on the associated Kummer surface are available through http://research.microsoft.com/en-us/downloads/ecd909b7-40af-4fd2-a215-b681e22d7084.

[5] Recent work [12] shows that when $\#\mathrm{Aut} > 2$, our estimates for the security level are slightly pessimistic.

[6] Note that to enable this implementation using the techniques described in [39], OpenSSL needs to be configured using "./Configure enable-ec_nistp_64_gcc_128."

ing the 125k cycle barrier for constant time implementations at the 128-bit security level.

We note that Table 2 reports implementations over prime fields only. For elliptic curves defined over quadratic extensions of large prime fields, Longa and Sica [48] report a non-constant time scalar multiplication in 91,000 cycles on the Sandy Bridge architecture, while their constant time version runs in 137,000 cycles. Over binary fields, Aranha *et al.* [3] perform a scalar multiplication on the Koblitz curve K-283 in 99,000 cycles on Sandy Bridge, while Oliveira et al. [57] recently announced a new speed record of 75,000 cycles on the same architecture. We note that both of these binary field implementations do not run in constant time.

With respect to the different arithmetic approaches from Sect. 3, we conclude that when using the prime $2^{127}-1$, the NIST-like approach is the way to go. In the more general comparison of $2^{128}-c_1$ versus $2^{64}\cdot(2^{63}-c_2)\pm 1$ for NIST-like and Montgomery-friendly primes, respectively, we found that the Montgomery-friendly primes outperform the former in practice. This was a surprising outcome and we hope that implementers of cryptographic schemes will consider this family of primes as well. The implementations (b) of “generic” and Kummer surface arithmetic highlight the practical advantage of the prime $2^{127}-1$ over the prime $2^{128}-c_1$: In both instances, the former is around 1.4 times faster than the latter.

7.4. *Generating Key Pairs With Precomputation*

Two cycle counts are reported for all of the implementations of Diffie–Hellman secret sharing benchmarked on eBACS [10]. The first is the “time to compute a shared secret,” which corresponds to the variable point scalar multiplications that we reported in Table 2. The second is the “time to generate a key pair," which corresponds to fixed-point scalar multiplications that allow precomputations on a known public generator. Our timings for the second case are reported in Table 3, where our fixed-point scalar multiplications employ the fixed-point comb method [47] and simultaneous addition technique [46]. In both settings, precomputed tables larger than 512 KB did not lower the cycle count. This is due to the size of the cache on our Intel Core i7, but this threshold size might be different on other platforms. We note that this technique (and the performance numbers in Table 3) only applies to the generic and GLV curves and that precomputation will not give rise to such drastic speedups in the case of the Kummer surface implementations.

Table 3. Performance timings in 10^3 cycles of $y^2=f(x)$, $\deg(f)=5$ with NIST-like reduction and precomputation.

Field char	Storage (KB)	10^3 cycles	Field char	Storage (KB)	10^3 cycles
$2^{127}-1$	64	53	$2^{128}-c$	64	81
	128	42		128	62
	256	36		256	53
	512	33		512	49
	1024	33		1024	49
	2048	33		2048	49

8. Kummer Chameleons

In this section, we explore curves that facilitate *both* efficient scalar multiplications on the Kummer surface and efficient scalar multiplications on the Jacobian using a GLV decomposition. Such curves give cryptographers the option of taking either route depending on the protocol at hand: For Diffie–Hellman protocols, working on the associated Kummer surface is the most efficient option, but if the pseudo-addition law on the Kummer surface is insufficient, the GLV method can be used on an associated curve. Since these curves can morph depending on the scenario, we call them *Kummer chameleons*.

We primarily focus on the two families that facilitate four-dimensional GLV decompositions. We start with the FKT family of curves to show an unfortunate drawback which prohibits us from using this Kummer/GLV duality over prime fields. We then move to the BK family of curves which does allow this duality in practice and provide some example Kummer chameleons in this case. For these special families, we also show the benefits of computing the Kummer surface parameters analytically (i.e., over $\mathbb{C}$). This approach tells us when we can (or cannot) expect to find practical Kummer parameters using the technique of extracting $\mathcal{K}$ from $\mathcal{C}_{\text{Ros}}$ in §5.2. It can additionally reveal when we are likely to find small surface constants, which guarantees solid speedups in practice. For an overview of computations involving the analytic Jacobian of a hyperelliptic curve, we refer to [71].

8.1. *Recognizing Kummer Parameters over* $\mathbb{C}$

We use an analytic approach to assist us in generating Kummer surfaces which are associated with a particular CM field. For each CM field, there is a collection of period matrices which correspond to the isomorphism classes of Jacobians of genus 2 curves with CM by that field, and thus with known possible group orders (see [71]). The theta functions can be evaluated at these period matrices, and approximations of the complex values of the associated theta constants can be used to recognize the minimal polynomials that they satisfy.

