Fast Cryptography in Genus 2 (Two is Greater than One)

Joppe W. Bos[1], Craig Costello[1⋆], Huseyin Hisil[2], and Kristin Lauter[1]

[1] Microsoft Research, Redmond, USA
[2] Yasar University, Izmir, Turkey

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 4-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 1980's, elliptic curve cryptography [40, 49] 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, 26, 22, 19]. Such attacks are not known, however, for *genus* 2 *curves*. In [28], 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 work over special prime fields, where the underlying field 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 [54].

⋆ 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 based on an earlier article: Fast Cryptography in Genus 2, EUROCRYPT, LNCS, Vol. 7881, pp. 194-210, , 10.1007/978-3-642-38348-9_12.

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 [33], 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 [47]. As an additional bonus, our implementations on the Kummer surface inherently run in constant-time, which is one of the major steps towards achieving a side-channel resistant implementation [41]. 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 for genus 1 curves, so higher dimensional scalar decomposition is possible without passing to an extension field [25, 24]. For prime fields we implement 4-dimensional GLV decompositions on Buhler-Koblitz (BK) curves [15] and on Furukawa-Kawazoe-Takahashi (FKT) curves [23], 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 [62] and another based on the special form of the prime when using Montgomery multiplication [50].

In addition, we put forward a multi-faceted 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 the Kummer surface, but if a more complicated protocol requires further group operations, one has the option to instead exploit a 4-dimensional GLV implementation using the same curve.

The paper is organized as follows. In Section 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 Section 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 Section 2.1. In Section 2.2 we outline the CM method, which is used several times in this work to generate secure curves. In Section 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 $C : y^2 = f(x)$, where $f(x)$ has degree 5 or 6 and has no double roots. We call C a *real* hyperelliptic curve if the degree of f is 6, and if such an $f(x)$ has a rational root in K, then we can birationally transform the curve so that f has degree 5 instead, in which case we say 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 denote this group by $\mathrm{Jac}(C)$, the Jacobian of C, which consists of degree zero divisors on the curve modulo principal divisors. For genus 2 hyperelliptic curves, each class has a unique *reduced* representative divisor consisting of at most two rational points (which are not reflections of each other) minus the point(s) at infinity. General Jacobian elements can 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. Throughout this paper we use the *Mumford representation* of general divisors $D = (x^2 + u_1x + u_0, v_1x + v_0) \in \mathrm{Jac}(C)$, and instead write $D = (u_1, u_0, v_1, v_0)$. This avoids confusion when x and y are used as two of the Kummer coordinates in Section 5. When working in homogeneous projective space, we write such divisors 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 C (over an underlying field) and using the classical Schoof-Pila [60, 57] algorithm to compute $\#\mathrm{Jac}(C)$, 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 [33] presented a fast version of the general Schoof-Pila algorithm that manages to compute the order of the Jacobian corresponding to any such a curve in around 1000 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 to 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 [32] 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 special prime fields that facilitate fast field arithmetic. The CM method works as follows. For a smooth, projective, irreducible genus 2 curve, C, over a prime field $\mathbb{F}_p$ with ordinary Jacobian $\mathrm{Jac}(C)$, 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 $\mathfrak{p}\overline{\mathfrak{p}} = p$. 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 $\mathrm{End}(\mathrm{Jac}(C)) \cong \mathcal{O}_K$) correspond to generators 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 K to ensure that $|\pi| = \sqrt{p}$. Since $\#\mathrm{Jac}(C)(\mathbb{F}_p) = (1 - \pi)(1 - \overline{\pi})(1 - \pi')(1 - \overline{\pi}')$, 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 2 possible group orders when $K \neq \mathbb{Q}(\zeta_5)$ is Galois cyclic and 4 possible group orders when K is non-Galois (see [21, Proposition 4] for the possibilities).

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 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 [70], the Chinese remainder theorem (CRT) method [21], and the p-adic method [31]. All of the class polynomials we used in this work were taken from Kohel's comprehensive Echidna database [42]. Upon computing the Igusa invariants, we can then reconstruct the curve $C/\mathbb{F}_p$ using the Mestre-Cardona-Quer algorithm [48].

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 C is not important, which is the case when searching for "generic" curves (see Section 4.2) and for curves facilitating arithmetic on the Kummer surface (see Section 5.5). Alternatively, we use the second approach when we need either a certain defining equation for C (e.g. the GLV curves in Section 6.2), or if we need to fix a particular CM field (e.g. the van Wamelen curves in Section 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* [59] and a typical optimization to lower the number of point additions is using *windows* [13] 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* [66] 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[3], reducing the total number of group operations even further; this is called the *signed windows* approach [53]. 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

[3] When referring to group elements, the term 'point' becomes 'divisor' in the case of hyperelliptic curves, but remains as 'point' for Kummer surface arithmetic in Section 5.

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 [51], 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 [67], 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. [14], and in [11] a comparison is made between the performance when using Montgomery multiplication [50] 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. Section 4 and Section 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 also 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 [62], which is used to implement fast arithmetic modulo the NIST primes [67]. 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 [50] 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 all $i \in \mathbb{Z}$ such that $0 \leq i < n$, 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 bounded as $0 \leq C < 2 \cdot p$, 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 [44] 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 [39, 1, 35] 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.

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 $\tilde{a} = a 2^{bn} \bmod p$ 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 [12] (which uses the binary version of the Euclidean algorithm from [37]) 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 $2^t \bmod 2^{127} - 1 = 2^{t \bmod 127}$.

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 technique.

4.1 Explicit formulas

We make use of the fast formulas for arithmetic on imaginary quadratic curves from [18], which employ homogeneous projective coordinates, and focus on reducing the total number of multiplications in projective point doublings, point additions and mixed additions.[4] Due to the small size of our fields, the cost of modular addition and subtraction compared to modular multiplication is relatively high. Hence, we optimize the formulas from [18] 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 $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$, and count multiplications by the f_i as full multiplications, unless they are zero.[5] Letting $\mathbf{m}$, $\mathbf{s}$ and $\mathbf{a}$ 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 Section 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 Section 2.3).

4.2 Curves

To find "generic" curves for comparison against the GLV and Kummer techniques, we searched Kohel's Echidna database [42] with two fixed primes that facilitate our chosen techniques for field arithmetic. We terminated the search when we found curves with Jacobians of prime

[4] Note that the formulas to compute the projective doubling from [18] can be sped up since the first multiplication to compute UU is redundant.

