Faster complete formulas and their application to curves with efficient endomorphisms

Diego F. Aranha
Department of Computer Science, Aarhus University, Denmark
dfaranha@cs.au.dk

Paulo L. Barreto
School of Engineering and Technology, University of Washington | Tacoma, US
pbarreto@uw.edu

November 2026

Abstract

We revisit scalar multiplication on prime curves $y^2 = x^3 + b$ with $p \equiv 1 \pmod 3$, a family that includes secp256k1, BN and BLS curves. Applying a 15-year-old technique in a new way, we improve the best complete addition from 12 to 11 multiplications (10 in mixed coordinates), the first improvement in 10 years: its final stage is a 3D cross product, which needs only five multiplications. In `libsecp256k1`, the new formulas make constant-time fixed-base scalar multiplication 5.2% faster. We also give a complete tripling formula that costs the same as incomplete Jacobian tripling, and combine both formulas into an exception-free width-3 τ-NAF with $\tau = 1 - \omega$ and a six-point table. In an operation-count model it is about 14% faster than exception-free GLV at 256 bits and 3–7% faster than the fastest incomplete GLV. In RELIC, on BLS12-381, BLS24-509 and BLS48-575, the τ-NAF in $\mathbb{G}_1$ is 5–7% faster than GLV with the same complete formulas, and its regular (constant-time) variant is 8–10% faster than regular 2-GLV with complete formulas and 3–7% faster than 2-GLV with incomplete Jacobian formulas. In $\mathbb{G}_2$, the τ-NAF is 4–11% faster than GLS in variable time, but in constant time the cheap Jacobian doubling over extension fields makes the exception-free variant 10–19% slower than Jacobian GLS. Finally, an 'imaginary' Toom-Cook variant for the extension fields makes $\mathbb{G}_T$ exponentiation 4–6% faster on BLS24-509 and BLS48-575, and a cyclotomic cube costing 0.77 multiplications brings the τ-adic method to $\mathbb{G}_T$, where it makes exponentiation 2–3% faster on BLS24-509 and 11% faster on BLS48-575.

1 Introduction

Complete formulas remove exceptional cases from elliptic-curve arithmetic, so constant-time code needs no branches and no exception analysis. Let M and S be the respective costs of multiplication and squaring in a field where the curve is defined. For prime-order short Weierstrass curves the standard choice is Renes, Costello and Batina (RCB) [1], costing $12M$ per addition in homogeneous coordinates. Curves $y^2 = x^3 + b$ over $\mathbb{F}_p$ with $p \equiv 1 \pmod 3$ also carry the endomorphism $\omega(x, y) = (\beta x, y)$, $\beta^3 = 1$. GLV uses it for a two-dimensional decomposition [3] that decomposes a scalar multiplication into two half-size ones. Recently, Xu, Yu, Han and Lu used the norm-3 element $\tau = 1 - \omega$ as the radix of a τ-NAF [4] inspired by binary Koblitz curves.

This report contributes:

- **An $11M$ complete addition ($10M$ mixed).** RCB's final stage is a cross product, computable with 5 multiplications instead of 6, for general a, $a = -3$ and $a = 0$ (Section 2).

- **A complete tripling at $4S + 6M$.** The 3-isogeny $\psi : E_b \to E_{-27b}$ has a complete homogeneous form with cost $2S + 3M$ (Section 3).
- **An exception-free τ-NAF.** Width 3, a U-invariant digit set and a six-point affine table, built entirely from complete formulas (Section 4).
- **A corrected cost comparison.** We correct the operation counts in [4], adjust the GLV comparison baseline, and evaluate the options at equal security properties (Section 5).
- **Faster $\mathbb{G}_T$ arithmetic.** An 'imaginary' Toom-Cook multiplication for extension fields over primes $p \equiv 3 \pmod 4$ (Section 6.1), and a cyclotomic cube that brings the τ-adic method to $\mathbb{G}_T$ exponentiation (Section 7).
- **Implementation and benchmarks.** Variable-time and constant-time τ-adic scalar multiplication in RELIC, measured against GLV in $\mathbb{G}_1$ of the BLS12, BLS24 and BLS48 families, with an extension to $\mathbb{G}_2$ through GLS, and the complete formulas in `libsecp256k1` for secp256k1 (Section 7). Along the way, we implement the 2-GLV method of [14], an adaptation of [13], at the level of optimization of the state of the art, which to our knowledge gives its first fair comparison with other methods: it is 3–4% faster than RELIC's previous regular GLV.

All formulas were checked against the affine and Jacobian ones (Section 8). Costs are field operations with $S = M$ unless stated, m_c denotes multiplication by a curve constant, and a an addition.

2 Faster complete addition: the cross-product block

For $E : Y^2Z = X^3 + aXZ^2 + bZ^3$, RCB's prefix computes $t_0 = X_1X_2$, $t_1 = Y_1Y_2$, $t_2 = Z_1Z_2$ and the three symmetric sums $t_3 = X_1Y_2 + X_2Y_1$, $t_4 = X_1Z_2 + X_2Z_1$, $t_5 = Y_1Z_2 + Y_2Z_1$ ($6M$ with Karatsuba). With $b_3 = 3b$ it forms

$$\begin{aligned} &A = t_3, && B = t_1 - at_4 - b_3t_2, && C = t_5,\\ &D = a(t_0 - at_2) + b_3t_4, && E = 3t_0 + at_2, && F = t_1 + at_4 + b_3t_2, \end{aligned}$$

and outputs $X_3 = AB - CD$, $Y_3 = ED + FB$, $Z_3 = CF + AE$ ($6M$). These six monomials form a 6-cycle in which each output takes two opposite edges. Up to a sign, the triple is a cross product:

$$(X_3, Y_3, Z_3) = (-w_2,\ w_1,\ -w_3), \qquad w = (A, F, D) \times (C, -E, B).$$

The cross product of two 3-vectors has bilinear rank 5. One decomposition with 8 additions:

$$\begin{aligned} &s = F + D, && r = B - C, && m_0 = sr, && m_1 = FC, \quad m_2 = D(E - r),\\ &m_3 = (A - s)B, && m_4 = AE, \\ &X_3 = m_0 + m_1 + m_3, && Y_3 = m_0 + m_1 + m_2, && Z_3 = m_1 + m_4. \end{aligned}$$

These are polynomial identities in free $A, \ldots, F$, so completeness is inherited from RCB. The rank-5 decomposition of the cross product is classical in algebraic complexity; we have not found it applied to complete addition laws before. An exhaustive search over the 396 rank-5 decompositions with coefficients in $\{-1, 0, 1\}$ finds none with fewer additions. Against the same prefix the trade is always $-1M + 5a$ (Table 1).

The technique comes from the twisted Hessian addition due to Hisil [5], formula `add-2010-h` in the EFD [6], which has the same 6-cycle final stage and improves $12M \to 11M$ the same way. The two tensors coincide up to relabelling; the decomposition above uses 8 additions against roughly 16 plus a doubling in Hisil's.