Although it can be difficult to analytically recognize the theta constants themselves, for special families it is often possible to recognize *quotients* of certain theta constants. In Tables 4 and 6, we give the minimal polynomials satisfied by all of the parameters required for the Kummer surface implementation for the FKT and BK families: The values E', F, G, H, y_0, z_0, t_0, y_0', z_0' and t_0' (as defined in Sect. 5). The coefficients of these minimal polynomials can be reduced modulo any prime p, and so for any p for which the polynomials have a consistent choice of roots modulo p, they can be used to define a Kummer surface over $\mathbb{F}_p$ such that the associated group order of $J_{\mathcal{C}}$ is known (from the CM field).

8.2. *The Kummer Surface of FKT Curves*

For curves of the form $y^2 = x^5 + ax$, the complex values (and corresponding minimal polynomials) of the required Kummer parameters are given in Table 4. We note that once we choose $i = \sqrt{-1}$ by sufficiently extending $\mathbb{F}_p$ (if necessary), all of the required constants are determined. Observe that two of the six surface constants that appear in

Table 4. Kummer parameters (and their minimal polynomials) over $\mathbb{C}$ for the FKT family.

$\mathcal{K}$ param.	E	F, G, H	y_0, t_0	z_0	y_0', t_0'	z_0'
Value $\in \mathbb{C}$	$17+31i$	$(3+i)/2$	1	$1-i$	$3+4i$	$-3-4i$
Min. poly.	$x^2-34x+1250$	$2x^2-6x+5$	$x-1$	x^2-2x+2	$x^2-6x+25$	$x^2+6x+25$

Table 5. Quotients appearing in the maps from $\mathcal{K}$ to the u-polynomial of $D \in J_{\mathcal{C}_{\text{Ros}}}$ for FKT curves.

u_x/u_t	u_y/u_t	u_z/u_t	u_x'/u_t'	u_y'/u_t'	u_z'/u_t'	d_x/d_t	d_y/d_t	d_z/d_t	u_t/d_t	u_t'/d_t
$-1-i$	$\frac{i-1}{2}$	1	$2i$	$-1-i$	$-1-i$	$\frac{i-1}{2}$	$-1-i$	1	i	$\frac{1-i}{2}$

each iteration of $\mathcal{K}(\texttt{SMUL})$ (of Algorithm 6) are 1, which immediately results in two fewer multiplications.

We further note that it is possible to recognize quotients of theta constants that appear in the maps from $\mathcal{K}$ to $J_{\mathcal{C}_{\text{Ros}}}$ in (4). In the case of FKT curves, Table 5 gives the values of all the quotients we need, which allows us to simplify the expressions in the map to the u-polynomial of a divisor $D \in J_{\mathcal{C}_{\text{Ros}}}$ as $u_0 = \frac{(2-2i)x-(1+i)y+2iz+2it}{(i-1)x-(2+2i)y+2z+2t}$, $u_1 = \frac{(2+2i)x-2y-2z+(1-i)t}{(i-1)x-(2+2i)y+2z+2t} - u_0 - 1$. Although the expressions for the v-polynomial expand to be more complicated, leaving them in factored form allows similar simplifications. The above maps take points on the Kummer surface points on $\mathcal{K}$ to points in $J_{\mathcal{C}_{\text{Ros}}}$ or $J_{\mathcal{C}'_{\text{Ros}}}$, where $\mathcal{C}_{\text{Ros}} : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu)$, and for which we can also recognize the Rosenhain invariants in $\mathbb{C}$ as $\lambda = (i+1)/2$, $\mu = i$ and $\nu = i+1$. Now, to reduce these values modulo p, we note that if $p \equiv 1 \pmod 4$, then $i = \sqrt{-1} \in \mathbb{F}_p$ and the Rosenhain model defined by those values is defined over $\mathbb{F}_p$. The curve $\mathcal{C} : y^2 = x^5 + ax$ can be rewritten as $y^2 = x(x-\alpha)(x+\alpha)(x-\alpha i)(x+\alpha i)$, where α is a non-trivial fourth root of $-a$. Clearly $\mathcal{C}$ and $\mathcal{C}_{\text{Ros}}$ can only be isomorphic over $\mathbb{F}_p$ if $\alpha \in \mathbb{F}_p$, which implies that $J_{\mathcal{C}}$ is isogenous over $\mathbb{F}_p$ to the product of two elliptic curves [24, Lemma 4]. Thus, $\mathcal{C}$ is not suitable for cryptographic applications in this case, since the group order of $J_{\mathcal{C}}$ is a product of factors of at most half the size of the total. If instead $p \equiv 3 \pmod 4$, then $i \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$, and from Table 4, it follows that the Kummer surface $\mathcal{K}$ is defined over $\mathbb{F}_{p^2}$, which destroys the arithmetic efficiency of the group law algorithms. Therefore, we conclude that the FKT family does not yield a secure and efficient (Gaudry-style) Kummer surface over prime fields.

