Four$\mathbb{Q}$: four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime

Craig Costello and Patrick Longa

Microsoft Research, USA
{craigco,plonga}@microsoft.com

Abstract. We introduce Four$\mathbb{Q}$, a high-security, high-performance elliptic curve that targets the 128-bit security level. At the highest arithmetic level, cryptographic scalar multiplications on Four$\mathbb{Q}$ can use a four-dimensional Gallant-Lambert-Vanstone decomposition to minimize the total number of elliptic curve group operations. At the group arithmetic level, Four$\mathbb{Q}$ admits the use of extended twisted Edwards coordinates and can therefore exploit the fastest known elliptic curve addition formulas over large prime characteristic fields. Finally, at the finite field level, arithmetic is performed modulo the extremely fast Mersenne prime $p = 2^{127} - 1$. We show that this powerful combination facilitates scalar multiplications that are significantly faster than all prior works. On Intel's Broadwell, Haswell, Ivy Bridge and Sandy Bridge architectures, our software computes a variable-base scalar multiplication in 50,000, 56,000, 69,000 cycles and 72,000 cycles, respectively; and, on the same platforms, our software computes a Diffie-Hellman shared secret in 80,000, 88,000, 104,000 cycles and 112,000 cycles, respectively. These results show that, in practice, Four$\mathbb{Q}$ is around four to five times faster than the original NIST P-256 curve and between two and three times faster than curves that are currently under consideration as NIST alternatives, such as Curve25519.

1 Introduction

This paper introduces a new, *complete* twisted Edwards [5] curve $\mathcal{E}(\mathbb{F}_{p^2}) : -x^2 + y^2 = 1 + dx^2y^2$, where p is the Mersenne prime $p = 2^{127} - 1$, and d is a non-square in $\mathbb{F}_{p^2}$. This curve, dubbed "Four$\mathbb{Q}$", arises as a special instance of recent constructions using $\mathbb{Q}$-curves [50, 28], and is thus equipped with an endomorphism ψ related to the p-power Frobenius map. In addition, it has *complex multiplication* (CM) by the order of discriminant $D = -40$, meaning it comes equipped with another efficiently computable, low-degree endomorphism ϕ [51].

We built an elliptic curve cryptography (ECC) library [16] that works inside the cryptographic subgroup $\mathcal{E}(\mathbb{F}_{p^2})[N]$, where N is a 246-bit prime. The endomorphisms ψ and ϕ do not give any practical speedup to Pollard's rho algorithm [46], which means the best known attack against the elliptic curve discrete logarithm problem (ECDLP) on $\mathcal{E}(\mathbb{F}_{p^2})[N]$ requires around $\sqrt{\pi N/4} \sim 2^{122.5}$ group operations on average. Thus, the cryptographic security of $\mathcal{E}$ (see §2.3 for more details) is closely comparable to other curves that target the 128-bit security level, e.g., [22, 41, 9, 6].

Our choice of curve and the accompanying library offer a range of advantages over existing curves and implementations:

- **Speed:** Four$\mathbb{Q}$'s library computes scalar multiplications significantly faster than all known software implementations of curve-based cryptographic primitives. It uses the endomorphisms ψ and ϕ to accelerate scalar multiplications via four-dimensional Gallant-Lambert-Vanstone (GLV)-style [23] decompositions. Four-dimensional decompositions have been used before [35, 41, 9], but not over *the* Mersenne prime[1]; this choice of field is significantly faster than any

[1] p stands alone as the only Mersenne prime suitable for high-security curves over quadratic extension fields. The next largest Mersenne prime is $2^{521} - 1$, which is suitable only for prime field curves targeting the 256-bit level.

neighboring fields and several works have studied its arithmetic [22, 40, 15]. The combination of extremely fast modular reductions and four-dimensional scalar decompositions makes for highly efficient scalar multiplications on $\mathcal{E}$. Furthermore, we can exploit the fastest known addition formulas for elliptic curves over large characteristic fields [34], which are complete on $\mathcal{E}$ since the above d is non-square [34, §3]. In Section 2, we explain why four-dimensional decompositions and this special underlying field were not previously partnered at the 128-bit security level.

– **Simplicity and concrete correctness:** Simplicity is a major priority in this work and in the development of our software; in some cases we sacrifice speed enhancements in order to design a more simple and compact algorithm (cf. §4.2 and Remark 7).
On input of *any* point $P \in \mathcal{E}(\mathbb{F}_{p^2})[N]$, validated as in Appendix A if necessary, and *any* integer scalar $m \in [0, 2^{256})$, our software does the following (strictly in constant-time and without exception):
  1. Computes $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$ using exactly[2] 68**M**, 27**S** and 49.5**A** – see Section 3.
  2. Decomposes m (e.g., in less than 200 Sandy Bridge cycles) into a multiscalar $(a_1, a_2, a_3, a_4) \in \mathbb{Z}^4$ such that each a_i is positive and at most 64 bits – see Section 4.
  3. Recodes the multiscalar (e.g., in less than 800 Sandy Bridge cycles) to ensure a simple and constant-time main loop – see Section 5.
  4. Computes a lookup table of 8 elements using exactly 7 complete additions, before executing the main loop using exactly 64 complete twisted Edwards double-and-add operations, and finally outputting $[m]P = [a_1]P + [a_2]\phi(P) + [a_3]\psi(P) + [a_4]\psi\phi(P)$ – see Section 5.

  This paper details each of the above steps explicitly, culminating in the full routine presented in Algorithm 2. Several prior works exploiting scalar decompositions have potential points of failure (cf. [33, §7], and §4.2), but crucially, and for the first time in the setting of four-dimensional decompositions, we accompany our routine with a robust proof of correctness – see Theorem 1.

– **Cryptographic versatility:** Four$\mathbb{Q}$ is intended to be used in the same way, i.e., using the same model, same coordinates and same explicit formulas, irrespective of the cryptographic protocol or nature of the intended scalar multiplication. Unlike implementations using ladders [4, 24, 9, 6], Four$\mathbb{Q}$ supports fast variable-base and fast fixed-base scalar multiplications, both of which use twisted Edwards coordinates; this serves as a basis for fast (ephemeral) Diffie-Hellman key exchange and fast Schnorr-like signatures. The presence of a single, complete addition law gives implementers the ability to easily wrap higher-level software and protocols around the Four$\mathbb{Q}$'s library exactly *as is*.

– **Public availability:** Prior works exploiting four-dimensional decompositions have either made code available that did not attempt to run in constant-time [9, 28], or not published code that did run in constant-time [41, 19]. Our library, which is publicly available [16], dispels any myths concerning the "complications" of making endomorphism-accelerated implementations run in constant-time [6, §1.2]. This software is compact, simple to read, and easy to audit. The library is largely written in portable C and includes two modular implementations of the arithmetic over $\mathbb{F}_{p^2}$: a portable implementation written in C and a high-performance implementation for x64 platforms written in C and optional x64 assembly. The library also permits the selection

[2] Here, and throughout, **I**, **M**, **S** and **A** are used to denote the respective costs of inversions, multiplications, squarings and additions in $\mathbb{F}_{p^2}$. We note that Frobenius operations amount to conjugations in $\mathbb{F}_p$, which are tallied as 0.5**A**.

(at build time) of whether or not endomorphisms are to be used for computing variable-based scalar multiplications. The code is accompanied by Magma scripts that can be used to verify the proofs of all claims and the claimed operation counts. Our aim is to make it easy for subsequent implementers to replicate the routine and, if desired, develop specialized code that is tailored to specific platforms (e.g., for further performance gains or with different memory constraints).

When the NIST curves [44] were standardized in 1999, many of the landmark discoveries in ECC (e.g., [23, 18, 22, 50]) were yet to be made. Four$\mathbb{Q}$ and its accompanying library represent the culmination of several of the best known ECC optimizations to date: it pulls together the extremely fast Mersenne prime, the fastest known large characteristic addition formulas [34], and the highest degree of scalar decompositions (there is currently no known way of achieving higher dimensional decompositions without exposing the ECDLP to attacks that are asymptotically much faster than Pollard rho). Subsequently, for generic scalar multiplications, Four$\mathbb{Q}$ performs around four to five times faster than the original NIST P-256 curve [27], between two and three times faster than curves that are currently under consideration as NIST alternatives, e.g., Curve25519 [4], and is also significantly faster than all of the other curves used to set previous speed records (see Section 6 for the comparisons). Interestingly, Four$\mathbb{Q}$ is still highly efficient if the endomorphisms ψ and ϕ are not used at all for computing generic scalar multiplications. In this case, Four$\mathbb{Q}$ performs about three times faster than the NIST P-256 curve and up to 1.6 times faster than Curve25519.

It is our belief that the demand for high-performance cryptography warrants the state-of-the-art in ECC to be part of the standardization discussion: this paper ultimately demonstrates the performance gains that are possible if such a curve was to be considered alongside the "conservative" choices.

2 The Curve: Four$\mathbb{Q}$

This section describes the proposed curve, where we adopt Smith's notation [48, 50] for the most part. We present the curve parameters in §2.1, shed some light on how the curve was found in §2.2, and discuss its cryptographic security in §2.3. Both §2.2 and §2.3 discuss that $\mathcal{E}$ is essentially one-of-a-kind, illustrating that there were no degrees of freedom in the choice of curve – see Remark 2.

2.1 A complete twisted Edwards curve

We will work over the quadratic extension field

$$\mathbb{F}_{p^2} := \mathbb{F}_p(i)\,, \quad \text{where} \quad p := 2^{127} - 1 \quad \text{and} \quad i^2 = -1.$$

We define $\mathcal{E}$ to be the twisted Edwards [5] curve

$$\mathcal{E}/\mathbb{F}_{p^2} : \; -x^2 + y^2 = 1 + dx^2y^2, \tag{1}$$

where

$$d := 125317048443780598345676279555970305165 \cdot i + 4205857648805777768770.$$

The set of $\mathbb{F}_{p^2}$-rational points satisfying the affine model for $\mathcal{E}$ forms a group: the neutral element is $\mathcal{O}_\mathcal{E} = (0, 1)$ and the inverse of a point (x, y) is $(-x, y)$. The fastest set of explicit formulas for

the addition law on $\mathcal{E}$ are due to Hisil, Wong, Carter and Dawson [34]: they use *extended twisted Edwards coordinates* to represent the affine point (x, y) on $\mathcal{E}$ by any projective tuple of the form $(X : Y : Z : T)$ for which $Z \neq 0$, $x = X/Z$, $y = Y/Z$ and $T = XY/Z$. Since d is not a square in $\mathbb{F}_{p^2}$, this set of formulas is also *complete* on $\mathcal{E}$ (see [5]), meaning that they will work without exception for all points in $\mathcal{E}(\mathbb{F}_{p^2})$.

The trace $t_{\mathcal{E}}$ of the p^2-power Frobenius endomorphism $\pi_{\mathcal{E}}$ of $\mathcal{E}$ is

$$t_{\mathcal{E}} = 136368062447564341573735631776713817674,$$

which reveals that

$$\#\mathcal{E}(\mathbb{F}_{p^2}) = p^2 + 1 - t_{\mathcal{E}} = 2^3 \cdot 7^2 \cdot N, \tag{2}$$

where N is a 246-bit prime. The cryptographic group we work with in this paper is $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

2.2 Where did this curve come from?

The curve $\mathcal{E}$ above comes from the family of $\mathbb{Q}$-curves of degree 2 – originally defined by Hasegawa [31] – that was recently used as one of the example families in Smith's general construction of $\mathbb{Q}$-curve endomorphisms [48, 50]. Certain examples of low-degree $\mathbb{Q}$-curves (including this family) were independently obtained through a different construction by Guillevic and Ionica [28], who also studied 4-dimensional decompositions arising from such curves possessing CM. In fact, $\mathcal{E}$ has a similar structure to the curve constructed in [28, Ex. 1], but is over the prime $p = 2^{127} - 1$.

For Δ a square-free integer, this family is defined over $\mathbb{Q}(\sqrt{\Delta})$ and is parameterized by $s \in \mathbb{Q}$ as

$$\tilde{\mathcal{E}}_{2,\Delta,s} \colon y^2 = x^3 - 6(5 - 3s\sqrt{\Delta})x + 8(7 - 9s\sqrt{\Delta}). \tag{3}$$

By definition [48, Def. 1], curves from this family are 2-isogenous (over $\mathbb{Q}(\Delta, \sqrt{-2})$) to their Galois conjugates ${}^{\sigma}\tilde{\mathcal{E}}_{2,\Delta,s}$. Smith reduces $\tilde{\mathcal{E}}_{2,\Delta,s}$ and ${}^{\sigma}\tilde{\mathcal{E}}_{2,\Delta,s}$ modulo primes p that are inert in $\mathbb{Q}(\sqrt{\Delta})$ to produce the curves $\mathcal{E}_{2,\Delta,s}$ and ${}^{\sigma}\mathcal{E}_{2,\Delta,s}$ defined over $\mathbb{F}_{p^2}$. He then composes the induced 2-isogeny from $\mathcal{E}_{2,\Delta,s}$ to ${}^{\sigma}\mathcal{E}_{2,\Delta,s}$ with the p-power Frobenius map from ${}^{\sigma}\mathcal{E}_{2,\Delta,s}$ back to $\mathcal{E}_{2,\Delta,s}$, which produces an efficiently computable degree $2p$ endomorphism ψ on $\mathcal{E}_{2,\Delta,s}$.