Variant	RCB block	$5M$ block
general a	$12M + 5m_c + 23a$	$11M + 5m_c + 28a$
general a, mixed	$11M + 5m_c + 17a$	$10M + 5m_c + 22a$
$a = -3$	$12M + 2m_c + 29a$	$11M + 2m_c + 34a$
$a = -3$, mixed	$11M + 2m_c + 23a$	$10M + 2m_c + 28a$
$a = 0$	$12M + 2m_c + 19a$	$11M + 2m_c + 24a$
$a = 0$, mixed	$11M + 2m_c + 13a$	$10M + 2m_c + 18a$

Table 1: Complete addition costs with the RCB final block and with the $5M$ block.

3 Complete tripling via the 3-isogeny

On $E_b : y^2 = x^3 + b$ the points with $x = 0$ form, with O, a subgroup of order 3 defined over $\mathbb{F}_p$. The quotient isogeny $\psi : E_b \to E_{-27b}$ is

$$\psi(x, y) = \left(\frac{x^3 + 4b}{x^2},\ \frac{y\,(x^3 - 8b)}{x^3} \right).$$

In homogeneous coordinates this gives $\big(X(4Y^2Z - 3X^3) \ :\ Y(9X^3 - 8Y^2Z) \ :\ X^3Z\big)$, which vanishes at O. Substituting $X^3 = Z(Y^2 - bZ^2)$ from the curve equation pulls out a common factor Z:

$$\psi(X : Y : Z) = \big(X(Y^2 + 3bZ^2) \ :\ Y(Y^2 - 9bZ^2) \ :\ Z(Y^2 - bZ^2)\big).$$

This costs $2S + 3M$ plus multiplications by the small constants b, $3b$ and $9b$. The isogeny is classical, but we have not found this complete homogeneous form in the literature.

Completeness. Write $\psi(X : Y : Z) = (X' : Y' : Z')$. The output is never $(0, 0, 0)$ on E_b. If $Z = 0$ the point is $O = (0 : 1 : 0)$ and $Y' = Y^3 \neq 0$. If $Z \neq 0$, $Z' = 0$ forces $Y^2 = bZ^2$, and $Y' = 0$ forces $Y = 0$ or $Y^2 = 9bZ^2$; each combination gives $bZ^2 = 0$, a contradiction. The image is correct everywhere: $O \mapsto O$ and $(0 : \pm\sqrt{b} : 1) \mapsto (0 : 1 : 0) = O$.

Tripling. Applying ψ again with $-27b$ lands on E_{729b}, which maps back to E_b by $\iota : (X : Y : Z) \mapsto (3X : Y : 27Z)$ since $729 = 3^6$. Hence

$$[3]P = \iota(\psi_{-27b}(\psi_b(P))), \qquad \text{cost } 4S + 6M,$$

which is complete and matches the incomplete Jacobian tripling of [4]. Writing [3] as an isogeny of degree 3 composed with its dual follows Doche, Icart and Kohel [7]. It uses no β, so it holds for every $j = 0$ curve, including $p \equiv 2 \pmod 3$.

The τ map. With $\omega(x, y) = (\beta x, y)$ and $\tau = 1 - \omega$, τ is ψ followed by the isomorphism $E_{-27b} \to E_b$, $(X : Y : Z) \mapsto \big((1 - \beta)X : Y : (1 - \beta)^3 Z\big)$, using $(1 - \beta)^2 = -3\beta$. Only the precomputation needs τ.

Doubling. The complete doubling used for GLV comparisons is

$$\big(2XY(Y^2 - 9bZ^2) \ :\ (Y^2 + 9bZ^2)^2 - 108b^2Z^4 \ :\ 8Y^3Z\big)$$

at $4M + 4S$. It is the point update of the pairing doubling step of Costello–Lange–Naehrig and Aranha et al. [9, 10], so it is not new. Over $\mathbb{F}_p$, where a squaring costs almost as much as a multiplication, the same point follows faster from Algorithm 9 of [1] at $6M + 2S$, which needs fewer additions and multiplications by small constants; our implementation uses that one over $\mathbb{F}_p$ and the formula above over extension fields.

4 Exception-free τ-NAF

Setting. Let $G \subset E_b(\mathbb{F}_p)$ have prime order $n \equiv 1 \pmod 3$, and let λ be the eigenvalue of ω on G, so $\omega P = \lambda P$. A scalar k is decomposed as $k \equiv k_1 + k_2\lambda \pmod n$ with $|k_i| \approx \sqrt{n}$ (Gauss-reduced lattice, Babai rounding) and rewritten as $a + b\tau$ with $a = k_1 + k_2$, $b = -k_2$. The element $a + b\tau \in \mathbb{Z}[\omega]$ has norm about n, so its τ-adic expansion has about $\log_3 n$ digits.

Width-3 digit set. By [4, Lemma 4.1], τ^3 divides $x + y\tau$ iff $9 \mid x$ and $3 \mid y$, so the residue classes mod τ^3 not divisible by τ are the 18 pairs ($x \bmod 9, y \bmod 3$) with $3 \nmid x$. The six units act freely on them, so three orbits suffice. We take

$$C = U \cdot \{\, 1,\ -2,\ -1 - \tau \,\}, \qquad U = \{\pm 1, \pm\omega, \pm\omega^2\},$$

whose 18 elements hit all classes with norms 1, 4 and 7, below the termination bound 27. Recoding is Algorithm 1 of [4] with moduli $(9, 3)$: subtract the digit for the current class whenever $3 \nmid a$, then divide by τ via $(a, b) \leftarrow (a+b, -a/3)$. Nonzero digits have density $1/(w+\frac{1}{2})$, simulated as $1/3.46$ for $w = 3$ and $1/4.41$ for $w = 4$ at 256 bits.

Evaluation. Using $\tau^2 = -3\omega$, a digit d at position $i = 2m + \delta$ contributes $[d(-\omega)^m] \cdot \tau^\delta \cdot 3^m$. The bracket is again a digit because C is U-invariant. The loop is therefore Horner in base 3 (Algorithm 1). Neither operation has exceptional cases, so R may start at O and no coincidence $R = \pm Q$ needs handling.

