Fast Cryptography in Genus 2

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 120 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 [36,42] 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 [8]). 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 [2,24,20,18]. Such attacks are not known, however, for *genus* 2 *curves*. In [26], 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. [5]) 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

[*] 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.

T. Johansson and P. Nguyen (Eds.): EUROCRYPT 2013, LNCS 7881, pp. 194–210, 2013.

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 [46].

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 [29], 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 [40]. 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 [37]. 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 [23,22]. For prime fields we implement 4-dimensional GLV decompositions on Buhler-Koblitz (BK) curves [14] and on Furukawa-Kawazoe-Takahashi (FKT) curves [21], 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 [52] and another based on the special form of the prime when using Montgomery multiplication [43].

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.

We refer to the full-version of this paper [10] for the specifications of all curves used and more detailed information.

2 Preliminaries

In this section we recall some basic facts and notation concerning genus 2 curves and briefly review the main techniques used to compute scalar multiplications.

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 $\text{Jac}(C)$, the Jacobian of C, which consists of degree zero divisors on the curve modulo principal divisors. Throughout this paper we use the *Mumford representation* of general divisors $D = (x^2 + u_1x + u_0, y - (v_1x + v_0)) \in \text{Jac}(C)$, 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 $Z \neq 0$.

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* [50] and a typical optimization to lower the number of point additions is using *windows* [12] 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* [55] 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[1], reducing the total number of group operations even further; this is called the *signed windows* approach [45]. See [7] for a summary of these techniques.

Adding an affine point to a projective point to obtain another projective point, often referred to as mixed addition, is usually faster than adding two projective points. In order to use these faster formulas, a common approach is to convert the precomputed projective points into their affine form. This requires an inversion for each point in the table. Using Montgomery's *simultaneous inversion* method [44], 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

[1] The term 'point' becomes 'divisor' in the case of hyperelliptic curves, but remains as 'point' for Kummer surface arithmetic in Section 5.

the FIPS 186-3 standard [56], NIST recommends the use of five prime fields when using the elliptic curve digital signature algorithm (but see also [4]). Such special primes have been studied from both a theoretical and practical point of view. 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. [13], and in [9] a comparison is made between the performance when using Montgomery multiplication [43] 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).

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 $0 \le a, b, c_h, c_\ell, \delta < 2^s$, write $c = a \cdot b = c_h \cdot 2^s + c_\ell \equiv c_\ell + \delta c_h \pmod{2^s - \delta}$ where $0 \le c_\ell + \delta c_h < (\delta+1)2^s$. 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 [52], which is used to implement fast arithmetic modulo the NIST primes [56]. We refer to this type of reduction as *NIST-like reduction*. When computing $a \cdot b \bmod p_{1271}$ with $0 \le a, b < p_{1271}$, one can first compute the multiplication $c = a \cdot b = c_1 \cdot 2^{128} + c_0$, where $0 \le c_1, c_0 < 2^{128}$. 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 $0 \le c' < 2^{128}$. 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.

Montgomery-Friendly Primes. Montgomery multiplication [43] 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 $r^{n-1} \le p < r^n$, and suppose we have an integer $0 \le X < p$. 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 $0 \le A, B < p$. Let $A = \sum_{i=0}^{n-1} a_i \cdot r^i$, where $0 \le a_i < r$, and start with $C = 0$. For all $i \in \mathbb{Z}$ such that $0 \le i < n$, 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 [39] 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 [35,1,31] 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.

Modular Inversion. When using the regular representation of integers, one can either use the (binary) extended GCD algorithm to compute the modular inversion or use the special form of the modulus to compute the inverse by using modular exponentiations. For instance, in the case of p_{1271}, one can exploit the congruence $a^{2^{127}-2} \equiv a^{-1} \pmod{p_{1271}}$. The situation when working in Montgomery form is slightly different. Given the Montgomery form $\tilde{a} = a2^{bn} \bmod p$ of an integer a, we want to compute the Montgomery inverse $\tilde{a}^{-1}2^{2bn} \equiv a^{-1}2^{bn} \pmod p$. This would require a classical inversion and modular multiplication, however we found that the approach presented in [11] (which uses the binary version of the Euclidean algorithm from [33]) 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 $\geq 2^k - c$.