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

Algorithm 1 Projective doubling for general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

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

$U_0'' \leftarrow U_0 \cdot Z$, $t_1 \leftarrow Z^2$,
$t_2 \leftarrow U_1^2$, $t_3 \leftarrow 2 \cdot t_2$,
$t_4 \leftarrow 2 \cdot U_0''$, $t_5 \leftarrow t_3 + t_4$,
$t_5 \leftarrow t_5 \cdot U_1$, $t_6 \leftarrow V_1^2$,
$t_7 \leftarrow f_2 \cdot t_1$, $t_6 \leftarrow t_7 - t_6$,
$t_6 \leftarrow t_6 \cdot Z$, $t_6 \leftarrow t_6 + t_5$,
$t_1 \leftarrow f_3 \cdot t_1$, $t_1 \leftarrow t_1 + t_2$,
$t_4 \leftarrow t_1 - t_4$, $t_4 \leftarrow t_4 + t_3$,
$V_0'' \leftarrow V_0 \cdot Z$, $t_1 \leftarrow U_1 \cdot V_1$,
$t_2 \leftarrow 2 \cdot t_1$, $t_1 \leftarrow t_1 + V_0''$,
$t_2 \leftarrow t_2 - V_0''$, $t_3 \leftarrow t_3 + U_0''$,
$t_3 \leftarrow V_1 \cdot t_3$, $t_5 \leftarrow t_3 \cdot t_4$,
$t_7 \leftarrow t_6 \cdot t_2$, $t_5 \leftarrow t_5 - t_7$,
$t_6 \leftarrow t_6 \cdot V_1$, $t_4 \leftarrow t_4 \cdot t_1$,
$t_4 \leftarrow t_4 - t_6$, $t_3 \leftarrow t_3 \cdot V_1$,
$t_1 \leftarrow t_1 \cdot t_2$, $t_3 \leftarrow t_3 - t_1$,
$t_1 \leftarrow t_5 \cdot t_4$, $t_2 \leftarrow t_3 \cdot t_4$,
$t_4 \leftarrow t_4^2$, $t_6 \leftarrow U_0'' \cdot t_4$,
$t_7 \leftarrow t_4 \cdot Z$, $t_4 \leftarrow t_4 \cdot U_1$,
$t_3 \leftarrow 2 \cdot t_3$, $t_3 \leftarrow t_3^2$,
$t_3 \leftarrow t_3 \cdot Z$, $t_2 \leftarrow 2 \cdot t_2$,
$U_0'' \leftarrow t_2 \cdot Z$, $V_1'' \leftarrow V_1 \cdot U_0''$,
$V_0'' \leftarrow V_0'' \cdot t_2$, $t_2 \leftarrow t_1 - t_4$,
$t_5 \leftarrow t_5^2$, $t_8 \leftarrow 2 \cdot t_3$,
$t_8 \leftarrow t_8 - t_2$, $t_8 \leftarrow t_8 - t_1$,
$t_8 \leftarrow t_8 \cdot U_1$, $t_8 \leftarrow t_8 + t_5$,
$t_5 \leftarrow 2 \cdot V_1''$, $t_8 \leftarrow t_8 + t_5$,
$V_1'' \leftarrow t_6 + V_1''$, $t_6 \leftarrow t_6 \cdot t_2$,
$U_1'' \leftarrow 2 \cdot t_2$, $U_1'' \leftarrow U_1'' - t_3$,
$t_2 \leftarrow U_1'' - t_2$, $t_4 \leftarrow t_4 - U_1''$,
$t_4 \leftarrow t_4 \cdot t_2$, $t_4 \leftarrow t_4 \cdot Z$,
$Z'' \leftarrow U_0'' \cdot Z$, $t_1 \leftarrow t_1 - U_1''$,
$U_1'' \leftarrow U_1'' \cdot Z''$, $U_0'' \leftarrow t_8 \cdot U_0''$,
$V_1'' \leftarrow V_1'' - t_8$, $V_1'' \leftarrow V_1'' \cdot t_7$,
$V_1'' \leftarrow t_4 - V_1''$, $V_0'' \leftarrow V_0'' \cdot t_7$,
$t_1 \leftarrow t_1 \cdot t_8$, $t_1 \leftarrow t_1 - t_6$,
$V_0'' \leftarrow t_1 - V_0''$, $Z'' \leftarrow Z'' \cdot t_7$

Algorithm 2 Projective addition between general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

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

$U_1'' \leftarrow U_1 \cdot Z'$, $U_0'' \leftarrow U_0 \cdot Z'$,
$t_1 \leftarrow V_0 \cdot Z'$, $t_2 \leftarrow V_0' \cdot Z$,
$t_1 \leftarrow t_1 - t_2$, $t_2 \leftarrow U_0' \cdot Z$,
$t_3 \leftarrow U_1' \cdot Z$, $t_4 \leftarrow t_3 \cdot t_2$,
$t_2 \leftarrow t_2 - U_0''$, $t_5 \leftarrow U_1'' - t_3$,
$t_6 \leftarrow U_1'' \cdot U_0''$, $t_4 \leftarrow t_4 - t_6$,
$t_6 \leftarrow V_1' \cdot Z$, $Z'' \leftarrow Z \cdot Z'$,
$t_7 \leftarrow V_1 \cdot Z'$, $t_8 \leftarrow t_7 - t_6$,
$t_6 \leftarrow t_7 + t_6$, $t_9 \leftarrow U_1''^2$,
$t_{10} \leftarrow Z'' \cdot t_2$, $t_{10} \leftarrow t_9 + t_{10}$,
$t_{11} \leftarrow t_3^2$, $t_3 \leftarrow U_1'' + t_3$,
$t_{12} \leftarrow t_{10} - t_{11}$, $t_{11} \leftarrow t_9 + t_{11}$,
$t_9 \leftarrow t_4 \cdot t_8$, $t_4 \leftarrow t_4 \cdot t_5$,
$t_5 \leftarrow t_1 \cdot t_5$, $t_1 \leftarrow t_1 \cdot t_{12}$,
$t_8 \leftarrow t_2 \cdot t_8$, $t_2 \leftarrow t_2 \cdot t_{12}$,
$t_1 \leftarrow t_9 + t_1$, $t_5 \leftarrow t_5 + t_8$,
$t_2 \leftarrow t_2 - t_4$, $t_4 \leftarrow t_5 \cdot Z''$,
$t_8 \leftarrow t_2 \cdot t_4$, $t_2 \leftarrow t_2^2$,
$t_5 \leftarrow t_5 \cdot t_4$, $t_4 \leftarrow t_1 \cdot t_4$,
$U_1'' \leftarrow U_1'' \cdot t_5$, $t_9 \leftarrow 2 \cdot t_4$,
$t_9 \leftarrow t_9 - t_2$, $t_{12} \leftarrow t_5 \cdot t_3$,
$t_9 \leftarrow t_9 - t_{12}$, $t_2 \leftarrow t_9 - t_2$,
$t_2 \leftarrow t_2 \cdot t_3$, $t_{11} \leftarrow t_5 \cdot t_{11}$,
$t_2 \leftarrow t_2 + t_{11}$, $t_2 \leftarrow t_2/2$,
$t_{12} \leftarrow Z'' \cdot t_5$, $U_0'' \leftarrow U_0'' \cdot t_{12}$,
$t_{12} \leftarrow t_8 \cdot t_{12}$, $t_{11} \leftarrow Z' \cdot t_{12}$,
$V_0'' \leftarrow t_{11} \cdot V_0$, $V_1'' \leftarrow t_{11} \cdot V_1$,
$t_{11} \leftarrow t_4 - t_9$, $t_4 \leftarrow U_1'' - t_4$,
$t_1 \leftarrow t_1^2$, $t_6 \leftarrow t_8 \cdot t_6$,
$t_1 \leftarrow t_1 \cdot Z''$, $t_1 \leftarrow t_1 + t_6$,
$t_1 \leftarrow t_1 - t_2$, $t_2 \leftarrow t_1 - U_0''$,
$t_5 \leftarrow t_2 \cdot t_5$, $t_2 \leftarrow t_9 \cdot t_{11}$,
$t_{11} \leftarrow t_1 \cdot t_{11}$, $t_6 \leftarrow U_1'' \cdot t_4$,
$t_6 \leftarrow t_6 + t_2$, $t_5 \leftarrow t_6 + t_5$,
$t_4 \leftarrow U_0'' \cdot t_4$, $t_{11} \leftarrow t_4 + t_{11}$,
$t_9 \leftarrow t_9 \cdot t_8$, $U_1'' \leftarrow t_9 \cdot Z''$,
$U_0'' \leftarrow t_1 \cdot t_8$, $t_5 \leftarrow t_5 \cdot Z''$,
$V_1'' \leftarrow t_5 - V_1''$, $V_0'' \leftarrow t_{11} - V_0''$,
$Z'' \leftarrow Z'' \cdot t_{12}$