Algorithm 1: Exception-free width-3 τ-NAF evaluation

Input: digits $d_0, \ldots, d_{\ell-1}$ of $a + b\tau$; table of cP and τcP for $c \in C$
Output: kP

$R \leftarrow O$;
for $m = \lfloor (\ell - 1)/2 \rfloor$ **to** 0 **do**
  $R \leftarrow [3]R$; `// complete tripling,` $4S + 6M$
  foreach *nonzero* d_i *with* $i = 2m + \delta$, $\delta \in \{0, 1\}$ **do**
    $R \leftarrow R + \tau^\delta \cdot [d_i(-\omega)^m] \cdot P$; `// complete mixed addition,` $10M$
return R;

Table. We store P, $-2P$ and $(-1 - \tau)P$ and their τ-images in affine form: six points, 12 field elements, 384 bytes for 256-bit p. All 36 points the loop needs follow from them: $\omega(x, y) = (\beta x, y)$, $\omega^2(x, y) = (-x - \beta x, y)$ and negation flips y, so reading a digit costs at most one multiplication by β. Storing βx for each point as well, 18 field elements, makes the images free, as in our implementation (Section 7). Precomputation is $2P$ and ρP for $\rho = 1 + \tau$, whose negations are the table entries, jointly ($16M$, as in [4]), three complete τ-images, and one batch inversion, after normalizing P if it is not affine: about $70M$ plus the inversions.

Conditions. The multiplications by $3b$, $9b$ and $27b$ must be cheap: $b = 7$ (secp256k1), 2 (BN-P254), 4 (BLS12-381), 1 (BLS24-509), 4 (BLS48-575) and 1 (BLS12-377) all qualify. The addition is exceptional only when $P - Q$ has order 2 [2, 1]. Scalar multiplication adds only points of G, whose order is odd, so it is exception-free on every curve above. When $\#E(\mathbb{F}_p)$ is also odd, as for secp256k1, BN-P254, BLS12-381 and BLS48-575, the addition is complete on all of $E(\mathbb{F}_p)$, which also covers inputs outside G. On BLS12-377, which has the point $(-1, 0)$ of order 2, and on BLS24-509, whose cofactor is even, cofactor clearing outside G is not covered.

5 Cost analysis and comparison with GLV

Model. Costs are in $\mathbb{F}_p$ multiplications with $S = M$, using simulated digit counts for τ-NAF and standard w-NAF densities for GLV (two half-length scalars, interleaved). Incomplete variants use Jacobian coordinates (doubling $7M$, mixed addition $11M$ [6]) with a co-Z table on an isomorphic curve, which needs no inversion. Exception-free variants use the complete homogeneous formulas (doubling $8M$, tripling $10M$, mixed addition $10M$) with an affine table and one inversion implemented as in Bernstein–Yang [16], taken as $100M$ at 256 bits and scaled linearly.

Correcting the accounting of [4]. The optimised τ-NAF (their Algorithm 3) triples once per pair of τ-positions, so it needs about $\frac{1}{2}\log_3 n$ triplings, not the $2\log_3 n/4.5$ counted in [4]. Simulating their recoding on 2,000 secp256k1-sized scalars gives 80.0 triplings against 71.8 counted, and $1551M$ against their $1476M$. Their GLV baseline also uses width 4, while width 5 is cheaper in their own model ($1690M$ against $1774M$). With both corrections their 256-bit margin is 8.3% rather than 16.7%, and about 7.5–7.8% rather than 17.6–18% at 384–512 bits.

Comparison. Table 2 gives absolute costs, each GLV column being the better of widths 4 and 5; Table 3 gives the resulting margins.

	GLV (best of $w = 4, 5$)		τ-NAF, $w = 3$	
Bits	incomplete	exception-free	incomplete	exception-free
256	$1535M$	$1671M$	$1392M$	$1434M$
384	$2220M$	$2446M$	$2053M$	$2122M$
512	$2903M$	$3221M$	$2713M$	$2809M$

Table 2: Scalar multiplication costs ($S = M$).

	τ-NAF vs GLV		
Bits	exc.-free vs exc.-free	exc.-free vs incomplete	incomplete vs incomplete
256	14.1%	6.6%	9.3%
384	13.3%	4.4%	7.5%
512	12.8%	3.2%	6.5%

Table 3: τ-NAF advantage over GLV.

Why completeness favours τ-NAF. Tripling costs the same in homogeneous and Jacobian coordinates, and the complete mixed addition ($10M$) is cheaper than the Jacobian one ($11M$). GLV's complete doubling costs $1M$ more than in Jacobian coordinates, on every one of about 128 doublings. Exception-freeness therefore costs τ-NAF about 3% at 256 bits and GLV about 9%. Measured in $\mathbb{G}_1$ of the BLS curves (Table 5), it costs τ-NAF 1–4% and GLV up to 8%, and GLV on BLS12-381 is even 2% faster with complete formulas, whose projective table needs no inversion.

Window choice. Width 4 saves about 10 additions over width 3 but needs a table of 18 points instead of 6, at about $180M$ more precomputation. With $16M$ additions, as in [4], width 4 wins; with 10–$11M$ additions, width 3 wins at 256 bits ($1388M$ against $1472M$ in the incomplete model), the two tie near 384 bits, and width 4 is marginally better at 512 bits.

Sensitivity. At $S = 0.8M$ the tripling $(4S + 6M)$ gains less than doubling $(2M + 5S)$, and the exception-free margin over exception-free GLV falls to about 13%. The method applies to $\mathbb{G}_1$ of BN and BLS families and to secp256k1. In BLS12-381 $\mathbb{G}_2$, where GLS gives a four-dimensional decomposition and $\mathbb{F}_{p^2}$ squarings are relatively cheap, a first estimate puts the gain at about 5% (incomplete) to 12–13% (exception-free); Section 7 measures 4% and 8%, as the complete tripling costs more doublings in cycles than in this model.

6 Faster Toom-Cook multiplication

Let $a(x) := a_0 + a_1x + a_2x^2$ and $b(x) := b_0 + b_1x + b_2x^2$. Their product is $c(x) := a(x)b(x) := c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$.

Computing $c(x)$ with the 3-way divide-and-conquer Karatsuba algorithm [17] requires 6 multiplications over the base field. A potentially more efficient method is the 3-way evaluate-and-interpolate Toom (also known as Toom-Cook) algorithm [18].

The Toom-3 algorithm, which specializes the Toom-Cook method for polynomials of degree less than 3, consists of evaluating $a(x)$ and $b(x)$ at five well-chosen points, yielding $c(x)$ at those points upon multiplication, then using Lagrange interpolation to recover $c(x)$ from its value at those points.

The usual setting for odd or zero characteristic evaluates $c(x)$ at points 0, 1, -1, -2, and ∞, where the last point simply means taking the coefficient of the term of largest degree.

While this choice is simple and tends to work well, it is not without drawbacks. Contrary to the evaluation phase (where only multiplications by powers of 2 and sign flips are needed), the interpolation phase requires at least one division by 3, which is not as convenient (and in fact can be quite clumsy, depending on the platform).