8.3. *The Kummer Surface of BK Curves*

For curves of the form $y^2 = x^5 + b$, the minimal polynomials for the required Kummer parameters are given in Table 6. Since these polynomials have degree larger than two, writing down the correct root corresponding to each Kummer parameter becomes more involved. Furthermore, these polynomials tell us that we cannot expect any Kummer constants to automatically be small. Nevertheless, they do help us deduce when it is possible to find practical Kummer parameters. For example, t_0 is a root of $\Phi_5(-x^2)$, which does not have any roots in $\mathbb{F}_p$ when $p \equiv 11 \pmod{20}$, yet splits into linear factors

Table 6. Kummer parameters (and their minimal polynomials) over $\mathbb{C}$ for the Buhler–Koblitz family.

Kummer parameter	Minimal polynomial
E, F	$x^2 - 20x - 400,\ x^8 - 11x^6 + 46x^4 - 96x^2 + 121$
G, H	$x^8 - 11x^6 + 46x^4 - 96x^2 + 121,\ x^2 + x - 1$
y_0, z_0	$x^4 - x^3 + x^2 - x + 1,\ x^8 - 4x^6 + 6x^4 + x^2 + 1$
t_0, y_0'	$x^8 - x^6 + x^4 - x^2 + 1,\ x^4 - 16x^3 + 46x^2 - 16x + 1$
z_0', t_0'	$25x^8 - 100x^7 + 460x^6 + 580x^5 + 286x^4 + 36x^3 - 4x^2 - 4x + 1$

when $p \equiv 1 \pmod{20}$. In fact, all of the polynomials in Table 6 split into linear factors in $\mathbb{F}_p$ for $p \equiv 1 \pmod{20}$; this agrees with our experiments which always extracted working Kummer parameters for BK curves when $p \equiv 1 \pmod{20}$ and always failed to do so when $p \equiv 11 \pmod{20}$.

The only minor drawback for the Kummer surface associated with the BK family is that, for primes congruent to 1 modulo 5, if the 2-torsion of $J_{\mathcal{C}}$ or $J_{\mathcal{C}'}$ is defined over $\mathbb{F}_p$, then 5 divides at least one of the two group orders. Hence, even in the best case the two group orders have cofactors of 16 and 80, which means either the curve or its twist will be around 1 bit less secure than the other. In this case, generators on the Kummer surface should be chosen which map back to the curve with cofactor 16. We give two examples of these Kummer chameleons below.

8.3.1. *BK Kummer Chameleon over a* 127*-bit Prime Field*

Let $p = 2^{64} \cdot (2^{63} - 1035383) + 1$, and let $\mathcal{C}/\mathbb{F}_p : y^2 = x^5 + 7^5$, the quadratic twist $\mathcal{C}'$ of which can be written as $\mathcal{C}' : y^2 = 7(x^5 + 7^5)$. The group orders are $\#J_{\mathcal{C}} = 2^4 \cdot r$ and $\#J_{\mathcal{C}'} = 2^4 \cdot 5 \cdot r'$, where

$$r=18092513943326593532100447217799657167771995357680607589566157707118911 00371,$$
$$r'=3618502788665318706446574743757939080623325651725094314883591277782613310 91,$$

are 250- and 248-bit primes, respectively. A degree 5 Rosenhain model $\mathcal{C}_{\mathrm{Ros}}$ isomorphic to $\mathcal{C}$ is given by the constants

$$\lambda=106611868196659112931082761926395923 33,\quad \mu=414466078838781044746547282339645840 14,$$
$$\nu=127213099918419761245342755241553487702,$$

for which one choice of the squared fundamental theta constants is

$$a^2=84491026685045794598730782355659170339,\ b^2=33186841131699432035082366865570982234,$$
$$c^2=85766492034541656770688027007588903688,\ d^2=1.$$

The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'=139180060863318125490801991597453057 70,\quad F=187625840664800037601345952054852599 83,$$
$$G=137599581973583773482954213814600348679,\ H=85766492034541656770688027007588903688.$$

A generator on $\mathcal{K}$ with order r that maps back to $J_{\mathcal{C}_{\text{Ros}}}$ is

$$P{=}[2](1,1,1,86011366689699880330600293725419043935).$$

8.3.2. *BK Kummer Chameleon over a* 128*-bit Prime Field*

Let $p = 2^{128} - 12091815$, and let $\mathcal{C}/\mathbb{F}_p : y^2 = x^5 + 17^5$, the quadratic twist $\mathcal{C}'$ of which can be written as $\mathcal{C}' : y^2 = 17(x^5 + 17^5)$. The group orders are $\#J_{\mathcal{C}} = 2^4 \cdot r$ and $\#J_{\mathcal{C}'} = 2^4 \cdot 5 \cdot r'$, where

$$r{=}7237005577332262215080031836658877787354274212851606663878680202836635096291$$
$$r'{=}1447401115466452442573268257885216407322324444568217420589854251119792109811,$$

are 253- and 250-bit primes, respectively. A degree 5 Rosenhain model $\mathcal{C}_{\text{Ros}}$ isomorphic to $\mathcal{C}$ is given by the constants

$$\lambda{=}69750747073243793503741945404989703593, \quad \mu{=}150179622307743074988869416441414313355,$$
$$\nu{=}227997816177602308074873451087733623676,$$

for which one choice of the squared fundamental theta constants is

$$a^2{=}311378520185987879249636451466710084857, \; b^2{=}194692299483499628396825108659644530161,$$
$$c^2{=}77818193869859233086004034646319310321, \quad d^2{=}1.$$

