J. Cryptol. (2014) 27: 248–283
DOI: 10.1007/s00145-012-9144-3

Journal of
CRYPTOLOGY

Four-Dimensional Gallant–Lambert–Vanstone Scalar Multiplication*

Patrick Longa
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
plonga@microsoft.com

Francesco Sica
Nazarbayev University, School of Science and Technology, 53 Kabanbay Batyr Ave., Astana, Kazakhstan
francesco.sica@nu.edu.kz

Communicated by Nigel P. Smart

Received 27 August 2012
Online publication 16 January 2013

Abstract. The GLV method of Gallant, Lambert, and Vanstone (CRYPTO 2001) computes any multiple kP of a point P of prime order n lying on an elliptic curve with a low-degree endomorphism Φ (called GLV curve) over $\mathbb{F}_p$ as

$$kP = k_1 P + k_2 \Phi(P) \quad \text{with } \max\bigl\{|k_1|, |k_2|\bigr\} \le C_1\sqrt{n}$$

for some explicit constant $C_1 > 0$. Recently, Galbraith, Lin, and Scott (EUROCRYPT 2009) extended this method to all curves over $\mathbb{F}_{p^2}$ which are twists of curves defined over $\mathbb{F}_p$. We show in this work how to merge the two approaches in order to get, for twists of any GLV curve over $\mathbb{F}_{p^2}$, a four-dimensional decomposition together with fast endomorphisms Φ, Ψ over $\mathbb{F}_{p^2}$ acting on the group generated by a point P of prime order n, resulting in a proven decomposition for any scalar $k \in [1, n]$ given by

$$kP = k_1 P + k_2 \Phi(P) + k_3 \Psi(P) + k_4 \Psi\Phi(P) \quad \text{with } \max_i\bigl(|k_i|\bigr) < C_2 n^{1/4}$$

for some explicit $C_2 > 0$. Remarkably, taking the best C_1, C_2, we obtain $C_2/C_1 < 412$, independently of the curve, ensuring in theory an almost constant relative speedup. In practice, our experiments reveal that the use of the merged GLV–GLS approach supports a scalar multiplication that runs up to 1.5 times faster than the original GLV method. We then improve this performance even further by exploiting the Twisted Edwards model and show that curves originally slower may become extremely efficient on this model. In addition, we analyze the performance of the method on a multicore setting and describe how to efficiently protect GLV-based scalar multiplication against several side-channel attacks. Our implementations improve the state-of-the-art performance of scalar multiplication on elliptic curves over large prime characteristic fields for a variety of scenarios including side-channel protected and unprotected cases with sequential and multicore execution.

* This is the full version of a paper published at Asiacrypt 2012.

Key words. Elliptic curves, GLV–GLS method, Scalar multiplication, Twisted Edwards curve, Side-channel protection, Multicore computation.

1. Introduction

The Gallant–Lambert–Vanstone (GLV) method is a generic approach to speed up the computation of scalar multiplication on some elliptic curves defined over fields of large prime characteristic. Given a curve with a point P of prime order n, it consists essentially in an algorithm that finds a decomposition of an arbitrary scalar multiplication kP for $k \in [1, n]$ into two scalar multiplications, with the new scalars having only about half the bitlength of the original scalar. This immediately enables the elimination of half the doublings by employing the Straus–Shamir trick for simultaneous point multiplication.

Whereas the original GLV method as defined in [13] works on curves over $\mathbb{F}_p$ with an endomorphism of small degree (GLV curves), Galbraith–Lin–Scott (GLS) in [11] have shown that over $\mathbb{F}_{p^2}$ one can expect to find many more such curves by basically exploiting the action of the Frobenius endomorphism. One can therefore expect that on the particular GLV curves, this new insight will lead to improvements over $\mathbb{F}_{p^2}$. Indeed, the GLS article itself considers four-dimensional decompositions on GLV curves with nontrivial automorphisms (corresponding to the degree one cases) but leaves the other cases open to investigation.

In this work, we generalize the GLS method to *all* GLV curves by exploiting fast endomorphisms Φ, Ψ over $\mathbb{F}_{p^2}$ acting on a cyclic group generated by a point P of prime order n to construct a proven decomposition with no heuristics involved for any scalar $k \in [1, n]$

$$kP = k_1 P + k_2 \Phi(P) + k_3 \Psi(P) + k_4 \Psi\Phi(P) \quad \text{with } \max_i\bigl(|k_i|\bigr) < Cn^{1/4}$$

for some explicitly computable C. In doing this we provide a reduction algorithm for the four-dimensional relevant lattice which runs in $O(\log^2 n)$ by implementing two Cornacchia-type algorithms [9,25], one in $\mathbb{Z}$, the other in $\mathbb{Z}[i]$. The algorithm is remarkably simple to implement and allows us to demonstrate an improved $C = O(\sqrt{s})$ (compared to the value obtained with LLL, which is only $\Omega(s^{3/2})$). Thus, it guarantees a relative speedup practically independent of the curve when moving from a two-dimensional to a four-dimensional GLV method over the same underlying field. If parallel computation is available, then the computation of kP can possibly be implemented (close to) four times faster in this case. When moving from two-dimensional GLV over $\mathbb{F}_p$ to the four-dimensional case over $\mathbb{F}_{p^2}$, our method still guarantees a relative speedup that is *quasi*-uniform among all GLV curves (see Sect. 8 for details). In fact, we present experimental results on different GLV curves that demonstrate that the relative speedup between the original GLV method and the proposed method (termed GLV–GLS in the remainder) is as high as 1.5 times.

Twisted Edwards curves [2] are efficient generalizations of the popular Edwards curves [10], which exhibit high-performance arithmetic. By exploiting this curve model, Galbraith, Lin, and Scott [12] showed that the GLS method can be improved in practice a further 10 %, approximately. Similar findings were later reported by Longa and Gebotys [23] (see also Longa [22]). Galbraith et al. also described how to write down

j-invariant 0 and 1728 curves in Edwards form to combine a four-dimensional decomposition with the fast arithmetic provided by this curve model. We exploit this approach and, most remarkably, lift the restriction to those special curves and show that in practice the GLV–GLS curves discussed in this work may achieve extremely high-performance and become virtually equivalent in terms of speed when written in Twisted Edwards form.

In the last years multiple works have incrementally shown the impact of using the GLS method for high performance [11,16,23]. However, it is still unclear how well the method behaves on settings where side-channel attacks are a threat. Since it is usually assumed that required countermeasures once in place degrade performance significantly, it is also unclear if the GLS method would retain its current superiority in the case of side-channel protected implementations. Here, we study this open problem and describe how to protect implementations based on the GLV–GLS method against timing attacks, cache attacks, and similar ones and still achieve very high performance. The techniques discussed naturally apply to GLV-based implementations in general. Finally, we discuss different strategies to implement GLV-based scalar multiplication on modern multicore processors, and include the case in which countermeasures against side-channel attacks are required.

The presented implementations corresponding to the GLV–GLS method improve the state-of-the-art performance of point multiplication for all the cases under study: protected and unprotected versions with sequential and parallel execution. For instance, on one core of an Intel Core i7-2600 processor and at roughly 128 bits of security, we compute an *unprotected* scalar multiplication in only 91,000 cycles (which is 1.34 times faster than a previous result reported by Hu, Longa, and Xu [16]) and a *side-channel protected* scalar multiplication in only 137,000 cycles (which is 1.42 times faster than the protected implementation presented by Bernstein et al. [3]).

Related Work Recently, a paper by Zhou, Hu, Xu, and Song [32] has shown that it is possible to combine the GLV and GLS approaches by introducing a three-dimensional version of the GLV method, which seems to work to a certain degree, with however no justification but through practical implementations. The first author together with Hu and Xu [16] studied the case of curves with j-invariant 0 and provided a bound for this particular case. Our analysis supplements [16] by considering all GLV curves and providing a unified treatment.

2. The GLV Method

In this section we briefly summarize the GLV method following [29]. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, and P be a point on this curve with prime order n such that the cofactor $h = \#E(\mathbb{F}_q)/n$ is small, say $h \le 4$. Let us consider a nontrivial endomorphism Φ defined over $\mathbb{F}_q$ and its characteristic polynomial $X^2 + rX + s$. In all the examples, r and s are actually small fixed integers, and q is varying in some family. By hypothesis there is only one subgroup of order n in $E(\mathbb{F}_q)$, implying that $\Phi(P) = \lambda P$ for some $\lambda \in [0, n-1]$, since $\Phi(P)$ has order dividing the prime n. In particular, λ is obtained as a root of $X^2 + rX + s$ modulo n.

Define the group homomorphism (the GLV reduction map)

$$\begin{aligned} \mathfrak{f}\colon \mathbb{Z}\times\mathbb{Z} &\to \mathbb{Z}/n \\ (i,j) &\mapsto i+\lambda j \pmod{n}. \end{aligned}$$

Let $\mathcal{K}=\ker\mathfrak{f}$. It is a sublattice of $\mathbb{Z}\times\mathbb{Z}$ of rank 2 since the quotient is finite. Let $\Bbbk>0$ be a constant (depending on the curve) such that we can find v_1, v_2 two linearly independent vectors of $\mathcal{K}$ satisfying $\max\{|v_1|_\infty, |v_2|_\infty\} < \Bbbk\sqrt{n}$, where $|\cdot|_\infty$ denotes the rectangle norm.[1] Express

$$(k,0)=\beta_1 v_1+\beta_2 v_2,$$

where $\beta_i\in\mathbb{Q}$. Then round β_i to the nearest integer $b_i=\lfloor\beta_i\rceil=\lfloor\beta_i+1/2\rfloor$ and let $v=b_1v_1+b_2v_2$. Note that $v\in\mathcal{K}$ and that $u\overset{\text{def}}{=}(k,0)-v$ is short. Indeed by the triangle inequality we have that

$$|u|_\infty \le \frac{|v_1|_\infty+|v_2|_\infty}{2} < \Bbbk\sqrt{n}.$$

If we set $(k_1,k_2)=u$, then we get $k\equiv k_1+k_2\lambda \pmod{n}$ or equivalently $kP=k_1P+k_2\Phi(P)$ with $\max(|k_1|,|k_2|)<\Bbbk\sqrt{n}$.

In [29], the optimal value of $\Bbbk$ (with respect to large values of n, i.e., large fields, keeping X^2+rX+s constant) is determined. Let $\Delta=r^2-4s$ be the discriminant of the characteristic polynomial of Φ. Then the optimal $\Bbbk$ is given by the following result.[2]

Theorem 1 [29, Theorem 4]. *Assuming that n is the norm of an element of $\mathbb{Z}[\Phi]$, the optimal value of $\Bbbk$ is*

$$\Bbbk=\begin{cases} \frac{\sqrt{s}}{2}\left(1+\frac{1}{|\Delta|}\right) & \text{if } r \text{ is odd,} \\ \frac{\sqrt{s}}{2}\sqrt{1+\frac{4}{|\Delta|}} & \text{if } r \text{ is even.} \end{cases}$$

3. The GLS Improvement

In 2009, Galbraith, Lin, and Scott [11] realized that we do not need to have $\Phi^2+r\Phi+s=0$ in $\mathrm{End}(E)$ but only in a subgroup of $E(\mathbb{F})$ for a specific finite field $\mathbb{F}$. In particular, considering $\Psi=\mathrm{Frob}_p$ the p-power Frobenius endomorphism of a curve E defined over $\mathbb{F}_p$, we know that $\Psi^m(P)=P$ for all $P\in E(\mathbb{F}_{p^m})$. While this tells nothing useful if $m=1,2$, it does offer new nontrivial relations for higher-degree extensions. The case $m=4$ is particularly useful here.

In this case, if $P\in E(\mathbb{F}_{p^4})\backslash E(\mathbb{F}_{p^2})$, then $\Psi^2(P)=-P$, and hence on the subgroup generated by P, Ψ satisfies the equation $X^2+1=0$. This implies that if $\Psi(P)$ is a multiple of P (which happens as soon as the order n of P is sufficiently large, say at least

[1] The rectangle norm of (x,y) is by definition $\max(|x|,|y|)$. As remarked in [29], we can replace it by any other metric norm. We will use the term “short” to denote smallness in the rectangle norm.

[2] There is a mistake in [29] in the derivation of $\Bbbk$ for odd values of r. This affects [29, Corollary 1] for curves E_2 and E_3, where the correct values of $\Bbbk$ are respectively $2/3$ and $4\sqrt{2}/7$.

$2p$), we can apply the previous GLV construction and split again a scalar multiplication as $kP = k_1P + k_2\Psi(P)$ with $\max(|k_1|, |k_2|) = O(\sqrt{n})$. Contrast this with the characteristic polynomial of Ψ which is $X^2 - a_pX + p$ for some integer a_p, a nonconstant polynomial to which we cannot apply efficiently the GLV paradigm.

For efficiency reasons, however, one does not work with $E/\mathbb{F}_{p^4}$ directly but with $E'/\mathbb{F}_{p^2}$ isomorphic to E *over* $\mathbb{F}_{p^4}$ but not over $\mathbb{F}_{p^2}$, that is, a quadratic twist over $\mathbb{F}_{p^2}$. In this case, it is possible that $\#E'(\mathbb{F}_{p^2}) = n \geq (p-1)^2$ be prime. Furthermore, if $\psi\colon E' \to E$ is an isomorphism defined over $\mathbb{F}_{p^4}$, then the endomorphism $\Psi = \psi \operatorname{Frob}_p \psi^{-1} \in \operatorname{End}(E')$ satisfies the equation $X^2 + 1 = 0$, and if $p \equiv 5 \pmod 8$, it can be defined over $\mathbb{F}_p$.

This idea is at the heart of the GLS approach, but it only works for curves over $\mathbb{F}_{p^m}$ with $m > 1$, and therefore it does not generalize the original GLV method but rather complements it.

4. Combining GLV and GLS

Let $E/\mathbb{F}_p$ be a GLV curve. As in Sect. 3, we will denote by $E'/\mathbb{F}_{p^2}$ a quadratic twist $\mathbb{F}_{p^4}$-isomorphic to E via the isomorphism $\psi\colon E \to E'$. We also suppose that $\#E'(\mathbb{F}_{p^2}) = nh$ where n is prime and $h \leq 4$. We then have the two endomorphisms of E', $\Psi = \psi \operatorname{Frob}_p \psi^{-1}$ and $\Phi = \psi\phi\psi^{-1}$, with ϕ the GLV endomorphism coming with the definition of a GLV curve. They are both defined over $\mathbb{F}_{p^2}$ since if σ is the nontrivial Galois automorphism of $\mathbb{F}_{p^4}/\mathbb{F}_{p^2}$, then $\psi^\sigma = -\psi$, so that $\Psi^\sigma = \psi^\sigma \operatorname{Frob}_p^\sigma(\psi^{-1})^\sigma = (-\psi)\operatorname{Frob}_p(-\psi^{-1}) = \Psi$, meaning that $\Psi \in \operatorname{End}_{\mathbb{F}_{p^2}}(E')$. Similarly for Φ, where we are using the fact that $\phi \in \operatorname{End}_{\mathbb{F}_p}(E)$. Notice that $\Psi^2 + 1 = 0$ and that Φ has the same characteristic polynomial as ϕ. Furthermore, since we have a large subgroup $\langle P\rangle \subset E'(\mathbb{F}_{p^2})$ of prime order, $\Phi(P) = \lambda P$ and $\Psi(P) = \mu P$ for some $\lambda, \mu \in [1, n-1]$. We will assume that Φ and Ψ, when viewed as algebraic integers, generate disjoint quadratic extensions of $\mathbb{Q}$. In particular, we are not dealing with Example 1 from Appendix A, but this can be treated separately with a quartic twist as described in Appendix B.