Interestingly, in an unpublished manuscript, Bodrato [19] describes a Toom-3.5 algorithm over $\mathbb{F}_5[x]$ with the evaluation points 0, ± 1, $\pm i$ and ∞, where $i := \sqrt{-1}$, and observes that it generalises to any characteristic $p \equiv 1 \pmod 4$, and more generally to any integral domain containing $\pm i$. When i lies in the base field $\mathbb{F}_p$, however, multiplying by it is an essentially general multiplication in $\mathbb{F}_p$, which eats into the savings. In an extension $\mathbb{F}_{p^2} \simeq \mathbb{F}_p[i]/\langle i^2 + 1\rangle$ with $p \equiv 3 \pmod 4$, multiplying by i costs only a negation, and this is the setting we exploit: a Toom-3 algorithm with the single imaginary point i, whose interpolation needs only halvings and multiplications by $1 \pm i$, with explicit operation counts and an application to the extension fields of pairing-friendly curves.

6.1 'Imaginary' Toom-Cook

Let p be a prime with $p \equiv 3 \pmod 4$, and consider an extension field $\mathbb{F}_{p^k}$ with $6 \mid k$. In that case it is possible to consider the quadratic extension $\mathbb{F}_{p^2} \simeq \mathbb{F}_p[i]/\langle i^2 + 1\rangle$ and represent $\mathbb{F}_{p^k}$ as an extension of $\mathbb{F}_{p^2}$ of degree $k/2$ (possibly as a tower with intermediate fields of degree between 2 and k).

We propose to evaluate $c(x)$ at the five points 0, 1, -1, i, and ∞. The evaluation matrix then becomes

$$M := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & i & -1 & -i & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

whose inverse is

$$M^{-1} := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ i & \frac{1-i}{4} & -\frac{1+i}{4} & -i/2 & i \\ -1 & 1/2 & 1/2 & 0 & -1 \\ -i & \frac{1+i}{4} & -\frac{1-i}{4} & i/2 & -i \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In other words, compute:

$$\begin{bmatrix} a(0) \\ a(1) \\ a(-1) \\ a(i) \\ a(\infty) \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & i & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \quad \begin{bmatrix} b(0) \\ b(1) \\ b(-1) \\ b(i) \\ b(\infty) \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & i & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix},$$

then interpolate:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ i & \frac{1-i}{2} & -\frac{1+i}{2} & -i & i \\ -1 & 1 & 1 & 0 & -1 \\ -i & \frac{1+i}{2} & -\frac{1-i}{2} & i & -i \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a(0)b(0) \\ a(1)b(1)/2 \\ a(-1)b(-1)/2 \\ a(i)b(i)/2 \\ a(\infty)b(\infty) \end{bmatrix}.$$

The evaluation matrix involves multiplication by i, but the cost is essentially the same as a sign flip, since $i(u + vi) = -v + ui$. At first glance, the interpolation matrix may seem more complex, but apart from the divisions by 2 and 4 (easily carried out with shifts and exclusive-ors), the relevant operations are multiplications by $1 + i$ and $1 - i$, which are cheap: $(1 + i)(u + vi) = (u - v) + (u + v)i$, $(1 - i)(u + vi) = (u + v) + (v - u)i$, involving one addition and one subtraction (notice that both of these values are needed in the interpolation phase, so they can be computed together, amortizing the cost).

Algorithms 2–4 implement these auxiliary operations over $\mathbb{F}_p$, Algorithm 5 the evaluation, and Algorithms 6 and 7 the multiplication and squaring, with the interpolations of Algorithms 8 and 9.

The cost lines count operations over $\mathbb{F}_p$ for coefficients in $\mathbb{F}_{p^2}$. In $\mathbb{F}_{p^{24}}$, a cubic extension of $\mathbb{F}_{p^8}$, the same steps run with the five products in $\mathbb{F}_{p^8}$ and the linear steps on each $\mathbb{F}_{p^2}$ component of the coefficients, with lazy reduction.

Squaring gains less than multiplication. The Chung–Hasan squaring [20] costs four squarings and one multiplication in the coefficient field, or 11 multiplications over $\mathbb{F}_p$ with coefficients in $\mathbb{F}_{p^2}$, so Algorithm 7 saves a single $\mathbb{F}_p$ multiplication while its interpolation needs many more additions, subtractions and halvings. One level up, with coefficients in $\mathbb{F}_{p^8}$, the same trade-off makes Toom-3 squaring in $\mathbb{F}_{p^{24}}$ slower than Chung–Hasan in RELIC, so our implementation uses Toom-3 only for multiplication.

7 Implementation and benchmarks

Implementation. The methods are part of RELIC introduced as `EP_MUL=TWNAF` and `TWREG` configurations, next to the GLV methods `LWNAF` (interleaved w-NAF) and `LWREG` (regular recoding). In $\mathbb{G}_1$, RELIC has since replaced `LWREG` with the 2-GLV method of Faraoun and El Mrabet [14], which adapts to prime curves the 2DT method of Aardal and Aranha for the binary curve GLS254 [13]: it reads one digit per position from a joint table of 8 combinations of P and $\psi(P)$. Its authors released a Rust implementation; ours is optimized like the other methods in RELIC, so that all methods are compared under similar levels of optimization. We report both versions. Both τ-adic methods share the recoding of Section 4, done in a single

Algorithm 2: $\mathsf{imul}(w)$: Multiplication by i

INPUT: $w := u + vi \in \mathbb{F}_{p^2}$.
OUTPUT: $iw \in \mathbb{F}_{p^2}$.
COST: 1 neg (over $\mathbb{F}_p$)
1: **return** $-v + ui$

Algorithm 3: $\mathsf{isub}(w_1, w_2)$: Subtraction and multiplication by i

INPUT: $w_1 := u_1 + v_1 i,\ w_2 := u_2 + v_2 i \in \mathbb{F}_{p^2}$.
OUTPUT: $i(w_1 - w_2) \in \mathbb{F}_{p^2}$.
COST: 2 sub (over $\mathbb{F}_p$)
1: **return** $(v_2 - v_1) + (u_1 - u_2)i$

Algorithm 4: $\mathsf{jmul}(w)$: Joint multiplication by $(1 \pm i)/2$

INPUT: $w := u + vi \in \mathbb{F}_{p^2}$.
OUTPUT: $(1+i)w/2,\ (1-i)w/2 \in \mathbb{F}_{p^2}$.
COST: 1 add, 1 sub, 1 neg, 2 hlv (all over $\mathbb{F}_p$)
1: $w_0 \leftarrow (u+v)/2$
2: $w_1 \leftarrow (u-v)/2$
3: **return** $w_1 + w_0 i,\ w_0 - w_1 i$

Algorithm 5: $\mathsf{eval}(a)$: Evaluation

INPUT: Polynomial $a(x) \in \mathbb{F}_{p^2}[x]$ of degree $\deg(a) < 3$.
OUTPUT: Evaluation of polynomial $a(x)$ at 0, 1, -1, i, and ∞.
COST: **5 add**, **5 sub** (all over $\mathbb{F}_p$)
1: $e_5 \leftarrow a_0 + a_2$ ▷ 2 add
2: $e_0 \leftarrow a_0$
3: $e_1 \leftarrow e_5 + a_1$ ▷ 2 add
4: $e_2 \leftarrow e_5 - a_1$ ▷ 2 sub
5: $e_3 \leftarrow a_0 - a_2 + \mathsf{imul}(a_1)$ ▷ NB: 1 add, 3 sub
6: $e_4 \leftarrow a_2$
7: **return** $(e_0, e_1, e_2, e_3, e_4)$ ▷ $a(0), a(1), a(-1), a(i), a(\infty)$