The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'{=}195234409713430807866582263199361727876, \; F{=}142475655262409749610168226227930172175,$$
$$G{=}25789434559921498757356883420864617479, \quad H{=}77818193869859233086004034646319310321.$$

A generator on $\mathcal{K}$ with order r that maps back to $J_{\mathcal{C}_{\text{Ros}}}$ is

$$P{=}[2](1,1,-1,330547215562037048968388688956419952626).$$

8.4. *Kummer Chameleons with Two-dimensional GLV*

Although we have focused on two families of genus 2 curves that offer four-dimensional GLV over prime fields, there are many more families that offer two-dimensional GLV [33, 44, 67]. We especially mention the families studied in [33, §4.3-4.4], which might be particularly attractive since the techniques in [33] make it practical to find twist-secure instances over $\mathbb{F}_p$ with $p = 2^{127} - 1$.

8.5. *GLV on the Kummer Surface?*

Gaudry [30] observed that there is a certain class of Kummer surfaces that come equipped with a simple endomorphism on the Kummer surface itself. If the squared fundamental theta constants are related by $b^2 = a^2 - c^2 - d^2$, then the doubling step in Algorithm 5 can be seen as a map $\phi : \mathcal{K} \to \mathcal{K}$ composed with itself, which means $\phi^2 = [2]$, and we must have that $\phi = [\sqrt{2}]$ on $\mathcal{K}$. It is natural to go looking for these Kummers within families of genus 2 curves that have RM by $\sqrt{2}$, whether the RM is *explicit* and *efficiently computable*[7] on the Jacobian or not. One such instance comes from the curves defined over the rationals by van Wamelen [70], the second example of which has CM by the quartic CM field $\mathbb{Q}(\sqrt{-2+\sqrt{2}})$. Over the two forms of prime field we prefer, we used the CM method to find many instances of these curves, and indeed we were always able to extract several Kummer parameterizations with $b^2 = a^2 - c^2 - d^2$: Two twist-secure examples of these are given at the end of this subsection. The question now becomes: *can we exploit this endomorphism and perform GLV on the Kummer surface itself?*

Since we are limited to pseudo-additions on $\mathcal{K}$, the standard GLV technique of merging the mini-scalars and proceeding with a standard addition chain does not apply in this scenario. In this case, to compute $[k]P$ from P and $\phi(P)$, we need a *two-dimensional differential-addition chain*. Such chains have already been well studied because of their application to multi-exponentiations in Montgomery coordinates [5,53,66]: In the two-dimensional case, this means computing $[m]P + [n]Q$ from the three starting values P, Q, and $P - Q$. This brings forward the main hurdle in achieving GLV on the Kummer surface, in that after computing $Q = \phi(P)$, we only have two of the three values that are needed to start the addition chain. In order to proceed we need either $Q + P$ or $Q - P$ on $\mathcal{K}$, which equivalently means we need an explicit and efficient way of computing the map $\phi^+ = \phi + [1]$ or the map $\phi^- = \phi - [1]$ on $\mathcal{K}$.

In estimating the performance gain that finding these maps would offer, we must mention two caveats. Firstly, we note that since the input difference into the pseudo-addition algorithm is no longer constant throughout the routine, we suffer an extra 6 $\mathbb{F}_p$ multiplications each time it is called—the inverses that were precomputed are now projectively scaled to on-the-fly multiplications. Furthermore, we are no longer performing additions and doublings concurrently throughout, and we therefore lose the benefit of the constant overlap between them. Nevertheless, using either of the chains given in [5,53] would mean performing less than half the total number of doubling and pseudo-addition operations than in the standard Kummer case, and this is more than enough motivation to pose the problem of finding a setting where ϕ^+ and/or ϕ^- are efficiently computable.