Algorithm 3 Mixed addition between general divisors on the Jacobian of $C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1 + f_0$.

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

$t_1 \leftarrow v_0 \cdot Z$, $V_0'' \leftarrow V_0 - t_1$,
$t_1 \leftarrow v_1 \cdot Z$, $t_2 \leftarrow t_1 + V_1$,
$t_1 \leftarrow t_1 - V_1$, $V_1'' \leftarrow u_1 \cdot Z$,
$t_3 \leftarrow V_1'' + U_1$, $t_4 \leftarrow u_0 \cdot Z$,
$t_5 \leftarrow V_1'' \cdot t_4$, $t_6 \leftarrow U_1 \cdot U_0$,
$t_6 \leftarrow t_6 - t_5$, $U_0'' \leftarrow U_0 - t_4$,
$t_5 \leftarrow V_1''^2$, $t_7 \leftarrow U_1^2$,
$U_1'' \leftarrow V_1'' - U_1$, $t_8 \leftarrow t_5 - t_7$,
$t_5 \leftarrow t_5 + t_7$, $t_7 \leftarrow Z \cdot U_0''$,
$t_8 \leftarrow t_7 + t_8$, $t_7 \leftarrow t_6 \cdot t_1$,
$t_1 \leftarrow U_0'' \cdot t_1$, $U_0'' \leftarrow U_0'' \cdot t_8$,
$t_6 \leftarrow t_6 \cdot U_1''$, $U_1'' \leftarrow V_0'' \cdot U_1''$,
$t_8 \leftarrow V_0'' \cdot t_8$, $t_7 \leftarrow t_7 - t_8$,
$t_1 \leftarrow t_1 - U_1''$, $U_0'' \leftarrow U_0'' - t_6$,
$t_8 \leftarrow U_0''^2$, $t_6 \leftarrow t_1 \cdot Z$,
$U_0'' \leftarrow U_0'' \cdot t_6$, $t_1 \leftarrow t_1 \cdot t_6$,
$V_1'' \leftarrow t_1 \cdot V_1''$, $t_5 \leftarrow t_1 \cdot t_5$,
$V_0'' \leftarrow t_7 \cdot t_6$, $t_6 \leftarrow t_6^2$,
$t_7 \leftarrow t_7^2$, $t_4 \leftarrow t_4 \cdot t_6$,
$t_6 \leftarrow U_0'' \cdot t_6$, $U_1'' \leftarrow 2 \cdot V_0''$,
$U_1'' \leftarrow U_1'' - t_8$, $t_2 \leftarrow U_0'' \cdot t_2$,
$t_7 \leftarrow t_7 \cdot Z$, $t_7 \leftarrow t_7 + t_2$,
$t_2 \leftarrow t_1 \cdot t_3$, $U_1'' \leftarrow U_1'' - t_2$,
$t_8 \leftarrow U_1'' - t_8$, $t_3 \leftarrow t_3 \cdot t_8$,
$t_3 \leftarrow t_3 + t_5$, $t_3 \leftarrow t_3/2$,
$t_7 \leftarrow t_7 - t_3$, $t_8 \leftarrow V_1'' - V_0''$,
$V_0'' \leftarrow V_0'' - U_1''$, $t_5 \leftarrow t_7 - t_4$,
$V_1'' \leftarrow V_1'' \cdot t_8$, $t_1 \leftarrow t_1 \cdot t_5$,
$t_1 \leftarrow t_1 + V_1''$, $V_1'' \leftarrow U_1'' \cdot V_0''$,
$V_1'' \leftarrow V_1'' + t_1$, $t_4 \leftarrow t_4 \cdot t_8$,
$V_0'' \leftarrow V_0'' \cdot t_7$, $V_0'' \leftarrow t_4 + V_0''$,
$t_4 \leftarrow t_6 \cdot v_1$, $V_1'' \leftarrow V_1'' - t_4$,
$U_1'' \leftarrow U_1'' \cdot Z$, $U_1'' \leftarrow U_1'' \cdot U_0''$,
$U_0'' \leftarrow t_7 \cdot U_0''$, $V_1'' \leftarrow Z \cdot V_1''$,
$Z'' \leftarrow Z \cdot t_6$, $t_7 \leftarrow Z'' \cdot v_0$,
$V_0'' \leftarrow V_0'' - t_7$

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 will differ from any other generic curves over the same prime fields, unless such curves are real (degree-6) curves which cannot be transformed into imaginary (degree-5) curves.

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 [42] and gives rise to a curve C over $\mathbb{F}_p$ whose Jacobian has prime order

$$r = 2894802230932904884816923999565902513845117797309155137410147573289258033225 9,$$

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

$$f_3 = 34744234758245218589390329770704207149, \quad f_2 = 13271361720934533507512505944425618802 1,$$
$$f_1 = 90907655901711006083734360528442376758, \quad f_0 = 66679866221737283378235608571799928 16.$$

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 [42] and gives rise to a curve C over $\mathbb{F}_p$ whose Jacobian has prime order

$$r = 1157920892373161954293422038010335541709316156518816573073080680797020899517 81,$$

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

$$f_3 = 31825824271720170945390138432856923665 3, \quad f_2 = 75380722035796344355219475510170298006,$$
$$f_1 = 12941608260346057927284769463099809923 7, \quad f_0 = 14386407277259944404677841670908267938 8.$$

5 The Kummer Surface

Gaudry [28] built on earlier observations by Chudnovsky and Chudnovsky [16] to show that scalar multiplication in genus 2 can be greatly accelerated by working on the Kummer surface associated to a Jacobian, rather than 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 [61].

5.1 The Squares-only Kummer Routine

The Kummer surface that was originally proposed for cryptography in [28] 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 that the fundamental Theta constants associated to a genus 2 curve cannot be defined over $\mathbb{F}_p$, but all of their squares can be.

Cosset [17] 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

$$\mathcal{K}: \quad E'xyzt = ((x^2 + y^2 + z^2 + t^2) - F(xt + yz) - G(xz + yt) - H(xy + zt))^2,$$

Algorithm 4 The Hadamard transform (H).