Algorithm 6: $\mathsf{mul}(a, b)$: Multiplication

INPUT: Polynomials $a(x),\ b(x) \in \mathbb{F}_{p^2}[x]$ of degrees $\deg(a) < 3,\ \deg(b) < 3$.
OUTPUT: Product polynomial $c(x) = a(x)b(x) \in \mathbb{F}_{p^2}[x]$.
COST: **15 mul**, **28 add**, **37 sub**, **10 hlv**, **2 neg** (all over $\mathbb{F}_p$)
1: **return** $\mathsf{interp}(\mathsf{eval}(a), \mathsf{eval}(b))$

Algorithm 7: $\mathsf{sqr}(a)$: Squaring

INPUT: Polynomial $a(x) \in \mathbb{F}_{p^2}[x]$ of degree $\deg(a) < 3$.
OUTPUT: Square polynomial $c(x) = a(x)^2 \in \mathbb{F}_{p^2}[x]$.
COST: **10 mul**, **23 add**, **22 sub**, **10 hlv**, **2 neg** (all over $\mathbb{F}_p$)
1: **return** $\mathsf{interp2}(\mathsf{eval}(a))$

carry pass with exact division by τ^3, and a table built with the complete τ map. With complete formulas in $\mathbb{G}_1$ the table stays projective, since an addition costs little more than a mixed one and this saves an inversion, as in RELIC's GLV tables. Otherwise it is normalized with one simultaneous inversion, which is faster in $\mathbb{G}_2$, where a multiplication costs more relative to an inversion. The images under ω and ω^2 are read as $(\beta x, y)$ and $(-x - \beta x, y)$, with βx computed once for each point, so the table holds 6 points and their βx for `TWNAF`. In $\mathbb{G}_1$, the GLV method likewise keeps βx with each point of its table, so that its images under the endomorphism cost no multiplication either.

- **`TWNAF`** evaluates Algorithm 1 in variable time, in the coordinates chosen at build time: complete homogeneous formulas, or incomplete Jacobian formulas.

- **`TWREG`** is a constant-time regular recoding with a nonzero digit at every third position, in base τ^3, following the regular τ-adic recodings of Oliveira et al. for Koblitz curves [11, 12]. The digits are $U \cdot \{1, 2, 1+\tau, 1-2\tau, 2+\tau, 1+2\tau\}$, so the table holds 12 points. The element is first incremented by $c \in \{0, 1\}$ so that τ does not divide it, and cP is subtracted at the end with masked copies. The expansion has a fixed length, derived from the norms of the reduced lattice basis (54 digits on BLS12-381 $\mathbb{G}_1$, followed by a top digit that is a unit). Each digit is read by scanning the six table entries for the parity of its position, which

Algorithm 8: $\mathsf{interp}(a, b)$: Interpolation for multiplication

INPUT: Evaluations of polynomials $a(x),\ b(x) \in \mathbb{F}_{p^2}[x]$ of degrees $\deg(a) < 3,\ \deg(b) < 3$ at 0, 1, -1, i, and ∞.
OUTPUT: $c(x) = a(x)b(x) \in \mathbb{F}_{p^2}[x]$.
COST: 5 Mul (= 15 mul, 10 add, 15 sub), 3 Add (= 6 add), 4 Sub (= 8 sub), 3 Hlv (= 6 hlv), 1 isub (= 2 sub), 2 jmul (= 2 add, 2 sub, 2 neg, 4 hlv), total: **15 mul**, **18 add**, **27 sub**, **10 hlv**, **2 neg** (all over $\mathbb{F}_p$)

1: $e_0 \leftarrow a(0) \cdot b(0)$
2: $e_1 \leftarrow a(1) \cdot b(1)\,/2$
3: $e_2 \leftarrow a(-1) \cdot b(-1)\,/2$
4: $e_3 \leftarrow a(i) \cdot b(i)\,/2$
5: $e_4 \leftarrow a(\infty) \cdot b(\infty)$
6: $e_5 \leftarrow e_0 + e_4$
7: $e_6, e_7 \leftarrow \mathsf{jmul}(e_1)$
8: $e_8, e_9 \leftarrow \mathsf{jmul}(e_2)$
9: $c_0 \leftarrow e_0$
10: $c_2 \leftarrow e_1 + e_2 - e_5$
11: $c_4 \leftarrow e_4$
12: $e_5 \leftarrow \mathsf{isub}(e_5, e_3)$
13: $c_1 \leftarrow e_7 - e_8 + e_5$
14: $c_3 \leftarrow e_6 - e_9 - e_5$
15: **return** $c(x) := c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$

Algorithm 9: $\mathsf{interp2}(a)$: Interpolation for squaring

INPUT: Evaluation of polynomial $a(x) \in \mathbb{F}_{p^2}[x]$ of degree $\deg(a) < 3$ at 0, 1, -1, i, and ∞.
OUTPUT: $c(x) = a(x)^2 \in \mathbb{F}_{p^2}[x]$.
COST: 5 Sqr (= 10 mul, 10 add, 5 sub), 3 Add (= 6 add), 4 Sub (= 8 sub), 3 Hlv (= 6 hlv), 1 isub (= 2 sub), 2 jmul (= 2 add, 2 sub, 2 neg, 4 hlv), total: **10 mul**, **18 add**, **17 sub**, **10 hlv**, **2 neg** (all over $\mathbb{F}_p$)

1: $e_0 \leftarrow a(0)^2$
2: $e_1 \leftarrow a(1)^2/2$
3: $e_2 \leftarrow a(-1)^2/2$
4: $e_3 \leftarrow a(i)^2/2$
5: $e_4 \leftarrow a(\infty)^2$
6: $e_5 \leftarrow e_0 + e_4$
7: $e_6, e_7 \leftarrow \mathsf{jmul}(e_1)$
8: $e_8, e_9 \leftarrow \mathsf{jmul}(e_2)$
9: $c_0 \leftarrow e_0$
10: $c_2 \leftarrow e_1 + e_2 - e_5$
11: $c_4 \leftarrow e_4$
12: $e_5 \leftarrow \mathsf{isub}(e_5, e_3)$
13: $c_1 \leftarrow e_7 - e_8 + e_5$
14: $c_3 \leftarrow e_6 - e_9 - e_5$
15: **return** $c(x) := c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$

is public, and selecting the coordinates and the power of ω with masks. Like the other methods it uses the configured coordinates. With the complete homogeneous formulas it is exception-free. With Jacobian formulas it is regular and constant time, but like Jacobian `LWREG` it falls back to branches in the rare exceptional cases.