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. We make use of the fast formulas for arithmetic on imaginary quadratic curves from [17], which employ homogeneous projective coordinates, and focus on reducing the total number of multiplications in projective point doublings, point additions and mixed additions.[2]

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.[3] 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 $34\mathbf{m} + 6\mathbf{s} + 34\mathbf{a}$. 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 $44\mathbf{m} + 4\mathbf{s} + 29\mathbf{a}$. 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 $37\mathbf{m} + 5\mathbf{s} + 29\mathbf{a}$. 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).

5 The Kummer Surface

Gaudry [26] built on earlier observations by Chudnovsky and Chudnovsky [15] 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 [51].

[2] Note that the formulas to compute the projective doubling from [17] can be sped up since the first multiplication to compute UU is redundant.

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

The Squares-only Kummer Routine. The Kummer surface that was originally proposed for cryptography in [26] 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 [5] 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.

Cosset [16] 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

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

$$\text{where} \quad E' = 4E^2a^2b^2c^2d^2, \quad E = \frac{ABCD}{(a^2d^2-b^2c^2)(a^2c^2-b^2d^2)(a^2b^2-c^2d^2)},$$

$$F = \frac{a^4-b^4-c^4+d^4}{a^2d^2-b^2c^2}, \quad G = \frac{a^4-b^4+c^4-d^4}{a^2c^2-b^2d^2}, \quad H = \frac{a^4+b^4-c^4-d^4}{a^2b^2-c^2d^2},$$

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} a^2 \\ b^2 \\ c^2 \\ d^2 \end{bmatrix}. \tag{1}$$

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 $\mathcal{K}$.

Extracting the Squared Kummer Surface Parameters. In [26] Gaudry showed the relationship between the Kummer surface and the isomorphic Rosenhain model of the genus 2 curve C, given as

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

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 $d^2 = 1$ [27] 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 $C_{\mathrm{Ros}} \cong C$ over the algebraic closure [25, §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). If the 2-torsion is rational, then 16 must divide the cardinality of $\text{Jac}(C)$ [26].

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 $C'_{\text{Ros}} \cong C'$ [26, §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 are overly involved (see [10, §5]). The best solution is to compute curves where both $\text{Jac}(C)$ and $\text{Jac}(C')$ have large prime order subgroups. Such curves and their associated Kummer surfaces are called *twist-secure* [29,28].

Implementation Details and Side-channel Resistance. When computing the scalar multiplication on a Kummer surface, the combined double and pseudo-addition routine is called for every bit in the scalar, except the first one. 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 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 we compute the combined doubling and pseudo-addition a fixed number of times.

6 GLV in Genus-2

The Gallant-Lambert-Vanstone (GLV) method [23] 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.

Endomorphisms. In general, algebraic curves over prime fields do not come equipped with a useful endomorphism ϕ, which means 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 $y^2 = x^5 + b$ [14] and Furukawa-Kawazoe-Takahashi (FKT) curves of the form $y^2 = x^5 + ax$ [21], 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 *real multiplication* (RM) facilitate 2-dimensional scalar decompositions [38,28]. To give an idea of the expected performance in such scenarios, we also present timings for a 2-dimensional GLV decomposition on FKT curves.

Scalar Decomposition Via Division. At Eurocrypt 2002, Park, Jeong and Lim [48] 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 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 [48, Table 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 [22] is very similar to using signed windows for a single scalar (see Section 2). 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 [22]. 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