Input: (x, y, z, t).
Output: $\mathtt{H}(x, y, z, t)$.
1. $t_1 \leftarrow x + y,\ \ t_2 \leftarrow z + t,$
   $t_3 \leftarrow x - y,\ \ t_4 \leftarrow z - t.$
2. $x \leftarrow t_1 + t_2,\ \ y \leftarrow t_1 - t_2,$
   $z \leftarrow t_3 + t_4,\ \ t \leftarrow t_3 - t_4.$
3. **return** (x, y, z, t).

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

Input: $P = (x\colon y\colon z\colon t)$ and constants $y_0, z_0, t_0, y_0', z_0', t_0'$.
Output: $[2]P = \mathtt{DBL}(P)$.
1. $x, y, z, t \leftarrow \mathtt{H}(x, y, z, t)$.
2. $x \leftarrow x^2,\ \ y \leftarrow y^2,\ \ z \leftarrow z^2,\ \ t \leftarrow t^2$.
3. $y \leftarrow y \cdot y_0',\ \ z \leftarrow z \cdot z_0',\ \ t \leftarrow t \cdot t_0'$.
4. $x, y, z, t \leftarrow \mathtt{H}(x, y, z, t)$.
5. $x \leftarrow x^2,\ \ y \leftarrow y^2,\ \ z \leftarrow z^2,\ \ t \leftarrow t^2$.
6. $y \leftarrow y \cdot y_0,\ \ z \leftarrow z \cdot z_0,\ t \leftarrow t \cdot t_0$.
7. **return** $(x\colon y\colon z\colon t)$.

$$\text{where}\quad E' = 4E^2a^2b^2c^2d^2,\quad E = \frac{ABCD}{(a^2d^2 - b^2c^2)(a^2c^2 - b^2d^2)(a^2b^2 - c^2d^2)},$$
$$F = \frac{a^4 - b^4 - c^4 + d^4}{a^2d^2 - b^2c^2},\quad G = \frac{a^4 - b^4 + c^4 - d^4}{a^2c^2 - b^2d^2},\quad H = \frac{a^4 + b^4 - c^4 - d^4}{a^2b^2 - c^2d^2},$$
$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} a^2 \\ b^2 \\ c^2 \\ d^2 \end{bmatrix}. \tag{1}$$

We write $(x\colon y\colon z\colon t) = (\vartheta_1(\mathbf{z})^2\colon \vartheta_2(\mathbf{z})^2\colon \vartheta_3(\mathbf{z})^2\colon \vartheta_4(\mathbf{z})^2)$ 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, $\mathcal{K}(\mathtt{DBL})$, computes the doubling $[2]P \in \mathcal{K}$ of a point $P \in \mathcal{K}$, while Algorithm 6, $\mathcal{K}(\mathtt{DBLADD})$, 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 [17]. Algorithm 7 computes the scalar multiple $[k]P \in \mathcal{K}$ of $P \in \mathcal{K}$ using a genus 2 version [28] of the Montgomery ladder [51]. The six surface constants that appear in the algorithms are defined as

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

Although the formulas in Algorithm 6 are presented for general inputs P, Q and $P - Q$, the inputs to $\mathcal{K}(\mathtt{DBLADD})$ 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 Line 9 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 Line 9 is modified to compute 3 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 [28, 6].

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

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: $([2]P, P + Q) = \texttt{DBLADD}(P, Q, P - Q)$.

1. $x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$, $x', y', z', t' \leftarrow \texttt{H}(x', y', z', t')$.
2. $X \leftarrow x \cdot x'$, $Y \leftarrow y \cdot y_0'$, $Z \leftarrow z \cdot z_0'$, $T \leftarrow t \cdot t_0'$.
3. $x \leftarrow x^2$, $y \leftarrow y \cdot Y$, $z \leftarrow z \cdot Z$, $t \leftarrow t \cdot T$.
4. $Y \leftarrow Y \cdot y'$, $Z \leftarrow Z \cdot z'$, $T \leftarrow T \cdot t'$.
5. $x, y, z, t \leftarrow \texttt{H}(x, y, z, t)$, $X, Y, Z, T \leftarrow \texttt{H}(X, Y, Z, T)$.
6. $x \leftarrow x^2$, $y \leftarrow y^2$, $z \leftarrow z^2$, $t \leftarrow t^2$.
7. $X \leftarrow X^2$, $Y \leftarrow Y^2$, $Z \leftarrow Z^2$, $T \leftarrow T^2$.
8. $y \leftarrow y \cdot y_0$, $z \leftarrow z \cdot z_0$, $t \leftarrow t \cdot t_0$.
9. $x' \leftarrow X/\bar{x}$, $y' \leftarrow Y/\bar{y}$, $z' \leftarrow Z/\bar{z}$, $t' \leftarrow T/\bar{t}$.
10. **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. $P_m \leftarrow P$, $P_p = \texttt{DBL}(P)$.
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** $(x : y : z : t)$.

5.2 Extracting the Squared Kummer Surface Parameters from C

In [28] Gaudry showed the relationship between the Kummer surface and the isomorphic Rosenhain model of the genus 2 curve C, given as

$$C_{\text{Ros}} : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu), \tag{3}$$

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

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

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 $d^2 = 1$ [30] and compute the other 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 C of genus 2, there are up to 120 unique Rosenhain triples λ, μ, ν that give an isomorphic representation $C_{\text{Ros}} \cong C$ over the algebraic closure [27, §2.2]. So for a given curve with rational 2-torsion, we can expect 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 Section 8.3). If the 2-torsion is rational, then 16 must divide the cardinality of $\text{Jac}(C)$ [28].

5.3 Mapping from $\mathcal{K}$ to $\text{Jac}(C)$

The maps from $\mathcal{K}$ to $\text{Jac}(C)$ were originally given by Gaudry [28] and tweaked for the squares only case by Cosset [17]. We reproduce them here for completeness, correcting a sign mistake introduced in the computation of v_0 in [17]. It should be noted that the map below

is not directly to the Jacobian of C, but rather to the Jacobian of the isomorphic curve C_{Ros} in Rosenhain form. The map takes $P = (x\colon y\colon z\colon t) \in \mathcal{K}$ 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

$$\begin{aligned}
\ell =& -\Big(\vartheta_{12}^2(\mathbf{z})\vartheta_7(\mathbf{z})^2 b^2c^2\vartheta_9^4 + \vartheta_{11}^2(\mathbf{z})\vartheta_9^2(\mathbf{z})a^2d^2\vartheta_7^4 + 2a^2b^2c^2d^2(xz+yt)\\
&+ \left(x^2+y^2+z^2+t^2 - F(xt+yz) - G(xz+yt) - H(xy+zt)\right)\frac{a^2c^2+b^2d^2}{E}\Big),\\
v_0 =& \sqrt{\ell \cdot \frac{\vartheta_8^2\vartheta_3^4\vartheta_1^4\vartheta_{14}^2(\mathbf{z})}{(\vartheta_{16}^2(\mathbf{z})b^2d^2\vartheta_{10}^2)^3}},\\
v_1 =& \frac{u_0^3 - u_0^2(u_1^2 + u_1 + (u_1+1)(\lambda+\mu+\nu) + \lambda\mu + \nu\lambda + \nu\mu) + u_0\lambda\mu\nu + u_1 v_0^2}{2v_0u_0},
\end{aligned} \tag{5}$$

where the λ, μ and ν are the particular choice of Rosenhain invariants corresponding to C_{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 [28, §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 $C_{\text{Ros}} \cong C$ or its twist $C'_{\text{Ros}} \cong C'$ [28, §5.2]. As long as the public generator $P \in \mathcal{K}$ is chosen so that it maps back to $\text{Jac}(C_{\text{Ros}})$, then any honest party participating in a Diffie-Hellman style protocol computes with multiples of P that also map back to $\text{Jac}(C_{\text{Ros}})$. However, an attacker could feed a party another point $P' \in \mathcal{K}$ that (unbeknownst to the party) maps back to C'_{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 $\text{Jac}(C)$ and $\text{Jac}(C')$ have large prime order subgroups. The ideal situation is to have $\text{Jac}(C) = 16 \cdot r$ and $\text{Jac}(C') = 16 \cdot r'$, 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, 32].