Recall that in this paper we fix $p = 2^{127} - 1$ for efficiency reasons. For this particular prime p and this family of $\mathbb{Q}$-curves, Smith's construction gives rise to precisely p non-isomorphic curves corresponding to each possible choice of $s \in \mathbb{F}_p$ [50, Prop. 1]. Varying s allows us to readily find curves belonging to this family with strong cryptographic group orders, each of which comes equipped with the endomorphism ψ that facilitates a two-dimensional scalar decomposition.

Seeking a four-dimensional (rather than two-dimensional) scalar decomposition on $\mathcal{E}_{2,\Delta,s}$ restricts us to a very small subset of possible s values. This is because we require the existence of another efficiently computable endomorphism on $\mathcal{E}_{2,\Delta,s}$, namely the low-degree GLV endomorphism ϕ on those instances of $\mathcal{E}_{2,\Delta,s}$ that possess CM over $\mathbb{Q}(\sqrt{\Delta})$. In [50, §9], Smith explains why there are only a handful of s values in any particular $\mathbb{Q}$-curve family that correspond to a curve with CM, before cataloging all such instances in the families of $\mathbb{Q}$-curves of degrees 2, 3, 5 and 7. In particular, up to isogeny and over any prime p, there are merely 13 values of s such that $\mathcal{E}_{2,\Delta,s}$ has CM over $\mathbb{Q}(\sqrt{\Delta})$. As is remarked in [50, §9], this scarcity of CM curves makes it highly unlikely that we will find a secure instance of a low-degree $\mathbb{Q}$-curve family with CM over any fixed prime p. This is the

reason why other authors chasing high speeds at the 128-bit security level have previously sacrificed the fast Mersenne prime $p = 2^{127} - 1$ in favor of a four-dimensional decomposition [41, 9]; one can always search through the small handfull of exceptional CM curves over many sub-optimal primes until a cryptographically secure instance is found. However, in the specific case of $p = 2^{127} - 1$, we actually get extremely lucky: our search through Smith's tables of exceptional $\mathbb{Q}$-curves with CM [50, Thm. 6] found one particular instance over $\mathbb{F}_{p^2}$ with a prime subgroup of 246-bits, namely $\mathcal{E}_{2,\Delta,s}$ with $s = \pm\frac{4}{9}$ and $\Delta = 5$. As is detailed in [50, §3], the specification of $\Delta = 5$ here does not dictate how we form the extension field $\mathbb{F}_{p^2}$ over $\mathbb{F}_p$; all quadratic extension fields of $\mathbb{F}_p$ are isomorphic, so we can take $s\sqrt{\Delta} = \pm\frac{4}{9}\sqrt{5}$ in (3) while still taking the reduction of $\tilde{\mathcal{E}}_{2,5,\pm\frac{4}{9}}$ modulo p to be $\mathcal{E}_{2,5,\pm\frac{4}{9}}/\mathbb{F}_{p^2}$ with $\mathbb{F}_{p^2} := \mathbb{F}_p(\sqrt{-1})$. To simplify notation, from hereon we fix $\tilde{\mathcal{E}}_{\mathrm{W}} := \tilde{\mathcal{E}}_{2,5,\pm\frac{4}{9}}$ and define $\mathcal{E}_{\mathrm{W}}$ as the reduction of $\tilde{\mathcal{E}}_{\mathrm{W}}$ modulo p, given as

$$\mathcal{E}_{\mathrm{W}}/\mathbb{F}_{p^2} : \; y^2 = x^3 - (30 - 8\sqrt{5})x + (56 - 32\sqrt{5}), \tag{4}$$

where the choice of the root $\sqrt{5}$ in $\mathbb{F}_{p^2}$ will be fixed in Section 3. We note that the short Weierstrass curve $\mathcal{E}_{\mathrm{W}}$ is not isomorphic to our twisted Edwards curve $\mathcal{E}$, but rather to a twisted Edwards curve $\hat{\mathcal{E}}$ that is $\mathbb{F}_{p^2}$-isogenous to $\mathcal{E}$. The reason we work with $\mathcal{E}$ rather than $\hat{\mathcal{E}}$ is because the curve constant d on $\mathcal{E}$ is non-square in $\mathbb{F}_{p^2}$, which is not the case for the curve constant $\hat{d}$ on $\hat{\mathcal{E}}$; as we mentioned above, d being a non-square ensures that the fastest known addition formulas are also complete on $\mathcal{E}$. The isogenies between $\mathcal{E}$ and $\hat{\mathcal{E}}$ are made explicit as follows.

Proposition 1. *Let $\hat{\mathcal{E}}/K$ and $\mathcal{E}/K$ be the twisted Edwards curves defined by $\hat{\mathcal{E}}/K\colon\ -x^2 + y^2 = 1 + \hat{d}x^2y^2$ and $\mathcal{E}/K\colon\ -x^2 + y^2 = 1 + dx^2y^2$. If $d = -(1 + 1/\hat{d})$, then the map*

$$\tau : \; \mathcal{E} \to \hat{\mathcal{E}}, \quad (x, y) \mapsto \left(\frac{2xy}{(x^2 + y^2)\sqrt{\hat{d}}}, \frac{x^2 - y^2 + 2}{y^2 - x^2} \right)$$

is a 4-isogeny, the dual of which is

$$\hat{\tau} : \; \hat{\mathcal{E}} \to \mathcal{E}, \quad (x, y) \mapsto \left(\frac{2xy\sqrt{\hat{d}}}{x^2 - y^2 + 2}, \frac{y^2 - x^2}{y^2 + x^2} \right).$$

Proof. We derive τ and $\hat{\tau}$ using the 2-isogenies ψ and $\hat{\psi}$ from [1, Theorem 3.2], together with the isomorphisms $\sigma_1\sigma_2$ and $\sigma_2\sigma_1$ in [1, Equations 15-16]. The isogeny τ is the composition of $\psi\colon \mathcal{E} \to L_{-d}$, of $\sigma_2\sigma_1\colon L_{-d} \to L_{1/(1+d)}$, and of $\hat{\psi}\colon L_{1/(1+d)} \to \hat{\mathcal{E}}$, i.e., $\tau = \hat{\psi}\sigma_2\sigma_1\psi$ in $\mathrm{Hom}(\mathcal{E}, \hat{\mathcal{E}})$. The isogeny $\hat{\tau}$ is the composition of $\psi\colon \hat{\mathcal{E}} \to L_{-\hat{d}}$, of $\sigma_1\sigma_2\colon L_{-\hat{d}} \to L_{1+1/\hat{d}}$, and of $\hat{\psi}\colon L_{1+1/\hat{d}} \to \mathcal{E}$, i.e., $\hat{\tau} = \hat{\psi}\sigma_1\sigma_2\psi$ in $\mathrm{Hom}(\hat{\mathcal{E}}, \mathcal{E})$. It follows that both τ and $\hat{\tau}$ are 4-isogenies. It is easily verified that $\tau\hat{\tau}$ corresponds to multiplication by 4 in $\mathrm{End}(\mathcal{E})$, so $\hat{\tau}$ is indeed the dual of τ [21, Theorem 9.6.21]. □

We note at once that if $\hat{d}$ is a square in K, then τ and $\hat{\tau}$ are defined over K. Fortunately, while the twisted Edwards curve $\hat{\mathcal{E}}$ corresponding to $\mathcal{E}_{\mathrm{W}}/\mathbb{F}_{p^2}$ has a square constant $\hat{d}$, our chosen isogenous curve $\mathcal{E}$ has the non-square constant $d = -(1+1/\hat{d})$. Our implementation will work solely in twisted Edwards coordinates on $\mathcal{E}$, but we will pass back and forth through $\mathcal{E}_{\mathrm{W}}$ (via $\hat{\mathcal{E}}$) when deriving explicit formulas for the endomorphisms ϕ and ψ in Section 3. We note that Hamburg used 4-isogenies (also derived from [1]) to a similar effect in [30].

Remark 1. We tried several other possibilities to achieve a non-square d before resorting to an isogeny (which ends up adding around 10 multiplications to each endomorphism – see Table 1). This included looking at the $a = -1$ twisted Edwards model corresponding to different choices of roots and corresponding to the Galois conjugate $\mathcal{E}_{\mathrm{W}}^{\sigma}$ of $\mathcal{E}_{\mathrm{W}}$, which is 2-isogenous to $\mathcal{E}_{\mathrm{W}}$ by definition; none of these options give rise to a non-square d. We note that using these isogenies once-off in each endomorphism computation is still much faster than using a slower, complete addition on $\hat{\mathcal{E}}$.

2.3 The cryptographic security of Four$\mathbb{Q}$

Pollard's rho algorithm [46] is the best known way to solve the ECDLP in $\mathcal{E}(\mathbb{F}_{p^2})[N]$. An optimized version of this attack which uses the negation map [54] requires around $\sqrt{\pi N/4} \sim 2^{122.5}$ group operations on average. We note that, unlike some of the typical GLV [23] or GLS [22] endomorphisms that can be used to speed up Pollard's rho algorithm [17], both ψ and ϕ on $\mathcal{E}$ do not facilitate any known advantage; neither of these endomorphisms have a small orbit and they are both more expensive to compute than an amortized addition. Thus, the known complexity of the ECDLP on $\mathcal{E}$ is comparable to various other curves used in the speed-record literature; optimized implementations of Pollard rho against any of the fastest curves in [4, 22, 41, 9, 19, 15, 45] would require between $2^{124.8}$ and $2^{125.8}$ group operations on average. Ideally, we would prefer not to have the factor 7^2 dividing $\#\mathcal{E}(\mathbb{F}_{p^2})$, but the resulting ($\sim 2.8$ bit) security degradation is a small price to pay for having the fastest field at the 128-bit level in conjunction with a four-dimensional scalar decomposition. As discussed in Remark 2 below, it was a long shot to try and find such a cryptographically secure $\mathbb{Q}$-curve with CM over $\mathbb{F}_{p^2}$ in Smith's tables in the first place, let alone one that also had the necessary torsion to support a twisted Edwards model.

Since $\mathcal{E}(\mathbb{F}_{p^2})$ has rational 2-torsion, it is easy to write down the corresponding abelian surface over $\mathbb{F}_p$ whose Jacobian is isogenous to the Weil restriction of $\mathcal{E}$ – see [47, Lemma 2.1 and Lemma 3.1]. But since the best known algorithm to solve the discrete logarithm problem on such abelian surfaces is again Pollard's rho algorithm, the Weil descent philosophy (cf. [25]) does not pose a threat here. Furthermore, the embedding degree of $\mathcal{E}$ with respect to N is $(N-1)/2$, making it infeasible to reduce the ECDLP into a finite field [43, 20].

We note that the largest prime factor dividing the group order of $\mathcal{E}$'s quadratic twist is 158 bits, but *twist-security* [4] is not an issue in this work: firstly, our software always validates input points (such validation is essentially free), and secondly, x-coordinate-only arithmetic (which is where twist-security makes sense) on $\mathcal{E}$ is not competitive with a four-dimensional decomposition that uses both coordinates.

In contrast to most currently standardized curves, the proposed curve is both defined over a quadratic extension field and has a small discriminant; one notable exception is `secp256k1` in the SEC standard [13], which is used in the Bitcoin protocol and also has small discriminant. However, it is important to note that there is no better-than-generic attack known to date that can exploit either of these two properties on $\mathcal{E}$. In fact, with respect to ECDLP difficulty, Koblitz, Koblitz and Menezes [36, §11] point out that slower, large discriminant curves, like NIST P-256 and Curve25519, may turn out to be less conservative than specially chosen curves with small discriminant.

Remark 2. It should be noted that there were no degrees of freedom in choosing the curve $\mathcal{E}$. Demanding the field $\mathbb{F}_{p^2}$ (with $p = 2^{127} - 1$) alongside a four-dimensional decomposition reveals that, of all the exceptional $\mathbb{Q}$-curves with CM tabulated in [50, Thm. 6], the 246-bit prime subgroup makes $\mathcal{E}$ the only known curve that comes close to the target 128-bit security level. Over this field,

the second strongest curve found in all of Smith's tables had a 215-bit prime subgroup. To find another strong curve supporting four-dimensional decompositions over this field would, to our knowledge, require increasing the degree of the $\mathbb{Q}$-curve family beyond $d = 7$, for which explicit constructions are currently limited [31, 50]. The discussion in §2.2 also shows why four-dimensional decompositions and our specially chosen underlying field have not been previously partnered at the 128-bit security level. Prior to [48], partnering the Galbraith-Lin-Scott (GLS) endomorphism [22] with a GLV endomorphism was the only known way to achieve such decompositions on elliptic curves over large characteristic fields, and a secure curve facilitating GLV+GLS had only been found over suboptimal primes [22]. One consequence of the constructions in [48] and [28] is that they increased the search space of curves that facilitate four-dimensional decompositions (over any fixed field).