8.5.1. *Van Wamelen "$[\sqrt{2}]$-on-$\mathcal{K}$" Curve over a* 127*-bit Prime Field.*

Let p be the Montgomery-friendly prime $p = 2^{64} \cdot (2^{63} - 107125) + 1$. The group orders of the Jacobian of $\mathcal{C}/\mathbb{F}_p : y^2 = -x^5 + 3x^4 + 2x^3 - 6x^2 - 3x + 1$ and its twist $\mathcal{C}'$ are given as $\#J_{\mathcal{C}} = 2^5 \cdot r$ and $\#J_{\mathcal{C}'} = 2^4 \cdot r'$, where

$$r = 904625697166511763040116799547618814004487139266761754879956154884593483799,$$
$$r' = 1809251394333023526590934270642281340029094439228748499659037824507445397703,$$

are 249- and 250-bit primes, respectively. An isomorphic Rosenhain model $\mathcal{C}_{\text{Ros}}$ of $\mathcal{C}$ is defined by the triple

[7] These terms are made precise in [33, Def. 1,2].

$$\lambda{=}168171229223321177769186812485517969948,\ \mu{=}9417430203573952280833540215738464957,$$
$$\nu{=}158753799019747225488353272269779504992,$$

for which one choice of the squared fundamental theta constants is

$$a^2{=}150321345934746312135040529601365675926,\ c^2{=}96985692613693010230188339033665775936,$$

with $d^2 = 1$ and $b^2 = a^2 - c^2 - d^2$. The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'{=}16, \qquad F{=}112722080887356168648583571057476016874,$$
$$G{=}39639675051441886978375755957603131604,\ H{=}133526895531650151065292246583602218570.$$

A compact generator on $\mathcal{K}$ is

$$P{=}[2](1,1,-1,129889658466772916887665811107285236509).$$

8.5.2. *Van Wamelen "[$\sqrt{2}$]-on-$\mathcal{K}$" Curve over a* 128*-bit Prime Field.*

Let p be the prime $p = 2^{128} - 6404735$. The group orders of the Jacobian of $\mathcal{C}/\mathbb{F}_p$: $y^2 = -x^5 + 3x^4 + 2x^3 - 6x^2 - 3x + 1$ and its twist $\mathcal{C}'$ are given as $\#J_{\mathcal{C}} = 2^5 \cdot r$ and $\#J_{\mathcal{C}'} = 2^4 \cdot r'$, where

$$r{=}3618502788666131107463347962322673561312709571881084013989949351691450367367,$$
$$r'{=}7237005577332262213019677201440096504580481109273330370637841367829704337687,$$

are both 252-bit primes. An isomorphic Rosenhain model $\mathcal{C}_{\mathrm{Ros}}$ of $\mathcal{C}$ is defined by the triple

$$\lambda{=}338853613323961976541294628448051393997,\ \mu{=}161826994076915014352261046382941880808,$$
$$\nu{=}177026619247046962189033582065109513190,$$

for which one choice of the squared fundamental theta constants is

$$a^2{=}186055185429089423029499828889903742105,\ c^2{=}293311051702477990348906969535446463740,$$

with $d^2 = 1$ and $b^2 = a^2 - c^2 - d^2$. The corresponding Kummer surface $\mathcal{K}$ is parameterized by

$$E'{=}16, \qquad F{=}308566990761521795503609453351512063008,$$
$$G{=}308454362983698080867749557083716129230,\ H{=}293367365591389847666836917669344430628.$$

A compact generator on $\mathcal{K}$ is

$$P{=}[2](1,1,-1,328931498180381025899390285257510062396).$$

9. Conclusions

We have given a taxonomy of the state of the art in genus 2 arithmetic over prime fields, with respect to its application in public-key cryptography. We studied two different approaches to achieve fast modular arithmetic and implemented these techniques in three settings: on "generic" genus 2 curves, on special genus 2 curves facilitating two- and four-dimensional GLV decompositions, and on the Kummer surface proposed by Gaudry [29]. Furthermore, we presented *Kummer chameleons*; curves which allow fast arithmetic on the Kummer surface as well as efficient arithmetic on the Jacobian that results from a GLV decomposition. Ultimately, we highlighted the practical benefits of genus 2 curves with our Kummer surface implementation—this sets a new software speed record at the 128-bit security level for computing constant time scalar multiplications compared to all previous elliptic curve and genus 2 implementations.

Acknowledgements

We wish to thank Pierrick Gaudry for his Kummer help when this project began, Dan Bernstein and Tanja Lange for several fruitful discussions during the preparation of this work, Patrick Longa for his advice on optimizing the GLV routines and extensive comments on this work, Michael Naehrig for proofreading early versions of this paper, and the anonymous Eurocrypt reviewers for their useful comments.

We make note of several works that have appeared since this paper was submitted. The follow-up work in [7] pointed out that our online Kummer implementation contained a mistake which might leak secret information to side-channel adversaries. We updated the code accordingly(http://hhisil.yasar.edu.tr/files/hisil20140312genus2.tar.gz) and the subsequent performance numbers are stated in Table 2. We would like to thank the authors for finding this mistake. In addition, the authors of [7] tailored the use of vector instructions to give a solid boost to the performance of the Kummer surface routine described in Sect. 5, and subsequently, their accompanying implementation currently offers the fastest constant-time scalar multiplications over large prime fields. Finally, the overall Diffie–Hellman speed record at the 128-bit security level was recently claimed by Aranha et al. [56], who use a binary field elliptic curve equipped with an endomorphism to achieve highly efficient, constant-time scalar multiplications in around 60,000 clock cycles on the Haswell architecture.