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 [33]. For the prime $p = 2^{128} - 34827$, we used the CM method to generate a twist-secure genus 2 curve.

Kummer Surface over $p = 2^{127}-1$. Gaudry and Schost [33] 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

$$\lambda = 283568639100782052886145506193140176 18, \quad \mu = 154040945529144206406682019582013187910,$$
$$\nu = 113206060534360680770189432771018826227.$$

The group orders of $\mathrm{Jac}(C) \cong \mathrm{Jac}(C_{\mathrm{Ros}})$ and $\mathrm{Jac}(C') \cong \mathrm{Jac}(C'_{\mathrm{Ros}})$ are given by $2^4 \cdot r$ and $2^4 \cdot r'$ respectively, where

$$r = 180925139433306555341467595505029059892350884363594131307776729780117962605 1,$$
$$r' = 180925139433306555357191732647120652144130617439968355857167262354635672633 9,$$

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

$$E' = 37299146226279590906389874065895056737, \quad F = 145242473685766417331928186098925456110,$$
$$G = 81667768061025231231209905783624370749, \quad H = 540582355476407258010377720836421071 70.$$

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

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

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

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 [42]. One choice of the Rosenhain model is given by the constants

$$\lambda = 45778738964487293478077907344653244 21, \quad \mu = 234789861994364729479821884660190521407,$$
$$\nu = 17433357352319216401635905869489526048 0.$$

The group orders of $\mathrm{Jac}(C) \cong \mathrm{Jac}(C_{\mathrm{Ros}})$ and $\mathrm{Jac}(C') \cong \mathrm{Jac}(C'_{\mathrm{Ros}})$ are given by $2^4 \cdot r$ and $2^4 \cdot r'$ respectively, where

$$r = 723700557733226221387377749983186995960300853730490726519494799558003962212 1,$$
$$r' = 723700557733226221407259562625411555923965370978554236151302174172125531660 1,$$

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

$$a^2 = 20124371314471321495627280078996599920 0, \quad b^2 = 146836762876902436263761475043196343 60,$$
$$c^2 = 33790404179921125742438390824466397006 3, \qquad d^2 = 1.$$

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

$$E' = 25388021028067100698903332051644035735 0, \quad F = 15901699995935891250345450670545167290 8,$$
$$G = 72998263011580475773814426394752329 07, \quad H = 13793062916001283675618873430828756806.$$

A generator on $\mathcal{K}$ with order r that maps back to $\mathrm{Jac}(C_{\mathrm{Ros}})$ is $P = (P_x : P_y : P_z : P_t)$ where

$$P_x = 1,\ P_y = 29512289488083576153799721930148668360 8,$$
$$P_z = 11682935711572123242076114651352673591 2 \text{ and } P_t = 99251552912154476320478841520348830750.$$

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

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 (pointers to) the in- and output. Since no lookup tables are used, and all modern cache sizes are large enough to hold the intermediate values from Algorithm 6 when using 128-bit arithmetic, the algorithm (and runtime) becomes independent of input almost for free. 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, we can either 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 Section 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 Section 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 2 (y_0, z_0 and t_0) and/or those appearing on Line 8 (y_0', z_0' and t_0') of Algorithm 6 by any non-zero 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 [25] 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 2-dimensional decompositions over prime fields. On the other hand, the genus 2 analogues of these curves, Buhler-Koblitz (BK) curves of the form $y^2 = x^5 + b$ [15] and Furukawa-Kawazoe-Takahashi (FKT) curves of the form $y^2 = x^5 + ax$ [23], have ϕ's whose minimal polynomials are of degree 4, which means that we can achieve 4-dimensional scalar decompositions on genus 2 curves over prime fields. Besides the two families above that offer 4-dimensional GLV decompositions, families of genus 2 curves with RM facilitate 2-dimensional scalar decompositions [43, 32]. To give an idea of the expected performance in such scenarios, we also present timings for a 2-dimensional GLV decomposition on FKT curves.

Dimension-4 GLV on BK Curves. To achieve a 4-dimensional GLV on curves of the form $C : y^2 = x^5 + b$, 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 $\mathrm{Jac}(C)$ efficiently in this scenario [15] (also see [23, §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 $(x, y) \in C$, then $(\xi_5 x, y) \in C$. 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 $T^4 + T^3 + T^2 + T + 1 = 0$.

Dimension-4 GLV on FKT Curves. Curves of the form $C : y^2 = x^5 + ax$ 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 $\mathrm{Jac}(C)$ in this scenario is also efficient [23]. Since the point $(x, y) = (0, 0) \in C$ induces a point of order 2 on $\mathrm{Jac}(C)$, 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 $(x, y) \in C$, 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 $T^4 + 1 = 0$.

Dimension-2 GLV on FKT Curves. The reason we chose FKT curves for the 2-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 2-dimensional decomposition by defining $\phi : (x, y) \mapsto ((\xi_5 + \xi_5^{-1})x, y)$ on C and extending $\mathbb{Z}$-linearly to general divisors under the canonical embedding of C in $\mathrm{Jac}(C)$. In this case ϕ satisfies the minimal polynomial $T^2 + T - 1$ on $\mathrm{Jac}(C)$.

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.