3 The Endomorphisms ψ and ϕ

In this section we derive explicit formulas for the two endomorphisms on $\mathcal{E}$. In what follows we use $c_{i,j,k,l}$ to denote the constant $i + j\sqrt{2} + k\sqrt{5} + l\sqrt{2}\sqrt{5}$ in $\mathbb{F}_{p^2}$, which is fixed by setting

$$\sqrt{2} := 2^{64} \quad \text{and} \quad \sqrt{5} := 873928070873369763180053688207072444 64 \cdot i.$$

For both ψ and ϕ, we start by deriving the explicit formulas on the short Weierstrass model $\mathcal{E}_{\mathrm{W}}$. As discussed in the previous section, we will pass back and forth between $\mathcal{E}$ and $\mathcal{E}_{\mathrm{W}}$ via the twisted Edwards curve $\hat{\mathcal{E}}$ that is 4-isogenous to $\mathcal{E}$ over $\mathbb{F}_{p^2}$. The maps between $\mathcal{E}$ and $\hat{\mathcal{E}}$ are given in Proposition 1, and we take the maps $\delta\colon \mathcal{E}_{\mathrm{W}} \to \hat{\mathcal{E}}$ and $\delta^{-1}\colon \hat{\mathcal{E}} \to \mathcal{E}_{\mathrm{W}}$ from [50, §5] (tailored to our $\hat{\mathcal{E}}$) as

$$\delta : (x, y) \mapsto \left(\frac{\gamma(x-4)}{y}, \frac{x-4-c_{0,2,0,1}}{x-4+c_{0,2,0,1}} \right) \text{ and } \delta^{-1} : (x, y) \mapsto \left(\frac{c_{0,2,0,1}(y+1)}{1-y} + 4, \frac{c_{0,2,0,1}(y+1)\gamma}{x(1-y)} \right),$$

where $\gamma^2 = c_{-12,-4,0,-2}$. The choice of the square root $\gamma \in \mathbb{F}_{p^2}$ becomes irrelevant in the compositions below.

3.1 Explicit formulas for ψ

There is almost no work to be done in deriving ψ on $\mathcal{E}$, since this is Smith's $\mathbb{Q}$-curve endomorphism corresponding to the degree-2 family to which $\mathcal{E}_{\mathrm{W}}$ belongs. We start with $\psi_{\mathrm{W}}\colon \mathcal{E}_{\mathrm{W}} \to \mathcal{E}_{\mathrm{W}}$, taken from [50, §5], as

$$\psi_{\mathrm{W}}\colon (x, y) \mapsto \left(\left(-\frac{x}{2} - \frac{c_{9,0,4,0}}{x-4} \right)^p, \left(-\frac{y}{i\sqrt{2}} \left(-\frac{1}{2} + \frac{c_{9,0,4,0}}{(x-4)^2} \right) \right)^p \right).$$

With ψ_{W} as above, we define $\psi\colon \mathcal{E} \to \mathcal{E}$ as the composition $\psi = \hat{\tau}\delta\psi_{\mathrm{W}}\delta^{-1}\tau$. In optimizing the explicit formulas for this composition, there is practically nothing to be gained by simplifying the full composition in the function field $\mathbb{F}_{p^2}(\mathcal{E})$. However, it is advantageous to optimize explicit formulas for the inner composition $(\delta\psi_{\mathrm{W}}\delta^{-1})$ in the function field $\mathbb{F}_{p^2}(\hat{\mathcal{E}})$. In fact, for both ψ and ϕ, optimized explicit formulas for this inner composition are faster than the respective endomorphisms ψ_{W} and ϕ_{W}, and are therefore much faster than computing the respective compositions individually.

Simplifying the composition $\delta\psi_{\mathrm{W}}\delta^{-1}$ in the function field $\mathbb{F}_{p^2}(\hat{\mathcal{E}})$ yields

$$(\delta\psi_{\mathrm{W}}\delta^{-1})\colon \hat{\mathcal{E}} \to \hat{\mathcal{E}}, \quad (x,y) \mapsto \left(\frac{2ix^p \cdot c_{-2,3,-1,0}}{y^p \cdot ((x^p)^2 \cdot c_{-140,99,0,0} + c_{-76,57,-36,24})}, \frac{c_{-9,-6,4,3} - (x^p)^2}{c_{-9,-6,4,3} + (x^p)^2} \right).$$

Note that each of the p-power Frobenius operations above amount to one $\mathbb{F}_p$ negation. As mentioned above, we compute the endomorphism $\psi = \hat{\tau}(\delta\psi_{\mathrm{W}}\delta^{-1})\tau$ on $\mathcal{E}$ by computing τ and $\hat{\tau}$ separately; see Section 3.4 for the operation counts.

3.2 Deriving explicit formulas for ϕ

We now derive the second endomorphism ϕ that arises from $\mathcal{E}$ admitting CM by the order of discriminant $D = -40$. We start by pointing out that there is actually multiple routes that could be taken in defining and deriving ϕ. The route we took in a preliminary version of this article was to use Stark's algorithm [51] (see also [12]). On input of the two curve constants in (4) and an integral basis $\{1, \beta\}$ for the ring of integers in $\mathbb{Q}(\sqrt{D})$, Stark's algorithm outputs two polynomials $f(x)$ and $g(x)$ defining the endomorphism $\phi_{\mathrm{W}}\colon \tilde{\mathcal{E}}_{\mathrm{W}} \to \tilde{\mathcal{E}}_{\mathrm{W}}$ as

$$\phi\colon (x,y) \mapsto \left(\frac{f(x)}{g(x)}, cy \left(\frac{f(x)}{g(x)} \right)' \right),$$

for a constant c in $\mathbb{Q}(\sqrt{D})$. The degrees of $f(x)$ and $g(x)$ are $N(\beta)$ and $N(\beta)-1$, where $N(\cdot)$ is the norm function from $\mathbb{Q}(\sqrt{D})$ down to $\mathbb{Q}$. In our case, since $\beta = \frac{1}{2}\sqrt{-40} = \sqrt{-10}$ defines an interal basis, the degrees of f and g were 10 and 9 respectively, and the resulting endomorphism ϕ_{W} on $\mathcal{E}_{\mathrm{W}}(\mathbb{F}_{p^2})[N]$ corresponded to scalar multiplication by $\sqrt{-10} \pmod N$.

The second possibility, which we use in this paper, produces an endomorphism of lower degree. This option was revealed to us in correspondence with Ben Smith, who pointed out that $\mathbb{Q}$-curves with CM can also be produced as the intersection of families of $\mathbb{Q}$-curves, and that our curve $\mathcal{E}$ is not only a degree-2 $\mathbb{Q}$-curve, but is also a degree-5 $\mathbb{Q}$-curve. Thus, the second endomorphism ϕ can be derived by first following the treatment in [50, §7] (see also [28, §3.3]) to derive ϕ_{W} as a 5-isogeny on $\mathcal{E}_{\mathrm{W}}$, which we do below.

Working in $\mathbb{Q}(\sqrt{5})[x]$, the 5-division polynomial (cf. [21, Def. 9.8.4]) of $\tilde{\mathcal{E}}_{\mathrm{W}}$ factors as $f(x)g(x)$, where $f(x) = x^2 + 4\sqrt{5}\cdot x + (18 - 4/5\sqrt{5})$ and $g(x)$ (which is of degree 10) are irreducible. The polynomial $f(x)$ defines the kernel of a 5-isogeny $\phi^{\sigma}_{\mathrm{W}}\colon \tilde{\mathcal{E}}_{\mathrm{W}} \to \tilde{\mathcal{E}}^{\sigma}_{\mathrm{W}}$. We use this kernel to compute $\phi^{\sigma}_{\mathrm{W}}$ via Vélu's formulae [53] (see also [37, §2.4]), reduce modulo p, and then compose with Frobenius $\pi_p\colon \mathcal{E}^{\sigma}_{\mathrm{W}} \to \mathcal{E}_{\mathrm{W}}$ to give $\phi_{\mathrm{W}}\colon \mathcal{E}_{\mathrm{W}} \to \mathcal{E}_{\mathrm{W}}, (x,y) \mapsto (x_{\phi_W}, y_{\phi_W})$, where

$$x_{\phi_W} = \left(\frac{x^5 + 8\sqrt{5}x^4 + (40\sqrt{5}+260)x^3 + (720\sqrt{5}+640)x^2 + (656\sqrt{5}+4340)x + (1920\sqrt{5}+960)}{5\left((x^2 + 4\sqrt{5}x - 1/5(4\sqrt{5}-90)\right)^2} \right)^p,$$

$$y_{\phi_W} = \left(\frac{-y\left(x^2 + (4\sqrt{5}-8)x - 12\sqrt{5} + 26\right)\left(x^4 + (8\sqrt{5}+8)x^3 + 28x^2 - (48\sqrt{5}+112)x - 32\sqrt{5} - 124\right)}{\left(\sqrt{5}(x^2 + 4\sqrt{5}x - 1/5(4\sqrt{5}-90))\right)^3} \right)^p,$$

As was the case with ψ in §3.1, it is advantageous to optimize formulas in $\mathbb{F}_{p^2}(\hat{\mathcal{E}})$ for the composition $(\delta\psi_{\mathrm{W}}\delta^{-1})$, which gives $(\delta\psi_{\mathrm{W}}\delta^{-1})\colon \hat{\mathcal{E}} \to \hat{\mathcal{E}}, (x,y) \mapsto (x_\phi, y_\phi)$, where

$$x_\phi = \left(\frac{c_{9,-6,4,-3}\cdot x \cdot (y^2 - c_{7,5,3,2}\cdot y + c_{21,15,10,7})\cdot(y^2 + c_{7,5,3,2}\cdot y + c_{21,15,10,7})}{(y^2 + c_{3,2,1,1}\cdot y + c_{3,3,2,1})\cdot(y^2 - c_{3,2,1,1}\cdot y + c_{3,3,2,1})} \right)^p,$$

$$y_\phi = \left(\frac{c_{15,10,6,4}\cdot(5y^4 + c_{120,90,60,40}\cdot y^2 + c_{175,120,74,54})}{5y\cdot(y^4 + c_{240,170,108,76}\cdot y^2 + c_{3055,2160,1366,966})} \right)^p.$$

Again, we use this to compute the full endomorphism $\psi = \hat{\tau}(\delta\psi_{\mathrm{W}}\delta^{-1})\tau$ on $\mathcal{E}$ by computing τ and $\hat{\tau}$ separately; see §3.4 for the operation counts.

3.3 Eigenvalues

The eigenvalues of the two endomorphisms ψ and ϕ play a key role in developing scalar decompositions. In this subsection we write them in terms of the curve parameters. From [50, Thm. 2], and given that we used a 4-isogeny τ and its dual to pass back and forth to $\mathcal{E}_{\mathrm{W}}$, the eigenvalues of ψ on $\mathcal{E}(\mathbb{F}_{p^2})[N]$ are

$$\lambda_\psi \;:=\; 4 \cdot \frac{p+1}{r} \pmod{N} \tag{5}$$

and $\lambda'_\psi := -\lambda_\psi \pmod{N}$, where r is an integer satisfying

$$2r^2 = 2p + t_{\mathcal{E}}. \tag{6}$$

To derive the eigenvalues for ϕ, we make use of the CM equation for $\mathcal{E}$, which (since $\mathcal{E}$ has CM by the order of discriminant $D = -40$) is

$$40V^2 = 4p^2 - t_{\mathcal{E}}^2, \tag{7}$$

for an integer V. We fix r and V to be the positive integers satisfying (6) and (7), namely

$$V := 4929397548930634471175140312327029681 4; \qquad r := 15437785290780909242.$$

Proposition 2. *The eigenvalues of ϕ on $\mathcal{E}(\mathbb{F}_{p^2})[N]$ are*

$$\lambda_\phi \;:=\; 4 \cdot \frac{(p-1)r^3}{(p+1)^2 V} \pmod{N} \tag{8}$$

and $\lambda'_\phi := -\lambda_\phi \pmod{N}$.

Proof. The endomorphism $\phi_{\mathrm{W}} \in \mathrm{End}(\mathcal{E}_{\mathrm{W}})$ has minimal polynomial $P_{\phi_{\mathrm{W}}}(T) = T^2 + 5$, so we first show that $\big((p-1)r^3/((p+1)^2V)\big)^2 \equiv -5 \bmod N$. To do this we rewrite (7) as $-5 = (t_{\mathcal{E}}^2 - 4p^2)/(8V^2) = (t_{\mathcal{E}} - 2p)r^2/(4V^2)$, which follows from (6). Since $t_{\mathcal{E}} \equiv p^2 + 1 \bmod N$, we have $-5 \equiv (p-1)^2r^2/(4V^2) \bmod N$, and using [50, Thm. 2] to replace the 4 on the denominator by $((p+1)/r)^4$ gives that the eigenvalue of ϕ_{W} is $(p-1)r^3/((p+1)^2V)$. Finally, the factor 4 in (8) comes from $\phi = \hat{\tau}\delta\phi_{\mathrm{W}}\delta^{-1}\tau$ where δ and δ^{-1} are isomorphisms and τ is a 4-isogeny with dual $\hat{\tau}$. □

3.4 Section summary

Table 1 summarizes the isogenies derived in this section, together with their exact operation counts. The reason that multiples of 0.5 appear in the additions column is that we count Frobenius operations (which amount to a negation in $\mathbb{F}_p$) as half an addition in $\mathbb{F}_{p^2}$. Four-dimensional scalar decompositions on $\mathcal{E}$ require the computation of $\phi(P)$, $\psi(P)$ and the composition $\psi(\phi(P))$; the ordering here is important since ψ is much faster than ϕ, meaning we actually compute ϕ once and ψ twice. We note that all sets of explicit formulas were derived assuming the inputs were projective points $(X : Y : Z)$ corresponding to a point $(X/Z, Y/Z)$ in the domain of the isogeny. Similarly, all

explicit formulas output the point $(X' : Y' : Z')$ corresponding to $(X'/Z', Y'/Z')$ in the codomain, and in the special cases when the codomain is $\mathcal{E}$ (i.e., for $\hat{\tau}$, ϕ, ψ and $-\psi\phi$), we also output the coordinate T' (or a related variant) corresponding to $T' = X'Y'/Z'$, which facilitates faster subsequent group law formulas on $\mathcal{E}$ – see §5.2.