Consider the biquadratic (Galois of degree 4, with Galois group $\mathbb{Z}/2 \times \mathbb{Z}/2$) number field $K = \mathbb{Q}(\Phi, \Psi)$. Let $\mathfrak{o}_K$ be its ring of integers. The following analysis is inspired by [29, Sect. 8].

We have $\mathbb{Z}[\Phi, \Psi] \subseteq \mathfrak{o}_K$. Since the degrees of Φ and Ψ are much smaller than n, the prime n is unramified in K, and the existence of λ and μ above means that n splits in $\mathbb{Q}(\Phi)$ and $\mathbb{Q}(\Psi)$, namely that n splits completely in K. There exists therefore a prime ideal $\mathfrak{n}$ of $\mathfrak{o}_K$ dividing $n\mathfrak{o}_K$, such that its norm is n. We can also suppose that $\Phi \equiv \lambda \pmod{\mathfrak{n}}$ and $\Psi \equiv \mu \pmod{\mathfrak{n}}$. The four-dimensional GLV–GLS method works as follows.

Consider the GLV–GLS reduction map F defined by

$$\begin{aligned} F\colon \mathbb{Z}^4 &\to \mathbb{Z}/n \\ (x_1, x_2, x_3, x_4) &\mapsto x_1 + x_2\lambda + x_3\mu + x_4\lambda\mu \pmod{n}. \end{aligned}$$

If we can find four linearly independent vectors $v_1, \dots, v_4 \in \ker F$ with $\max_i |v_i|_\infty \le Cn^{1/4}$ for some constant $C > 0$, then for any $k \in [1, n-1]$, we write

$$(k, 0, 0, 0) = \sum_{j=1}^{4} \beta_j v_j$$

with $\beta_j \in \mathbb{Q}$. As in the GLV method, one performs a Babai rounding to obtain the closest lattice vector $v = \sum_{j=1}^{4} \lfloor \beta_j \rceil v_j$ and defines

$$u = (k, 0, 0, 0) - v = (k_1, k_2, k_3, k_4).$$

We then get

$$kP = k_1 P + k_2 \Phi(P) + k_3 \Psi(P) + k_4 \Psi\Phi(P) \quad \text{with } \max_i\bigl(|k_i|\bigr) \le 2Cn^{1/4}. \qquad (1)$$

We next focus on the study of $\ker F$ in order to find a reduced basis v_1, v_2, v_3, v_4 with an explicit C. We can factor the GLV–GLS map F as

$$\begin{tikzcd}
\mathbb{Z}^4 \arrow[r, "f"] & \mathbb{Z}[\Phi, \Psi] \arrow[r, "\text{reduction}", "\text{mod } \mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi]"'] & \mathbb{Z}/n \\
(x_1, x_2, x_3, x_4) \arrow[r, mapsto] & x_1 + x_2\Phi + x_3\Psi + x_4\Phi\Psi \arrow[r, mapsto] & x_1 + x_2\lambda + x_3\mu + x_4\lambda\mu \pmod n.
\end{tikzcd}$$

Notice that the kernel of the second map (reduction mod $\mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi]$) is exactly $\mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi]$. This can be seen as follows. The reduction map factors as

$$\mathbb{Z}[\Phi, \Psi] \longrightarrow \mathfrak{o}_K \longrightarrow \mathfrak{o}_K/\mathfrak{n} \cong \mathbb{Z}/n,$$

where the first arrow is inclusion, and the second is reduction mod $\mathfrak{n}$ corresponding to reducing the x_i's mod $\mathfrak{n} \cap \mathbb{Z} = n\mathbb{Z}$ and using $\Phi \equiv \lambda$, $\Psi \equiv \mu \pmod{\mathfrak{n}}$. But the kernel of this map consists precisely of elements of $\mathbb{Z}[\Phi, \Psi]$ which are in $\mathfrak{n}$, and that is what we want.

Moreover, since the reduction map is surjective, we obtain an isomorphism $\mathbb{Z}[\Phi, \Psi]/\mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi] \cong \mathbb{Z}/n$, which says that the index of $\mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi]$ inside $\mathbb{Z}[\Phi, \Psi]$ is n. Since the first map f is an isomorphism, we get that $\ker F = f^{-1}(\mathfrak{n} \cap \mathbb{Z}[\Phi, \Psi])$ and that $\ker F$ has index $[\mathbb{Z}^4 : \ker F] = n$ inside $\mathbb{Z}^4$.

We can also produce a basis of $\ker F$ by the following observation. Let $\Phi' = \Phi - \lambda$, $\Psi' = \Psi - \mu$, and hence $\Phi'\Psi' = \Phi\Psi - \lambda\Psi - \mu\Phi + \lambda\mu$. In matrix form,

$$\begin{pmatrix} 1 \\ \Phi' \\ \Psi' \\ \Phi'\Psi' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 \\ -\mu & 0 & 1 & 0 \\ \lambda\mu & -\mu & -\lambda & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \Phi \\ \Psi \\ \Phi\Psi \end{pmatrix}.$$

Since the determinant of the square matrix is 1, we deduce that $\mathbb{Z}[\Phi, \Psi] = \mathbb{Z}[\Phi', \Psi']$. But in this new basis, we claim that

$$\mathfrak{n} \cap \mathbb{Z}\bigl[\Phi', \Psi'\bigr] = n\mathbb{Z} + \mathbb{Z}\Phi' + \mathbb{Z}\Psi' + \mathbb{Z}\Phi'\Psi'.$$

Indeed, reverse inclusion ($\supseteq$) is easy since $\Phi', \Psi', \Phi'\Psi' \in \mathfrak{n}$ and so is n, because $\mathfrak{n}$ divides $n\mathfrak{o}_K$ is equivalent to $\mathfrak{n} \supseteq n\mathfrak{o}_K$. On the other hand, the index of both sides in $\mathbb{Z}[\Phi', \Psi']$ is n, which can only happen, once an inclusion is proved, if the two sides are equal. Using the isomorphism f, we see that a basis of $\ker F \subset \mathbb{Z}^4$ is therefore given by

$$w_1 = (n, 0, 0, 0), w_2 = (-\lambda, 1, 0, 0), w_3 = (-\mu, 0, 1, 0), w_4 = (\lambda\mu, -\mu, -\lambda, 1).$$

The LLL algorithm [20] then finds, for a given basis $w_1, \ldots, w_4$ of $\ker F$, a reduced[3] basis $v_1, \ldots, v_4$ in polynomial time (in the logarithm of the norm of the w_i's) such that (cf. [8, Theorem 2.6.2, p. 85])

$$\prod_{i=1}^{4} |v_i|_\infty \le 8\big[\mathbb{Z}^4 \colon \ker F\big] = 8n. \tag{2}$$

Lemma 1. *Let Φ and Ψ be as defined at the beginning of this section,*

$$\begin{aligned} \mathcal{N} \colon \mathbb{Z}^4 &\to \mathbb{Z} \\ (x_1, x_2, x_3, x_4) &\mapsto \sum_{\substack{i_1, i_2, i_3, i_4 \ge 0 \\ i_1+i_2+i_3+i_4=4}} b_{i_1,i_2,i_3,i_4} x_1^{i_1} x_2^{i_2} x_3^{i_3} x_4^{i_4} \end{aligned}$$

be the norm of an element $x_1 + x_2\Phi + x_3\Psi + x_4\Phi\Psi \in \mathbb{Z}[\Phi, \Psi]$, where the b_{i_1,i_2,i_3,i_4}'s lie in $\mathbb{Z}$. Then, for any nonzero $v \in \ker F$, one has

$$|v|_\infty \ge \frac{n^{1/4}}{(\sum_{\substack{i_1,i_2,i_3,i_4 \\ i_1+i_2+i_3+i_4=4}} |b_{i_1,i_2,i_3,i_4}|)^{1/4}}. \tag{3}$$

Proof. For $v \in \ker F$, we have $\mathcal{N}(v) \equiv 0 \pmod{n}$, and if $v \ne 0$, we must therefore have $|\mathcal{N}(v)| \ge n$. On the other hand, if we did not have (3), then every component of v would be strictly less than the right-hand side, and plugging this upper bound in the definition of $|\mathcal{N}(v)|$ would yield a quantity $< n$, a contradiction. □

Let B be the denominator of the right-hand side of (3). Then (2) and (3) imply that

$$|v_i|_\infty \le 8B^3 n^{1/4}, \quad i = 1, 2, 3, 4. \tag{4}$$

Remark 1. In our case, where $\Psi^2 + 1 = 0$ and $\Phi^2 + r\Phi + s = 0$, we get as norm function

$$\begin{aligned} &x_1^4 + s^2x_2^4 + x_3^4 + s^2x_4^4 - 2rx_1^3x_2 - 2rsx_1x_2^3 - 2rx_3^3x_4 - 2rsx_3x_4^3 \\ &\quad + \big(r^2 + 2s\big)x_1^2x_2^2 + 2x_1^2x_3^2 + \big(r^2 - 2s\big)x_1^2x_4^2 \\ &\quad + \big(r^2 - 2s\big)x_2^2x_3^2 + 2s^2x_2^2x_4^2 + \big(r^2 + 2s\big)x_3^2x_4^2 \\ &\quad - 2rx_1^2x_3x_4 - 2rsx_2^2x_3x_4 - 2rx_1x_2x_3^2 - 2rsx_1x_2x_4^2 + 8sx_1x_2x_3x_4, \end{aligned}$$

[3] The estimates are usually given for the Euclidean norm of the vectors. But it is easy to see that the rectangle norm is upper bounded by the Euclidean norm.

and therefore,

$$B = \bigl(4 + 4s^2 + 8s + 8|r| + 8|r|s + 2\bigl(r^2 + 2s\bigr) + 2|r^2 - 2s|\bigr)^{1/4}. \tag{5}$$

From (1) and (4) we have proved the following theorem.

Theorem 2. *Let $E/\mathbb{F}_p$ be a GLV curve, and $E'/\mathbb{F}_{p^2}$ a twist, together with the two efficient endomorphisms Φ and Ψ, where everything is defined as at the start of this section. Suppose that the minimal polynomial of Φ is $X^2 + rX + s = 0$. Let $P \in E'(\mathbb{F}_{p^2})$ be a generator of the large subgroup of prime order n. There exists an efficient algorithm which for any $k \in [1, n]$ finds integers k_1, k_2, k_3, k_4 such that*

$$kP = k_1 P + k_2 \Phi(P) + k_3 \Psi(P) + k_4 \Psi\Phi(P) \quad \textit{with} \ \max_i\bigl(|k_i|\bigr) \le 16B^3 n^{1/4}$$

and

$$B = \bigl(4 + 4s^2 + 8s + 8|r| + 8|r|s + 2\bigl(r^2 + 2s\bigr) + 2\bigl|r^2 - 2s\bigr|\bigr)^{1/4}.$$

5. Uniform Improvements and a Tale of Two Cornacchia Algorithms

The previous analysis is only the first step of our work. It shows that the GLV–GLS method works as predicted in a four-way decomposition on twists of GLV curves over $\mathbb{F}_{p^2}$. However, the constant B^3 involved is rather large and, hence, does not guarantee a non-negligible gain when switching from two to four dimensions (especially on those GLV curves with more complicated endomorphism rings). A much deeper argument allows us to prove the following result.

Theorem 3. *When performing an optimal lattice reduction on* $\ker F$, *it is possible to decompose any $k \in [1, n]$ into integers k_1, k_2, k_3, k_4 such that*

$$kP = k_1 P + k_2 \Phi(P) + k_3 \Psi(P) + k_4 \Psi\Phi(P)$$

with $\max_i(|k_i|) < 103(\sqrt{1 + |r| + s})\, n^{1/4}$.

The significance of this theorem lies in the improvement of the constant $16B^3$, which is $\Omega(s^{3/2})$ in Theorem 2, to a value that is an absolute constant times greater than the minimal bound for the two-dimensional GLV method (Theorem 1). Hence, this guarantees in practice a more uniform improvement when switching from two-dimensional to four-dimensional GLV independently of the curve.

To prove Theorem 3, first note that Lemma 1 gives a rather poor bound when applied to more than one vector, as is done three times for the proof of Theorem 2. A more direct treatment of the reduced vectors of $\ker F$ becomes necessary, and this is done via a modification of the original GLV approach. This results in a new, easy-to-implement lattice reduction algorithm which employs two Cornacchia-type algorithms [8, Sect. 1.5.2], one in $\mathbb{Z}$ (as in the original GLV method), the other one in $\mathbb{Z}[i]$ (Gaussian Cornacchia).

The full proof of Theorem 3 via the new lattice reduction algorithm can be found in Appendix D.

5.1. *The Euclidean Algorithm in $\mathbb{Z}$*

The first step is to find $\nu = a + ib \in \mathbb{Z}[i]$ such that $|\nu|^2 = a^2 + b^2 = n$, i.e., a Gaussian prime above n. Recall that n splits in $\mathbb{Z}[i]$. Let $\nu = a + ib$ a prime above n. We can furthermore assume that $\nu P = aP + biP := aP + b\Psi(P) = 0$ since $\nu\bar{\nu}P = nP = 0$, and hence either $\bar{\nu}P$ is a nonzero multiple of P and therefore $\nu P = 0$, or else $\bar{\nu}P = 0$, so that in any case one of the Gaussian primes (WLOG ν) above n will have $\nu P = 0$. We can find ν by Cornacchia's algorithm [8, Sect. 1.5.2], which is a truncated form of the Euclidean algorithm. For completeness and consistency with what will follow, we recall how this is done.