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

curve/prime-GLV dimension	r (bits)	$\max\{\lvert k_\ell\rvert\}$ (bits) / freq. (%)					
$C_{\mathrm{BK}}/\mathbb{F}_{p_{127m}}$ - 4	254	64/21.0401	**63/ 59.2356**	62/18.1809	61/14.456	60/0.0928	$\leq$ 59/0.0050
$C_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}}$- 4	253	63/1.0009	**62/60.2852**	61/35.6103	60/2.9155	59/0.1773	$\leq$ 58/0.0108
$C_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}}$- 2	253	**126/50.0546**	125/37.3918	124/9.3992	123/2.3752	122/0.5803	$\leq$ 121/0.1989
$C_{\mathrm{BK}}/\mathbb{F}_{p_{128n}}$ - 4	256	65/0.0006	64/37.5937	**63/56.1647**	62/5.8495	61/0.3660	$\leq$ 60/0.0255
$C_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}}$- 4	255	64/23.3766	**63/64.2568**	62/11.6045	61/0.7160	60/0.0433	$\leq$ 59/0.0028
$C_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}}$- 2	255	**127/50.0756**	126/37.4652	125/9.3466	124/2.3331	123/0.5841	$\leq$ 122/0.1954

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 Section 3.2) where $\mu = -p_{127m}^{-1} \bmod 2^{64} = -1$. The Jacobians of the curves $C_{\mathrm{BK}}/\mathbb{F}_{p_{127m}} : y^2 = x^5 + 17$ and $C_{\mathrm{FKT}}/\mathbb{F}_{p_{127m}} : y^2 = x^5 + 17x$ have orders $\#\mathrm{Jac}(C_{\mathrm{BK}}) = r$ and $\#\mathrm{Jac}(C_{\mathrm{FKT}}) = 2 \cdot r'$, where

$$r = 289480223093288765951155679942144885248233282097238663354835636342417789127 51,$$
$$r' = 144740111546644382990239325534322540076961984661664556618833340927958802334 41$$

are 254- and 253-bit primes respectively.

GLV Curves Over a 128-bit Prime Field. Let $p_{128n} = 2^{128} - 24935$. The Jacobians of the curves $C_{\mathrm{BK}}/\mathbb{F}_{p_{128n}} : y^2 = x^5 + 3^7$ and $C_{\mathrm{FKT}}/\mathbb{F}_{p_{128n}} : y^2 = x^5 + 3^7 x$ have orders $\#\mathrm{Jac}(C_{\mathrm{BK}}) = r$ and $\#\mathrm{Jac}(C_{\mathrm{FKT}}) = 2 \cdot r'$ respectively, where

$$r = 115792089237316195401210495125503591471546519982099914586091636775415022457661,$$
$$r' = 578960446186580977065424241431272795958172016886380858825690668693068991608 01.$$

are 256- and 255-bit primes respectively.

6.3 Scalar Decomposition via Division

At Eurocrypt 2002, Park, Jeong and Lim [56] 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 4-dimensional cases (BK and FKT) it takes 20 multiplications to fully decompose k, and in the 2-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 [56, 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 top row which reports the statistics corresponding to 4-dimensional decompositions on Buhler-Koblitz curves with r being 254 bits. The first column indicates that around 21% of scalars decomposed to 4 mini-scalars where the maximum bit length was 64, whilst the second column reports that around 59% of scalars decomposed to 4 mini-scalars $\{k_1, k_2, k_3, k_4\}$ 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 [24] is very similar to using signed windows for a single scalar (see Section 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 [24]. 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 Section 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 applied when adding any element in a lookup table. In our implementations, the first approach is faster in the 4-dimensional case and the second approach is faster in the 2-dimensional case.

7 Results and Discussion

In Section 7.1 we discuss our code and the benchmarking environment we used. We present the main results in Section 7.2 and discuss them further in Section 7.3. In Section 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

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. 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 '?'.

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 [36]	1	✗	$2^{256} - 587$	255	2	126.8	227
2-GLV [47]	1	✗	$2^{256} - 11733$	256	6	127.0	145
surf127eps [34]	2	✓	$2^{127} - 735$	251	2	124.8	236
NISTp-224 [67, 38]	1	✓	$2^{224} - 2^{96} + 1$	224	2	111.8	302
NISTp-256 [67]	1	?	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	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	✗	$2^{64} \cdot (2^{63} - 27443) + 1$	254	10	125.7	156
(a) GLV-4-FKT	2	✗	$2^{64} \cdot (2^{63} - 27443) + 1$	253	8	125.3	156
(a) GLV-2-FKT	2	✗	$2^{64} \cdot (2^{63} - 27443) + 1$	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 SUPERCOP toolkit for measuring the performance of cryptographic software (see [10]). The implementations labeled (b) are made publicly available through [10]. 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 [58] algorithm that exploits additional automorphisms [20, 71]. If r is the largest prime factor of a group with #Aut automorphisms, we compute the security level s as $s = \log_2(\sqrt{\frac{\pi r}{2\#\mathrm{Aut}}})$. We also indicate if the implementation runs in constant time, an important step towards achieving side channel resistance [41].

The implementations in the top part of the table are obtained from eBACS, except for [67] and [47]. The standardized NIST curves [67], 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 [47] 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.

[6] Note that to enable this implementation using the techniques described in [38], OpenSSL needs to be configured using "./Configure enable-ec_nistp_64_gcc_128".

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 Section 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 4-dimensional GLV implementations manage to outperform the previous fastest genus 2 implementation [34]. Prior to this work, the fastest curve arithmetic reported on eBACS was due to Bernstein [4], whilst Longa and Sica [47] 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 [47]. Our approach (b) on the Kummer surface sets a new software speed record by breaking 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 [47] 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.* [55] 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 Section 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 [46] and simultaneous addition technique [45]. 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 speedupss 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 4-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 C_{Ros} in Section 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 [69].

8.1 Recognising Kummer Parameters over $\mathbb{C}$

We use an analytic approach to assist us in generating Kummer surfaces which are associated to 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 [69]). 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 Section 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 $\text{Jac}(C)$ is known (from the CM field).

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

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

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 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 $\mathrm{Jac}(C_{\mathrm{Ros}})$ 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 $D \in \mathrm{Jac}(C_{\mathrm{Ros}})$ as

$$u_0 = \frac{(2-2i)x - (1+i)y + 2iz + 2it}{(i-1)x - (2+2i)y + 2z + 2t}, \quad u_1 = \frac{(2+2i)x - 2y - 2z + (1-i)t}{(i-1)x - (2+2i)y + 2z + 2t} - u_0 - 1.$$

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 divisors on $\mathrm{Jac}(C_{\mathrm{Ros}})$ or $\mathrm{Jac}(C'_{\mathrm{Ros}})$, where $C_{\mathrm{Ros}} : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu)$, 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 $C : y^2 = x^5 + ax$ 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 C and C_{Ros} can only be isomorphic over $\mathbb{F}_p$ if $\alpha \in \mathbb{F}_p$, which implies that $\mathrm{Jac}(C)$ is isogenous over $\mathbb{F}_p$ to the product of two elliptic curves [23, Lemma 4]. Thus C is not suitable for cryptographic applications in this case, since the group order of $\mathrm{Jac}(C)$ 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 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 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

Table 5. Quotients appearing in the maps from $\mathcal{K}$ to the u-polynomial of $D \in \mathrm{Jac}(C_{\mathrm{Ros}})$ 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	$\frac{1-i}{2}$

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

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

down the correct root corresponding to each Kummer parameter becomes more involved. Furthermore, these polynomials tell us that we can not 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 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 to the BK family is that, for primes congruent to 1 modulo 5, if the 2-torsion of $\mathrm{Jac}(C)$ or $\mathrm{Jac}(C')$ 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.