Table 1 reveals that, on input of a projective point in $\mathcal{E}(\mathbb{F}_{p^2})[N]$, the total cost of the three maps ϕ, ψ and $\psi\phi$ is $68\mathbf{M} + 27\mathbf{S} + 49.5\mathbf{A}$. Computing the maps using these explicit formulas requires the storage of 16 constants in $\mathbb{F}_{p^2}$, and at any stage of the endomorphism computations, requires the storage of at most 7 temporary variables.

Table 1. Summary of isogenies used in the derivation of the three endomorphisms ϕ, ψ and $\phi\psi$ on $\mathcal{E}$, together with the cost of their explicit formulas. Here **M**, **S** and **A** respectively denote the costs of one multiplication, one squaring and one addition in $\mathbb{F}_{p^2}$.

isogeny	domain & codomain	degree	no. fixed constants	no. temp variables	cost		
					M	**S**	**A**
τ	$\mathcal{E} \to \hat{\mathcal{E}}$	4	1	2	5	3	5
$\hat{\tau}$	$\hat{\mathcal{E}} \to \mathcal{E}$	4	1	2	5	3	4
$(\delta\phi_{\mathrm{W}}\delta^{-1})$	$\hat{\mathcal{E}} \to \hat{\mathcal{E}}$	$5p$	10	7	20	5	11.5
$(\delta\psi_{\mathrm{W}}\delta^{-1})$	$\hat{\mathcal{E}} \to \hat{\mathcal{E}}$	$2p$	4	2	9	2	5.5
ϕ	$\mathcal{E} \to \mathcal{E}$	$80p$	11	7	30	11	20.5
ψ		$32p$	5	2	19	8	14.5
$\psi\phi$		$2560p$	-	7	19	8	14.5
	total cost $(\phi, \psi, \psi\phi)$		**16**	**7**	**68**	**27**	**49.5**

4 Optimal Scalar Decompositions

Let λ_ψ and λ_ϕ be as fixed in (5) and (8). In this section we show how to compute, for any integer scalar $m \in \mathbb{Z}$, a corresponding 4-dimensional multiscalar $(a_1, a_2, a_3, a_4) \in \mathbb{Z}^4$ such that $m \equiv a_1 + a_2\lambda_\phi + a_3\lambda_\psi + a_4\lambda_\phi\lambda_\psi \pmod N$, such that $0 \le a_i < 2^{64} - 1$ for $i = 1, 2, 3, 4$, and such that a_1 is odd (which facilitates faster scalar recodings and multiplications – see Section 5). An excellent reference for general scalar decompositions in the context of elliptic curve cryptography is Smith's article [49], where it is shown how to write down short lattice bases for scalar decompositions directly from the curve parameters. Here, we show how to further reduce such short bases into bases that are, in the context of multiscalar multiplications, *optimal*.

4.1 Babai rounding and optimal bases

Following [49, §1], we define the *lattice of zero decompositions* as

$$\mathcal{L} := \langle\, (z_1, z_2, z_3, z_4) \in \mathbb{Z}^4 \mid z_1 + z_2\lambda_\phi + z_3\lambda_\psi + z_4\lambda_\phi\lambda_\psi \equiv 0 \pmod N \rangle,$$

so that the set of decompositions for $m \in \mathbb{Z}/N\mathbb{Z}$ is the lattice coset $(m, 0, 0, 0) + \mathcal{L}$. For a given basis $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4)$ of $\mathcal{L}$, and on input of any $m \in \mathbb{Z}$, the *Babai rounding* technique [2] computes $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Q}^4$ as the unique solution to $(m, 0, 0, 0) = \sum_{i=1}^4 \alpha_i \mathbf{b}_i$, and subsequently computes the multiscalar

$$(a_1, a_2, a_3, a_4) = (m, 0, 0, 0) - \sum_{i=1}^{4} \lfloor \alpha_i \rceil \cdot \mathbf{b}_i.$$

It follows that $(a_1, a_2, a_3, a_4) - (m, 0, 0, 0) \in \mathcal{L}$, so $m \equiv a_1 + a_2\lambda_\phi + a_3\lambda_\psi + a_4\lambda_\phi\lambda_\psi \pmod N$. Since $-1/2 \leq x - \lfloor x \rceil \leq 1/2$, this technique finds the unique element in $(m, 0, 0, 0) + \mathcal{L}$ that lies inside the parallelepiped[3] defined by $\mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} \,|\, \mathbf{x} \in [-1/2, 1/2)^4\}$, i.e., Babai rounding maps $\mathbb{Z}$ onto $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$. For a given m, the length of the corresponding multiscalar multiplication is then determined by the infinity norm, $||\cdot||_\infty$, of the corresponding element (a_1, a_2, a_3, a_4) in $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$.

Since our scalar multiplications must run in time independent of m, the speed of the multiscalar exponentiations will depend on the worst case, i.e., on the maximal infinity norm taken across all elements in $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$. Or, equivalently, the speed of routine will depend on the *width* of the smallest 4-cube whose convex body contains $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$. This width depends only on the choice of $\mathbf{B}$, so this gives us a natural way of finding a basis that is optimal for our purposes. We make this concrete in the following definition, which is stated for an arbitrary lattice of dimension n. Definition 1 simplifies the situation by looking for the smallest n-cube containing $\mathcal{P}(\mathbf{B})$, rather than $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^n$, but our candidate bases will always be orthogonal enough such that the conditions are equivalent in practice.

Definition 1 (Babai-optimal bases). *We say that a basis $\mathbf{B}$ of a lattice $\mathcal{L} \in \mathbb{R}^n$ is* Babai-optimal *if the width of the smallest n-cube containing the parallelepiped $\mathcal{P}(\mathbf{B})$ is minimal across all bases for $\mathcal{L}$.*

We note immediately that taking the n *successive minima* under $||\cdot||_\ell$, for any $\ell \in \{1, 2, \ldots, \infty\}$, will not be Babai-optimal in general. Indeed, for our specific lattice $\mathcal{L}$, neither the $||\cdot||_2$-reduced basis (output from LLL [38]) or the $||\cdot||_\infty$-reduced basis (in the sense of Lovász and Scarf [42]) are Babai-optimal.

For very low dimensions, such as those used in ECC scalar decompositions, we can find a Babai-optimal basis via straightforward *enumeration* as follows. Starting with any reasonably small basis $\mathbf{B}' = (\mathbf{b}'_1, \ldots, \mathbf{b}'_n)$, like the ones in [49], we compute the width, $w(\mathbf{B}')$, of the smallest n-cube whose convex body contains $\mathcal{P}(\mathbf{B}')$; by the definition of $\mathcal{P}$, this is $w(\mathbf{B}') = \max_{1 \leq j \leq n} \{\sum_{i=1}^{n} |\mathbf{b}'_i[j]|\}$. We then enumerate the set S of all vectors $\mathbf{v} \in \mathcal{L}$ such that $||\mathbf{v}||_\infty \leq w(\mathbf{B}')$; any vector not in S cannot be in a basis whose width is smaller than $\mathbf{B}'$. We can then test all possible bases $\mathbf{B}$, that are formed as combinations of n linearly independent vectors in S, and choose one corresponding to the minimal value of $w(\mathbf{B})$.

Proposition 3. *A Babai optimal basis for our zero decomposition lattice $\mathcal{L}$ is given by $\mathbf{B} := (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4)$, where*

$$
\begin{aligned}
224 \cdot \mathbf{b}_1 &:= (16(-60\alpha + 13r - 10),\ 4(-10\alpha - 3r + 12),\ 4(-15\alpha + 5r - 13),\ -13\alpha - 6r + 3), \\
8 \cdot \mathbf{b}_2 &:= (32(5\alpha - r),\ -8,\ 8,\ 2\alpha + r), \\
224 \cdot \mathbf{b}_3 &:= (16(80\alpha - 15r + 18),\ 4(18\alpha - 3r - 16),\ 4(-15\alpha - 9r + 15),\ 15\alpha + 8r + 3\alpha), \\
448 \cdot \mathbf{b}_4 &:= (16(-360\alpha + 77r + 42),\ 4(42\alpha + 17r + 72),\ 4(85\alpha - 21r - 77),\ (-77\alpha - 36r - 17)),
\end{aligned}
$$

for V and r as fixed in Section 3, and (since $V \equiv 0 \bmod r$) where $\alpha := V/r \in \mathbb{Z}$.

Proof. Straightforward but lengthy calculations using (2), (5), (6), (7) and (8) reveal that $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ are all in $\mathcal{L}$. Another direct calculation reveals that the determinant of $\langle \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4 \rangle$ is $\left(100V^4 + 20r^2(r^2+2)V^2 + r^4(r^2-2)^2\right)/(1568r^4)$, which simplifies to N under (2), (6) and (7),

[3] This is a translate (by $-\frac{1}{2}(\sum_{i=1}^{4} \mathbf{b}_i)$) of the *fundamental parallelepiped*, which is defined using $\mathbf{x} \in [0, 1)^4$.

so $\mathbf{B}$ is a basis for $\mathcal{L}$. To show that $\mathbf{B}$ is Babai-optimal, we set $\mathbf{B}' = \mathbf{B}$ and compute $w(\mathbf{B}') = \max_{1\le j\le 4}\left\{\sum_{i=1}^{4}|\mathbf{b}'_i[j]|\right\}$, which (at $j=1$) is $w(\mathbf{B}') = (245\alpha + 120r + 17)/448$. Enumeration under $||\cdot||_\infty$ yields exactly 128 vectors (up to sign) in $S = \{\mathbf{v} \in \mathcal{L} \mid ||\mathbf{v}||_\infty \le w(\mathbf{B}')\}$; none of the rank 4 bases formed from S have a width smaller than $\mathbf{B}$. □

The size of the set S in the above proof depends on the quality of the initial basis $\mathbf{B}'$. For the proof, it suffices to start with the Babai-optimal basis $\mathbf{B}$ itself, but in practice we will usually start with a basis that is not optimal according to Definition 1. In our case we computed the basis in Proposition 3 by first writing down a short basis using Smith's methodology [49]. We input this into the LLL algorithm [38] to obtain an *LLL-reduced* basis $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_4, \mathbf{b}_3)$; these are also the four successive minima under $||\cdot||_2$. We then input this basis into the algorithm of Lovász and Scarf [42]; this forced the requisite changes to output a basis consisting of the four successive minima under $||\cdot||_\infty$, namely $(\mathbf{b}_1, \mathbf{b}_1 + \mathbf{b}_4, \mathbf{b}_2, \mathbf{b}_1 + \mathbf{b}_3)$. Using this as our input $\mathbf{B}'$ into the enumeration gave a set S of size 282, which we exhaustively searched to find $\mathbf{B}$.

Remark 3. In practice, when the dimension of the lattice $\mathcal{L}$ is very small, a more bovine approach to enumerating under $||\cdot||_\infty$ is to instead enumerate under $||\cdot||_2$ by using the simple fact that, for any $\mathbf{v} \in \mathcal{L}$, $||\mathbf{v}||_2 \le \sqrt{n}\cdot||\mathbf{v}||_\infty$ (to see this, maximize the 2-norm of $\mathbf{v}$ with respect to a fixed infinity norm). For example, Magma's [11] `ShortVectors`$(\mathcal{L}, n\cdot\gamma^2)$ command will enumerate the set of all vectors of 2-norm up $\sqrt{n}\cdot|\gamma|$; any vectors with a larger 2-norm than this will necessarily have an larger infinity norm than $|\gamma|$.

We now describe a simple scalar decomposition that uses Babai rounding on the optimal basis above. Note that, since V and r are fixed, the four $\hat{\alpha}_i$ values below are fixed integer constants.

Proposition 4. *For a given integer m, and the basis $\mathbf{B} := (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4)$ in Proposition 3, let (a_1, a_2, a_3, a_4) be the multiscalar defined as*

$$(a_1, a_2, a_3, a_4) = (m, 0, 0, 0) - \sum_{i=1}^{4} \lfloor \alpha_i \rceil \cdot \mathbf{b}_i,$$

where $\alpha_i = \hat{\alpha}_i \cdot m/N$, with

$$\begin{aligned}
6272r^3\cdot\hat{\alpha}_1 &= 540V^3 + 10r(27r-4)V^2 + 6r^2(9r^2-2r+18)V + r^3(27r+4)(r^2-2),\\
25088r^3\cdot\hat{\alpha}_2 &= 1020V^3 + 10r(47r-8)V^2 + 2r^2(51r^2+26r+102)V + r^3(47r+8)(r^2-2),\\
25088r^3\cdot\hat{\alpha}_3 &= 220V^3 + 10r(11r+16)V^2 + 2r^2(11r^2-46r+22)V + r^3(11r-16)(r^2-2),\\
1792r^3\cdot\hat{\alpha}_4 &= 60V^3 + 30r^2V^2 + 2r^2(3r^2+2r+6)V + 3r^4(r^2-2).
\end{aligned}$$

Then $m \equiv a_1 + a_2\lambda_\phi + a_3\lambda_\psi + a_4\lambda_\psi\phi \pmod N$ and $|a_1|, |a_2|, |a_3|, |a_4| < 2^{62}$.