Let $\mu \in [1, n]$ be such that $\mu \equiv i \pmod{n}$, with i being defined by $\Psi(P) = iP$. Actually, in the GLS approach [11], it has been pointed out that this value of μ can be readily computed from $\#E(\mathbb{F}_p)$. The extended Euclidean algorithm to compute the gcd of n and μ produces three terminating sequences of integers $(r_j)_{j\geq 0}$, $(s_j)_{j\geq 0}$ and $(t_j)_{j\geq 0}$ such that

$$\begin{pmatrix} r_{j+2} & s_{j+2} & t_{j+2} \\ r_{j+1} & s_{j+1} & t_{j+1} \end{pmatrix} = \begin{pmatrix} -q_{j+1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{j+1} & s_{j+1} & t_{j+1} \\ r_j & s_j & t_j \end{pmatrix}, \quad j \geq 0, \tag{6}$$

for some integer $q_{j+1} > 0$ and initial data

$$\begin{pmatrix} r_1 & s_1 & t_1 \\ r_0 & s_0 & t_0 \end{pmatrix} = \begin{pmatrix} \mu & 0 & 1 \\ n & 1 & 0 \end{pmatrix}. \tag{7}$$

This means that at step $j \geq 0$,

$$r_j = q_{j+1} r_{j+1} + r_{j+2}$$

and similarly for the other sequences. The sequence $(q_j)_{j\geq 1}$ is *uniquely* defined by imposing that the previous equation be the integer division of r_j by r_{j+1}. In other terms, $q_{j+1} = \lfloor r_j / r_{j+1} \rfloor$. This implies by induction that all the sequences are well defined in the integers, together with the following properties.

Lemma 2. *The sequences $(r_j)_{j\geq 0}$, $(s_j)_{j\geq 0}$, and $(t_j)_{j\geq 0}$ defined by (6) and (7) with $q_{j+1} = \lfloor r_j / r_{j+1} \rfloor$ satisfy the following properties, valid for all $j \geq 0$.*

1. $r_j > r_{j+1} \geq 0$ *and* $q_{j+1} \geq 1$,
2. $(-1)^j s_j \geq 0$ *and* $|s_j| < |s_{j+1}|$ (*this last inequality valid for* $j \geq 1$),
3. $(-1)^{j+1} t_j \geq 0$ *and* $|t_j| < |t_{j+1}|$,
4. $s_{j+1} r_j - s_j r_{j+1} = (-1)^{j+1} r_1$,
5. $t_{j+1} r_j - t_j r_{j+1} = (-1)^j r_0$,
6. $r_0 s_j + r_1 t_j = r_j$.

These properties lie at the heart of the original GLV algorithm. They imply in particular via property 1 that the algorithm terminates (once r_j reaches zero) and that it has $O(\log n)$ steps, as $r_j = q_{j+1}r_{j+1} + r_{j+2} \geq r_{j+1} + r_{j+2} > 2r_{j+2}$. Note that properties 1, 2, and 3 imply that properties 4 and 5 can be rewritten in our case respectively as

$$|s_{j+1}r_j| + |s_j r_{j+1}| = \mu \quad \text{and} \quad |t_{j+1}r_j| + |t_j r_{j+1}| = n. \tag{8}$$

The Cornacchia (as well as the GLV) algorithm does not make use of the full sequences $(r_j), (s_j)$, and (t_j) but rather stops at the $m \geq 0$ such that $r_m \geq \sqrt{n}$ and $r_{m+1} < \sqrt{n}$. An application of (8) with $j = m$ yields $|t_{m+1}r_m| < n$ or $|t_{m+1}| < \sqrt{n}$. Since by property 6 we have $r_{m+1} - \mu t_{m+1} = ns_{m+1} \equiv 0 \pmod{n}$, we deduce that $r_{m+1}^2 + t_{m+1}^2 = (r_{m+1} - \mu t_{m+1})(r_{m+1} + \mu t_{m+1}) \equiv 0 \pmod{n}$. Moreover, $t_{m+1} \neq 0$ by property 3, so that $0 < r_{m+1}^2 + t_{m+1}^2 < n + n = 2n$, which therefore implies that $r_{m+1}^2 + t_{m+1}^2 = n$ and finally that $\nu = r_{m+1} - it_{m+1}$.

We present here the pseudo-code of this Euclidean algorithm in $\mathbb{Z}$.

Algorithm 1 (Cornacchia's GCD in $\mathbb{Z}$).

```
Input: $n \equiv 1 \pmod 4$ prime, $1 < \mu < n$ such that $\mu^2 \equiv -1 \pmod n$.
Output: $\nu = \nu_{(R)} + i\nu_{(I)}$ Gaussian prime dividing $n$, such that
$\nu P = 0$.

1. initialize:
   $r_0 \leftarrow n, r_1 \leftarrow \mu, r_2 \leftarrow n,$
   $t_0 \leftarrow 0, t_1 \leftarrow 1, t_2 \leftarrow 0,$
   $q \leftarrow 0.$
2. main loop:
   while $r_2^2 \geq n$ do
       $q \leftarrow \lfloor r_0/r_1 \rfloor,$
       $r_2 \leftarrow r_0 - qr_1, r_0 \leftarrow r_1, r_1 \leftarrow r_2,$
       $t_2 \leftarrow t_0 - qt_1, t_0 \leftarrow t_1, t_1 \leftarrow t_2.$
3. return:
   $\nu = r_1 - it_1, \nu_{(R)} = r_1, \nu_{(I)} = -t_1$
```

5.2. *The Euclidean Algorithm in* $\mathbb{Z}[i]$

In the previous subsection we have given a meaning to zP, where $z \in \mathbb{Z}[i]$, and we have seen how to construct ν, a Gaussian prime such that $\nu P = 0$. By identifying[4] $(x_1, x_2, x_3, x_4) \in \mathbb{Z}^4$ with $(z_1, z_2) = (x_1 + ix_3, x_2 + ix_4) \in \mathbb{Z}[i]^2$, we can rewrite the 4-GLV reduction map F of Sect. 4 as (using the same letter F by abuse of notation)

$$F : \mathbb{Z}[i]^2 \to \mathbb{Z}[i]/\nu \cong \mathbb{Z}/n$$
$$(z_1, z_2) \mapsto z_1 + \lambda z_2 \pmod{\nu}.$$

This F should be compared with the map $\mathfrak{f}$ of Sect. 2. In mimicking the GLV original paper [13] we would like to apply the extended Euclidean algorithm (defined exactly as

[4] It is important to keep in mind that this association is only an isomorphism of abelian groups ($\mathbb{Z}$-modules). However, $\mathbb{Z}[i]^2$ is also endowed with a structure of $\mathbb{Z}[i]$-module.

before, with integer divisions occurring in $\mathbb{Z}[i]$, henceforth denoted EGEA in short for extended Gaussian Euclidean algorithm) to the pair $(r_0, r_1) = (\lambda, \nu)$ if $\lambda \geq \sqrt{2}\,|\nu|$ and $(r_0, r_1) = (\lambda + n, \nu)$ otherwise (the latter case being exceptionally rare). This should output short vectors in $\mathbb{Z}[i]^2$, which we can transform into short vectors in $\mathbb{Z}^4$ using the previous isomorphism, thus proving Theorem 3 by the Babai rounding argument given in Sect. 4.

What are the difficulties in following this path? Let us note that properties 4, 5, and 6 of Lemma 2 still hold and property 1 holds in modulus (in particular, the algorithm terminates). However, in the analysis of this algorithm, especially in [29], a crucial role is played by (8), in order to derive a bound on $|s_{j+1}r_j|$ and $|s_j r_{j+1}|$ from a bound on

$$s_{j+1}r_j - s_j r_{j+1} = (-1)^{j+1}\nu \tag{9}$$

in the present case. This fact, as we saw, stems from the alternating sign of the sequence (s_j), which results from taking a canonical form of integer division with positive quotients q_{j+1} and nonnegative remainders r_{j+2}, a property which is not available here. Nevertheless, we can still use a similar reasoning using (9), provided that the arguments of $s_{j+1}r_j$ and $s_j r_{j+1}$ are not too close, so as to avoid a high degree of cancellation. In other terms, in order to follow the argument of [29, Theorem 1], we need a property of the kind

$$|s_{j+1}r_j - s_j r_{j+1}| \leq M \quad \Longrightarrow \quad \max\bigl(|s_{j+1}r_j|, |s_j r_{j+1}|\bigr) \leq cM$$

for some explicit absolute constant c (equal to 1 in [29]). This is in general impossible to attain because in the EGEA, in contrast to the usual extended Euclidean algorithm, we have no control over the arguments of the r_j's or the s_j's. However, *in most cases* something of the sort can be proved. This is the content of Lemma 4 (Appendix D). We define the corresponding indices (terms) of the sequences r_j, s_j as "good" when this happens. If all the terms were good, then the proof of [29, Theorem 1] could be carried over to proving Theorem 3 without almost any change (the final constant of the theorem would be different, depending on c). However, this is not the case, and the main difficulty here lies in the treatment of the terms which are not good (called therefore "bad"). The surprising fact is that we can still control the contribution of bad terms to our advantage (see Lemma 5) and, ultimately, the combination of Lemmas 4 and 5 becomes the main ingredient in the proof of Theorem 3. All above makes the reasoning noticeably more sophisticated than in [29].

We now turn to the description of the EGEA. The first observation is that in the case of Gaussian integers, there can be 2, 3, or 4 possible choices for a remainder in the jth step of the integer division $r_j = q_{j+1}r_{j+1} + r_{j+2}$. It turns out that choosing at each step $j \geq 0$ of the EGEA a remainder r_{j+2} with smallest modulus will yield Theorem 3.

We give the pseudo-code of Cornacchia's Algorithm in $\mathbb{Z}[i]$ in two forms, working with complex numbers (see Algorithm 2) and separating real and imaginary parts (see Algorithm 3, Appendix C).

Algorithm 2 (EGEA or Cornacchia's algorithm in $\mathbb{Z}[i]$—compact form).

```
Input: $\nu$ Gaussian prime dividing $n$ rational prime, $1 < \lambda < n$
such that $\lambda^2 + r\lambda + s \equiv 0 \pmod{n}$.
Output: Two $\mathbb{Z}[i]$-linearly independent vectors $v_1$ & $v_2$ of $\ker F \subset$
$\mathbb{Z}[i]^2$ of rectangle norms $< 51.5(\sqrt{1+|r|+s})n^{1/4}$.

1. initialize:
   If $\lambda^2 \geq 2n$ then
       $r_0 \leftarrow \lambda$,
   else
       $r_0 \leftarrow \lambda + n$,
   $r_1 \leftarrow \nu$, $r_2 \leftarrow n$,
   $s_0 \leftarrow 1$, $s_1 \leftarrow 0$, $s_2 \leftarrow 0$,
   $q \leftarrow 0$.
2. main loop:
   while $|r_2|^4(1+|r|+s)^2 \geq n$ do
       $q \leftarrow$ closest Gaussian integer to $r_0/r_1$,
       $r_2 \leftarrow r_0 - qr_1$, $r_0 \leftarrow r_1$, $r_1 \leftarrow r_2$,
       $s_2 \leftarrow s_0 - qs_1$, $s_0 \leftarrow s_1$, $s_1 \leftarrow s_2$.
3. return:
   $v_1 = (r_0, -s_0)$, $v_2 = (r_1, -s_1)$
```

Remark 2. In the case of the LLL algorithm, we have not managed to demonstrate a bound as good as the one obtained with our lattice reduction algorithm.

Remark 3. Nguyen and Stehlé [26] have produced an efficient lattice reduction in four dimensions which finds successive minima and hence produces a decomposition with relatively good bounds. Our algorithm represents a very simple and easy-to-implement alternative that may be ideal for certain cryptographic libraries.

6. GLV–GLS using the Twisted Edwards Model

The GLV–GLS method can be sped up in practice by writing down GLV–GLS curves in the Twisted Edwards model. Note that arithmetic on j-invariant 0 Weierstrass curves is already very efficient. However, some GLV curves do not exhibit such high-speed arithmetic. In particular, curves in Examples 3–6 from Appendix A have Weierstrass coefficients $a_4 \cdot a_6 \neq 0$ for curve parameters a_4 and a_6, and hence they have more expensive point doubling (even more if we consider the extra multiplication by the twisted parameter u when using the GLS method). So the impact of using Twisted Edwards is expected to be especially significant for these curves. In fact, if we consider that suitable parameters can be always chosen, the use of Twisted Edwards curves isomorphic to the original Weierstrass GLV–GLS curves *uniformizes* the performance of all of them.

Let us illustrate how to produce a Twisted Edwards GLV–GLS curve with the GLV curve from Example 4, Appendix A. First, consider its quadratic twist over $\mathbb{F}_{p^2}$

$$E'/\mathbb{F}_{p^2}\colon x^3 - \frac{15}{2}u^2x - 7u^3 = (x+2u) \cdot \left(x^2 - 2ux - \frac{7}{2}u^2\right).$$

The change of variables $x_1 = x + 2u$ transforms E' into

$$y^2 = x_1^3 - 6ux_1^2 + \frac{9u^2}{2}x_1.$$

Let $\beta = 3u/\sqrt{2} \in \mathbb{F}_{p^2}$ and substitute $x_1 = \beta x'$ to get

$$\frac{1}{\beta^3}y^2 = x'^3 - \frac{6u}{\beta}x'^2 + x',$$

and this is a Montgomery curve $M_{A,B} : Bv^2 = u^3 + Au^2 + u$, where $A \neq \pm 2$, $B \neq 0$, with

$$B = \frac{1}{\beta^3} = \frac{2\sqrt{2}}{27u^3}, \qquad A = -\frac{6u}{\beta} = -2\sqrt{2}.$$

The corresponding Twisted Edwards GLV–GLS curve is then $E_{a,d}$: $ax^2 + y^2 = 1 + dx^2y^2$ with

$$a = \frac{A+2}{B} = 27u^3\left(\frac{\sqrt{2}}{2} - 1\right), \qquad d = \frac{A-2}{B} = -27u^3\left(\frac{\sqrt{2}}{2} + 1\right).$$

The map $E' \to E_{a,d}$ is

$$(x, y) \mapsto \left(\frac{x+2u}{\beta y}, \frac{x+2u-\beta}{x+2u+\beta}\right) = (X, Y)$$

with inverse

$$(X, Y) \mapsto \left(\frac{\beta - 2u + (\beta + 2u)Y}{1 - Y}, \frac{1+Y}{(1-Y)X}\right).$$

We now specify the formulas for Φ and Ψ, obtained by composing these endomorphisms on the Weierstrass model with the birational maps above. We found an extremely appealing expression in the case where $u = 1 + i$ and $i^2 = -1$. Then $\beta = 3u/\sqrt{2} = 3\zeta_8$ where ζ_8 is a primitive 8th root of unity. We have

$$\Phi(X, Y) = \left(-\frac{(\zeta_8^3 + 2\zeta_8^2 + \zeta_8)XY^2 + (\zeta_8^3 - 2\zeta_8^2 + \zeta_8)X}{2Y}, \frac{(\zeta_8^2 - 1)Y^2 + 2\zeta_8^3 - \zeta_8^2 + 1}{(2\zeta_8^3 + \zeta_8^2 - 1)Y^2 - \zeta_8^2 + 1}\right)$$

and

$$\Psi(X, Y) = \left(\zeta_8 X^p, \frac{1}{Y^p}\right).$$

In this case,

$$a = 54\left(\zeta_8^3 - \zeta_8^2 + 1\right), \qquad d = -54\left(\zeta_8^3 + \zeta_8^2 - 1\right).$$

Finally, one would want to use the efficient formulas given in [15] for the case $a = -1$. After ensuring that $-a$ be a square in $\mathbb{F}_{p^2}$, we use the map $(x, y) \mapsto (x/\sqrt{-a}, y)$ to convert to the isomorphic curve $-x^2 + y^2 = 1 + d'x^2y^2$, where $d' = -d/a$.