Table 1. 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 [4,6]	1	✓	$2^{255} - 19$	253	2	125.8	182
ecfp256e [32]	1	✕	$2^{256} - 587$	255	2	126.8	227
Longa-Sica 2-GLV [40]	1	✕	$2^{256} - 11733$	256	6	127.0	145
surf127eps [30]	2	✓	$2^{127} - 735$	251	2	124.8	236
NISTp-224 [56,34]	1	✓	$2^{224} - 2^{96} + 1$	224	2	111.8	302
NISTp-256 [56]	1	?	p_1	256	2	127.8	658
(a) generic127	2	✕	$2^{127} - 1$	254	2	126.8	295
(b) generic127	2	✕	$2^{127} - 1$	254	2	126.8	248
(b) generic128	2	✕	$2^{128} - 173$	257	2	127.8	364
(a) Kummer	2	✓	$2^{127} - 1$	251	2	124.8	139
(b) Kummer	2	✓	$2^{127} - 1$	251	2	124.8	117
(b) Kummer	2	✓	$2^{128} - 237$	253	2	125.8	166
(a) GLV-4-BK	2	✕	p_2	254	10	125.7	156
(a) GLV-4-FKT	2	✕	p_2	253	8	125.3	156
(a) GLV-2-FKT	2	✕	p_2	253	8	125.3	220
(b) GLV-4-BK	2	✕	$2^{128} - 24935$	256	10	126.7	164
(b) GLV-4-FKT	2	✕	$2^{128} - 24935$	255	8	126.3	167
(b) GLV-2-FKT	2	✕	$2^{128} - 24935$	255	8	126.3	261

using Montgomery's simultaneous inversion method (see Section 2), 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 this section we present our performance results and compare them with the current state-of-the-art.

Benchmark Setting and Code. All of the implementations in Table 1 were run on an Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz with hyperthreading turned off and over-clocking ("turbo boost") disabled. The implementations labeled (a) use the Montgomery-friendly primes. They have been compiled using Microsoft Visual Studio 2012 and run on 64-bit Windows, where the timings are obtained using the time stamp counter instruction `rdtsc` over several thousand scalar multiplications. The implementations labeled (b) use the NIST-like approach and have been compiled with gcc 4.6.3 to run on 64-bit

Linux, where the timings are obtained using the SUPERCOP toolkit for measuring the performance of cryptographic software (see [8]). The implementations labeled (b) are made publicly available through [8]. 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.

Results. Table 1 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 [49] algorithm that exploits additional automorphisms [19,58]. If r is the largest prime factor of a group with #Aut automorphisms, we compute the security level s as $s = \log_2(\sqrt{\frac{\pi r}{2\#\text{Aut}}})$. We also indicate if the implementation runs in constant time, an important step towards achieving side channel resistance [37].

The implementations in the top part of the table are obtained from eBACS, except for [56] and [40]. The standardized NIST curves [56], one of which is at a lower security level, are both obtained from the benchmark program included in OpenSSL 1.0.1.[4] The implementation from [40] 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.

Discussion. The first thing to observe from Table 1 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 [30]. Prior to this work, the fastest curve arithmetic reported on eBACS was due to Bernstein [4], whilst Longa and Sica [40] 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 [40]. Our approach (b) on the Kummer surface sets a new software speed record by breaking the 120k cycle barrier for constant time implementations at the 128-bit security level.

We note that Table 1 reports implementations over prime fields only. For elliptic curves defined over quadratic extensions of large prime fields, Longa and Sica [40] 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.* [47] 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.

[4] Note, to enable this implementation, using the techniques described in [34], OpenSSL needs to be configured using "./Configure enable-ec_nistp_64_gcc_128".

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 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.

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. 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. 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 [57].

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 [57]). The theta functions can be evaluated at these period matrices, and approximations of the complex values of quotients 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 2 and 3, 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 which define the surface (see Section 5), and the constants y_0, z_0, t_0, y_0', z_0' and t_0' which are needed in the

Table 2. Kummer parameters (and their minimal polynomials) over $\mathbb{C}$ for the FKT family

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

doubling and pseudo-addition operations (see [10, §5]). The coefficients of these minimal polynomials can be reduced modulo any prime p, 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 $\mathrm{Jac}(C)$ is known (from the CM field).

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 2. 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 are 1, which immediately results in two fewer multiplications (see [10, §5]).

Mapping points from the Kummer surface to the associated Jacobian(s) actually takes points on $\mathcal{K}$ to divisors on $\mathrm{Jac}(C_{\mathrm{Ros}})$ or $\mathrm{Jac}(C'_{\mathrm{Ros}})$, where C_{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, 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 [21, 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 2 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.

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 3. Since these polynomials have degree larger than two, writing down the correct root corresponding to each Kummer parameter becomes more involved. Furthermore, these polynomials tell us that we 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 3 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 $p \equiv 11 \pmod{20}$.

Table 3. Kummer parameters (and their minimal polynomials) over $\mathbb{C}$ for the Buhler-Koblitz family

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

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.