The addition of Section 2 is RELIC's homogeneous addition for every curve, and the tripling of Section 3 its homogeneous tripling for $a = 0$, so the GLV baselines also use the $11M$ addition. The width-3 τ-adic methods ignore the window width; the GLV methods were built with widths 4 and 5, and each table entry below is the faster of the two. Other improvements made along the way, such as cheap multiplication by small b and skipping operations on the point at infinity, were applied to all methods.

Setup. Intel Xeon Silver 4314 (Ice Lake-SP) at 2.4 GHz with Turbo Boost disabled, GCC 15.2 with `-O3 -march=native`. Prime-field arithmetic uses RELIC's x64 assembly backends (`MULX`), with 6, 8 and 9 limbs for BLS12-381, BLS24-509 and BLS48-575. They implement $\mathbb{F}_p$ multiplication in 136, 222 and 282 cycles and addition in 16, 22 and 25. For secp256k1 we measure libsecp256k1 (upstream commit `e74a018`, default build with `-O2` and the 5×52 field with x86-64 assembly, multiplication about 60 cycles), and a modified copy where the same complete formulas replace the constant-time Jacobian ones. The modified copy also inlines the field's weak normalization, and we apply that change to upstream too, so that only the group formulas differ. Each operation runs on 2000 random scalars and points, with all methods interleaved on the same inputs and checked to agree. We report the median of user-space core cycles (`perf_event_open`) on one pinned core, and the smaller of two full passes. The two passes differ by at most 1.7% for every scalar multiplication in the tables. RELIC's tests of

point arithmetic and scalar multiplication pass in these builds, and its tests of points, extension fields, pairings and protocols pass in GMP builds for the same curves.

Point operations. Table 4 gives the formula costs in cycles. The $11M$ addition is 4–5% faster than RCB in $\mathbb{G}_1$ of the BLS curves, gaining more as a multiplication grows relative to an addition. The complete tripling costs 1.36–1.43 complete doublings, against $10/8 = 1.25$ in the model of Section 5: in cycles, its two squarings and its multiplications by the constants $3b$ and $9b$ are not free. The Jacobian columns are the incomplete formulas used by the incomplete variants. The secp256k1 row comes from libsecp256k1, which has only the $5M$ block and no Jacobian tripling. Its Jacobian columns are the constant-time formulas that the complete ones replace (`gej_add_ge`, `gej_double`). The mixed addition handles the doubling case without branches.

	homogeneous, complete						Jacobian, incomplete		
	add		mixed add						
Curve	RCB	$5M$	RCB	$5M$	dbl	tpl	madd	dbl	tpl
secp256k1[†]	–	865	–	788	507	723	839[‡]	430[‡]	–
BLS12-381	1,887	1,820	1,679	1,609	1,185	1,695	1,654	1,100	1,531
BLS24-509	3,050	2,913	2,720	2,588	1,870	2,618	2,606	1,715	2,429
BLS48-575	3,886	3,711	3,492	3,316	2,390	3,255	3,253	2,088	3,032

Table 4: Point operations in $\mathbb{G}_1$, in cycles. [†]libsecp256k1; the other rows are RELIC. [‡]libsecp256k1's constant-time Jacobian formulas, the baseline of its constant-time code; the variable-time ones take 705 and 436 cycles.

Scalar multiplication. Tables 5 and 6 give the results, and Table 7 the number of precomputed points of each method. In $\mathbb{G}_1$ of the BLS curves:

- With complete formulas on both sides, the τ-NAF is 5–7% faster than GLV, against about 13% in the model at 384–512 bits. The model's exception-free GLV normalizes its table with one inversion; RELIC's GLV method keeps the table projective, which is faster in $\mathbb{G}_1$, so the measured margins are below the model's. With incomplete Jacobian formulas on both sides it is 4–7% faster.
- The constant-time `TWREG` is 11–13% faster than the previous `LWREG` with the same complete formulas, and 6–11% faster than it with incomplete Jacobian formulas, which is constant time but not exception-free. Against 2-GLV, which is 3–4% faster than the previous `LWREG`, the margins are 8–10% and 3–7%. In $\mathbb{G}_1$, exception-freeness therefore comes with a speed-up rather than a cost. With Jacobian formulas on both sides, `TWREG` is 10–12% faster than the previous `LWREG` and 7–8% faster than 2-GLV.

$\mathbb{G}_2$. The methods extend to GLS curves: when $\psi^{S/2}$ acts as a unit $u \in \mathbb{Z}[\omega]$ for an S-dimensional decomposition $\sum_{i<S} k_i\psi^i$, the scalar regroups as $\sum_{i<S/2}(k_i + k_{i+S/2}u)\,\psi^i$, whose elements of $\mathbb{Z}[\omega]$ are recoded and interleaved. On $\mathbb{G}_2$ of the three BLS curves the τ-NAF stays 8–11% faster than GLS with complete formulas and 4–9% with Jacobian formulas (Tables 5 and 6). The regular variant gains less: with the same formulas, `TWREG` ranges from 1% slower on BLS12-381 to 4–5% faster on BLS24-509 and BLS48-575 with complete formulas, and from 3% slower to 2% faster with Jacobian ones. Over $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^4}$ and $\mathbb{F}_{p^8}$, the Jacobian doubling costs only 0.71–0.82 complete doublings, so exception-free constant time does not pay off there: `TWREG` with complete formulas loses 10–19% to Jacobian GLS.

	variable time				constant time			
	GLV w-NAF		τ-NAF		GLV regular		τ regular	
Curve	Jac.	compl.	Jac.	compl.	Jac.	compl.	Jac.	compl.
$\mathbb{G}_1$								
BLS12-381	266	262	248	250	337	347	297	301
2-GLV [14]					324	333		
BLS24-509	615	630	578	596	757	781	674	694
2-GLV [14]					728	750		
BLS48-575	920	990	885	924	1,128	1,212	1,016	1,065
2-GLV [14]					1,095	1,179		
$\mathbb{G}_2$								
BLS12-381	441	493	421	452	540	592	558	596
BLS24-509	2,738	3,419	2,557	3,096	3,502	4,347	3,504	4,174
BLS48-575	11,494	14,101	10,414	12,498	15,220	18,565	14,878	17,577

Table 5: Scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$, in thousands of cycles. "Jac." uses incomplete Jacobian formulas, "compl." the complete homogeneous ones. GLV columns are the faster of widths 4 and 5; in $\mathbb{G}_2$ they are GLS. In $\mathbb{G}_1$, the GLV regular columns are RELIC's previous `LWREG`, measured with the same setup before it was replaced, and the indented lines its current one, 2-GLV [14].