Proof. The tuple $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Q}^4$ is the unique solution to $(m, 0, 0, 0) = \sum_{i=1}^{4}\alpha_i\mathbf{b}_i$, so $m \equiv a_1 + a_2\lambda_\phi + a_3\lambda_\psi + a_4\lambda_\phi\lambda_\psi \pmod N$. The bounds on the a_i follow from all 16 corners of the parallelepiped $\mathcal{P}(\mathbf{B})$ having all four coordinates of absolute value less than 2^{62}. □

4.2 Handling round-off errors

The decomposition described in Proposition 4 requires the computation of four roundings $\lfloor \frac{\hat{\alpha}_i}{N} \cdot m \rceil$, where m is the input scalar and the four $\hat{\alpha}_i$ and N are fixed curve constants. Following [10, §4.2], one efficient way of performing these roundings is to choose a power of 2 greater than the denominator N, say μ, and precompute the fixed curve constants $\ell_i = \lfloor \frac{\hat{\alpha}_i}{N} \cdot \mu \rceil$, so that $\lfloor \frac{\hat{\alpha}_i}{N} \cdot m \rceil$ can be computed at runtime as $\lfloor \frac{\ell_i \cdot m}{\mu} \rfloor$, and the division by μ can be computed as a simple shift.

It is correctly noted in [10, §4.2] that computing the rounding in this way means the answer can be out by 1 in some cases, but it is further said that "*in practice this does not affect the size of the multiscalars*". While this assertion may have been true in [10], in general this will not be the case, particularly when we wish to bound the size of the multiscalars as tightly as possible. We address this issue on $\mathcal{E}$ starting with the following lemma.

Lemma 1. *Let $\hat{\alpha}$ be any integer, and let m, N and μ be positive integers with $m < \mu$. Then*

$$\left\lfloor \frac{\hat{\alpha} m}{N} \right\rceil - \left\lfloor \left\lfloor \frac{\hat{\alpha} \mu}{N} \right\rceil \cdot \frac{m}{\mu} \right\rfloor$$

is either 0 or 1.

Proof. We use $x - 1/2 \leq \lfloor x \rceil \leq x + 1/2$ and $x - 1 < \lfloor x \rfloor \leq x$ to see that the above value is greater than $-1/2 - m/(2\mu)$ and less than $3/2 + m/(2\mu)$. Since $m < \mu$, the value therefore lies strictly between -1 and 2 and the result follows from it being an integer. □

Lemma 1 says that, so long as we choose μ to be greater than the maximum size of our input scalars m, our fast method of approximating $\lfloor \frac{\hat{\alpha}_i}{N} \cdot m \rceil$ will either give the correct answer, or it will be $\lfloor \frac{\hat{\alpha}_i}{N} \cdot m \rceil - 1$. It is easy to see that larger choices of μ decrease the probability of a rounding error. For example, on 10 million random decompositions of integers between 0 and N with $\mu = 2^{246}$, roughly 2.2 million trials gave at least one error in the α_i; when $\mu = 2^{247}$, roughly 1.7 million trials gave at least one error; when $\mu = 2^{256}$, 4333 trials gave an error; and, taking $\mu = 2^{269}$ was the first power of two that gave no errors.

Prior works have seemingly addressed this problem by taking μ to be large enough so that the chance of roundoff errors are very (perhaps even exponentially) small. However, no matter how large μ is chosen, the existence of a permissible scalar whose decomposition gives a roundoff error is still a possibility[4], and this could violate constant-time promises.

In this work, and in light of Theorem 1, we instead choose to sacrifice some speed by guaranteeing that roundoff errors are always accounted for. Rather than assuming that $(a_1, a_2, a_3, a_4) = \sum_{i=1}^{4} (\alpha_i - \lfloor \alpha_i \rceil) \mathbf{b}_i$, we account for the approximation $\tilde{\alpha}_i$ to $\lfloor \alpha_i \rceil$ (described in Lemma 1) by allowing

$$(a_1, a_2, a_3, a_4) = \sum_{i=1}^{4} (\alpha_i - \tilde{\alpha}_i)\, \mathbf{b}_i = \sum_{i=1}^{4} (\alpha_i - (\lfloor \alpha_i \rceil - \epsilon_i))\, \mathbf{b}_i,$$

for all sixteen combinations arising from $\epsilon_i \in \{0, 1\}$, for $i = 1, 2, 3, 4$. This means that all integers less than μ will decompose to a multiscalar in $\mathbb{Z}^4$ whose coordinates lie inside the parallelepiped $\mathcal{P}_\epsilon(\mathbf{B}) := \{\mathbf{Bx} \,|\, \mathbf{x} \in [-1/2, 3/2)^4\}$. Theorem 1 permits scalars as any 256-bit strings, so we fix

[4] This is not technically true: so long as the set of permissible scalars is finite, there will always be a μ large enough to round all scalar decompositions accurately, but finding or proving this is, to our knowledge, very difficult.

$\mu := 2^{256}$ from here on, which also means that division by μ will correspond to a shift of machine words. The edges of $\mathcal{P}_\epsilon(\mathbf{B})$ are twice as long as those of $\mathcal{P}(\mathbf{B})$, so the number of points in $\mathcal{P}_\epsilon(\mathbf{B}) \cap \mathbb{Z}^4$ is $\mathrm{vol}(\mathcal{P}_\epsilon) = 16N$. We note that, even though the number of permissible scalars far exceeds $16N$, the decomposition that maps integers in $[0, \mu)$ to multiscalars in $\mathcal{P}_\epsilon(\mathbf{B}) \cap \mathbb{Z}^4$ is certainly no longer *onto*; almost all of the μ scalars will map into $\mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$, since the chance of roundoff errors that take us into $\mathcal{P}_\epsilon(\mathbf{B}) - \mathcal{P}(\mathbf{B})$ is small. Plainly, the width of smallest 4-cube containing $\mathcal{P}_\epsilon(\mathbf{B})$ is also twice that of the 4-cube containing $\mathcal{P}(\mathbf{B})$, so (in the sense of Definition 1) our basis is still Babai-optimal. Nevertheless, the bounds in Proposition 4 no longer apply, which is one of the issues addressed in the next subsection.

4.3 All-positive multiscalars

Many points in $\mathcal{P}_\epsilon(\mathbf{B}) \cap \mathbb{Z}^4$ have coordinates that are far greater than 2^{62} in absolute value, and in addition, the majority of them will have coordinates that are both positive and negative. Dealing with such signed multiscalars can require an additional iteration in the main loop of the scalar multiplication, so in this subsection we use an offset vector in $\mathcal{L}$ to find a translate of $\mathcal{P}_\epsilon(\mathbf{B})$ that contains points whose four coordinates are always positive. We note that this does not save the additional iteration mentioned above, but (at no cost) it does simplify the scalar recoding, such that we do not have to deal with multiscalars that can have negative coordinates. Such offset vectors were used in two dimensions in [15, §4].

From the proof of Proposition 3, we have that the width of the smallest 4-cube containing $\mathcal{P}_\epsilon(\mathbf{B})$ is $2{\cdot}(245\alpha{+}120r{+}17)/448$, which lies between 2^{63} and 2^{64}. Thus, the optimal situation is to translate of $\mathcal{P}_\epsilon(\mathbf{B})$ (using a vector in $\mathcal{L}$) that fits inside the convex body of the 4-cube $\mathcal{H} = \{2^{64}{\cdot}\mathbf{x} \,|\, \mathbf{x} \in [0,1]^4\}$. In fact, as we discuss in the next paragraph, we actually want to find two unique translates of $\mathcal{P}_\epsilon(\mathbf{B})$ inside $\mathcal{H}$.

The scalar recoding described in Section 5 requires that the first component of the multiscalar (a_1, a_2, a_3, a_4) is odd. In the case that a_1 is even, which happens around half of the time, previous works have employed this "odd-only" recoding by instead working with the multiscalar $(a_1 - 1, a_2, a_3, a_4)$, and adding the point P to the value output by the main loop (cf. [45, Alg. 4] and [19, Alg. 2]). Of course, in a constant-time routine, this scalar update and point addition must be performed regardless of the parity of a_1, and the correct scalars and results must be masked in and out of the main loop accordingly. In this work we simplify the situation by using offset vectors in $\mathcal{L}$ to achieve the same result; this has the added advantage of avoiding an extra point addition. We do this by finding two vectors $\mathbf{c}, \mathbf{c}' \in \mathcal{L}$ such that $\mathbf{c} + \mathcal{P}_\epsilon(\mathbf{B})$ and $\mathbf{c}' + \mathcal{P}_\epsilon(\mathbf{B})$ both lie inside $\mathcal{H}$, and such that precisely one of $(a_1, a_2, a_3, a_4) + \mathbf{c}$ and $(a_1, a_2, a_3, a_4) + \mathbf{c}'$ has a first component that is odd. This is made explicit in the full scalar decomposition described below.

Proposition 5 (Scalar Decompositions). *Let $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4)$ be the basis in Proposition 3, let $\mu = 2^{256}$, and define the four curve constants $\ell_i := \lfloor \hat{\alpha}_i \cdot \mu/N \rceil$ for $i = 1, 2, 3, 4$, with the $\hat{\alpha}_i$ as given in Proposition 4. Let $\mathbf{c} = 2\mathbf{b}_1 - \mathbf{b}_2 + 5\mathbf{b}_3 + 2\mathbf{b}_4$ and $\mathbf{c}' = 2\mathbf{b}_1 - \mathbf{b}_2 + 5\mathbf{b}_3 + \mathbf{b}_4$ in $\mathcal{L}$. For any integer $m \in [0, 2^{256})$, let $\tilde{\alpha}_i = \lfloor \ell_i m/\mu \rfloor$, and let (a_1, a_2, a_3, a_4) be given by*

$$\begin{aligned}
a_1 &= m - \tilde{\alpha}_1 \cdot \mathbf{b}_1[1] - \tilde{\alpha}_2 \cdot \mathbf{b}_2[1] - \tilde{\alpha}_3 \cdot \mathbf{b}_3[1] - \tilde{\alpha}_4 \cdot \mathbf{b}_4[1], \\
a_2 &= \quad -\tilde{\alpha}_1 \cdot \mathbf{b}_1[2] - \tilde{\alpha}_2 \cdot \mathbf{b}_2[2] - \tilde{\alpha}_3 \cdot \mathbf{b}_3[2] - \tilde{\alpha}_4 \cdot \mathbf{b}_4[2], \\
a_3 &= \quad -\tilde{\alpha}_1 \cdot \mathbf{b}_1[3] - \tilde{\alpha}_2 \cdot \mathbf{b}_2[3] - \tilde{\alpha}_3 \cdot \mathbf{b}_3[3] - \tilde{\alpha}_4 \cdot \mathbf{b}_4[3], \\
a_4 &= \quad -\tilde{\alpha}_1 \cdot \mathbf{b}_1[4] - \tilde{\alpha}_2 \cdot \mathbf{b}_2[4] - \tilde{\alpha}_3 \cdot \mathbf{b}_3[4] - \tilde{\alpha}_4 \cdot \mathbf{b}_4[4].
\end{aligned}$$

Then both of the multiscalars $(a_1, a_2, a_3, a_4) + \mathbf{c}$ *and* $(a_1, a_2, a_3, a_4) + \mathbf{c}'$ *are valid decompositions of* m*, have all four coordinates positive and less than* 2^{64}*, and precisely one of them has a first coordinate that is odd.*

Proof. Lemma 1 gives $\tilde{\alpha}_i = \lfloor \alpha_i \rceil - \epsilon_i$ with $\epsilon_i \in \{0, 1\}$ for $i = 1, 2, 3, 4$, where $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \mathbb{Q}^4$ is the unique solution to $(m, 0, 0, 0) = \sum_{i=1}^{4} \alpha_i \mathbf{b}_i$ in Proposition 4. The point (a_1, a_2, a_3, a_4) above is given as $(a_1, a_2, a_3, a_4) = (m, 0, 0, 0) - \sum_{i=1}^{4} \tilde{\alpha}_i \mathbf{b}_i$, and since $\mathbf{c}, \mathbf{c}' \in \mathcal{L}$, then both $(a_1, a_2, a_3, a_4) + \mathbf{c}$ and $(a_1, a_2, a_3, a_4) + \mathbf{c}'$ are in $(m, 0, 0, 0) + \mathcal{L}$, so are valid decompositions of m. Furthermore, $(a_1, a_2, a_3, a_4) + \mathbf{c} = \sum_{i=1}^{4} (\alpha_i - (\lfloor \alpha_i \rceil - \epsilon_i))\mathbf{b}_i + \mathbf{c}$ is in $\mathcal{P}_\epsilon(\mathbf{B}) + \mathbf{c}$, and similarly $(a_1, a_2, a_3, a_4) + \mathbf{c}'$ is in $\mathcal{P}_\epsilon(\mathbf{B}) + \mathbf{c}'$. All sixteen corners of $\mathcal{P}_\epsilon(\mathbf{B}) + \mathbf{c}$ and $\mathcal{P}_\epsilon(\mathbf{B}) + \mathbf{c}'$ are inside the convex body of $\mathcal{H}$, meaning they have all four coordinates positive and less than 2^{64}. Precisely, one of the multiscalars having an odd first coordinate follows from the difference of the two multiscalars being $\mathbf{c} - \mathbf{c}' = \mathbf{b}_4$, whose first coordinate is $(-360\alpha + 77r + 42)/28$, which is odd. □

The scalar decomposition described in Proposition 5 outputs two multiscalars. Our decomposition routine uses a bitmask to select and output the one with an odd first coordinate in constant time. We conclude this section with a remark concerning an alternative decomposition strategy.