BK Kummer Chameleon over a 127-bit Prime Field. Let $p = 2^{64} \cdot (2^{63} - 1035383) + 1$, and let $C/\mathbb{F}_p : y^2 = x^5 + 7^5$, the quadratic twist C' of which can be written as $C' : y^2 = 7(x^5 + 7^5)$. The group orders are $\#\mathrm{Jac}(C) = 2^4 \cdot r$ and $\#\mathrm{Jac}(C') = 2^4 \cdot 5 \cdot r'$, where

$$r = 180925139433265935321004472177996571677719953576806075895661577071189110037 1,$$
$$r' = 361850278866531870644657474375793908062332565172509431488359127778261331091,$$

are 250- and 248-bit primes respectively. A degree 5 Rosenhain model C_{Ros} isomorphic to 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

$$a^2 = 844910266850457945987307823556591703 39, \quad b^2 = 331868411316994320350823668655709822 34,$$
$$c^2 = 857664920345416567706880270075889036 88, \qquad d^2 = 1.$$

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

$$E' = 139180060863318125490801991597453057 70, \quad F = 187625840664800037601345952054852599 83,$$
$$G = 137599581973583773482954213814600348679, \quad H = 857664920345416567706880270075889036 88.$$

A generator on $\mathcal{K}$ with order r that maps back to $\mathrm{Jac}(C_{\mathrm{Ros}})$ is

$$P = [2](1, 1, 1, 860113666896998803306002937254190439 35).$$

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

$$r = 72370055773322622150800318366588777873542742128516066638786802028366350962 91,$$
$$r' = 14474011154664524425732682578852164073223244445682174205898542511197921098 11,$$

are 253- and 250-bit primes respectively. A degree 5 Rosenhain model C_{Ros} isomorphic to C is given by the constants

$$\lambda = 69750747073243793503741945404989703593, \quad \mu = 15017962230774307498886941644141431335 5,$$
$$\nu = 22799781617760230807487345108773362367 6,$$

for which one choice of the squared fundamental theta constants is

$$a^2 = 31137852018598787924963645145671008485 7, \quad b^2 = 19469229948349962839682510865964453016 1,$$
$$c^2 = 77818193869859233086004034646319310321, \qquad d^2 = 1.$$

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

$$E' = 19523440971343080786658226319936172787 6, \quad F = 14247565526240974961016822622793017217 5,$$
$$G = 25789434559921498757356883420864617479, \quad H = 77818193869859233086004034646319310321.$$

A generator on $\mathcal{K}$ with order r that maps back to $\mathrm{Jac}(C_{\mathrm{Ros}})$ is

$$P = [2](1, 1, -1, 33054721556203704896838868895641995262 6).$$

8.4 Kummer Chameleons with 2-dimensional GLV

Although we have focused on two families of genus 2 curves that offer 4-dimensional GLV over prime fields, there are many more families that offer 2-dimensional GLV [43, 64, 32]. We especially mention the family due to Mestre [48], which was studied further in [32, §4.4]. This family might be particularly attractive since the techniques in [32] make it practical to find twist-secure instances over $\mathbb{F}_p$ with $p = 2^{127} - 1$. Working analytically, we observed that small Kummer constants are often obtained if we take special instances of the families with efficiently computable RM. An example from the family due to Tautz, Tops and Verberkmoes [65] (also see [43, §5.1]) is the Kummer surface associated to the curve $y^2 = x(x^4 - x^2 + 1)$, which yields $t_0 = 1$ over $\mathbb{C}$, so the techniques in [32, §4.4] could be used (over many primes) to find twist-secure curves that can take advantage of this.

8.5 GLV on the Kummer Surface?

Gaudry [29] 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 [68], 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

[7] These terms are made precise in [32, Def. 1,2].

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 [52, 5, 63]: 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 [52, 5] 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.

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 $C/\mathbb{F}_p : y^2 = -x^5 + 3x^4 + 2x^3 - 6x^2 - 3x + 1$ and its twist C' are given as $\#\mathrm{Jac}(C) = 2^5 \cdot r$ and $\#\mathrm{Jac}(C') = 2^4 \cdot r'$, where

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

are 249- and 250-bit primes respectively. An isomorphic Rosenhain model C_{Ros} of C is defined by the triple

$$\lambda = 168171229223321177769186812485517969948, \quad \mu = 941743020357395228083354021573846495 7,$$
$$\nu = 158753799019747225488353272269779504992,$$

for which one choice of the squared fundamental theta constants is

$$a^2 = 150321345934746312135040529601365675926, \quad c^2 = 969856926136930102301883390336657759 36,$$

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

$$E' = 16, \quad F = 1127220808873561686485835710574760168 74,$$
$$G = 396396750514418869783757559576031316 04, \quad H = 133526895531650151065292246583602218570.$$

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

$$P = [2](1, 1, -1, 1298896584667729168876658111072852365 09).$$

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 $C/\mathbb{F}_p : y^2 = -x^5 + 3x^4 + 2x^3 - 6x^2 - 3x + 1$ and its twist C' are given as $\#\mathrm{Jac}(C) = 2^5 \cdot r$ and $\#\mathrm{Jac}(C') = 2^4 \cdot r'$, where

$$r = 36185027886661311074633479623226735613127095718810840139899493516914503673 67,$$
$$r' = 72370055773322622130196772014400965045804811092733303706378413678297043376 87,$$

are both 252-bit primes. An isomorphic Rosenhain model C_{Ros} of C is defined by the triple

$$\lambda = 33885361332396197654129462844805139399 7, \quad \mu = 16182699407691501435226104638294188080 8,$$
$$\nu = 17702661924704696218903358206510951319 0,$$

for which one choice of the squared fundamental theta constants is

$$a^2 = 18605518542908942302949982888990374210 5, \quad c^2 = 29331105170247799034890696953544646374 0,$$

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

$$E' = 16, \quad F = 30856699076152179550360945335151206300 8,$$
$$G = 30845436298369808086774955708371612923 0, \quad H = 29336736559138984766683691766934443062 8.$$

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

$$P = [2](1, 1, -1, 32893149818038102589939028525751006239 6).$$

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 2-and 4-dimensional GLV decompositions, and on the Kummer surface proposed by Gaudry [28]. 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.

After the publication of this paper, 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[8] and the subsequent performance numbers are stated in Table 2. We would like to thank the authors of [7] for finding this mistake.