7. Side-Channel Protection and Parallelization of the GLV–GLS Method

Given the potential threat posed by attacks that exploit timing information to deduce secret keys ([7,19]), many works have proposed countermeasures to minimize the risks and achieve the so-called constant-time execution during cryptographic computations. In general, to avoid leakage, the execution flow should be independent of the secret key. This means that conditional branches and secret-dependent table lookup indices should be avoided [4,18]. There are *five* key points that are especially vulnerable during the computation of scalar multiplication: inversion, modular reduction in field operations, precomputation, scalar recoding, and double-and-add execution.

A well-known technique that is secure and easy to implement for inverting any field element a consists of computing the exponentiation $a^{p-2} \bmod p$ using a short addition chain for $p - 2$.

To protect field operations, one may exploit conditional move instructions typically found on modern x86 and x64 processors (a.k.a. `cmove`). Since conditional checks happen during operations such as addition and subtraction as part of the reduction step, it is standard practice to replace conditional branches with the conditional move instruction. Luckily, these conditional branches are highly unpredictable, and, hence, the substitution above does not only make the execution constant-time but also more efficient in most cases. An exception happens when performing modular reduction during a field multiplication or squaring, where a final correction step could happen very rarely, and hence a conditional branch may be more efficient.

For the case of precomputation in the setting of elliptic curves, recent work [18] and later [3] showed how to enable the use of precomputed points by employing constant-time table lookups that mask the extraction of points, which is a known technique in the literature (see, for example, [5]). In our implementations (see Sect. 8), we exploit a similar approach based on `cmove` and conditional vector instructions instead, which is expected to achieve higher performance on some platforms than implementations based on logical instructions (see Listing 1 in [18]). Note that it is straightforward to enable the use of signed-digit representations that allow negative points by performing a second table lookup between the point selected in the first table lookup and its negated value.

To protect the scalar recoding and its corresponding double-and-add algorithm, one needs a regular pattern execution. Based on a method in [27], Joye and Tunstall [17] proposed a constant-time recoding that supports a regular execution double-and-add algorithm that exploits precomputations. The nonzero density of the method is $1/(w - 1)$, where w is the window width. Therefore, there is certain loss in performance in comparison with an unprotected version with nonzero density $1/(w + 1)$. In GLV-based implementations one has to deal with more than one scalar, and these scalars are scanned simultaneously, using interleaving [13], for instance, during multi-exponentiation. So there are two issues that arise. First, how are the several scalars aligned with respect to their zero and nonzero digit representation? And second, how do we guarantee the same representation length for all scalars so that no dummy operations are required? The first issue is inherently solved by the recoding algorithm itself. The input is always an odd number, which means that, from left to right, one obtains the execution pattern $(w-1)$ doublings, d additions, $(w-1)$ doublings, d additions, $\ldots$, $(w-1)$ doublings and d additions, for d-dimensional GLV. For dealing with even numbers, one may employ the technique described in [17] in a constant-time fashion, namely, scalars k_i that are even are replaced by k_i+1 and scalars that are odd are replaced by k_i+2 (the correction, also constant-time, is performed after the scalar multiplication computation using d point additions). A solution to the second issue was also hinted by [17]. We present in Appendix E the modified recoding algorithm that outputs a regular pattern representation with fixed length. Note that in the case of Twisted Edwards, one can alternatively use *unified* addition formulas that also work for doubling (see [2,15] for details). However, our analysis indicates that this approach is consistently slower because of the high cost of these unified formulas in comparison to doubling and the extra cost incurred by the increase in constant-time table lookup accesses.

7.1. *Multicore Computation and Its Side-Channel Protection*

Parallelization of scalar multiplication over prime fields is particularly difficult on modern multicore processors. This is due to the difficulty to perform point operations concurrently when executing the double-and-add algorithm from left to right. From right to left parallelization is easier, but performance is hurt because the use of precomputations is cumbersome. Hence, parallelization should be ideally performed at the field arithmetic level. Unfortunately, current multicore processors still impose a severe overhead for thread creation/destruction. During our tests, we observed overheads of a few thousands of cycles on modern 64-bit CPUs (that is, much more costly than a point addition or doubling). Given this limitation, for the GLV method, it seems the ideal approach (from a speed perspective) to let each core manage a separate scalar multiplication with k_i. This is simple to implement, minimizes thread management overhead and also eases the task of protecting the implementation against side-channel attacks since each scalar can be recoded using Algorithm 4, Appendix E. Using d cores, the total cost of a protected d-dimensional GLV l-bit scalar multiplication (disregarding precomputation) is approximately l/d doublings and $l/((w-1)\cdot d)$ mixed additions. A somewhat slower approach (but more power efficient) would be to let one core manage all doublings and let one or two extra cores manage the additions corresponding to nonzero digits. For instance, for dimension four and three cores, the total cost (disregarding precomputation) is approximately l/d doublings and $l/((w-1)\cdot d)$ *general* additions, always that the latency of $(w-1)$ doublings be equivalent or greater than the addition part (otherwise, the cost is dominated by nonmixed additions).

8. Performance Analysis and Experimental Results

For our analysis and experiments, we consider the *five* curves below: two GLV curves in Weierstrass form with and without nontrivial automorphisms, their corresponding GLV–GLS counterparts, and one curve in Twisted Edwards form isomorphic to the GLV–GLS curve E_3' (see below).

- GLV–GLS curve with j-invariant 0 in Weierstrass form $E_1'/\mathbb{F}_{p_1^2} : y^2 = x^3 + 9u$, where $p_1 = 2^{127} - 58309$ and $\#E_1'(\mathbb{F}_{p_1^2}) = r$, where r is a 254-bit prime. We use $\mathbb{F}_{p_1^2} = \mathbb{F}_{p_1}[i]/(i^2+1)$ and $u = 1 + i \in \mathbb{F}_{p_1^2}$. E_1' is the quadratic twist of the curve in Example 2, Appendix A. $\Phi(x,y) = \lambda P = (\xi x, y)$ and $\Psi(x,y) = \mu P = (u^{(1-p)/3}x^p, u^{(1-p)/2}y^p)$, where $\xi^3 = 1 \mod p_1$. We have that $\Phi^2 + \Phi + 1 = 0$ and $\Psi^2 + 1 = 0$.
- GLV curve with j-invariant 0 in Weierstrass form $E_2/\mathbb{F}_{p_2} : y^2 = x^3 + 2$, where $p_2 = 2^{256} - 11733$, and $\#E_2(\mathbb{F}_{p_2})$ is a 256-bit prime. This curve corresponds to Example 2, Appendix A.
- GLV–GLS curve in Weierstrass form $E_3'/\mathbb{F}_{p_3^2} : y^2 = x^3 - 15/2\, u^2 x - 7u^3$, where $p_3 = 2^{127} - 5997$ and $\#E_3'(\mathbb{F}_{p_3^2}) = 8r$, where r is a 251-bit prime. We use $\mathbb{F}_{p_3^2} = \mathbb{F}_{p_3}[i]/(i^2+1)$ and $u = 1+i \in \mathbb{F}_{p_3^2}$. E_3' is the quadratic twist of a curve isomorphic to the one in Example 4, Appendix A. The formula for $\Phi(x,y) = \lambda P$ can be easily derived from $\psi(x,y)$, and $\Psi(x,y) = \mu P = (u^{(1-p)}x^p, u^{3(1-p)/2}y^p)$. It can be verified that $\Phi^2 + 2 = 0$ and $\Psi^2 + 1 = 0$.
- GLV–GLS curve in Twisted Edwards form $E_{T3}'/\mathbb{F}_{p_3^2} : -x^2 + y^2 = 1 + dx^2y^2$, where

$$d = 170141183460469231731687303715884099728$$
$$+ 116829086847165810221872975542241037773\, i,$$

  $p_3 = 2^{127} - 5997$ and $\#E_{T3}'(\mathbb{F}_{p_3^2}) = 8r$, where r is a 251-bit prime. We use again $\mathbb{F}_{p_3^2} = \mathbb{F}_{p_3}[i]/(i^2+1)$ and $u = 1 + i \in \mathbb{F}_{p_3^2}$. E_{T3}' is isomorphic to curve E_3' above and was obtained following the procedure in Sect. 6. The formulas for $\Phi(x,y)$ and $\Psi(x,y)$ are also given in Sect. 6. It can be verified that $\Phi^2 + 2 = 0$ and $\Psi^2 + 1 = 0$.
- GLV curve $E_4/\mathbb{F}_{p_4} : y^2 = x^3 - 15/2\, x - 7$, where $p_4 = 2^{256} - 45717$ and $\#E_4(\mathbb{F}_{p_4}) = 2r$, where r is a 256-bit prime. This curve is isomorphic to the curve in Example 4, Appendix A.

For our experiments, we also explored the case of $p = 2^{128} - c$, with a relatively small integer c, for GLV–GLS curves. We finally decided on $p = 2^{127} - c$ because it was consistently faster thanks to the use of lazy reduction in the multiplication over $\mathbb{F}_{p^2}$ [21] at the expense of a slight reduction in security.

Let us first analyze the performance of the GLV–GLS method over $\mathbb{F}_{p^2}$ in comparison with the traditional 2-GLV case over $\mathbb{F}_p$. We assume the use of a pseudo-Mersenne prime of the form $p = 2^m - c$ with small c (for our targeted curves, groups with (near) prime order cannot be constructed using the attractive Mersenne prime $p = 2^{127} - 1$). Given that we have a proven ratio $C_2/C_1 < 412$ that is independent of the curve, the only values left that could affect significantly a uniform speedup between GLV–GLS and 2-GLV are the quadratic nonresidue β used to build $\mathbb{F}_{p^2}$ as $\mathbb{F}_p[i]/(i^2 - \beta)$, the value of the twisting parameter u, and the cost of applying the endomorphisms Φ and Ψ. In particular, if $|\beta| > 1$, a few extra additions (or a multiplication by a small constant) are required per $\mathbb{F}_{p^2}$ multiplication and squaring. Luckily, for all the GLV curves listed in Appendix A, one can always use a suitably chosen modulus p so that $|\beta|$ can be one

Table 1. Operation counts and performance for scalar multiplication at approximately 128 bits of security. To determine the total costs, we consider 1i = 66m, 1s = 0.76m, and 1a = 0.18m for E'_1, E'_3, and E'_{T3}; and 1I = 290M, 1S = 0.85M, and 1A = 0.18M for E_2 and E_4. The cost ratio of multiplications over $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ is M/m = 0.91. These values and the performance figures (in cycles) were obtained by benchmarking full implementations on a single core of a 3.4 GHz Intel Core i7-2600 (Sandy Bridge) processor.

Curve	Method	Operation Count	Total Cost	Gain	Performance	Gain
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 32pts.	2i + 617m + 404s + 847a	1209m	51 %	99,000cc	53 %
$E_2(\mathbb{F}_{p_2})$	2-GLV, 16pts.	1I + 904M + 690S + 1240A	2004M ≈ 1824m	–	151,000cc	–
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	1i + 742m + 225s + 767a	1117m	97 %	91,000cc	102 %
$E'_3(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	2i + 678m + 581s + 1200a	1468m	50 %	121,000cc	52 %
$E_4(\mathbb{F}_{p_4})$	2-GLV, 16pts.	1I + 950M + 970S + 1953A	2416M ≈ 2199m	–	184,000cc	–

or at least very close to it. Similar comments apply to the twisting parameter u. In this case, the extra cost (equivalent to a few additions) is added to the cost of point doubling always that the curve parameter a in the Weierstrass equation be different to zero (e.g., it does not affect j-invariant 0 curves). In the case of Twisted Edwards, we applied a better strategy, that is, we eliminated the twisting parameter u in the isomorphic curve. The cost of applying Φ and Ψ does depend on the chosen curve, and it could be relatively expensive. If computing $\Phi(P)$, $\Psi(P)$ or $\Psi\Phi(P)$ is more expensive than point addition, then its use can be limited to only one application (i.e., multiples of those values—if using precomputations—should be computed with point additions). Further, the extra cost can be minimized by choosing the optimal window width for each k_i.

To illustrate how the parameters above may affect the performance gain, we detail in Table 1 estimates for the cost of computing a scalar multiplication with our representative curves. For the remainder, we use the following notation: M, S, A, and I represent field multiplication, squaring, addition, and inversion over $\mathbb{F}_p$, respectively, and m, s, a, and i represent the same operations over $\mathbb{F}_{p^2}$. Side-channel protected multiplication and squaring are denoted by m_s and s_s. We consider the cost of addition, subtraction, negation, multiplication by 2, and division by 2 as equivalent. For the targeted curves in Weierstrass form, a mixed addition consists of 8 multiplications, 3 squarings, and 7 additions, and a general addition consists of 12 multiplications, 4 squarings, and 7 additions. For E'_1 and E_2, a doubling consists of 3 multiplications, 4 squarings, and 7 additions, and for E'_3 and E_4, a doubling consists of 3 multiplications, 6 squarings, and 12 additions. For Twisted Edwards, we consider the use of mixed homogeneous/extended homogeneous projective coordinates [15]. In this case, a mixed addition consists of 7 multiplications and 7 additions, a general addition consists of 8 multiplications and 6 or 7 additions, and a doubling consists of 4 multiplications, 3 squarings, and 5 additions. We also assume the use of interleaving [13] with width-w nonadjacent form (wNAF) and the use of the LM scheme for precomputing points on the Weierstrass curves [24] (see also [22, Chap. 3]).

According to our theoretical estimates, it is expected that the relative speedup when moving from 2-GLV to GLV-GLS be as high as 1.5 times, approximately. To confirm our findings, we realized full implementations of the methods. Experimental results, also displayed in Table 1, closely follow our estimates and confirm that speedups in practice are about 1.52 times. Most remarkably, the use of the Twisted Edwards model

Table 2. Operation counts for scalar multiplication at approximately 128 bits of security using curves E'_1, E_2, and E'_{T3} in up to four variants: unprotected and side-channel protected implementations with sequential and multicore execution. To determine the total costs we consider $1\mathrm{i} = 66\mathrm{m}$, $1\mathrm{s} = 0.76\mathrm{m}$, and $1\mathrm{a} = 0.18\mathrm{m}$ for unprotected versions of E'_1 and E'_{T3}; $1\mathrm{i} = 79\mathrm{m}_s$, $1\mathrm{s}_s = 0.81\mathrm{m}_s$, and $1\mathrm{a} = 0.17\mathrm{m}_s$ for protected versions of E'_1 and E'_{T3}; $\mathrm{t} = 0.83\mathrm{m}_s$ for E'_1 (32pts.); $\mathrm{t} = 1.28\mathrm{m}_s$ for E'_{T3} (36pts.); $\mathrm{t} = 0.78\mathrm{m}_s$ for E'_{T3} (20pts.); and $1\mathrm{I} = 290\mathrm{M}$, $1\mathrm{S} = 0.85\mathrm{M}$, and $1\mathrm{A} = 0.18\mathrm{M}$ for E_2. In our case, $\mathrm{M}/\mathrm{m} = 0.91$ and $\mathrm{m}_s/\mathrm{m} = 1.11$. These values were obtained by benchmarking full implementations on a 3.4 GHz Intel Core i7-2600 (Sandy Bridge) processor.