Remark 4. Another way of dealing with negative components in multiscalars (a_1, a_2, a_3, a_4) that lie in $\mathcal{P}_\epsilon(\mathbf{B})$ is to always compute all eight points $\pm P$, $\pm\phi(P)$, $\pm\psi(P)$ and $\pm\psi(\phi(P))$, before selecting the correct combination according to the signs of the a_i, and proceeding with the multiscalar $(|a_1|, |a_2|, |a_3|, |a_4|)$. In this case the $|a_i|$ would then be at most 63 bits each, rather than the 64 bits that we have above. Moreover, there are many translates of $\mathcal{P}_\epsilon(\mathbf{B})$ inside $\mathcal{H}' = \{2^{63} \cdot \mathbf{x} \,|\, \mathbf{x} \in [-1, 1]^4\}$, so we can readily find two that differ by a vector in $\mathcal{L}$ whose first component is odd. This would require additional point negations and four masked selections, but would save us a double-and-add operation in the main loop. In our software we opted for the slower approach for its obvious simplicity benefits: our code avoids the sign-dependent maskings so that the outputs of scalar decomposition and endomorphism routines are fed into the main loop independently of one another.

5 The Scalar Multiplication

This section describes the full scalar multiplication of $P \in \mathcal{E}(\mathbb{F}_{p^2})$ by an integer $m \in [0, 2^{256})$, pulling together the endomorphisms and scalar decompositions derived in the previous two sections.

5.1 Recoding the multiscalar

The "all-positive" multiscalar (a_1, a_2, a_3, a_4) that is obtained from the decomposition described in Proposition 5 could be fed *as is* into a simple 4-way multiexponentiation (e.g., the 4-dimensional version of [52]) to achieve an efficient scalar multiplication. However, more care needs to be taken to obtain an efficient routine that also runs in constant-time. For example, we need to guarantee that the main loop iterates in the same number of steps, which would not currently be the case since $\max_j(\log_2(|a_j|))$ can be several integers less than 64. As another example, a straightforward multiexponentiation could leak information in the case that the i-th bit of all four a_j values was 0, which would result in a "do-nothing" rather than a non-trivial addition.

To achieve an efficient constant-time routine, we adopt the general recoding Algorithm from [19, Alg. 1], and tailor it to scalar multiplications on Four$\mathbb{Q}$. This results in Algorithm 1 below, which is presented in two flavors: one that is geared towards the general reader and one that is geared

towards implementers (we note that the lines do not coincide for the most part). On input of any multiscalar (a_1, a_2, a_3, a_4) produced by Proposition 5, Algorithm 1 outputs an equivalent multiscalar (b_1, b_2, b_3, b_4) with $b_j = \sum_{i=0}^{64} b_j[i] \cdot 2^i$ for $b_j[i] \in \{-1, 0, 1\}$ and $j = 1, 2, 3, 4$, such that we always have $b_1[64] = 1$ and such that $b_1[i]$ is non-zero for every $i = 0, \ldots, 63$. This fixes the length of the main loop and ensures that each addition step of the multiexponentiation requires an addition by something other than the neutral element.

Another benefit of Algorithm 1 is that $b_j[i] \in \{0, b_1[i]\}$ for $j = 2, 3, 4$; as was exploited in [19], this "sign-alignment" means that the lookup table used in our multiexponentiation only requires 8 elements, rather than the 16 that would be required in a naïve multiexponentiation that uses (a_1, a_2, a_3, a_4). More specifically, since $b_1[i]$ (which is to be multiplied by P) is always non-zero, every element of the lookup table T must contain P, so we have

$$T[u] := P + [u_0]\phi(P) + [u_1]\psi(P) + [u_2]\psi(\phi(P)),$$

where $u = (u_2, u_1, u_0)_2$ for $u = 0, \ldots, 7$. In Proposition 1 we present and prove the three required properties of the output multiscalars. We point out that the recoding must itself be implemented in constant-time; the implementer-friendly version shows that Algorithm 1 indeed lends itself to such an implementation. We further note that the outputs of the two versions are formatted differently: the left side outputs the multiscalar (b_1, b_2, b_3, b_4), while the right side instead outputs the corresponding lookup table indices (the d_i) and the masks (the m_i) used to select the correct signs of the lookup elements. That is, $(m_{64}, \ldots, m_0)$ corresponds to the binary expansion of b_1 and $(d_{64}, \ldots, d_0)$ corresponds to the binary expansion of $b_2 + 2b_3 + 4b_4$.

Algorithm 1 Four$\mathbb{Q}$ multiscalar recoding: reader-friendly (left) and implementer-friendly (right).

Input: four positive integers $a_j = (0, a_j[63], \ldots, a_j[0])_2 \in \{0, 1\}^{65}$ less than 2^{64} for $1 \leq j \leq 4$ and with a_1 odd.

Output: four integers $b_j = \sum_{i=0}^{64} b_j[i] \cdot 2^i$, with $b_j[i] \in \{-1, 0, 1\}$.

1: $b_1[64] = 1$
2: **for** $i = 0$ **to** 64 **do**
3: **if** $i \neq 64$ **then**
4: $b_1[i] = 2a_1[i+1] - 1$
5: **for** $j = 2$ **to** 4 **do**
6: $b_j[i] = b_1[i] \cdot a_j[0]$
7: $a_j = \lfloor a_j/2 \rfloor - \lfloor b_j[i]/2 \rfloor$
8: **return** $(b_j[64], \ldots, b_j[0])$ for $1 \leq j \leq 4$.

Output: $(d_{64}, \ldots, d_0)$ with $0 \leq d_i < 7$, and $(m_{64}, \ldots, m_0)$ with $m_i \in \{-1, 0\}$.

1: $m_{64} = -1$
2: **for** $i = 0$ **to** 63 **do**
3: $d_i = 0$
4: $m_i = -a_1[i+1]$
5: **for** $j = 2$ **to** 4 **do**
6: $d_i = d_i + (a_j[0] \ll (j-2))$
7: $c = (a_1[i+1] \,|\, a_j[0]) \wedge a_1[i+1]$
8: $a_j = (a_j \gg 1) + c$
9: $d_{64} = a_2 + 2a_3 + 4a_4$
10: **return** $(d_{64}, \ldots, d_0)$ and $(m_{64}, \ldots, m_0)$.

Proposition 6. *The four integers b_1, b_2, b_3 and b_4 output from Algorithm 1 are such that:*

(Property 1)	$b_j = a_j$,	*for* $1 \leq j \leq 4$;
(Property 2)	$b_1[i] \in \{-1, 1\}$,	*for* $0 \leq i \leq 64$;
(Property 3)	$b_j[i] \in \{0, b_1[i]\}$,	*for* $2 \leq j \leq 4$ *and* $0 \leq i \leq 64$.

Proof. We refer to the lines in the "reader-friendly" version. Property 3 follows immediately from Line 6 and Property 2 follows immediately from Lines 1 and 4. Property 1 with $j = 1$ also follows

from Lines 1 and 4 since $b_1 = 2^{64} + \sum_{i=0}^{63}(2a_1[i+1]-1)\cdot 2^i = (2^{64} - \sum_{i=0}^{63} 2^i) + \sum_{i=0}^{63} a_1[i+1]\cdot 2^{i+1} = 1 + \sum_{i=1}^{64} a_1[i]\cdot 2^i$, which is a_1 because $a_1[0] = 1$. It remains to prove Property 1 for $j = 2, 3, 4$, so for j as any of them, let a_j^i and b_j^i respectively denote the intermediate values of the integers a_j and b_j immediately after the execution of Line 7 of the i-th iteration, and note that $b_j^i = \sum_{k=0}^{i} b_j[k]\cdot 2^k$. We claim that the value $2^{i+1}\cdot a_j^i + b_j^i$ is invariant as follows. Line 6 gives $b_j^{i+1} = b_j^i + 2^{i+1}\cdot b_1[i]\cdot a_j^i[0]$ and Line 7 gives $a_j^{i+1} = \lfloor a_j^i/2 \rfloor - \lfloor b_j[i]/2 \rfloor = (a_j^i - a_j^i[0])/2 - \lfloor b_1[i]\cdot a_j^i[0]/2 \rfloor$. We then write $2^{i+2}\cdot a_j^{i+1} + b_j^{i+1} = 2^{i+1}\left((a_j^i - a_j^i[0])/2 - \lfloor b_1[i]a_j^i[0]/2 \rfloor\right) + \left(b_j^i + 2^{i+1}\cdot b_1[i]\cdot a_j^i[0]\right)$. Evaluating this expression for the four possible combinations of $a_j^i[0] \in \{0,1\}$ and $b_1[i] \in \{-1,1\}$ always gives $2^{i+2}\cdot a_j^{i+1} + b_j^{i+1} = 2^{i+1}\cdot a_j^i + b_j^i$, and hence the claimed invariance. Viewing the end of the first ($i = 0$) iteration yields that this invariant is the input integer a_j, and this invariance clearly holds until the last iteration ($i = 64$), at which point $a_j^{64} = 0$, giving $b_j^{64} = a_j$ as the output integer. □

5.2 Fast addition formulas

The fastest set of explicit formulas for the addition law on $\mathcal{E}$ are due to Hisil, Wong, Carter and Dawson [34]: they use *extended twisted Edwards coordinates* to represent the affine point (x, y) on $\mathcal{E}$ by any projective tuple of the form $(X\colon Y\colon Z\colon T)$ for which $Z \neq 0$, $x = X/Z$, $y = Y/Z$ and $T = XY/Z$.

Starting with the alternatives discussed in Hisil's thesis [32, §5.1.4], some minor modifications of the original formulas in [34] have proven to facilitate a more friendly implementation in certain scenarios. For example, Hamburg [29, §3.2] uses the tuple (X, Y, Z, T_a, T_b) to represent a point $(X\colon Y\colon Z\colon T)$ in extended twisted Edwards coordinates, where T_a and T_b are any field elements such that $T = T_aT_b$. In our case, a careful analysis of the full scalar multiplication routine and explicit formulas revealed that there are four alternative point representations that can be used to achieve a faster scalar multiplication. Table 2 summarizes these alternative representations, denoted $\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3$ and $\mathbf{R}_4$, and Table 3 summarizes the costs of the three functions that we need to convert between these representations, as well as the three functions used to compute the group law on $\mathcal{E}$. We point out that the function $\texttt{ADD}(P, Q)$ is equivalent to (and implemented as) $\texttt{ADD}(P, Q) = \texttt{ADD\_core}(\texttt{R1toR3}(P), Q)$, and reiterate that the three group law functions in Table 3 have no exceptions.

Table 2. Different representations of a point in extended twisted Edwards coordinates.

rep.	representation of $(X\colon Y\colon Z\colon T)$
$\mathbf{R_1}$	(X, Y, Z, T_a, T_b)
$\mathbf{R_2}$	$(X+Y, Y-X, 2Z, 2dT)$
$\mathbf{R_3}$	$(X+Y, Y-X, Z, T)$
$\mathbf{R_4}$	(X, Y, Z)

Table 3. Summary of conversion and addition functions, together with the (respective) representation of input and output points.

function	input	rep(s).	output	rep.	cost		
					M	**S**	**A**
R1toR2	P	$\mathbf{R_1}$	P	$\mathbf{R_2}$	2	-	4
R1toR3	P	$\mathbf{R_1}$	P	$\mathbf{R_3}$	1	-	2
R2toR4	P	$\mathbf{R_2}$	P	$\mathbf{R_4}$	-	-	2
ADD_core	P, Q	$\mathbf{R_3}, \mathbf{R_2}$	$P+Q$	$\mathbf{R_1}$	7	-	4
ADD	P, Q	$\mathbf{R_1}, \mathbf{R_2}$	$P+Q$	$\mathbf{R_1}$	8	-	6
DBL	P	$\mathbf{R_4}$	$[2]P$	$\mathbf{R_1}$	3	4	6

5.3 The full routine

We now present Algorithm 2: the full scalar multiplication routine. This is immediately followed by Theorem 1, which proves that Algorithm 2 computes the correct result in a constant number

(and fixed sequence) of operations. This proof also provides more details on the steps summarized in Algorithm 2; in particular, it specifies the representations of all points in order to state the total number of $\mathbb{F}_{p^2}$ operations. Algorithm 2 assumes that the input point P is in $\mathcal{E}(\mathbb{F}_{p^2})[N]$, i.e., has been validated according to Appendix A. As such we assume that P is input as a projective point represented using $\mathbf{R}_1$ (or $\mathbf{R}_4$, since the endomorphisms only need the first three coordinates – see Table 2).

Algorithm 2 Four$\mathbb{Q}$'s scalar multiplication on $\mathcal{E}(\mathbb{F}_{p^2})[N]$.

Input: Point $P \in \mathcal{E}(\mathbb{F}_{p^2})[N]$ and integer scalar $m \in [0, 2^{256})$.
Output: $[m]P$.