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 [38,53,28]. We especially mention the family due to Mestre [41], which was studied further in [28, §4.4]. This family might be particularly attractive since the techniques in [28] 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 [54] (also see [38, §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 [28, §4.4] could be used (over many primes) to find twist-secure curves that can take advantage of this.

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 [26]. 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.

References

1. Acar, T., Shumow, D.: Modular reduction without pre-computation for special moduli. Technical report, Microsoft Research (2010)
2. Adleman, L., DeMarrais, J., Huang, M.: 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. Aranha, D.F., Faz-Hernández, A., López, J., Rodríguez-Henríquez, F.: Faster implementation of scalar multiplication on Koblitz curves. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 177–193. Springer, Heidelberg (2012)
4. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006)
5. Bernstein, D.J.: Elliptic vs. Hyperelliptic, part I. Talk at ECC, slides at (September 2006), `http://cr.yp.to/talks/2006.09.20/slides.pdf`
6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011)
7. Bernstein, D.J., Lange, T.: Analysis and optimization of elliptic-curve single-scalar multiplication. In: *Finite Fields and Applications*. Contemporary Mathematics Series, vol. 461, pp. 1–19. American Mathematical Society (2008)
8. Bernstein, D.J., Lange, T. (eds). eBACS: ECRYPT Benchmarking of Cryptographic Systems, `http://bench.cr.yp.to` (accessed October 4, 2012)
9. Bos, J.W.: High-performance modular multiplication on the Cell processor. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 7–24. Springer, Heidelberg (2010)
10. Bos, J.W., Costello, C., Hisil, H., Lauter, K.: Two is greater than one. Cryptology ePrint Archive, Report 2012/670 (2012), `http://eprint.iacr.org/`
11. Bos, J.W., Kaihara, M.E., Kleinjung, T., Lenstra, A.K., Montgomery, P.L.: Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. Int. J. of Applied Cryptography 2(3), 212–228 (2012)
12. Brauer, A.: On addition chains. Bulletin of the American Mathematical Society 45, 736–739 (1939)
13. Brown, M., Hankerson, D., López, J., Menezes, A.: Software implementation of the NIST elliptic curves over prime fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250–265. Springer, Heidelberg (2001)
14. Buhler, J., Koblitz, N.: Lattice basis reduction, Jacobi sums and hyperelliptic cryptosystems. Bull. of the Australian Math. Soc. 58(1), 147–154 (1998)
15. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics 7, 385–434 (1986)
16. Cosset, R.: Factorization with genus 2 curves. Math. Comp. 79(270), 1191–1208 (2010)
17. Costello, C., Lauter, K.: Group law computations on Jacobians of hyperelliptic curves. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 92–117. Springer, Heidelberg (2012)

18. Diem, C.: On the discrete logarithm problem in class groups of curves. Math. Comp. 80, 443–475 (2011)
19. Duursma, I.M., Gaudry, P., Morain, F.: Speeding up the discrete log computation on curves with automorphisms. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 103–121. Springer, Heidelberg (1999)
20. Enge, A.: Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. Math. Comp. 71, 729–742 (2002)
21. Furukawa, E., Kawazoe, M., Takahashi, T.: Counting points for hyperelliptic curves of type $\mathrm{y}^2 = \mathrm{x}^5 + \mathrm{ax}$ over finite prime fields. In: SAC 2003. LNCS, vol. 3006, pp. 26–41. Springer, Heidelberg (2003)
22. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. J. Cryptology 24(3), 446–469 (2011)
23. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)
24. Gaudry, P.: An algorithm for solving the discrete log problem on hyperelliptic curves. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 19–34. Springer, Heidelberg (2000)
25. Gaudry, P.: Algorithmique des courbes hyperelliptiques et applications à la cryptologie. PhD thesis, École polytechnique (2000), `http://www.lix.polytechnique.fr/Labo/Pierrick.Gaudry/publis/`
26. Gaudry, P.: Fast genus 2 arithmetic based on theta functions. Journal of Mathematical Cryptology 1(3), 243–265 (2007)
27. Gaudry, P.: Personal communication (2011)
28. Gaudry, P., Kohel, D.R., Smith, B.A.: Counting points on genus 2 curves with real multiplication. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 504–519. Springer, Heidelberg (2011)
29. Gaudry, P., Schost, É.: Genus 2 point counting over prime fields. J. Symb. Comp. 47(4), 368–400 (2012)
30. Gaudry, P., Thomé, E.: The mp$\mathbb{F}_q$ library and implementing curve-based key exchanges. In: SPEED 2007, pp. 49–64 (2007), `http://www.loria.fr/~gaudry/publis/mpfq.pdf`
31. Hamburg, M.: Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309 (2012), `http://eprint.iacr.org/`
32. Hisil, H., Wong, K.K.-H., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 326–343. Springer, Heidelberg (2008)
33. Kaliski Jr., B.S.: The Montgomery inverse and its applications. IEEE Transactions on Computers 44(8), 1064–1065 (1995)
34. Käsper, E.: Fast elliptic curve cryptography in openSSL. In: Danezis, G., Dietrich, S., Sako, K. (eds.) FC 2011 Workshops 2011. LNCS, vol. 7126, pp. 27–39. Springer, Heidelberg (2012)
35. Knežević, M., Vercauteren, F., Verbauwhede, I.: Speeding up bipartite modular multiplication. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 166–179. Springer, Heidelberg (2010)
36. Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48(177), 203–209 (1987)
37. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)