Curve	Method	Protection	# Cores	Operation Count	Total Cost
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	no	1	$1\mathrm{i} + 742\mathrm{m} + 225\mathrm{s} + 767\mathrm{a}$	$1117\mathrm{m}$
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 36pts.	yes	1	$1\mathrm{i} + 1014\mathrm{m}_s + 217\mathrm{s}_s + 997\mathrm{a} + 68\mathrm{t}$	$1525\mathrm{m}_s \approx 1693\mathrm{m}$
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	no	4	$1\mathrm{i} + 420\mathrm{m} + 198\mathrm{s} + 484\mathrm{a}$	$724\mathrm{m}$
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 20pts.	yes	4	$1\mathrm{i} + 503\mathrm{m}_s + 196\mathrm{s}_s + 532\mathrm{a} + 22\mathrm{t}$	$848\mathrm{m}_s \approx 941\mathrm{m}$
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 32pts.	no	1	$2\mathrm{i} + 617\mathrm{m} + 404\mathrm{s} + 847\mathrm{a}$	$1209\mathrm{m}$
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 36pts.	yes	1	$2\mathrm{i} + 849\mathrm{m}_s + 489\mathrm{s}_s + 1001\mathrm{a} + 68\mathrm{t}$	$1630\mathrm{m}_s \approx 1809\mathrm{m}$
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 32pts.	no	4	$2\mathrm{i} + 371\mathrm{m} + 316\mathrm{s} + 593\mathrm{a}$	$850\mathrm{m}$
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 36pts.	yes	4	$2\mathrm{i} + 425\mathrm{m}_s + 335\mathrm{s}_s + 637\mathrm{a} + 17\mathrm{t}$	$977\mathrm{m}_s \approx 1084\mathrm{m}$
$E'_1(\mathbb{F}_{p_1^2})$	non-GLV, 8pts.	no	1	$2\mathrm{i} + 1169\mathrm{m} + 1169\mathrm{s} + 2141\mathrm{a}$	$2575\mathrm{m}$
$E_2(\mathbb{F}_{p_2})$	2-GLV, 16pts.	no	1	$1\mathrm{I} + 904\mathrm{M} + 690\mathrm{S} + 1240\mathrm{A}$	$2004\mathrm{M} \approx 1824\mathrm{m}$
$E_2(\mathbb{F}_{p_2})$	2-GLV, 16pts.	no	2	$1\mathrm{I} + 681\mathrm{M} + 615\mathrm{S} + 1103\mathrm{A}$	$1692\mathrm{M} \approx 1540\mathrm{m}$

pushes performance even further. In Table 1, the expected gains for E'_{T3} are 31 % and 97 % in comparison with 4-GLV–GLS and 2-GLV in Weierstrass form (respectively). In practice, we achieved similar speedups, namely, 33 % and 102 % (respectively). Likewise, a rough analysis indicates that a Twisted Edwards GLV–GLS curve for a j-invariant 0 curve would achieve roughly similar speed to E'_{T3}, which means that in comparison to its corresponding Weierstrass counterpart the gains are in the order of 9 % and 66 % (respectively). This highlights the impact of using Twisted Edwards especially over those GLV–GLS curves relatively slower in the Weierstrass model. Timings were registered on a single core of a 3.4 GHz Intel Core i7-2600 (Sandy Bridge) processor.

Let us now focus on curves E'_1, E_2, and E'_{T3} to assess performance of implementations targeting *four* scenarios of interest: unprotected and side-channel protected versions with sequential and multicore execution. Operation counts for computing a scalar multiplication at approximately 128 bits of security for the different cases are displayed in Table 2. The techniques to protect and parallelize our implementations are described in Sect. 7. In particular, the execution flow and memory address access of side-channel protected versions are not secret and are fully independent of the scalar. For our versions running on several cores, we used OpenMP. We use an implementation in which each core is in charge of one scalar multiplication with k_i. Given the high cost of thread creation/destruction, this approach guarantees the fastest computation in our case (see Sect. 7 for a discussion). Note that these multicore figures are only relevant for scenarios in which latency rather than throughput is targeted. Finally, we consider the cost of constant-time table lookups (denoted by t) given its nonnegligible cost in protected implementations.

Focusing on curve E_1', it can be noted a significant cost reduction when switching from non-GLV to a GLV–GLS implementation. The speedup is more than twofold for sequential, unprotected versions. Significant improvements are also expected when using multiple cores. A remarkable factor 3 speedup is expected when using GLV–GLS on four cores in comparison with a traditional execution (listed as non-GLV).

In general for our targeted GLV–GLS curves, the speedup obtained by using four cores is in between 1.42–1.80 times. Interestingly, the improvement is greater for protected implementations since the overhead of using a regular pattern execution is minimized when distributing computation among various cores. Remarkably, protecting implementations against timing attacks slowdowns performance by a factor in between 1.28–1.52, approximately. On the other hand, in comparison with curve E_2, an optimal execution of GLV–GLS on four cores is expected to run 1.81 times faster than an optimal execution of the standard 2-GLV on two cores.

To confirm our findings, we implemented the different versions using curves E_1', E_2, and E_{T3}'. To achieve maximum performance and ease the task of parallelizing and protecting the implementations, we wrote our own standalone software without employing any external library. For our experiments we used a 3.4 GHz Intel Core i7-2600 processor, which contains four cores. The timings in terms of clock cycles are displayed in Table 3. As can be seen, closely following our analysis, GLV–GLS achieves a twofold speedup over a non-GLV implementation on a single core. Parallel execution improves performance by up to 1.76 times for side-channel protected versions. In comparison with the non-GLV implementation, the four-core implementation runs 3 times faster. Our results also confirm the lower-than-expected cost of adding side-channel protection. Sequential versions lose about 50 % in performance, whereas parallel versions only lose about 28 %. The relative speedup when moving from 2-GLV to GLV-GLS on j-invariant 0 curves is 1.53 times, closely following the theoretical factor-1.5 speedup estimated previously. Four-core GLV–GLS supports a computation that runs 1.81 times faster than the standard 2-GLV on two cores. Finally, in practice our Twisted Edwards curve achieves up to 9 % speedup on the sequential, nonprotected scenario in comparison with the efficient j-invariant 0 curve based on Jacobian coordinates.

Comparison to Related Work Let us now compare our best numbers with recent results in the literature for elliptic curves over large prime characteristic fields. Focusing on one-core unprotected implementations, the first author together with Hu and Xu reported in [16] 122,000 cycles for a j-invariant 0 Weierstrass curve on an Intel Core i7-2600 (Sandy Bridge) processor. We report 91,000 cycles with the GLV–GLS Twisted Edwards curve E_{T3}', improving that number by a factor-1.34 speedup. We benchmarked on the same processor the side-channel protected software recently presented by Bernstein et al. [3] and obtained 194,000 cycles. Thus, our protected implementation, which runs in 137,000 cycles, is 1.42 times faster. Our result is also 1.12 times faster than the recent implementation by Hamburg [14].

It is also relevant to mention very recent results in settings other than elliptic curves over large prime characteristic fields. Taverne et al. [31] reported a protected implementation of a binary Edwards curve that runs in 225,000 cycles on an Intel Core i7-2600 (Sandy Bridge) machine, which is 1.64 times slower than our corresponding result. Aranha et al. [1] presented an implementation of the Koblitz curve K-283 that runs in

Table 3. Point multiplication timings (in clock cycles), 64-bit processor.

Curve	Method	Protection	# Cores	Core i7
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	no	1	**91,000**
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 36pts.	yes	1	137,000
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 16pts.	no	4	**61,000**
$E'_{T3}(\mathbb{F}_{p_3^2})$	4-GLV–GLS, 20pts.	yes	4	78,000
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 32pts.	no	1	99,000
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 36pts.	yes	1	145,000
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 32pts.	no	4	70,000
$E'_1(\mathbb{F}_{p_1^2})$	4-GLV–GLS, 36pts.	yes	4	89,000
$E'_1(\mathbb{F}_{p_1^2})$	non-GLV, 8pts.	no	1	201,000
$E_2(\mathbb{F}_{p_2})$	2-GLV, 16pts.	no	1	151,000
$E_2(\mathbb{F}_{p_2})$	2-GLV, 16pts.	no	2	127,000

99,000 cycles on the same machine, which is 9 % slower than our GLV–GLS Twisted Edwards curve E'_{T3} (unprotected sequential execution). Aranha et al. do not report timings for side-channel protected implementations. A faster (although also unprotected) implementation of a GLS binary curve over a quadratic extension field of characteristic two was recently announced in ECC2012. The running time in this case is about 73,000 cycles on the same Sandy Bridge processor [28]. These results highlight the significant impact of the carryless multiplier on the efficiency of characteristic two fields in the newest Intel processors. Efficient implementations on genus-2 (hyperelliptic) curves were recently reported in Bos et al. [6]. For instance, a protected implementation on a Kummer surface over a prime field runs in approximately 117,000 cycles on an Intel Core i7-3520M (Ivy Bridge) processor. Note that this processor architecture is in general more efficient than Sandy Bridge.

To the best of our knowledge, we have presented the first scalar multiplication implementation running on multiple cores that is protected against timing attacks, cache attacks, and several others.

9. Conclusion

We have shown how to generalize the GLV scalar multiplication method by combining it with Galbraith–Lin–Scott's ideas to perform a proven almost fourfold speedup on GLV curves over $\mathbb{F}_{p^2}$. We have introduced a new and easy-to-implement reduction algorithm, consisting in two applications of the extended Euclidean algorithm, one in $\mathbb{Z}$ and the other in $\mathbb{Z}[i]$. The refined bound obtained from this algorithm has allowed us to get a relative improvement from 2-GLV to 4-GLV–GLS practically independent of the curve. Our analysis and experimental results on different GLV curves show that in practice one should expect a factor-1.5 speedup, approximately. We improve performance even further by exploiting the Twisted Edwards model over a larger set of curves and show that this approach is especially significant to certain GLV curves with slow arithmetic in the Weierstrass model. This makes available to implementers new curves that achieve close to optimal performance. Moreover, we have shown how to protect GLV-based implementations against certain side-channel attacks with relatively low overhead and carried out a performance analysis on modern multicore processors. Our implementations of the generalized GLV–GLS method improve the state-of-the-art performance of elliptic curve point multiplication over fields of large prime characteristic for multiple scenarios: unprotected and side-channel protected versions with sequential and parallel execution. Finally, we have produced new families of GLV curves and written all such curves (up to isomorphism) with nontrivial endomorphisms of degree ≤ 3.

Acknowledgements

We thank the reviewers, Mike Scott and Dan Bernstein for their helpful comments. Also, we would like to thank Diego Aranha for his advice on multicore programming, Joppe Bos for his help on looking for efficient chains for implementing modular inversion, and Craig Costello and Kristin Lauter for helping us to detect a typo on a curve parameter in a previous paper version.

Appendix A. Examples

We give a few examples of GLV curves, which are curves defined over $\mathbb{C}$ with complex multiplication by a quadratic integer of small norm, corresponding to an endomorphism ϕ of small degree.[5] They make up an exhaustive list, up to isomorphism, in increasing order of endomorphism degree up to degree 3. While the first four examples appear in the previous literature, the next ones (degree 3) are new and have been computed with the Stark algorithm [30].

Example 1. Let $p \equiv 1 \pmod 4$ be a prime. Define an elliptic curve E over $\mathbb{F}_p$ by

$$y^2 = x^3 + ax.$$

If β is an element of order 4, then the map ϕ defined in the affine plane by

$$\phi(x, y) = (-x, \beta y)$$

is an endomorphism of E defined over $\mathbb{F}_p$ with $\operatorname{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\sqrt{-1}]$ since ϕ satisfies the equation[6]

$$\phi^2 + 1 = 0.$$

Example 2. Let $p \equiv 1 \pmod 3$ be a prime. Define an elliptic curve E over $\mathbb{F}_p$ by

$$y^2 = x^3 + b.$$

[5] By small we mean really small, usually less than 5. In particular, for cryptographic applications, the degree is much smaller than the field size.

[6] This is the only case where we cannot apply Lemma 1. It needs a separate treatment, given in Appendix B.

If γ is an element of order 3, then we have an endomorphism ϕ defined over $\mathbb{F}_p$ by

$$\phi(x, y) = (\gamma x, y),$$

and $\mathrm{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\frac{1+\sqrt{-3}}{2}]$ since ϕ satisfies the equation

$$\phi^2 + \phi + 1 = 0.$$

Example 3. Let $p > 3$ be a prime such that -7 is a quadratic residue modulo p. Define an elliptic curve E over $\mathbb{F}_p$ by

$$y^2 = x^3 - \frac{3}{4}x^2 - 2x - 1.$$

If $\xi = (1 + \sqrt{-7})/2$ and $a = (\xi - 3)/4$, then we get the $\mathbb{F}_p$-endomorphism ϕ defined by

$$\phi(x, y) = \left(\frac{x^2 - \xi}{\xi^2(x - a)}, \frac{y(x^2 - 2ax + \xi)}{\xi^3(x - a)^2}\right),$$

and $\mathrm{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\frac{1+\sqrt{-7}}{2}]$ since ϕ satisfies the equation

$$\phi^2 - \phi + 2 = 0.$$

Example 4. Let $p > 3$ be a prime such that -2 is a quadratic residue modulo p. Define an elliptic curve E over $\mathbb{F}_p$ by

$$y^2 = 4x^3 - 30x - 28$$

together with the $\mathbb{F}_p$-endomorphism ϕ defined[7] by

$$\phi(x, y) = \left(-\frac{2x^2 + 4x + 9}{4(x + 2)}, y\frac{2x^2 + 8x - 1}{4\sqrt{-2}(x + 2)^2}\right).$$

We have $\mathrm{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\sqrt{-2}]$ since ϕ satisfies the equation

$$\phi^2 + 2 = 0.$$

Example 5. Let $p > 3$ be a prime such that -11 is a quadratic residue modulo p. We define the elliptic curve E over $\mathbb{F}_p$

$$y^2 = x^3 - \frac{13824}{539}x + \frac{27648}{539}$$

[7] We take the opportunity to correct a typo found and transmitted in many sources, where a y factor was absent in the second coordinate. Its sign is irrelevant.