Compute endomorphisms:
1: Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$ using the explicit formulas summarized in Table 1.
Precompute lookup table:
2: Compute $T[u] = P + [u_0]\phi(P) + [u_1]\psi(P) + [u_2]\psi(\phi(P))$ for $u = (u_2, u_1, u_0)_2$ in $0 \leq u \leq 7$.
Scalar decomposition:
3: Decompose m into the multiscalar (a_1, a_2, a_3, a_4) as in Proposition 5.
Scalar recoding:
4: Recode (a_1, a_2, a_3, a_4) into $(d_{64}, \ldots, d_0)$ and $(m_{64}, \ldots, m_0)$ using Algorithm 1. Write $s_i = 1$ if $m_i = -1$ and $s_i = -1$ if $m_i = 0$.
Main loop:
5: $Q = s_{64} \cdot T[d_{64}]$
6: **for** $i = 63$ **to** 0 **do**
7: $\quad Q = [2]Q$
8: $\quad Q = Q + s_i \cdot T[d_i]$
9: **return** Q

Theorem 1. *For* every *point* $P \in \mathcal{E}(\mathbb{F}_{p^2})[N]$ *and* every *non-negative integer* m *less than* 2^{256}*, Algorithm 2 computes* $[m]P$ *correctly using a fixed sequence of exactly* 1**I**, 842**M**, 283**S**, 950.5**A** *and a fixed sequence of integer and table-lookup operations.*

Proof. We proceed through each line of Algorithm 2 and refer back to Table 3 for definitions and costs of lower level functions. As discussed in Remark 6, ϕ and ψ compute correctly for all points in $\mathcal{E}(\mathbb{F}_{p^2})[N]$, including the neutral point $(0, 1)$. Line 1 therefore requires exactly 68**M**, 27**S** and 49.5**A** (see Table 1), at which point we have P, $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$ in $\mathbf{R}_1$. Before computing the lookup table, we convert formats and take $P \leftarrow$ `R1toR2`(P), $\phi(P) \leftarrow$ `R1toR3`$(\phi(P))$, $\psi(P) \leftarrow$ `R1toR3`$(\psi(P))$, $\psi(\phi(P)) \leftarrow$ `R1toR3`$(\psi(\phi(P)))$ at a cost of 5**M** and 10**A**. Executing Line 2 then requires exactly 7 executions of `ADD_core`, which costs 49**M** and 28**A**. The output of these additions are in $\mathbf{R}_1$; in preparation for a faster main loop, they are all converted to $\mathbf{R}_2$ at a cost of 14**M** and 28**A**. Line 3 requires only integer operations and Proposition 5 proves that it computes a correct, all-positive decomposition (a_1, a_2, a_3, a_4) for every $0 \leq m < \mu = 2^{256}$, such that $0 < a_i < 2^{64} - 1$. For Line 4, Proposition 6 proves that Algorithm 1 computes a correct recoding of (a_1, a_2, a_3, a_4) using a fixed sequence of bit operations. Line 5 uses one point negation (costing 1**A**) and one table lookup to extract the initial value of Q; this is converted to $\mathbf{R}_4$ using `R2toR4` which costs 2**A**. What follows in Line 6-8 is 64 point doublings (`DBL`), 64 point additions (`ADD`), 64 point negations and 64 table lookups, which costs 704**M**, 256**S** and 832**A**. We reiterate that these group operations all work without exception. Finally, Line 9 requires a normalization which incurs 1**I** and 2**M**, and the tallied operation count is as claimed. □

Remark 5. Providing exact counts for the integer operations in the scalar decomposition is not too meaningful, since this depends on the underlying architecture. It suffices to say that, for example, the entire scalar decomposition requires less than 200 clock cycles on both the Sandy Bridge and Ivy Bridge architectures. In addition, we note that implementers targeting different architectures may find it advantageous exploit other trade-offs (within the explicit formulas) and arrive at different operation counts in $\mathbb{F}_{p^2}$ than those stated in Theorem 1.

6 Performance Analysis and Results

This section shows that, at the 128-bit security level, Four$\mathbb{Q}$ is significantly faster than all other known curve-based primitives. We reiterate that our software runs in constant-time and is therefore fully protected against timing and cache attacks.

6.1 Operation counts

We begin with a first-order comparison based on operation counts between Four$\mathbb{Q}$ and two other efficient curve-based primitives that are defined over large prime characteristic fields and that target the 128-bit security level: the twisted Edwards GLV+GLS curve defined over $\mathbb{F}_{p^2}$ with $p = 2^{127} - 5997$ proposed in [41], and the genus 2 Kummer surface defined over $\mathbb{F}_p$ with $p = 2^{127} - 1$ that was proposed in [26]; we dub these "GLV+GLS" and "Kummer" below. Both of these curves have recently set speed records on a variety of platforms (see [19] and [6]). Table 4 summarizes the operation counts for one variable-base scalar multiplication on Four$\mathbb{Q}$, GLV+GLS and Kummer. In the right-most column we approximate the cost in terms of prime field operations (using the standard assumption that 1 base field squaring is approximately 0.8 base field multiplications), where we round each tally to the nearest integer. For the GLV+GLS and Four$\mathbb{Q}$ operation counts, we assume that one multiplication over $\mathbb{F}_{p^2}$ involves 3 multiplications and 5 additions/subtractions over $\mathbb{F}_p$ (when using Karatsuba) and one squaring over $\mathbb{F}_{p^2}$ involves 2 multiplications and 3 additions/subtractions over $\mathbb{F}_p$.

Table 4. Operation counts for variable-base scalar multiplications on three different curves targeting the 128-bit security level. In the case of the Kummer surface, we additionally use a "word-mul" column to count the number of special multiplications of a general element in $\mathbb{F}_p$ by a small (i.e., one-word) constant – see [6].

primitive	prime	op. count over $\mathbb{F}_{p^2}$				approx. op. count over $\mathbb{F}_p$			
	char. p	inv	mul	sqr	add	inv	mul	add	word-mul
Four$\mathbb{Q}$	$2^{127} - 1$	1	842	283	950.5	1	3092	6960	-
GLV+GLS	$2^{127} - 5997$	1	833	191	769	1	2885	6278	-
Kummer	$2^{127} - 1$	-	-	-	-	1	4319	8032	2008

Table 4 shows that the GLV+GLS routine from [41] requires slightly fewer operations than Four$\mathbb{Q}$. This can mainly be explained by the faster endomorphisms, but (as we will see in Table 5) this difference is more than made up for by the faster modular arithmetic and superior simplicity of Four$\mathbb{Q}$. Table 4 shows that Four$\mathbb{Q}$ requires far fewer operations (in the same ground field) than Kummer; it is therefore expected, in general, that implementations based on Four$\mathbb{Q}$ outperform Kummer implementations for computing variable-base scalar multiplications.

6.2 Experimental Results

To evaluate performance, we wrote a standalone library supporting Four$\mathbb{Q}$ – see [16]. The library's design pursues modularity and code reuse, and leverages the simplicity of Four$\mathbb{Q}$'s arithmetic. It also facilitates the addition of specialized code for different platforms and applications: the core functionality of the library is fully written in portable C and works together with pluggable implementations of the arithmetic over $\mathbb{F}_{p^2}$ (and a few other complementary functions). The first release version of the library comes with two of those pluggable modules: a portable implementation written in C and a high-performance implementation for x64 platforms written in C and optional x64 assembly. The library computes all of the basic elliptic curve operations including variable-base, fixed-base and double-scalar multiplications, making it suitable for a wide range of cryptographic protocols. In addition, the software permits the selection (at build time) of whether or not the endomorphisms ψ and ϕ are to be exploited in variable-based scalar multiplications.

In Table 5, we compare Four$\mathbb{Q}$'s performance with other state-of-the-art implementations documented in the literature. Our benchmarks cover a wide range of x64 processors, from high-end architectures (e.g., Intel's Haswell) to low-end architectures (e.g., Intel's Atom). To cast the performance numbers in the context of a real-world protocol, we choose to illustrate Four$\mathbb{Q}$'s performance in one round of an ephemeral Diffie-Hellman (DH) key exchange. This means that both parties can generate their public keys using a fixed-base scalar multiplication and generate the shared secret using a variable-base scalar multiplication. Exploiting such precomputations to generate *truly* ephemeral public keys agrees with the comments made by Bernstein and Lange in [8, §1], e.g., that "*forward secrecy is at its strongest when a key is discarded immediately after its use*". Thus, Table 5 shows the execution time (in terms of clock cycles) for both variable-base and fixed-base scalar multiplications. We note that the laddered implementations in [4, 9, 6] only compute variable-base scalar multiplications, which is why we use the cost of two variable-base scalar multiplications to approximate the cost of ephemeral DH in those cases. For the Four$\mathbb{Q}$ and GLV+GLS implementations, precomputations for the fixed-base scalar multiplications occupied 7.5KB and 6KB of storage, respectively.

Table 5 shows that, in comparison with traditional curves, Four$\mathbb{Q}$ is 2.2–2.9 times faster than the Curve25519 implementations in [3, 14] and up to 5.6 times faster than the curve P-256 implementation in [27], when computing variable-base scalar multiplications. When considering the results for the DH key exchange, Four$\mathbb{Q}$ performs 1.9–3.7 times faster than Curve25519 and up to 4.4 times faster than curve P-256.

In terms of comparisons to the previously fastest implementations, variable-base scalar multiplications using our software are between 1.24 and 1.29 times faster than the Kummer [9, 6] and the GLV+GLS [19] implementations on AMD's Kaveri and Intel's Atom Pineview, Sandy Bridge and Ivy Bridge. The Kummer implementation for Haswell in [6] is particularly fast because it takes advantage of the powerful AVX2 vector instructions. Nevertheless, our implementation (which does not currently exploit vector instructions to accelerate the field arithmetic) is still 1.09x faster in the case of variable-base scalar multiplication. Moreover, in practice we expect a much larger advantage. For example, in the case of the DH key exchange, we leverage the efficiency of fixed-base scalar multiplications to achieve a factor 1.39x speedup over the Kummer implementation on Haswell. For the rest of platforms considered in Table 5, a DH shared secret using the Four$\mathbb{Q}$ software can be computed 1.6–1.7 times faster than a DH secret using the Kummer software in [6]. We note that the eBACS website [7] and [6] report different results for the same Kummer software on the same platform (i.e., Titan0): eBACS reports 60,556 Haswell cycles whereas [6] claims 54,389 Haswell

Table 5. Performance results (expressed in terms of thousands of clock cycles) of state-of-the-art implementations of various curves targeting the 128-bit security level on various x64 platforms. Benchmark tests were taken with Intel's TurboBoost and AMD's TurboCore disabled and the results were rounded to the nearest 1000 clock cycles. The benchmarks for the Four$\mathbb{Q}$ and GLV+GLS implementations were done on 1.66GHz Intel Atom N570 Pineview, 3.4GHz Intel Core i7-2600 Sandy Bridge, 3.4GHz Intel Core i7-3770 Ivy Bridge, 3.4GHz Intel Core i7-4770 Haswell, 2.30GHz Intel Core i5-5300U Broadwell and 3.1GHz AMD A8 PRO-7600B Kaveri. All the results for Four$\mathbb{Q}$, excepting on the Atom Pineview platform, were obtained with version 2.0 of Four$\mathbb{Q}$lib [16]. For the Kummer implementations [9, 6] and Curve25519 implementation [3], Atom Pineview, Sandy Bridge, Ivy Bridge and Haswell benchmarks were taken from eBACS [7] (machines h2atom, h6sandy, h9ivy and titan0), while AMD Kaveri and Intel Broadwell benchmarks were obtained by running eBACS' SUPERCOP toolkit on the corresponding targeted machine. The benchmarks for curve NIST P-256 were taken directly from [27] and the second set of Curve25519 benchmarks were taken directly from [14].

proc.	operation	**Four$\mathbb{Q}$ (this work)**	GLV+GLS [19]	Kummer [9]	Kummer [6]	Curve25519 [3]	Curve25519 [14]	P-256 [27]
Atom Pineview	var-base	**442**	-	556	-	1,109	-	-
	fixed-base	**217**	-	-	-	-	-	-
	ephem. DH	**659**	-	1,112	-	2,218	-	-
Sandy Bridge	var-base	**74**	92	123	89	194	157	400
	fixed-base	**42**	51	-	-	-	54	90
	ephem. DH	**116**	143	246	178	388	211	490
Ivy Bridge	var-base	**71**	89	119	88	183	159	-
	fixed-base	**39**	49	-	-	-	52	-
	ephem. DH	**110**	138	238	176	366	211	-
Haswell	var-base	**59**	-	111	61	162	-	312
	fixed-base	**33**	-	-	-	-	-	67
	ephem. DH	**92**	-	222	122	324	-	379
Broadwell	var-base	**50**	-	-	61	144	-	-
	fixed-base	**30**	-	-	-	-	-	-
	ephem. DH	**80**	-	-	122	288	-	-
AMD Kaveri	var-base	**122**	-	151	164	301	-	-
	fixed-base	**65**	-	-	-	-	-	-
	ephem. DH	**187**	-	302	328	602	-	-

cycles. This difference in performance raises questions regarding accuracy. The results that we obtained after running the eBACS' SUPERCOP toolkit on our own targeted Haswell machine seem to confirm that the results claimed in [6] for the Kummer were measured with TurboBoost enabled.

We recently benchmarked our software on an Intel Broadwell machine and observed, when computing variable-base scalar multiplications, factor-2.88 and factor-1.22 speedups in comparison with the Curve25519 implementation from [3] and the Kummer implementation from [6], respectively. This highlights the impressive performance of Four$\mathbb{Q}$ on modern CPUs without even considering the additional speedup that can be obtained through the use of fixed-base scalar multiplications or the use of vector instructions.

We note that in the case of binary elliptic curves the fastest implementation in the literature is due to Oliveira et al. [45]. Although its performance is just slightly slower than Four$\mathbb{Q}$'s performance for computing variable-base scalar multiplications on Haswell ([45] computes a variable-base scalar multiplication in 62 thousand cycles, according to eBACS [7]), it is not as competitive on other platforms (e.g., it costs 114 and 120 thousand cycles on Ivy and Sandy Bridge architectures [7]).

Software from [45] does not currently support fixed-base scalar multiplications.

Four$\mathbb{Q}$ without endomorphisms. Our library can be built with a version of the variable-base scalar multiplication function that does not exploit the endomorphisms ψ and ϕ to accelerate computations (note that fixed-base scalar multiplications do not exploit these endomorphisms by default). In this case, Four$\mathbb{Q}$ computes one variable-base scalar multiplication in (respectively) 92, 104, 127, 134 and 803 thousand cycles on the Broadwell, Haswell, Ivy Bridge, Sandy Bridge and Atom Pineview processors used for our experiments. These results are up to 3 times faster than the corresponding results for NIST P-256 and between 1.2–1.6 times faster than the corresponding results for Curve25519.