[8] `http://hhisil.yasar.edu.tr/files/hisil20140312genus2.tar.gz`

References

1. T. Acar and D. Shumow. Modular reduction without pre-computation for special moduli. Technical report, Microsoft Research, 2010.
2. L. Adleman, J. DeMarrais, and M. Huang. A subexponential algorithm for discrete logarithms over hyperelliptic curves of large genus over GF(q). *Theoretical Computer Science*, 226(1-2):7–18, 1999.
3. D. F. Aranha, A. Faz-Hernández, J. López, and F. Rodríguez-Henríquez. Faster implementation of scalar multiplication on Koblitz curves. In A. Hevia and G. Neven, editors, *LATINCRYPT*, volume 7533 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2012.
4. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, Heidelberg, 2006.
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, and 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, and B.-Y. Yang. High-speed high-security signatures. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2011.
9. D. J. Bernstein and T. Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In G. L. Mullen, D. Panario, and I. E. Shparlinski, editors, *Finite Fields and Applications*, volume 461 of *Contemporary Mathematics Series*, pages 1–19. American Mathematical Society, 2008.
10. D. J. Bernstein and 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 and T. Helleseth, editors, *Arithmetic of Finite Fields – WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 7–24. Springer, Heidelberg, 2010.
12. J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. *International Journal of Applied Cryptography*, 2(3):212–228, 2012.
13. A. Brauer. On addition chains. *Bulletin of the American Mathematical Society*, 45:736–739, 1939.
14. M. Brown, D. Hankerson, J. López, and A. Menezes. Software implementation of the NIST elliptic curves over prime fields. In D. Naccache, editor, *CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer, Heidelberg, 2001.
15. J. Buhler and N. Koblitz. Lattice basis reduction, Jacobi sums and hyperelliptic cryptosystems. *Bulletin of the Australian Mathematical Society*, 58(1):147–154, 1998.
16. D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7:385–434, December 1986.
17. R. Cosset. Factorization with genus 2 curves. *Math. Comput.*, 79(270):1191–1208, 2010.
18. C. Costello and K. Lauter. Group law computations on Jacobians of hyperelliptic curves. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 92–117. Springer, 2011.
19. C. Diem. On the discrete logarithm problem in class groups of curves. *Mathematics of Computation*, 80:443–475, 2011.
20. I. M. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *Asiacrypt 1999*, volume 1716 of *Lecture Notes in Computer Science*, pages 103–121. Springer, Heidelberg, 1999.
21. K. Eisentrager and K. Lauter. A CRT algorithm for constructing genus 2 curves over finite fields. *AGCT-11*, 2007.
22. A. Enge. Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. *Mathematics of Computation*, 71:729–742, 2002.
23. E. Furukawa, M. Kawazoe, and T. Takahashi. Counting points for hyperelliptic curves of type $y^2 = x^5 + ax$ over finite prime fields. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2003.
24. S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011.

25. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
26. P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. *Eurocrypt*, 1807:19–34, 2000.
27. 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.
28. P. Gaudry. Fast genus 2 arithmetic based on theta functions. *Journal of Mathematical Cryptology JMC*, 1(3):243–265, 2007.
29. 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.
30. P. Gaudry. Personal communication, 2011.
31. P. Gaudry, T. Houtmann, D. R. Kohel, C. Ritzenthaler, and A. Weng. The 2-adic CM method for genus 2 curves with application to cryptography. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2006.
32. P. Gaudry, D. R. Kohel, and B. A. Smith. Counting points on genus 2 curves with real multiplication. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 504–519. Springer, 2011.
33. P. Gaudry and É. Schost. Genus 2 point counting over prime fields. *J. Symb. Comput.*, 47(4):368–400, 2012.
34. P. Gaudry and E. Thomé. The mp$\mathbb{F}_q$ library and implementing curve-based key exchanges. In *Software Performance Enhancement for Encryption and Decryption – SPEED 2007*, pages 49–64, 2007. `www.loria.fr/~gaudry/publis/mpfq.pdf`.
35. M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/`.
36. H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In J. Pieprzyk, editor, *Asiacrypt 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 326–343. Springer, Heidelberg, 2008.
37. B. S. Kaliski Jr. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44(8):1064–1065, 1995.
38. E. Käsper. Fast elliptic curve cryptography in OpenSSL. In G. Danezis, S. Dietrich, and K. Sako, editors, *Financial Cryptography Workshops*, volume 7126 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2012.
39. M. Knežević, F. Vercauteren, and I. Verbauwhede. Speeding up bipartite modular multiplication. In M. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields – WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 166–179. Springer Berlin / Heidelberg, 2010.
40. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.
41. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Crypto 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, Heidelberg, 1996.
42. D. R. Kohel. Databases for Elliptic Curves and Higher Dimensional Analogues (Echidna). `http://echidna.maths.usyd.edu.au/kohel/dbs/`.
43. D. R. Kohel and B. A. Smith. Efficiently computable endomorphisms for hyperelliptic curves. In F. Hess, S. Pauli, and M. E. Pohst, editors, *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 495–509. Springer, 2006.
44. A. K. Lenstra. Generating RSA moduli with a predetermined portion. In K. Ohta and D. Pei, editors, *Asiacrypt'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg, 1998.
45. C. H. Lim and H. S. Hwang. Speeding up elliptic scalar multiplication with precomputation. In J. Song, editor, *Information Security and Cryptology - ICISC'99*, volume 1787 of *Lecture Notes in Computer Science*, pages 102–119. Springer, 2000.
46. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 1994.
47. P. Longa and F. Sica. Four-dimensional Gallant-Lambert-Vanstone scalar multiplication. In X. Wang and K. Sako, editors, *Asiacrypt 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 718–739. Springer, 2012.
48. 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.

49. V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Crypto 1985*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, Heidelberg, 1986.
50. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
51. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
52. P. L. Montgomery. Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via lucas chains. URL: `ftp://ftp.cwi.nl/pub/pmontgom/Lucas.ps.gz`, 1992.
53. F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 24:531–544, 1990.
54. National Security Agency. Fact sheet NSA Suite B Cryptography. `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`, 2009.
55. T. Oliveira, F. Rodríguez-Henríquez, and J. López. New timings for scalar multiplication using a new set of coordinates. Rump session talk at ECC 2012, October 2012.
56. Y.-H. Park, S. Jeong, and J. Lim. Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms. In L. R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2002.
57. J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp*, 55(192):745–763, 1990.
58. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.
59. A. Scholz. Aufgabe 253. *Jahresbericht der deutschen Mathematiker-Vereingung*, 47:41–42, 1937.
60. R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp*, 44(170):483–494, 1985.
61. N. P. Smart and S. Siksek. A fast Diffie-Hellman protocol in genus 2. *J. Cryptology*, 12(1):67–73, 1999.
62. J. A. Solinas. Generalized Mersenne numbers. Technical Report CORR 99–39, Centre for Applied Cryptographic Research, University of Waterloo, 1999.
63. M. Stam. *Speeding up Subgroup Cryptosystems*. PhD thesis, Technische Universiteit Eindhoven, May 2003.
64. K. Takashima. A new type of fast endomorphisms on Jacobians of hyperelliptic curves and their cryptographic application. *IEICE Transactions*, 89-A(1):124–133, 2006.
65. W. Tautz, J. Top, and A. Verberkmoes. Explicit hyperelliptic curves with real multiplication and permutation polynomials. *Canad. J. Math*, 43(5):1055–1064, 1991.
66. E. G. Thurber. On addition chains $l(mn) \leq l(n) - b$ and lower bounds for $c(r)$. *Duke Mathematical Journal*, 40:907–913, 1973.
67. 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`.
68. P. B. van Wamelen. Examples of genus two CM curves defined over the rationals. *Math. Comput.*, 68(225):307–320, 1999.
69. 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, and B. Sturmfels, editors, *Discovering Mathematics with Magma*, volume 19 of *Algorithms and Computation in Mathematics*, pages 117–135. Springer Berlin Heidelberg, 2006.
70. A. Weng. Constructing hyperelliptic curves of genus 2 suitable for cryptography. *Mathematics of Computation*, 72(241):435–458, 2003.
71. M. J. Wiener and R. J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography – (SAC) 1998*, volume 1556 of *Lecture Notes in Computer Science*, pages 190–200. Springer New York, 1999.