with $a = (1 + \sqrt{-11})/2$ and the endomorphism ϕ defined by

$$\begin{aligned}
&\phi(x, y) \\
&= \Bigg(\frac{(-\frac{539}{5184}a + \frac{539}{1728})x^3 + (\frac{28}{27}a - \frac{35}{18})x^2 + (-\frac{92}{9}a + \frac{8}{3})x + \frac{1728}{77}a + \frac{192}{77}}{(\frac{2695}{5184}a - \frac{539}{864})x^2 + (-\frac{217}{54}a + \frac{49}{18})x + \frac{64}{9}a - \frac{4}{3}}, y \\
&\quad \frac{(\frac{3773}{373248}a - \frac{18865}{995328})x^3 + (-\frac{2695}{20736}a + \frac{539}{3456})x^2 + (\frac{7}{432}a - \frac{91}{144})x + \frac{20}{27}a + \frac{1}{9}}{(-\frac{18865}{1492992}a + \frac{116963}{995328})x^3 + (\frac{7007}{20736}a - \frac{539}{432})x^2 + (-\frac{791}{432}a + \frac{581}{144})x + \frac{74}{27}a - \frac{35}{9}} \Bigg)
\end{aligned}$$

such that $\mathrm{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\frac{1+\sqrt{-11}}{2}]$. The characteristic polynomial of ϕ is

$$\phi^2 - \phi + 3 = 0.$$

Example 6. Let $p > 3$ be a prime such that -3 is a quadratic residue mod p. We define the elliptic curve E over $\mathbb{F}_p$

$$y^2 = x^3 - \frac{3375}{121}x + \frac{6750}{121}$$

with the endomorphism ϕ defined by

$$\begin{aligned}
\phi(x, y) = \Bigg(&-\frac{1331x^3 - 10890x^2 + 81675x - 189000}{33(11x - 45)^2}, \\
&y\, \frac{1331x^3 - 16335x^2 + 7425x + 43875}{3\sqrt{-3}(11x - 45)^3} \Bigg)
\end{aligned}$$

such that[8] $\mathrm{End}(E) = \mathbb{Z}[\phi] \cong \mathbb{Z}[\sqrt{-3}]$. The characteristic polynomial of ϕ is

$$\phi^2 + 3 = 0.$$

Appendix B. Treatment of the $j = 1728$ Curve

Let $a \in \mathbb{F}_p$, and $u \in \mathbb{F}_{p^8}$ be such that $u^4 \in \mathbb{F}_{p^2}$. Let E_1 be the curve of equation $y^2 = x^3 + ax$ over $\mathbb{F}_{p^8}$ and E_1' the curve of equation $y^2 = x^3 + au^4x$ over $\mathbb{F}_{p^2}$. Then E_1 is isomorphic to E_1' over $\mathbb{F}_{p^8}$ via the isomorphism $\psi \colon E_1 \to E_1'$ defined by $\psi(x, y) = (u^2x, u^3y)$. In other terms, E_1' is a quartic twist of E_1. We define $\Psi = \psi \,\mathrm{Frob}_p\, \psi^{-1} \in \mathrm{End}(E_1')$.

[8] This is the first example where the endomorphism ring is not the maximal order of its field of fractions. It can be summarily seen as follows: $\mathrm{End}(E) \supseteq \mathbb{Z}[\sqrt{-3}]$. If not equal, then it must be the full ring of integers $\mathbb{Z}[\frac{1+\sqrt{-3}}{2}]$. This would imply that $j = 0$, as there is only $h(-3) = 1$ isomorphism class of elliptic curves with complex multiplication by $\mathbb{Z}[\frac{1+\sqrt{-3}}{2}]$, given in Example 2 (see [30] for an abridged description of the theory of complex multiplication). This is clearly not the case here. Alternatively, one can see that there would exist a nontrivial automorphism (a primitive cube root of unity) corresponding to $\frac{-1+\sqrt{-3}}{2}$. A direct computation then shows that this is impossible.

Then, since $\mathrm{Frob}_p^8 = 1$ and $\mathrm{Frob}_p^4 \neq 1$ on $E_1(\mathbb{F}_{p^8})$, we have $\Psi^8 = 1$ on $E_1'(\mathbb{F}_{p^2})$, and, if $u \notin \mathbb{F}_{p^4}$, then $\Psi^4 + 1 = 0$. In this case, we can proceed as in Sect. 4, defining $\Phi = \Psi^2$. Note that $K = \mathbb{Q}(\Psi)$ is a quartic field with $\mathfrak{o}_K = \mathbb{Z}[\Psi]$. Lemma 1 still holds with the norm function of $K/\mathbb{Q}$. Finally, Theorem 2 is true with $16B^3$ replaced by $16 \cdot 24^{3/4} \approx 173.49$.

Appendix C. Cornacchia's algorithm in $\mathbb{Z}[i]$—real & imaginary parts

Algorithm 3 (EGEA or Cornacchia's algorithm in $\mathbb{Z}[i]$—real & imaginary parts).

```
Input: $\nu$ Gaussian prime dividing $n$ rational prime, $1 < \lambda < n$
such that $\lambda^2 + r\lambda + s \equiv 0 \pmod{n}$.
Output: Four $\mathbb{Z}$-linearly independent vectors $v_1, v_2, v_3$, and $v_4 \in$
$\ker F \subset \mathbb{Z}^4$ of rectangle norms $< 51.5(\sqrt{1+|r|+s})n^{1/4}$.

1. initialize:
   If $\lambda^2 \geq 2n$ then
      $r_{0,(R)} \leftarrow \lambda$,
   else
      $r_{0,(R)} \leftarrow \lambda + n$,
   $r_{0,(I)} \leftarrow 0$,
   $r_{1,(R)} \leftarrow \nu_{(R)}, r_{1,(I)} \leftarrow \nu_{(I)}$,
   $r_{2,(R)} \leftarrow n, r_{2,(I)} \leftarrow 0$,
   $s_{0,(R)} \leftarrow 1, s_{0,(I)} \leftarrow 0$,
   $s_{1,(R)} \leftarrow 0, s_{1,(I)} \leftarrow 0$,
   $s_{2,(R)} \leftarrow 0, s_{2,(I)} \leftarrow 0$,
   $q_{(R)} \leftarrow 0, q_{(I)} \leftarrow 0$.
2. main loop:
   while $(r_{2,(R)}^4 + 2r_{2,(R)}^2 r_{2,(I)}^2 + r_{2,(I)}^4)(1+|r|+s)^2 \geq n$ do
      $q_{(R)} \leftarrow \left\lceil \frac{r_{0,(R)}r_{1,(R)} + r_{0,(I)}r_{1,(I)}}{r_{1,(R)}^2 + r_{1,(I)}^2} \right\rfloor$,
      $q_{(I)} \leftarrow \left\lceil \frac{r_{0,(I)}r_{1,(R)} - r_{0,(R)}r_{1,(I)}}{r_{1,(R)}^2 + r_{1,(I)}^2} \right\rfloor$,
      $r_{2,(R)} \leftarrow r_{0,(R)} - (q_{(R)}r_{1,(R)} - q_{(I)}r_{1,(I)})$,
      $r_{2,(I)} \leftarrow r_{0,(I)} - (q_{(R)}r_{1,(I)} + q_{(I)}r_{1,(R)})$,
      $r_{0,(R)} \leftarrow r_{1,(R)}, r_{1,(R)} \leftarrow r_{2,(R)}$,
      $r_{0,(I)} \leftarrow r_{1,(I)}, r_{1,(I)} \leftarrow r_{2,(I)}$,
      $s_{2,(R)} \leftarrow s_{0,(R)} - (q_{(R)}s_{1,(R)} - q_{(I)}s_{1,(I)})$,
      $s_{2,(I)} \leftarrow s_{0,(I)} - (q_{(R)}s_{1,(I)} + q_{(I)}s_{1,(R)})$,
      $s_{0,(R)} \leftarrow s_{1,(R)}, s_{1,(R)} \leftarrow s_{2,(R)}$,
      $s_{0,(I)} \leftarrow s_{1,(I)}, s_{1,(I)} \leftarrow s_{2,(I)}$.
3. return:
   $v_1 = (r_{0,(R)}, -s_{0,(R)}, r_{0,(I)}, -s_{0,(I)})$, $v_2 = (r_{1,(R)}, -s_{1,(R)}, r_{1,(I)}, -s_{1,(I)})$,
   $v_3 = (-r_{0,(I)}, s_{0,(I)}, r_{0,(R)}, -s_{0,(R)})$, $v_4 = (-r_{1,(I)}, s_{1,(I)}, r_{1,(R)}, -s_{1,(R)})$.
```

Appendix D. Proof of Theorem 3

This section is devoted to proving that Algorithms 2 and 3 produce a reduced basis of $\ker F$ of rectangle norm $< 51.5(\sqrt{1+|r|+s})n^{1/4}$. The proof of the decomposition of k follows from the deduction recalled in Sect. 4.

Let us note first, about the running time, that it is known that the extended Euclidean algorithm runs in $O(\log^2 n)$ bits. The same analysis will also show that its Gaussian version runs in $O(\log^2 n)$ bits since its number of steps is also logarithmic. In short, this works as follows: if $b_j = \lfloor \log_2(|r_j|) \rfloor$ (i.e., the bitsize of $|r_j|$), then step j of the EGEA necessitates to find q_{j+1} and then r_{j+2}. One can show that integer division of two h-bit Gaussian integers with an ℓ-bit quotient runs in $O(h(\ell+1))$ binary operations. Finding q_{j+1} has therefore a runtime $O(b_j(c_{j+1}+1))$, where $c_{j+1} = \lfloor \log_2(|q_{j+1}|) \rfloor = b_j - b_{j+1} + O(1)$. Similarly, knowing q_{j+1}, computing r_{j+2} can be done in $O(b_{j+1}c_{j+1}) + O(b_{j+1}) = O(b_{j+1}(b_j - b_{j+1})) + O(b_{j+1})$. If $S = O(\log n)$ is the number of steps of the EGEA, the total runtime is less than a constant times

$$\sum_{j=0}^{S} b_j(b_j - b_{j+1}) + b_j = O\big(b_0^2 + b_0 S\big) = O\big(\log^2 n\big).$$

In the following, whenever $z \in \mathbb{C}^*$, its argument value $\arg(z)$ will be always chosen in $(-\pi, \pi]$. By *lattice square* we mean a square of side length one with vertices in $\mathbb{Z}[i]$. We single out eight *exceptional lattice squares*, which are those lattice squares with a vertex of modulus 1 (that is, ± 1 or $\pm i$) but not containing the origin as a vertex. Our analysis of the EGEA rests on the following lemmas.

Lemma 3 (A geometric property of squares). *There exists an absolute real constant* $\theta \approx 2.45861$ (*with* $2\arctan 2 < \theta$) *such that, for any point P of a lattice square, different from the vertices, letting V_1 be the closest vertex to P, there exists another vertex* $V_2 \neq V_1$ *with* $\theta \le \widehat{V_1 P V_2} \le \pi$. (*Note that* $V_1 P \le 1/\sqrt{2}$.)

Proof. This is one case where a picture is worth one thousand words. We refer to Fig. D.1 for a visual explanation of why the argument works. The dotted and dashed circle arcs are centered on the vertices and have radius $1/\sqrt{2}$. The plain circle arcs have the following property: for any point P on them, the two square vertices V and V' belonging to them make an angle of θ with P, in other terms, $|\widehat{VPV'}| = \theta$. Therefore points between two bigger arcs (in one of the two almond-shaped regions) "look" at the diagonally opposite vertices marking the intersection of these arcs with an angle between θ and π. We then choose the closest vertex to get a distance $\le 1/\sqrt{2}$. In case P is at the intersection of the two almond-shaped regions (in the "blown square"), we may have to choose one region where one of the vertices is at distance $\le 1/\sqrt{2}$, but this is always possible, since the dashed and dotted disks cover everything. Finally, if P does not belong to the union of the two almond-shaped regions, then it lies inside one of the smaller plain disks, where its angle between two appropriate consecutive vertices will also be between θ and π. Furthermore, by choosing the closest vertex V_1 to P, we have $V_1 P < 1/\sqrt{2}$. □

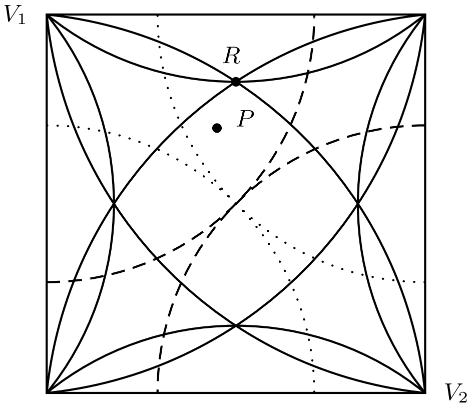

Fig. D.1. Existence of θ

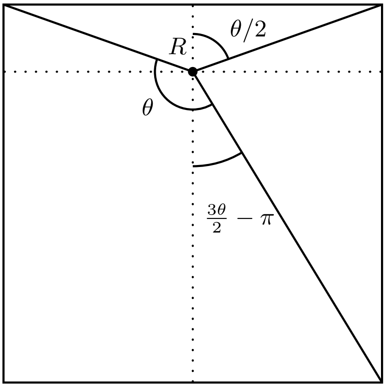

Fig. D.2. Calculation of $\cot(\theta/2)$

It remains to explain how we can calculate θ, or rather its value on the usual trigonometric functions $\sin\theta$ and $\cos\theta$ (which is what we really need later), since we can show that they are algebraic numbers expressible by radicals, but $\theta/\pi \notin \mathbb{Q}$.

We concentrate on finding the Cartesian coordinates of $R = (1/2, 1-u/2)$, appearing in Fig. D.1, supposing the vertices are the origin, $(1,0)$, $(1,1)$, and $(0,1)$. Our aim is then to find $u \in (0,1)$. A look at Fig. D.2 shows the disposition of the angles, so that $u = \cot(\theta/2)$ and $2-u = \cot(3\theta/2-\pi) = \cot(3\theta/2)$. The triplication formulas for the cotangent then show that u satisfies the equation

$$u + \frac{3u-u^3}{1-3u^2} = 2 \quad \Longleftrightarrow \quad 2u^3 - 3u^2 - 2u + 1 = 0.$$

Solving it yields that the root we are looking for is

$$u = \frac{(\sqrt{3}i+1)(12\sqrt{237}i-54)^{\frac{1}{3}}}{12 2^{\frac{1}{3}}} + \frac{72^{\frac{1}{3}}(1-\sqrt{3}i)}{4(12\sqrt{237}i-54)^{\frac{1}{3}}} + \frac{1}{2} \approx 0.3554157,$$

where the determination of the cube root is the one in the first quadrant.

Remark 4. One can see that $\theta/\pi \notin \mathbb{Q}$ in the following way.

$$\cot(\theta/2) = i\frac{e^{i\theta/2}+e^{-i\theta/2}}{e^{i\theta/2}-e^{-i\theta/2}},$$

and supposing for contradiction that $\theta/\pi \in \mathbb{Q}$, we would have that $e^{i\theta}$ is a root of unity. The preceding equality shows that then $\cot(\theta/2)$ belongs to a cyclotomic extension of $\mathbb{Q}$, whose Galois group is abelian. But we have seen that the irreducible polynomial of $\cot(\theta/2)$ is $2x^3-3x^2-2x+1$, with discriminant 316, not a rational square. Therefore, its Galois group is the nonabelian S_3, a contradiction.