38. Kohel, D.R., Smith, B.A.: Efficiently computable endomorphisms for hyperelliptic curves. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 495–509. Springer, Heidelberg (2006)
39. Lenstra, A.K.: Generating RSA moduli with a predetermined portion. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 1–10. Springer, Heidelberg (1998)
40. Longa, P., Sica, F.: Four-dimensional Gallant-Lambert-Vanstone scalar multiplication. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 718–739. Springer, Heidelberg (2012)
41. Mestre, J.-F.: Couples de Jacobiennes isogenes de courbes hyperelliptiques. Preprint, arXiv (2009), `http://arxiv.org/abs/0902.3470`
42. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
43. Montgomery, P.L.: Modular multiplication without trial division. Math. Comp. 44(170), 519–521 (1985)
44. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comp. 48(177), 243–264 (1987)
45. Morain, F., Olivos, J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. Informatique Théorique et Applications/Theoretical Informatics and Applications 24, 531–544 (1990)
46. National Security Agency. Fact sheet NSA Suite B Cryptography (2009), `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`
47. Oliveira, T., Rodríguez-Henríquez, F., López, J.: New timings for scalar multiplication using a new set of coordinates. In: Rump Session Talk at ECC 2012 (2012)
48. Park, Y.-H., Jeong, S., Lim, J.: Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 197–208. Springer, Heidelberg (2002)
49. Pollard, J.M.: Monte Carlo methods for index computation (mod p). Math. Comp. 32(143), 918–924 (1978)
50. Scholz, A.: Aufgabe 253. Jahresbericht der deutschen Mathematiker-Vereingung 47, 41–42 (1937)
51. Smart, N.P., Siksek, S.: A fast Diffie-Hellman protocol in genus 2. J. Cryptology 12(1), 67–73 (1999)
52. Solinas, J.A.: Generalized Mersenne numbers. Technical Report CORR 99–39, Centre for Applied Cryptographic Research, University of Waterloo (1999)
53. Takashima, K.: A new type of fast endomorphisms on Jacobians of hyperelliptic curves and their cryptographic application. IEICE Trans. 89-A(1), 124–133 (2006)
54. Tautz, W., Top, J., Verberkmoes, A.: Explicit hyperelliptic curves with real multiplication and permutation polynomials. Canad. J. Math 43(5), 1055–1064 (1991)
55. Thurber, E.G.: On addition chains $l(mn) \leq l(n) - b$ and lower bounds for $c(r)$. Duke Mathematical Journal 40, 907–913 (1973)
56. 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`
57. van Wamelen, P.: Computing with the analytic Jacobian of a genus 2 curve. In: *Discovering Mathematics with Magma*. Algorithms and Computation in Mathematics, vol. 19, pp. 117–135. Springer, Heidelberg (2006)
58. Wiener, M.J., Zuccherato, R.J.: Faster attacks on elliptic curve cryptosystems. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 190–200. Springer, Heidelberg (1999)