References

[1] T. Acar, D. Shumow, Modular reduction without pre-computation for special moduli. Technical report, Microsoft Research, 2010

[2] L. Adleman, J. DeMarrais, M. Huang, A subexponential algorithm for discrete logarithms over hyperelliptic curves of large genus over GF(q). *Theor. Comput. Sci.***226**(1–2), 7–18 (1999)

[3] D.F. Aranha, A. Faz-Hernández, J. López, F. Rodríguez-Henríquez, Faster implementation of scalar multiplication on Koblitz curves, in A. Hevia, G. Neven,editors, *LATINCRYPT*. Lecture Notes in Computer Science, vol. 7533 (Springer, 2012), pp. 177–193

[4] D.J. Bernstein, Curve25519: New Diffie–Hellman speed records,in M. Yung, Y. Dodis, A. Kiayias, T. Malkin, editors, *Public Key Cryptography—PKC 2006*. Lecture Notes in Computer Science, vol. 3958 (Springer, Heidelberg, 2006), pp. 207–228

[5] D.J. Bernstein, Differential addition chains. URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf, February 2006

[6] D.J. Bernstein, Elliptic vs. Hyperelliptic, part I. Talk at ECC (slides at http://cr.yp.to/talks/2006.09.20/slides.pdf,), September 2006

[7] D.J. Bernstein, C. Chuengsatiansup, T. Lange, P. Schwabe, Kummer strikes back: new DH speed records. Cryptology ePrint Archive, Report 2014/134, 2014. http://eprint.iacr.org/

[8] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang, High-speed high-security signatures, in B. Preneel, T. Takagi, editors, *CHES*. Lecture Notes in Computer Science, vol. 6917 (Springer, 2011), pp. 124–142

[9] D.J. Bernstein, T. Lange, Analysis and optimization of elliptic-curve single-scalar multiplication, in G.L. Mullen, D. Panario, I.E. Shparlinski, editors, *Finite Fields and Applications*. *Contemporary Mathematics Series*, vol. 461 (American Mathematical Society, 2008), pp. 1–19

[10] D.J. Bernstein, T. Lange (editors), eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to, accessed 4 October 2012

[11] J.W. Bos, High-performance modular multiplication on the Cell processor, in M.A. Hasan, T. Helleseth, editors, *Arithmetic of Finite Fields - WAIFI 2010*. Lecture Notes in Computer Science, vol. 6087 (Springer, Heidelberg, 2010), pp. 7–24

[12] J.W. Bos, C. Costello, A. Miele, Elliptic and hyperelliptic curves: A practical security analysis, in H. Krawczyk, editor, *Public Key Cryptography—PKC 2014*. Lecture Notes in Computer Science, vol. 8383 (Springer, 2014), pp. 203–220

[13] J.W. Bos, M.E. Kaihara, T. Kleinjung, A.K. Lenstra, P.L. Montgomery, Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. *Int. J. Appl. Cryptogr.***2**(3), 212–228 (2012)

[14] A. Brauer, On addition chains. *Bull. Am. Math. Soc.***45**, 736–739 (1939)

[15] M. Brown, D. Hankerson, J.López, A. Menezes, Software implementation of the NIST elliptic curves over prime fields, in D. Naccache, editor, *CT-RSA*. Lecture Notes in Computer Science, vol. 2020 (Springer, Heidelberg, 2001), pp. 250–265

[16] J. Buhler, N. Koblitz, Lattice basis reduction, Jacobi sums and hyperelliptic cryptosystems. *Bull. Aust. Math. Soc.***58**(1), 147–154 (1998)

[17] D.V. Chudnovsky, G.V. Chudnovsky, Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Adv. Appl. Math.***7**, 385–434 (1986)

[18] R. Cosset, Factorization with genus 2 curves. *Math. Comput.***79**(270), 1191–1208 (2010)

[19] C. Costello, K. Lauter, Group law computations on Jacobians of hyperelliptic curves, in A. Miri, S. Vaudenay, editors, *Selected Areas in Cryptography*. Lecture Notes in Computer Science, vol. 7118 (Springer, 2011), pp. 92–117

[20] C. Diem, On the discrete logarithm problem in class groups of curves. *Math. Comput.***80**, 443–475 (2011)

[21] I.M. Duursma, P. Gaudry, F. Morain, Speeding up the discrete log computation on curves with automorphisms, in K.-Y. Lam, E. Okamoto, C. Xing, editors, *Asiacrypt 1999*. Lecture Notes in Computer Science, vol. 1716 (Springer, Heidelberg, 1999), pp. 103–121

[22] K. Eisentrager, K. Lauter, A CRT algorithm for constructing genus 2 curves over finite fields. *AGCT-11* (2007)

[23] A. Enge, Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. *Math. Comput.***71**, 729–742 (2002)