Remark 5. When applying Lemma 3, it is essential that we be able to choose from the set of all vertices of the lattice square which ones are the adequate V_1 and V_2. Since the only excluded quotient q_j is zero, it means that we must be careful to avoid all four squares which have the origin as a vertex. But this follows from the fact that at all steps $j \geq 0$ we always have $|r_j/r_{j+1}| \geq \sqrt{2}$.

Define $\Theta = \arctan 2 - \pi/3$ and $A = 1/\sin\Theta = 2\sqrt{5}(8+5\sqrt{3})/\sqrt{13+4\sqrt{3}} \approx 16.6902$. In the following analysis of the EGEA, it will be useful to make the following distinction between indices.

Definition 1 (Good and bad j's). A step $j \geq 0$ of the EGEA will be called *bad* if, during the $(j-1)$th step, among all four choices of q_j as a vertex of the lattice square containing r_{j-1}/r_j (and consequent choice of r_{j+1} and s_{j+1}, noting that for the purpose of this definition, we do not require that $|r_{j+1}| < |r_j|$), we always have $s_j s_{j+1} r_{j+1} \neq 0$ and

$$\left|\arg\left(\frac{s_{j+1}r_j}{s_j r_{j+1}}\right)\right| < \Theta.$$

Otherwise j is called *good*.

Remark 6. Note that $j=0$ and $j=1$ are always good since $s_1 = 0$.

Lemma 4 (Use of good j's). *If j is good, then for some choice of r'_{j+1} as per preceding definition (and relative s'_{j+1}), we have*

$$\left|s'_{j+1}r_j - s_j r'_{j+1}\right| \geq \sin\Theta \max\left(\left|s'_{j+1}r_j\right|, \left|s_j r'_{j+1}\right|\right),$$

and, therefore, since r_{j+1} in the EGEA has smallest modulus among all four choices r'_{j+1}, we have

$$\max\bigl(|s_j r_{j+1}|, |s_{j+1} r_j|\bigr) \le (A+1)|\nu|.$$

Proof. Notice that the result holds trivially if $s_j s'_{j+1} r'_{j+1} = 0$. Otherwise, this is a straightforward application of a general inequality about complex numbers that we can express as follows: let $\zeta \in \mathbb{C}^*$ with $\pi \ge |\arg(\zeta)| \ge \Theta$. We claim that under these conditions, $|1-\zeta| \ge \sin\Theta$. Indeed, writing $\zeta = re^{i\psi}$ with $\Theta \le \psi \le \pi$, we have

$$|1-\zeta|^2 = \bigl(1 - re^{i\psi}\bigr)\bigl(1 - re^{-i\psi}\bigr) = 1 - 2r\cos\psi + r^2.$$

First note that we can suppose $\psi \le \pi/2$, otherwise clearly $|1-\zeta| \ge 1$. The last expression in r, when viewed as a quadratic polynomial, has minimum (over $\mathbb{R}$) equal to $-\Delta/4 = -(4\cos^2\psi - 4)/4 = \sin^2\psi \ge \sin^2\Theta$. Therefore $|1-\zeta| \ge \sin\Theta$, thereby proving our claim. The first part of the lemma will follow by applying the claim to $\zeta = s'_{j+1} r_j / s_j r'_{j+1}$ and $\zeta = s_j r'_{j+1} / s'_{j+1} r_j$ successively.

The second part follows from

$$|s_j r_{j+1}| \le \bigl|s_j r'_{j+1}\bigr| \le A\bigl|s'_{j+1} r_j - s_j r'_{j+1}\bigr| = A|\nu|,$$

and therefore,

$$|s_{j+1} r_j| = |s_{j+1} r_j - s_j r_{j+1} + s_j r_{j+1}| \le |s_{j+1} r_j - s_j r_{j+1}| + |s_j r_{j+1}| \le |\nu| + A|\nu|.$$

□

Remark 7. We have seen in the course of the proof the preceding lemma the following fact: if $\zeta \in \mathbb{C}^*$ with $\pi \ge |\arg(\zeta)| \ge \psi$, then $|1-\zeta| \ge \sin\psi$. This is equivalent to the following assertion (set $\zeta = 1 - \xi$), used in the proof of the next lemma: if $|\xi| < \sin\psi$, then $|\arg(1-\xi)| < \psi$.

The next result is crucial in controlling what happens in the bad cases. Its proof is rather elaborated.

Lemma 5 (Bad-j behavior of s_j). *If j is bad, then*

$$|s_{j+1}| \le 2\sqrt{2}|s_{j-1}| \quad \textit{and} \quad |s_j| \le |s_{j-1}|.$$

Proof. We first suppose that the point P corresponding to r_{j-1}/r_j does not belong to an exceptional lattice square. Let V_1 and V_2 as in Lemma 3 correspond respectively to q_j and q'_j. Upon defining $r'_{j+1} = r_{j-1} - q'_j r_j$, since $r_{j+1} = r_{j-1} - q_j r_j$, Lemma 3 states that $\pi \ge |\arg((q_j - r_{j-1}/r_j)/(q'_j - r_{j-1}/r_j))| = |\arg(r_{j+1}/r'_{j+1})| \ge \theta$. By the definition of "bad" we have, denoting $s'_{j+1} = s_{j-1} - q'_j s_j$,

$$\left|\arg\left(\frac{s_{j+1} r_j}{s_j r_{j+1}}\right)\right| < \Theta \quad \text{and} \quad \left|\arg\left(\frac{s'_{j+1} r_j}{s_j r'_{j+1}}\right)\right| < \Theta,$$

and this yields

$$\begin{aligned}\left|\arg\left(\frac{s_{j+1}r'_{j+1}}{s'_{j+1}r_{j+1}}\right)\right| &= \left|\arg\left(\frac{s_{j+1}r_j}{s_j r_{j+1}}\right)+\arg\left(\frac{s_j r'_{j+1}}{s'_{j+1}r_j}\right)\right| \\ &\le \left|\arg\left(\frac{s_{j+1}r_j}{s_j r_{j+1}}\right)\right|+\left|\arg\left(\frac{s_j r'_{j+1}}{s'_{j+1}r_j}\right)\right| < 2\Theta.\end{aligned}$$

We deduce

$$\begin{aligned}\left|\arg\left(\frac{s_{j+1}r'_{j+1}}{s'_{j+1}r_{j+1}}\right)+\arg\left(\frac{r_{j+1}}{r'_{j+1}}\right)\right| &\ge \left|\left|\arg\left(\frac{r_{j+1}}{r'_{j+1}}\right)\right|-\left|\arg\left(\frac{s_{j+1}r'_{j+1}}{s'_{j+1}r_{j+1}}\right)\right|\right| \\ &> \theta-2\Theta > \frac{2\pi}{3},\end{aligned}$$

while, on the other hand,

$$\left|\arg\left(\frac{s_{j+1}r'_{j+1}}{s'_{j+1}r_{j+1}}\right)\right|+\left|\arg\left(\frac{r_{j+1}}{r'_{j+1}}\right)\right| < 2\Theta+\pi < \theta-\frac{2\pi}{3}+\pi < \frac{4\pi}{3},$$

which together imply

$$\left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| > \frac{2\pi}{3}. \tag{D.1}$$

Now assume that $|s_j| > |s_{j-1}|$. Then $|q_j s_j| > \sqrt{2}\,|s_{j-1}|$ and $|q'_j s_j| > \sqrt{2}\,|s_{j-1}|$, since the quotients q_j, q'_j are Gaussian integers of modulus different from zero or one. Furthermore, since there is at most one Gaussian integer of modulus $\sqrt{2}$ in a lattice square, we have that either $|q_j s_j| > 2|s_{j-1}|$ or $|q'_j s_j| > 2|s_{j-1}|$. Therefore, by Remark 7,

$$\left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right)\right| = \left|\arg\left(1-\frac{s_{j-1}}{q_j s_j}\right)\right| \le \frac{\pi}{4},$$

and, similarly,

$$\left|\arg\left(\frac{q'_j s_j - s_{j-1}}{q'_j s_j}\right)\right| \le \frac{\pi}{4}$$

with at least one of them being $\le \pi/6$. We then get, using that $|\arg(q_j/q'_j)| \le \pi/4$,

$$\begin{aligned}\left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| &= \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right)+\arg\left(\frac{q_j}{q'_j}\right)+\arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right| \\ &\le \frac{\pi}{4}+\frac{\pi}{4}+\frac{\pi}{6}=\frac{2\pi}{3},\end{aligned}$$

contradicting (D.1).

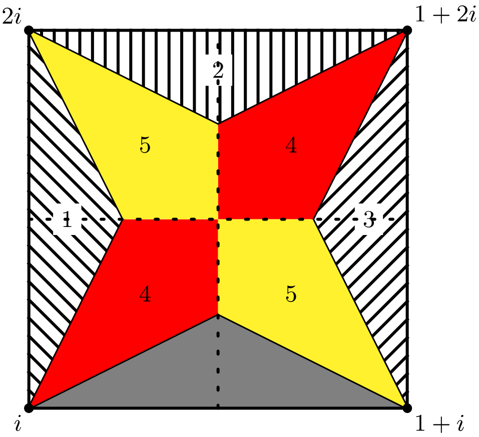

Fig. D.3. Exceptional regions

Exceptional Choices of V_1 *and* V_2 We now discuss the exceptional cases where q_j or $q'_j = \pm 1, \pm i$, which need to be handled ad hoc. By symmetry (without loss of generality), we place ourselves in the case where r_{j-1}/r_j lies in the lattice square of vertices $i, 1+i, 1+2i, 2i$. It then belongs to one of the five regions labeled 1 to 5 on Fig. D.3. Note that the open gray region is off-limits since $|r_{j-1}/r_j| \geq \sqrt{2}$. Each of these regions contains two lattice points, which as before we will denote V_1 for the one closest to the point P of affix r_{j-1}/r_j and V_2 for the other one (in case P lies on the boundary between two or more zones, their distinction is immaterial). Note that V_1 is closest to P among *all* four vertices.

Region 1 (delimited by a triangle of vertices $i, 1/4+3i/2, 2i$): In this case, $|\arg(r_{j+1}/r'_{j+1})| = \widehat{V_1 P V_2} \geq 2\arctan 2$. Supposing to fix notations that $q_j = i$ and $q'_j = 2i$, we have, assuming that $|s_j| > |s_{j-1}|$ and using Remark 7,

$$\left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right)\right| \leq \frac{\pi}{2}, \quad \left|\arg\left(\frac{q'_j s_j - s_{j-1}}{q'_j s_j}\right)\right| \leq \frac{\pi}{6}.$$

On the other hand, a reasoning similar to the one leading to (D.1) with the value $2\arctan 2$ instead of θ will show that again $|\arg(s_{j+1}/s'_{j+1})| > 2\pi/3$, which leads to a contradiction since

$$\begin{aligned}\left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| &= \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right| \\ &\leq \frac{\pi}{2} + 0 + \frac{\pi}{6} = \frac{2\pi}{3}.\end{aligned}$$

The other four cases are treated similarly, and we briefly outline them.

Region 2 (delimited by a triangle of vertices $2i, 1+7i/4, 1+2i$): Here $|\arg(r_{j+1}/r'_{j+1})| \geq 2\arctan 2$. Letting $q_j = 2i, q'_j = 1+2i$, one can show, assuming that $|s_j| > |s_{j-1}|$, that

$$\frac{2\pi}{3} < \left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| = \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right|$$
$$\leq \frac{\pi}{6} + \left(\frac{\pi}{2} - \arctan 2\right) + \left(\frac{\pi}{2} - \arctan 2\right) < \frac{\pi}{2} < \frac{2\pi}{3},$$

a contradiction.

Region 3 (delimited by a triangle of vertices $1 + 2i, 3/4 + i/2, 1 + i$): Here $|\arg(r_{j+1}/r'_{j+1})| \geq 2\arctan 2$. Letting $q_j = 1 + i, q'_j = 1 + 2i$, one can show, assuming that $|s_j| > |s_{j-1}|$, that

$$\frac{2\pi}{3} < \left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| = \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right|$$
$$\leq \frac{\pi}{4} + \arctan(1/3) + \left(\frac{\pi}{2} - \arctan 2\right) = \frac{\pi}{2} < \frac{2\pi}{3},$$

a contradiction.

Region 4 (the red zone): Here $|\arg(r_{j+1}/r'_{j+1})| \geq \pi - \arctan 2 + \arctan(2/3)$. Letting $q_j = i, q'_j = 1 + 2i$, one can show, assuming that $|s_j| > |s_{j-1}|$, that

$$\begin{aligned}
&\frac{5\pi}{3} - 3\arctan(2) + \arctan(2/3)\\
&\quad < \left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right|\\
&\quad = \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right|\\
&\quad \leq \frac{\pi}{2} + \left(\frac{\pi}{2} - \arctan 2\right) + \left(\frac{\pi}{2} - \arctan 2\right)\\
&\quad < \frac{5\pi}{3} - 3\arctan(2) + \arctan(2/3),
\end{aligned}$$

a contradiction.

Region 5 (the yellow zone): Here $|\arg(r_{j+1}/r'_{j+1})| \geq \pi - \arctan(4/7)$. Letting $q_j = 1+i$, $q'_j = 2i$, one can show, assuming that $|s_j| > |s_{j-1}|$, that

$$\begin{aligned}
&\frac{5\pi}{3} - 3\arctan(2) + \arctan(2/3)\\
&\quad < \left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right|\\
&\quad = \left|\arg\left(\frac{q_j s_j - s_{j-1}}{q_j s_j}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{q'_j s_j}{q'_j s_j - s_{j-1}}\right)\right|\\
&\quad \leq \frac{\pi}{4} + \frac{\pi}{4} + \frac{\pi}{6} = \frac{2\pi}{3}\\
&\quad < \frac{5\pi}{3} - 3\arctan(2) + \arctan(2/3),
\end{aligned}$$

a contradiction.