Acknowledgements. We thank Michael Naehrig for several discussions throughout this work, and Joppe Bos, Sorina Ionica and Greg Zaverucha for their comments on an earlier version of this paper. We are especially thankful to Ben Smith for pointing out the better option for ϕ in §3.2.

References

1. O. Ahmadi and R. Granger. On isogeny classes of Edwards curves over finite fields. Cryptology ePrint Archive, Report 2011/135, 2011. `http://eprint.iacr.org/`.
2. L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
3. D. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 124–142. Springer, 2011.
4. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006.
5. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 389–405. Springer, 2008.
6. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: New DH speed records. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 317–337. Springer, 2014.
7. D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems, accessed on May 19, 2015. `http://bench.cr.yp.to/results-dh.html`.
8. D. J. Bernstein and T. Lange. Hyper-and-elliptic-curve cryptography. *LMS Journal of Computation and Mathematics*, 17(A):181–202, 2014.
9. J. W. Bos, C. Costello, H. Hisil, and K. E. Lauter. Fast cryptography in genus 2. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*. Springer, 2013.
10. J. W. Bos, C. Costello, H. Hisil, and K. E. Lauter. High-performance scalar multiplication using 8-dimensional GLV/GLS decomposition. In G. Bertoni and J. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 331–348. Springer, 2013.
11. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Computation*, 24(3):235–265, 1997.
12. A. Bostan, F. Morain, B. Salvy, and É. Schost. Fast algorithms for computing isogenies between elliptic curves. *Mathematics of Computation*, 77(263):1755–1778, 2008.
13. Certicom Research. Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters, v2.0. Standard SEC2, Certicom, 2010.
14. T. Chou. Fastest Curve25519 implementation ever. Workshop on Elliptic Curve Cryptography Standards, 2015. `http://www.nist.gov/itl/csd/ct/ecc-workshop.cfm`.
15. C. Costello, H. Hisil, and B. Smith. Faster compact Diffie-Hellman: Endomorphisms on the x-line. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 183–200. Springer, 2014.
16. C. Costello and P. Longa. Four$\mathbb{Q}$lib. `http://research.microsoft.com/en-us/projects/fourqlib/`, 2015.

17. I. M. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In K. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology - ASIACRYPT '99*, volume 1716 of *LNCS*, pages 103–121. Springer, 1999.
18. H. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.
19. A. Faz-Hernández, P. Longa, and A. H. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves (extended version). *J. Cryptographic Engineering*, 5(1):31–52, 2015.
20. G. Frey, M. Müller, and H. Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Transactions on Information Theory*, 45(5):1717–1719, 1999.
21. S. D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.
22. S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011.
23. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 190–200. Springer, 2001.
24. P. Gaudry. Fast genus 2 arithmetic based on Theta functions. *J. Mathematical Cryptology*, 1(3):243–265, 2007.
25. P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symbolic Computation*, 44(12):1690–1702, 2009.
26. P. Gaudry and E. Schost. Genus 2 point counting over prime fields. *J. Symbolic Computation*, 47(4):368–400, 2012.
27. S. Gueron and V. Krasnov. Fast prime field elliptic curve cryptography with 256 bit primes. *J. Cryptographic Engineering*, 5(2):141–151, 2015.
28. A. Guillevic and S. Ionica. Four-dimensional GLV via the Weil restriction. In K. Sako and P. Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *LNCS*, pages 79–96. Springer, 2013.
29. M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/`.
30. M. Hamburg. Twisting Edwards curves with isogenies. Cryptology ePrint Archive, Report 2014/027, 2014. `http://eprint.iacr.org/`.
31. Y. Hasegawa. $\mathbb{Q}$-curves over quadratic fields. *Manuscripta Mathematica*, 94(1):347–364, 1997.
32. H. Hisil. *Elliptic curves, group law, and efficient computation*. PhD thesis, Queensland University of Technology, 2010.
33. H. Hisil and C. Costello. Jacobian coordinates on genus 2 curves. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 338–357. Springer, 2014.
34. H. Hisil, K. K. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, 2008.
35. Z. Hu, P. Longa, and M. Xu. Implementing 4-dimensional GLV method on GLS elliptic curves with j-invariant 0. *Des. Codes Cryptography*, 63(3):331–343, 2012.
36. A. H. Koblitz, N. Koblitz, and A. Menezes. Elliptic curve cryptography: The serpentine course of a paradigm shift. *J. Number theory*, 131(5):781–814, 2011.
37. D. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkeley, 1996.
38. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
39. C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroupp. In *Advances in Cryptology - CRYPTO'97*, pages 249–263, 1997.
40. P. Longa and C. Gebotys. Efficient techniques for high-speed elliptic curve cryptography. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *LNCS*, pages 80–94. Springer, 2010.
41. P. Longa and F. Sica. Four-dimensional Gallant-Lambert-Vanstone scalar multiplication. *J. Cryptology*, 27(2):248–283, 2014.
42. L. Lovász and H. E. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 17(3):751–764, 1992.
43. A. Menezes, S. A. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In C. Koutsougeras and J. S. Vitter, editors, *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 80–89. ACM, 1991.

44. National Institute of Standards and Technology (NIST). 186-2. Digital Signature Standard (DSS). *Federal Information Processing Standards (FIPS) Publication*, 2000.
45. T. Oliveira, J. López, D. F. Aranha, and F. Rodríguez-Henríquez. Two is the fastest prime: Lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014.
46. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.
47. J. Scholten. Weil restriction of an elliptic curve over a quadratic extension, URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.7987&rep=rep1&type=pdf`. 2004.
48. B. Smith. Families of fast elliptic curves from $\mathbb{Q}$-curves. In K. Sako and P. Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *LNCS*, pages 61–78. Springer, 2013.
49. B. Smith. Easy scalar decompositions for efficient scalar multiplication on elliptic curves and genus 2 Jacobians. *Contemporary Mathematics Series*, 637:15, 2015.
50. B. Smith. The $\mathbb{Q}$-curve construction for endomorphism-accelerated elliptic curves. *J. Cryptology (to appear)*, 2015.
51. H. M. Stark. Class-numbers of complex quadratic fields. In *Modular functions of one variable I*, volume 320 of *Lecture Notes in Math*, pages 153–174, 1973.
52. E. G. Straus. Addition chains of vectors. *American Mathematical Monthly*, 70(806-808):16, 1964.
53. J. Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:A238–A241, 1971.
54. M. J. Wiener and R. J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In S. E. Tavares and H. Meijer, editors, *Selected Areas in Cryptography (SAC'98)*, volume 1556 of *LNCS*. Springer, 1999.

A Point validation

The main scalar multiplication routine (in Algorithm 2) assumes that the input point lies in $\mathcal{E}(\mathbb{F}_{p^2})[N]$. However, since we have $\#\mathcal{E}(\mathbb{F}_{p^2}) = 392 \cdot N$, and in light of *small subgroup attacks* [39] that can be carried out in certain scenarios, here we briefly mention how our software enables the assertion (if desired) that scalar multiplications only accept points in $\mathcal{E}(\mathbb{F}_{p^2})[N]$. We then discuss why this validation is not done via more sophisticated means.

On input of a point $P = (x, y) \in \mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$, we

(i) Validate that $P \in \mathcal{E}$, i.e, assert that $-x^2 + y^2 = 1 + dx^2y^2$, otherwise reject P and abort.
(ii) Compute $P \leftarrow [392]P$. Plainly, since $(392)_2 = (1, 1, 0, 0, 0, 1, 0, 0, 0)$, this sequence involves 8 doubling and 2 addition operations. We note that these operations are independent of secret data and so constant-time strategies are irrelevant here.

Our method of achieving step (ii) above prompts the question as to why we do not absorb this cofactor into the decomposition. Indeed, this would be faster than multiplying input points by the cofactor outside of the multiexponentiation routine, since (as with the rest of the scalar) it compresses the length of the required loop by a factor close to 4.

The difficulty in "killing the cofactor" in this way arises because the initial decomposition in Proposition 4 maps *all* integers in the coset $m + N\mathbb{Z}$ to the same multiscalar $(a_1, a_2, a_3, a_4) \in \mathcal{P}(\mathbf{B}) \cap \mathbb{Z}^4$. Thus, if we instead take $\tilde{m} = 392m$ where m is chosen uniformly in an interval of length at least N, then the decomposition of $\tilde{m}$ will produce the same multiscalar as all integers in $\tilde{m} + N\mathbb{Z}$, and in particular, will produce the same multiscalar as the unique representative of this coset in $[0, N)$, the distribution of which is (approximately) uniform modulo 392. Subsequently, we cannot simply force scalars to be a multiple of the cofactor and expect the decomposition in Proposition 4 to respect this divisibility.

One possibility of dealing with the above problem is to use a sublattice $\mathcal{L}'$ of index 392 in $\mathcal{L}$, such that $\det(\mathcal{L}') = \#\mathcal{E}(\mathbb{F}_{p^2})$. This way, for any $m \in [0, N)$, then $\tilde{m} = 392m$ is the unique representative of the coset $\tilde{m} + 392N\mathbb{Z}$, and the decomposition *will* respect the divisibility of $\tilde{m}$ by 392.

Unfortunately though, the above approach cannot work on $\mathcal{E}(\mathbb{F}_{p^2})$. One reason for this is that there are small prime order subgroups of $\mathcal{E}(\mathbb{F}_{p^2})$ that are not fixed by both ψ and ϕ. The group structure of $\mathcal{E}$ is $\mathcal{E}(\mathbb{F}_{p^2}) \cong \mathbb{Z}/8\mathbb{Z} \times (\mathbb{Z}/7\mathbb{Z})^2 \times \mathbb{Z}/N\mathbb{Z}$, meaning that the entire 7-torsion is $\mathbb{F}_{p^2}$-rational; it consists of 8 linearly independent cyclic subgroups of order 7. On the one hand, ψ fixes each of these subgroups and has a consistent eigenvalue on $\mathcal{E}[7]$, namely $\psi|_{\mathcal{E}[7]} = [2]|_{\mathcal{E}[7]}$, but on the other hand, ϕ only fixes two of the subgroups and therefore does not have an eigenvalue on all of $\mathcal{E}[7]$. Subsequently, there is no meaningful way to build the (sub)lattice $\mathcal{L}'$ above such that it encompasses the action of ϕ and ψ on all points in $\mathcal{E}(\mathbb{F}_{p^2})$. Another obstruction concerning the use of $\mathcal{L}'$ is that $\mathcal{E}(\mathbb{F}_{p^2})[8]$ contains elements in the kernels of ϕ and ψ, since $\ker(\tau)$ is the four points of exact order 8 in $\mathcal{E}(\mathbb{F}_{p^2})$. This is why we choose to avoid any complications or exceptions, opting for a simple double-and-add sequence to compute $P \mapsto [392]P$.

Our code leaves the cofactor killing as an option to the user: the scalar multiplication API includes an input for enabling or disabling it during the computation. It is important to note that, in real-world scenarios, the cofactor killing can only be disabled when the input point is in $\mathcal{E}(\mathbb{F}_{p^2})[N]$, e.g., is a known (multiple of a) public generator that is asserted to be of order N. We point out that, even with cofactor killing enabled, the total cost of scalar multiplications is still significantly faster than all known implementations (many of which do not kill cofactors, e.g., [6]).

Remark 6. An observation that is important in the proof of Theorem 1 is that the ($\mathbb{F}_{p^2}$-rational parts of the) kernels of all of the isogenies in Table 1 are killed by the map $P \mapsto [392]P$, and furthermore, that all of the explicit formulas derived in this work are well behaved on $\mathcal{E}(\mathbb{F}_{p^2})[N]$, including the neutral point $(0, 1)$. This also means that, besides the two steps above, we never have to perform further checks or blacklist certain inputs into the main scalar multiplication.

Remark 7. There is a way to absorb *part* of the cofactor into a multiexponentiation. The explicit formulas for ϕ and ψ are well behaved on $\mathcal{E}(\mathbb{F}_{p^2})[4]$, mapping all four elements to the neutral point $(0, 1)$ on $\mathcal{E}$. Thus, the maps ϕ and ψ can be extended to $\mathcal{E}(\mathbb{F}_{p^2})[4N] \setminus \mathcal{E}(\mathbb{F}_{p^2})[4]$ without modifying the eigenvalues (we must remove the 4-torsion since both maps are equivalent to [0] when restricted to $\mathcal{E}(\mathbb{F}_{p^2})[4]$.). This means that we could instead work with the lattice $\tilde{\mathcal{L}} = \langle (z_1, z_2, z_3, z_4) \in \mathbb{Z}^4 \mid z_1 + z_2\lambda_\phi + z_3\lambda_\psi + z_4\lambda_\phi\lambda_\psi \equiv 0 \pmod{4N} \rangle$ of index 4 in $\mathcal{L}$, decompose scalars of the form $\tilde{m} = 4m$ for $m \in [0, N)$, and begin with $P \mapsto [98]P$, saving two doublings. In fact, the Babai-optimal basis $\tilde{\mathbf{B}}$ of $\tilde{\mathcal{L}}$ is such that all 16 corners of $\mathcal{P}(\tilde{\mathbf{B}})$ still have absolute value less than 2^{62}, meaning that these doublings could be absorbed into the multiexponentiation for free (if a similar treatment beginning with the analogue of Proposition 4 were carried out). However, this would require a new decomposition which means a new set of fixed constants; this is why we chose the simplified but slightly slower option of killing the full cofactor outside of the multiexponentiation.