[24] E. Furukawa, M. Kawazoe, T. Takahashi, Counting points for hyperelliptic curves of type $\mathrm{y}^2 = \mathrm{x}^5 + \mathrm{ax}$ over finite prime fields, in M. Matsui, R.J. Zuccherato, editors, *Selected Areas in Cryptography*. Lecture Notes in Computer Science, vol. 3006 (Springer, 2003), pp. 26–41

[25] S.D. Galbraith, X. Lin, M. Scott, Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptol.***24**(3), 446–469 (2011)

[26] R.P. Gallant, R.J. Lambert, S.A. Vanstone, Faster point multiplication on elliptic curves with efficient endomorphisms, in J. Kilian, editor, *CRYPTO*. Lecture Notes in Computer Science, vol. 2139 (Springer, 2001), pp. 190–200

[27] P. Gaudry, An algorithm for solving the discrete log problem on hyperelliptic curves. *Eurocrypt*, **1807**, 19–34 (2000)

[28] P. Gaudry, Algorithmique des courbes hyperelliptiques et applications à la cryptologie. PhD thesis, École polytechnique. http://www.lix.polytechnique.fr/Labo/Pierrick.Gaudry/publis/ (2000)

[29] P. Gaudry, Fast genus 2 arithmetic based on theta functions. *J. Math. Cryptol. JMC***1**(3), 243–265 (2007)

[30] P. Gaudry, Genus 2 formulae based on Theta functions and their implementation. Talk at ECC http://mathsci.ucd.ie/gmg/ECC2007Talks/ecc07-gaudry2.pdf, September 2007

[31] P. Gaudry, Personal communication (2011)

[32] P. Gaudry, T. Houtmann, D.R. Kohel, C. Ritzenthaler, A. Weng, The 2-adic CM method for genus 2 curves with application to cryptography, in X. Lai, K. Chen, editors, *ASIACRYPT*. Lecture Notes in Computer Science, vol. 4284 (Springer, 2006), pp. 114–129

[33] P. Gaudry, D.R. Kohel, B.A. Smith, Counting points on genus 2 curves with real multiplication, in D.H. Lee, X. Wang, editors,*ASIACRYPT*. Lecture Notes in Computer Science, vol. 7073 (Springer, 2011), pp. 504–519

[34] P. Gaudry, É. Schost, Genus 2 point counting over prime fields. *J. Symb. Comput.***47**(4), 368–400 (2012)

[35] P. Gaudry, E. Thomé, The mp$\mathbb{F}_q$ library and implementing curve-based key exchanges, in *Software Performance Enhancement for Encryption and Decryption—SPEED 2007*, pp. 49–64 (2007). www.loria.fr/~gaudry/publis/mpfq.pdf

[36] M. Hamburg, Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/

[37] H. Hisil, K.K.-H. Wong, G. Carter, E. Dawson, Twisted Edwards curves revisited, in J. Pieprzyk, editor, *Asiacrypt 2008*. Lecture Notes in Computer Science, vol. 5350 (Springer, Heidelberg, 2008), pp. 326–343

[38] B.S. Kaliski Jr, The Montgomery inverse and its applications. *IEEE Trans. Comput.***44**(8), 1064–1065 (1995)

[39] E. Käsper, Fast elliptic curve cryptography in OpenSSL, in G. Danezis, S. Dietrich, K. Sako, editors, *Financial Cryptography Workshops*. Lecture Notes in Computer Science, vol. 7126 (Springer, 2012) pp. 27–39

[40] M. Knežević, F. Vercauteren, I. Verbauwhede, Speeding up bipartite modular multiplication, in M. Hasan, T. Helleseth, editors, *Arithmetic of Finite Fields - WAIFI 2010*. Lecture Notes in Computer Science, vol. 6087 (Springer, Berlin / Heidelberg, 2010), pp. 166–179

[41] N. Koblitz, Elliptic curve cryptosystems. *Math. Comput.***48**(177), 203–209 (1987)

[42] P.C. Kocher, Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems, in N. Koblitz, editor, *Crypto 1996*. Lecture Notes in Computer Science, vol. 1109 (Springer, Heidelberg, 1996), pp. 104–113

[43] D.R. Kohel, Databases for Elliptic Curves and Higher Dimensional Analogues (Echidna). http://echidna.maths.usyd.edu.au/kohel/dbs/

[44] D.R. Kohel, B.A. Smith, Efficiently computable endomorphisms for hyperelliptic curves, in F. Hess, S. Pauli, M.E. Pohst, editors, *ANTS*. Lecture Notes in Computer Science, vol. 4076 (Springer, 2006), pp. 495–509

[45] A.K. Lenstra, Generating RSA moduli with a predetermined portion, in K. Ohta, D. Pei, editors, *Asiacrypt'98*. Lecture Notes in Computer Science, vol. 1514 (Springer, Berlin/Heidelberg, 1998), pp. 1–10

[46] C.H. Lim, H.S. Hwang, Speeding up elliptic scalar multiplication with precomputation, in J. Song, editor, *Information Security and Cryptology—ICISC'99*. Lecture Notes in Computer Science, vol. 1787 (Springer, 2000), pp. 102–119