$\mathbb{G}_T$. Table 8 measures the 'imaginary' Toom-Cook multiplication of Section 6.1 in $\mathbb{G}_T$ of BLS24-509 and BLS48-575. It replaces RELIC's Karatsuba products in $\mathbb{F}_{p^{24}}$, and in $\mathbb{F}_{p^{48}}$ the $\mathbb{F}_{p^{24}}$ products of its Karatsuba step, while squarings and cyclotomic squarings stay unchanged. The products by the sparse line functions, whose top coefficient over $\mathbb{F}_{p^8}$ is zero, use a Toom-2.5 variant instead: for $b(x) = b_0 + b_1 x$, the product $a(x)b(x)$ follows from $a(1)b(1)$, $a(-1)b(-1)$, $a_0 b_0$ and $a_2 b_1$, four products in $\mathbb{F}_{p^8}$ against five with Karatsuba, and needs no evaluation at i. Exponentiation gets 4–6% faster, variable time or constant time, and the pairing 2–3%. The gains are limited because squarings, which Toom-Cook leaves unchanged, dominate exponentiation.

Cyclotomic cubing and τ-adic exponentiation. The method of Section 4 carries over to $\mathbb{G}_T$ when $6 \mid k$. The Frobenius map π acts on $\mathbb{G}_T$ with $\pi^{k/3}$ as a cube root of unity ω, so the exponent decomposes along powers of π as in $\mathbb{G}_2$, its coefficients in $\mathbb{Z}[\omega]$ are recoded as τ-adic NAFs, and the loop of Algorithm 1 runs with a cyclotomic cube in place of each tripling and a multiplication in place of each addition. Write $a \in \mathbb{F}_{p^k}$ as $A + Bs + Cs^2$ with $A, B, C \in \mathbb{F}_{p^{k/3}}$ and $s^3 = z$, where the conjugation $\bar{\cdot}$, the $p^{k/2}$-power Frobenius, maps s to $-s$, and let N be the norm from $\mathbb{F}_{p^{k/3}}$ to $\mathbb{F}_{p^{k/6}}$. Membership in the cyclotomic subgroup, $a^{p^{k/2}+1} = 1$, gives $A\bar{A} = 1 - z(B\bar{C} - \bar{B}C)$, $\bar{A}B - A\bar{B} = -zN(C)$ and $A\bar{C} + \bar{A}C = N(B)$, hence

$$a^3 = 9A(A^2 - \bar{A}) + 1 + 3\big(B(3A^2 - 2\bar{A}) - zN(C)\big)s + 3\big(C(3A^2 - 2\bar{A}) + N(B)\big)s^2,$$

with three multiplications and one squaring in $\mathbb{F}_{p^{k/3}}$ and two norms. Ben Abdelghani and Haddaji [8] recently used cyclotomic cubing in the final exponentiation, in $\mathbb{F}_{q^6}$ with $q \equiv 1 \pmod 6$. In RELIC this cube costs 0.77 multiplications in $\mathbb{F}_{p^{12}}$, $\mathbb{F}_{p^{24}}$ and $\mathbb{F}_{p^{48}}$, against 0.37–0.41 for a cyclotomic squaring. The τ-adic exponentiation is then 2–3% faster than the GLS one on BLS24-509, depending on the product, and 11% faster on BLS48-575; together with Toom-Cook it is 6.1% and 15% faster than the Karatsuba baseline (Table 8). On BLS12-381 it is 6% slower.

secp256k1. Table 9 gives libsecp256k1's scalar multiplications: its two GLV methods for a variable point, and its fixed-base comb. `ecmult` is the variable-time interleaved w-NAF with Jacobian formulas and the effective-affine table. `ecmult_const` is the constant-time signed-digit

	τ-NAF vs GLV w-NAF			τ regular vs GLV regular		
Curve	compl./compl.	Jac./Jac.	compl./Jac.	compl./compl.	Jac./Jac.	compl./Jac.
$\mathbb{G}_1$						
BLS12-381	−4.6%	−7.0%	−6.0%	−13.3%	−11.7%	−10.7%
2-GLV [14]				−9.6%	−8.2%	−7.1%
BLS24-509	−5.5%	−5.9%	−3.1%	−11.2%	−10.9%	−8.4%
2-GLV [14]				−7.5%	−7.4%	−4.8%
BLS48-575	−6.6%	−3.8%	+0.4%	−12.1%	−10.0%	−5.6%
2-GLV [14]				−9.6%	−7.3%	−2.7%
$\mathbb{G}_2$						
BLS12-381	−8.2%	−4.4%	+2.6%	+0.6%	+3.3%	+10.4%
BLS24-509	−9.4%	−6.6%	+13.0%	−4.0%	+0.0%	+19.2%
BLS48-575	−11.4%	−9.4%	+8.7%	−5.3%	−2.2%	+15.5%

Table 6: Relative cost of the τ-adic methods against GLV, from Table 5 (negative is faster); "compl./Jac." compares the τ-adic method with complete formulas against GLV with Jacobian formulas. The indented lines compare `TWREG` with 2-GLV [14].

Curve	GLV w-NAF	τ-NAF	GLV regular	2-GLV	τ regular
$\mathbb{G}_1$					
BLS12-381	8	6	8	8	12
BLS24-509	8	6	8	8	12
BLS48-575	8	6	8	8	12
$\mathbb{G}_2$					
BLS12-381	16	12	8	–	24
BLS24-509	32 / 64	24	16	–	48
BLS48-575	64	48	32	–	96

Table 7: Precomputed points of the methods in Table 5. The GLV w-NAF tables depend on the faster width, given as Jacobian / complete where they differ; the other counts do not depend on the formulas. In $\mathbb{G}_1$, "GLV regular" is the previous `LWREG`; in $\mathbb{G}_2$ it is RELIC's regular GLS. Points are affine, except with complete formulas the GLV w-NAF and τ-adic tables in $\mathbb{G}_1$, which stay projective. The GLV w-NAF tables in $\mathbb{G}_1$ and the τ-adic tables also keep βx for each point, for the images under the endomorphism.

method, with 5-bit groups and a 16-point table. `ecmult_gen` is a constant-time comb with 43 blocks of 6 teeth, so it performs 44 mixed additions and no doublings. With the complete formulas, `ecmult_const` becomes exception-free and 10.2% slower; the model of Section 5 predicts about 9% for GLV. Against the constant-time Jacobian formulas they replace, the complete mixed addition is 6.1% cheaper, but the complete doubling costs 18% more. So the fixed-base comb, which only adds, gets 5.2% faster with the complete formulas, while the variable-base method, with 125 doublings and 51 additions, gets slower. Per bit of scalar, the complete tripling costs $723/\log_2 3 = 456$ cycles, against 507 for the complete doubling and 430 for the Jacobian one. This is the margin a τ-adic method would work with, but libsecp256k1 has no τ-adic method, so we have no measurement of one.