We have thus proved that in any case $|s_j| \leq |s_{j-1}|$. To show the first part of the lemma, we proceed similarly, although there is a slight difference. We assume at first that $|s_{j+1}| > 2\,|s_{j-1}|$ *and* $|s'_{j+1}| > \sqrt{2}\,|s_{j-1}|$. Then, by Remark 7,

$$\left|\arg\left(\frac{s_{j+1} - s_{j-1}}{s_{j+1}}\right)\right| = \left|\arg\left(1 - \frac{s_{j-1}}{s_{j+1}}\right)\right| \leq \frac{\pi}{6}$$

and

$$\left|\arg\left(\frac{s'_{j+1} - s_{j-1}}{s'_{j+1}}\right)\right| \leq \frac{\pi}{4}.$$

Proceeding as previously,

$$\begin{aligned}
\left|\arg\left(\frac{s_{j+1}}{s'_{j+1}}\right)\right| &= \left|\arg\left(\frac{s_{j+1}}{s_{j+1} - s_{j-1}}\right) + \arg\left(\frac{q_j}{q'_j}\right) + \arg\left(\frac{s'_{j+1} - s_{j-1}}{s'_{j+1}}\right)\right|\\
&\leq \frac{\pi}{4} + \frac{\pi}{4} + \frac{\pi}{6} = \frac{2\pi}{3},
\end{aligned}$$

again contradicting (D.1), which also holds in the exceptional cases, as we have just seen. Therefore, $|s_{j+1}| \leq 2\,|s_{j-1}|$ or $|s'_{j+1}| \leq \sqrt{2}\,|s_{j-1}|$ (or both). In the first case, we are done. Otherwise, since $s_{j+1} + q_j s_j = s'_{j+1} + q'_j s_j = s_{j-1}$, we derive

$$|s_{j+1}| \leq \left|s'_{j+1}\right| + \left|q_j - q'_j\right||s_j| \leq \sqrt{2}|s_{j-1}| + \sqrt{2}|s_{j-1}| = 2\sqrt{2}|s_{j-1}|$$

by the already proved second part of the lemma and the fact that q_j, q'_j correspond to two vertices of the same lattice square, so that $|q_j - q'_j| \leq \sqrt{2}$. □

Lemma 6 (Lower bound on generic vectors of $\ker F$). *For any nonzero* $(z_1, z_2) \in \ker F$, *we have*

$$\max\bigl(|z_1|, |z_2|\bigr) \geq \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}}.$$

In particular, for any $j \geq 0$, *we have*

$$\max\bigl(|r_j|, |s_j|\bigr) \geq \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}}.$$

Proof. This proof uses an argument already appearing in the proof of the original GLV algorithm (see [29]) and Lemma 1. If $(0,0) \neq (z_1, z_2) \in \ker F$, then $z_1 + \lambda z_2 \equiv 0 \pmod{\nu}$. If λ' is the other root of $X^2 + rX + s \pmod{n}$, we get that

$$z_1^2 - rz_1z_2 + sz_2^2 \equiv (z_1 + \lambda z_2)\bigl(z_1 + \lambda' z_2\bigr) \equiv 0 \pmod{\nu}.$$

Since $X^2 + rX + s$ is irreducible in $\mathbb{Q}(i)$ because the two quadratic fields are linearly disjoint, we therefore have $|z_1^2 - rz_1z_2 + sz_2^2| \geq |\nu|$. On the other hand, if

$$\max\bigl(|z_1|, |z_2|\bigr) < \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}},$$

then

$$\bigl|z_1^2 - rz_1z_2 + sz_2^2\bigr| \leq |z_1|^2 + |r||z_1||z_2| + s|z_2|^2 < |\nu|,$$

a contradiction. To show the second part, it suffices to note that since $r_0 s_j + \nu t_j = r_j$ (where, as mentioned previously, $r_0 = \lambda$ or $\lambda + n$), we have that

$$0 \equiv \nu t_j = r_j - r_0 s_j \equiv r_j - \lambda s_j \pmod{\nu},$$

so that $(r_j, -s_j) \in \ker F$ for every $j \geq 0$. □

Proof of Theorem 3. It remains here to show the improved bound, which brings us to finding four $\mathbb{Q}$-linearly independent vectors of $\ker F$ of rectangle norm bounded by $Cn^{1/4}$. Define $m \geq 1$ as the index such that

$$|r_m| \geq \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}} \quad \text{and} \quad |r_{m+1}| < \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}}. \tag{D.2}$$

Let us consider an index $j \leq m$. If it is good, then by Lemma 4 we have $|s_{j+1} r_j| \leq (A+1)|\nu|$, and therefore, since $(|r_j|)$ is a decreasing sequence,

$$|s_{j+1}| \leq 2\sqrt{2}(A+1)\sqrt{1+|r|+s}\sqrt{|\nu|}. \tag{D.3}$$

On the other hand if it is bad, then let $l < j$ be the largest good index less than j. By Lemma 5 and Lemma 4 we have

$$\frac{|s_{j+1}|}{2\sqrt{2}} \le |s_{j-1}| \le |s_{j-2}| \le \cdots \le |s_l| \le (A+1)\frac{|\nu|}{|r_{l+1}|}$$
$$\le (A+1)\sqrt{1+|r|+s}\sqrt{|\nu|},$$

and therefore in any case (D.3) holds. Applying this to $j = m-1$ and $j = m$, we find that

$$\max\bigl(|s_m|, |s_{m+1}|\bigr) \le 2\sqrt{2}(A+1)\sqrt{1+|r|+s}\sqrt{|\nu|}. \tag{D.4}$$

Moreover, using

$$s_{m+1}r_m - s_m r_{m+1} = (-1)^{m+1}\nu, \tag{D.5}$$

from (D.2) and (D.4) we deduce

$$|s_{m+1}r_m| \le |\nu| + |s_m r_{m+1}| \le |\nu| + 2\sqrt{2}(A+1)|\nu|.$$

In addition, by Lemma 6 we must have

$$|s_{m+1}| \ge \frac{\sqrt{|\nu|}}{\sqrt{1+|r|+s}},$$

which therefore implies that

$$|r_m| \le \bigl(2\sqrt{2}(A+1)+1\bigr)\sqrt{1+|r|+s}\sqrt{|\nu|}.$$

This last equation, together with (D.4) and (D.2), shows that the two vectors $v_1 = (r_m, -s_m), v_2 = (r_{m+1}, -s_{m+1}) \in \ker F \subset \mathbb{Z}[i]^2$ have rectangle norms bounded by $C\sqrt{|\nu|} = Cn^{1/4}$ for $C = (2\sqrt{2}(A+1)+1)\sqrt{1+|r|+s} < 51.5\sqrt{1+|r|+s}$ (that these two vectors belong to $\ker F$ was shown in the proof of Lemma 6).

We can find two more vectors by noticing that (D.5) implies that v_1 and v_2 are $\mathbb{Q}(i)$-linearly independent. Therefore, the vectors $v_1, v_2, v_3 = iv_1, v_4 = iv_2$ are $\mathbb{Q}$-linearly independent. They all belong to $\ker F$ and have rectangle norms bounded by $Cn^{1/4}$. In view of the fact that the Euclidean norm upper-bounds the rectangle norm, the corresponding vectors in $\mathbb{Z}^4$ also have rectangle norms bounded by $Cn^{1/4}$, thus concluding the proof of the theorem, since these are exactly the four vectors returned by Algorithm 3. □

Appendix E. Side-Channel Resistant Recoding Algorithm for d-Dimensional GLV Scalar Multiplication

Below is the recoding algorithm with regular execution that, likewise, enables a scalar multiplication with regular pattern execution. It modifies Algorithm 6 from [17] to support a recoding of fixed length. To apply the recoding to GLV-based scalar multiplication, Algorithm 4 has to be applied to each scalar derived from the GLV decomposition.

Algorithm 4 (Regular Pattern Recoding Algorithm with Fixed Length).

```
Input: $k$ odd, dimension $d$ of $l$-bit GLV scalar multiplication,
and window width $w$.
Output: $n = (k_t, \ldots, k_0)$, where $k_i \in \{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$.

$t = \lceil l/(d \cdot (w-1)) \rceil$
for $i = 0$ to $(t-1)$ do
    $k_i = (k \mod 2^w) - 2^{w-1}$
    $k = (k - k_i)/2^{w-1}$
end
$k_t = k$
```

Proof. The correctness of Algorithm 4 can be found in [17]. We still need to show that all scalars in a GLV-based scalar multiplication can be represented with exactly $t + 1 = \lceil l/(d \cdot (w-1)) \rceil + 1$ digits. For all practical purposes, it can be assumed that the GLV method decomposes an l-bit scalar into $\lceil l/d \rceil$-bit scalars. What is left is to show that every $(w-1)$ bits of the scalar are converted to exactly $(w-1)$ digits (padding with zeroes in the most significant positions if required). This is precisely what happens since at every iteration $(w-1)$ bits are replaced by $(w-2)$ zero digits and one nonzero digit. The updating step sets to one the least significant bit of the new partial value k. After all iterations are done this bit is assigned to k_t as the $(t+1)$th digit. □

References

[1] D.F. Aranha, A. Faz-Hernandez, J. Lopez, F. Rodriguez-Henriquez, Faster implementation of scalar multiplication on Koblitz curves, in *Proceedings of Latincrypt 2012*. LNCS, vol. 7533 (Springer, Berlin, 2012), pp. 177–193

[2] D.J. Bernstein, P. Birkner, M. Joye, T. Lange, C. Peters, Twisted Edwards curves, in *Proceedings of AFRICACRYPT 2008*, ed. by S. Vaudenay. LNCS, vol. 5023 (Springer, Berlin, 2008), pp. 389–405

[3] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang, High-speed high-security signatures, in *Proceedings of CHES 2011*, ed. by B. Preneel, T. Takagi. LNCS, vol. 6917 (Springer, Berlin, 2011), pp. 124–142

[4] D.J. Bernstein, Curve25519: New Diffie–Hellman speed records, in *Proceedings of PKC 2006*. LNCS, vol. 3958 (Springer, Berlin, 2006), pp. 207–228

[5] D.J. Bernstein, CPU traps and pitfalls. Talk at Emerging Topics in Cryptographic Design and Cryptanalysis, Pythagorion, Samos, 2007. Available at: http://cr.yp.to/talks/2007.05.04/slides.pdf

[6] J. Bos, C. Costello, H. Hisil, K. Lauter, Two is greater than one. Cryptology ePrint Archive, Report 2012/670, 2012. Available at: http://eprint.iacr.org/2012/670

[7] D. Brumley, D. Boneh, Remote timing attacks are practical, in *Proceedings of the 12th USENIX Security Symposium*, ed. by S. Mangard, F.-X. Standaert. LNCS, vol. 6225 (Springer, Berlin, 2003), pp. 80–94

[8] H. Cohen, *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics, vol. 138 (Springer, Berlin, 1996)

[9] G. Cornacchia, Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$. *G. Mat. Battaglini* **46**, 33–90 (1908)

[10] H. Edwards, A normal form for elliptic curves. *Bull. Am. Math. Soc.* **44**, 393–422 (2007)

[11] S.D. Galbraith, X. Lin, M. Scott, Endomorphisms for faster elliptic curve cryptography on a large class of curves, in *Proceedings of EUROCRYPT 2009*, ed. by A. Joux. LNCS, vol. 5479 (Springer, Berlin, 2009), pp. 518–535

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

[13] R.P. Gallant, J.L. Lambert, S.A. Vanstone, Faster point multiplication on elliptic curves with efficient endomorphisms, in *Advances in Cryptology—Proceedings of CRYPTO 2001*, ed. by J. Kilian. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 190–200

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

[15] H. Hisil, K. Wong, G. Carter, E. Dawson, Twisted Edwards curves revisited, in *Proceedings of ASIACRYPT 2008*, ed. by J. Pieprzyk. LNCS, vol. 5350 (Springer, Berlin, 2008), pp. 326–343

[16] Z. Hu, P. Longa, M. Xu, Implementing 4-dimensional GLV method on GLS elliptic curves with j-invariant 0. *Des. Codes Cryptogr.* **63**(3), 331–343 (2012). Also in Cryptology ePrint Archive, Report 2011/315, http://eprint.iacr.org/2011/315

[17] M. Joye, M. Tunstall, Exponent recoding and regular exponentiation algorithms, in *Proceedings of Africacrypt 2003*, ed. by M. Joye. LNCS, vol. 5580 (Springer, Berlin, 2009), pp. 334–349

[18] E. Kasper, Fast elliptic curve cryptography in OpenSSL, in *2nd Workshop on Real-Life Cryptographic Protocols and Standardization* (2011)

[19] P.C. Kocher, Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems, in *Advances in Cryptology—Proceedings of CRYPTO 1996*, ed. by N. Koblitz. LNCS, vol. 1109 (Springer, Berlin, 1996), pp. 104–113

[20] A.K. Lenstra, H.W. Lenstra Jr., L. Lovász, Factoring polynomials with rational coefficients. *Math. Ann.* **261**, 513–534 (1982)

[21] P. Longa, Elliptic curve cryptography at high speeds. Talk at the 15th Workshop on Elliptic Curve Cryptography (ECC 2011), INRIA, France, 2011. Available at: http://ecc2011.loria.fr/slides/longa.pdf

[22] P. Longa, High-speed elliptic curve and pairing-based cryptography. PhD thesis, University of Waterloo, 2011. Available at: http://hdl.handle.net/10012/5857

[23] P. Longa, C. Gebotys, Efficient techniques for high-speed elliptic curve cryptography, in *Proceedings of CHES 2010*, ed. by S. Mangard, F.-X. Standaert. LNCS, vol. 6225 (Springer, Berlin, 2010), pp. 80–94

[24] P. Longa, A. Miri, New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields, in *Proceedings of PKC 2008*, ed. by R. Cramer. LNCS, vol. 4939 (Springer, Berlin, 2008), pp. 229–247

[25] F. Morain, Courbes elliptiques et tests de primalité. PhD thesis, Université de Lyon I, 1990. Available at: http://www.lix.polytechnique.fr/Labo/Francois.Morain/Articles/publisfm.php, Chap. 2: On Cornacchia's algorithm (joint with J.-L. Nicolas)

[26] P.Q. Nguyen, D. Stehlé, Low-dimensional lattice basis reduction revisited, in *Algorithmic Number Theory, Proceedings of 6th International Symposium, ANTS-VI*, Burlington, VT, USA, 13–18 June 2004, ed. by D.A. Buell. LNCS, vol. 3076 (Springer, Berlin, 2004), pp. 338–357

[27] K. Okeya, T. Takagi, The width-w NAF method provides small memory and fast elliptic curve scalars multiplications against side-channel attacks, in *Proceedings of CT-RSA 2003*, ed. by M. Joye. LNCS, vol. 2612 (Springer, Berlin, 2003), pp. 328–342

[28] F. Rodriguez-Henriquez, Private communication, 2012

[29] F. Sica, M. Ciet, J.-J. Quisquater, Analysis of the Gallant–Lambert–Vanstone method based on efficient endomorphisms: Elliptic and hyperelliptic curves, in *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*, ed. by H. Heys, K. Nyberg. LNCS, vol. 2595 (Springer, Berlin, 2002), pp. 21–36

[30] H.M. Stark, Class-numbers of complex quadratic fields, in *Modular Functions of One Variable, I, Proc. Internat. Summer School*, Univ. Antwerp, Antwerp, 1972. Lecture Notes in Mathematics, vol. 320 (Springer, Berlin, 1973), pp. 153–174

[31] J. Taverne, A. Faz-Hernandez, D.F. Aranha, F. Rodriguez-Henriquez, D. Hankerson, J. Lopez, Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. *J. Cryptograph. Eng.* **1**, 187–199 (2011)

[32] Z. Zhou, Z. Hu, M. Xu, W. Song, Efficient 3-dimensional GLV method for faster point multiplication on some GLS elliptic curves. *Inf. Process. Lett.* **77**(262), 1075–1104 (2010)