[47] C.H. Lim, P.J. Lee, More flexible exponentiation with precomputation, in Y. Desmedt, editor, *CRYPTO*. Lecture Notes in Computer Science, vol. 839 (Springer, 1994), pp. 95–107

[48] P. Longa, F. Sica, Four-dimensional Gallant–Lambert–Vanstone scalar multiplication, in X. Wang, K. Sako, editors, *Asiacrypt 2012*. Lecture Notes in Computer Science, vol. 7658 (Springer, 2012), pp. 718–739

[49] J.-F. Mestre, Couples de jacobiennes isogenes de courbes hyperelliptiques. Preprint, arXiv http://arxiv.org/abs/0902.3470, or see http://www.lix.polytechnique.fr/smith/Mestre--families.pdf (2009)

[50] V.S. Miller, Use of elliptic curves in cryptography, in H.C. Williams, editor, *Crypto 1985*. Lecture Notes in Computer Science, vol. 218 (Springer, Heidelberg, 1986), pp. 417–426

[51] P.L. Montgomery, Modular multiplication without trial division. *Math. Comput.***44**(170), 519–521 (1985)

[52] P.L. Montgomery, Speeding the Pollard and elliptic curve methods of factorization. *Math. Comput.***48**(177), 243–264 (1987)

[53] P.L. Montgomery, Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via lucas chains. ftp://ftp.cwi.nl/pub/pmontgom/Lucas.ps.gz (1992)

[54] F. Morain, J. Olivos, Speeding up the computations on an elliptic curve using addition–subtraction chains. *Inform. Théor. Appl. Theor. Inform. Appl.***24**, 531–544 (1990)

[55] National Security Agency, Fact sheet NSA Suite B Cryptography. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml (2009)

[56] T. Oliveira, J.López, D.F. Aranha, F. Rodríguez-Henríquez, Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptogr. Eng.***4**(1), 3–17 (2014)

[57] T. Oliveira, F. Rodríguez-Henríquez, J.López, New timings for scalar multiplication using a new set of coordinates. Rump session talk at ECC 2012 October 2012 (2012)

[58] Y.-H. Park, S. Jeong, J. Lim, Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms, in L.R. Knudsen, editor, *EUROCRYPT*. Lecture Notes in Computer Science, vol. 2332 (Springer, 2002), pp. 197–208

[59] J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comput.***55**(192), 745–763 (1990)

[60] J.M. Pollard, Monte Carlo methods for index computation (mod p). *Math. Comput.***32**(143), 918–924 (1978)

[61] A. Scholz, Aufgabe 253. *Jahresbericht der deutschen Mathematiker-Vereingung***47**, 41–42 (1937)

[62] R. Schoof, Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comput.***44**(170), 483–494 (1985)

[63] D. Shanks, Class number, a theory of factorization, and genera, in D.J. Lewis, editor, *Symposia in Pure Mathematics*, vol. 20 (American Mathematical Society, 1971), pp. 415–440

[64] N.P. Smart, S. Siksek, A fast Diffie–Hellman protocol in genus 2. *J. Cryptol.***12**(1), 67–73 (1999)

[65] J.A. Solinas, Generalized Mersenne numbers. Technical Report CORR 99–39, Centre for Applied Cryptographic Research, University of Waterloo (1999)

[66] M. Stam, *Speeding up Subgroup Cryptosystems*. PhD thesis, Technische Universiteit Eindhoven, May 2003 (2003)

[67] K. Takashima, A new type of fast endomorphisms on Jacobians of hyperelliptic curves and their cryptographic application. *IEICE Trans.***89-A**(1), 124–133 (2006)

[68] E.G. Thurber, On addition chains $l(mn) \leq l(n) - b$ and lower bounds for $c(r)$. *Duke Math. J.***40**, 907–913 (1973)

[69] U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-3, 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[70] P.B. van Wamelen, Examples of genus two CM curves defined over the rationals. *Math. Comput.***68**(225), 307–320 (1999)

[71] P.B. van Wamelen, Computing with the analytic Jacobian of a genus 2 curve, in W. Bosma, J. Cannon, M. Bronstein, A.M. Cohen, H. Cohen, D. Eisenbud, B. Sturmfels, editors, *Discovering Mathematics with Magma*. Algorithms and Computation in Mathematics, vol. 19 (Springer, Berlin Heidelberg, 2006), pp. 117–135

[72] A. Weng, Constructing hyperelliptic curves of genus 2 suitable for cryptography. *Math. Comput.***72**(241), 435–458 (2003)

[73] M.J. Wiener, R.J. Zuccherato, Faster attacks on elliptic curve cryptosystems, in S. Tavares, H. Meijer, editors, *Selected Areas in Cryptography—(SAC) 1998*. Lecture Notes in Computer Science, vol. 1556 (Springer New York, 1999), pp. 190–200