8 Verification

- **Formulas.** RELIC's tests compare the complete addition, mixed addition, doubling and tripling in homogeneous coordinates with the affine and Jacobian formulas on random points, and check the identity on either side, $P = -Q$ and inputs that alias the output, on

Curve	$\mathbb{F}_{p^k}$ product	exp. (var. time)	exp. (τ-adic)	exp. (const. time)	pairing
BLS24-509	Karatsuba	4,468	4,350	5,672	10,356
	Toom-Cook	4,299	4,197	5,424	10,100
	change	−3.8%	−3.5%	−4.4%	−2.5%
	τ-adic vs var. time		−2.4%		
BLS48-575	Karatsuba	20,556	18,220	25,028	41,439
	Toom-Cook	19,620	17,388	23,636	40,322
	change	−4.6%	−4.6%	−5.6%	−2.7%
	τ-adic vs var. time		−11.4%		

Table 8: Exponentiation in $\mathbb{G}_T$ and the optimal ate pairing, in thousands of cycles. The τ-adic column uses cyclotomic cubing; the others are RELIC's GLS methods. "change" compares Toom-Cook with Karatsuba; "τ-adic vs var. time" compares the two exponentiations with Toom-Cook products. The products in $\mathbb{F}_{p^{24}}$ and $\mathbb{F}_{p^{48}}$ are RELIC's Karatsuba ones or the Toom-Cook ones of Section 6.1 (median of 500 random exponents and 100 pairings, width 5, smaller of two passes).

	Jacobian	complete	change
`ecmult` (variable base, variable time, w-NAF)	96,400	–	–
`ecmult_const` (variable base, constant time, regular)	122,500	135,100	+10.2%
`ecmult_gen` (fixed base, constant time, comb)	51,900	49,200	−5.2%

Table 9: libsecp256k1 scalar multiplication on secp256k1, in cycles (median of 2,000 random scalars, two passes within 0.5%). "Jacobian" is upstream, "complete" the copy with the complete formulas.

every curve of Section 7. The cross-product block is a polynomial identity in free $A, \ldots, F$, which we also checked symbolically.

- **Recoding.** The width-3 digit set is checked at load time to cover all 18 classes with norms below 27. Simulated densities agree with $1/(w + \frac{1}{2})$ to within 1–2%.
- **Constant time.** The regular variant has no secret-dependent branches or memory accesses by construction. Unlike the constant-time variant of [4], which uses two coefficient sets and τ^4-blocks, it uses one digit set in base τ^3 on complete formulas. The fixed-versus-random test of dudect [15], timed in core cycles, finds no leakage on BLS12-381, with the fixed scalar either random or the short $2^{64}+1$, whose subscalars are short or zero: $|t| < 1.6$ after about 21 million measurements in $\mathbb{G}_1$, and $|t| < 2.5$ after about 14 million in $\mathbb{G}_2$. The lattice decomposition of the scalar, shared with RELIC's GLV methods, and the regular recoding are constant time for scalars of absolute value below the group order: the two together and the recoding alone give $|t| < 2.3$ after 79–435 million measurements.

AI disclosure

The authors conceived the main ideas and formulas in this work and wrote their initial implementation. The 5M block formula was derived in an exploratory Claude session, and the imaginary Toom-Cook was discovered manually, without AI. Assistance from Claude Opus 5.5 was used mainly in the execution/refinement: under the authors' direction and review, it wrote and revised parts of the RELIC and libsecp256k1 code, such as the scalar multiplication templates, optimizations and constant-time fixes, ran the benchmarks and the dudect experiments, and produced the tables from their output. The authors verified the results and take full responsibility for the content.

References

[1] J. Renes, C. Costello, L. Batina. Complete addition formulas for prime order elliptic curves. EUROCRYPT 2016, LNCS 9665, pp. 403–428.

[2] W. Bosma, H. W. Lenstra Jr. Complete systems of two addition laws for elliptic curves. Journal of Number Theory 53, pp. 229–240, 1995.

[3] R. Gallant, R. Lambert, S. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. CRYPTO 2001, LNCS 2139, pp. 190–200.

[4] G. Xu, W. Yu, K. Han, P. Lu. On efficient computations of $y^2 = x^3 + b/\mathbb{F}_p$ for primes $p \equiv 1 \pmod 3$. IACR ePrint 2024/1906, version of 30 September 2026.

[5] H. Hisil. Elliptic curves, group law, and efficient computation. PhD thesis, Queensland University of Technology, 2010.

[6] D. J. Bernstein, T. Lange. Explicit-Formulas Database. `https://hyperelliptic.org/EFD/`

[7] C. Doche, T. Icart, D. R. Kohel. Efficient scalar multiplication by isogeny decompositions. PKC 2006, LNCS 3958, pp. 191–206.

[8] L. Ben Abdelghani, W. Haddaji. Optimized final exponentiation for optimal ate pairings using cyclotomic cubing. IACR ePrint 2026/885.

[9] C. Costello, T. Lange, M. Naehrig. Faster pairing computations on curves with high-degree twists. PKC 2010, LNCS 6056, pp. 224–242.

[10] D. F. Aranha, K. Karabina, P. Longa, C. Gebotys, J. López. Faster explicit formulas for computing pairings over ordinary curves. EUROCRYPT 2011, LNCS 6632, pp. 48–68.

[11] T. Oliveira, D. F. Aranha, J. López, F. Rodríguez-Henríquez. Fast point multiplication algorithms for binary elliptic curves with and without precomputation. SAC 2014, LNCS 8781, pp. 324–344.

[12] T. Oliveira, J. López, D. Cervantes-Vázquez, F. Rodríguez-Henríquez. Koblitz curves over quadratic fields. Journal of Cryptology 32(3), pp. 867–894, 2019.

[13] M. A. Aardal, D. F. Aranha. 2DT-GLS: Faster and exception-free scalar multiplication in the GLS254 binary curve. SAC 2022, LNCS 13742, pp. 53–74.

[14] K. M. Faraoun, N. El Mrabet. Optimizing and securing GLV multiplication over BLS pairings-friendly curves. Designs, Codes and Cryptography 93(9), pp. 3641–3669, 2025. Code at `https://github.com/kamel78/efficient-rust-pairings`.

[15] O. Reparaz, J. Balasch, I. Verbauwhede. Dude, is my code constant time? DATE 2017, pp. 1697–1702.

[16] D. J. Bernstein, B.-Y. Yang. Fast constant-time gcd computation and modular inversion. IACR TCHES 2019(3), pp. 340–398.

[17] A. Karatsuba, Yu. Ofman. Multiplication of multidigit numbers on automata. Soviet Physics Doklady 7, pp. 595–596, 1963.

[18] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. Soviet Mathematics Doklady 3, pp. 714–716, 1963.

[19] M. Bodrato. Notes on low degree Toom-Cook multiplication with small characteristic. Unpublished manuscript, 2009. http://marco.bodrato.it/papers/Bodrato-SmallCharacteristicLowDegreeToom.pdf

[20] J. Chung, M. A. Hasan. Asymmetric squaring formulae. 18th IEEE Symposium on Computer Arithmetic (ARITH 2007), pp. 113–122.