Faster Compact Diffie-Hellman: Endomorphisms on the x-line

Craig Costello[1], Huseyin Hisil[2], and Benjamin Smith[3,4]

[1] Microsoft Research, Redmond, USA
craigco@microsoft.com
[2] Yasar University, Izmir, Turkey
huseyin.hisil@yasar.edu.tr
[3] INRIA (Équipe-projet GRACE), France
[4] LIX (Laboratoire d'Informatique), École polytechnique, France
smith@lix.polytechnique.fr

Abstract. We describe an implementation of fast elliptic curve scalar multiplication, optimized for Diffie–Hellman Key Exchange at the 128-bit security level. The algorithms are compact (using only x-coordinates), run in constant time with uniform execution patterns, and do not distinguish between the curve and its quadratic twist; they thus have a built-in measure of side-channel resistance. The core of our construction is a suite of two-dimensional differential addition chains driven by efficient endomorphism decompositions, built on curves selected from a family of $\mathbb{Q}$-curve reductions over $\mathbb{F}_{p^2}$ with $p = 2^{127} - 1$. We include state-of-the-art experimental results for twist-secure, constant-time, x-coordinate-only scalar multiplication.
Keywords: Elliptic curve cryptography, scalar multiplication, twist-secure, side channel attacks, endomorphism, Kummer variety, addition chains, Montgomery curve.

1 Introduction

In this paper, we discuss the design and implementation of state-of-the-art Elliptic Curve Diffie–Hellman key exchange (ECDH) primitives for security level of approximately 128 bits. The major priorities for our implementation are

1. **Compactness**: We target x-coordinate-only systems. These have the advantage of shorter keys, simple and fast algorithms, and (in a well-designed system) the ability to use arbitrary x-values, not just legitimate x-coordinates of points on a curve (the "illegitimate" values are simply x-coordinates on the quadratic twist). For x-coordinate ECDH, the elliptic curve exists only to supply formulæ for scalar multiplications, and a hard elliptic curve discrete logarithm problem (ECDLP) to underwrite a hard computational Diffie-Hellman problem (CDHP) on x-coordinates. The users should not have to verify whether given values correspond to points on a curve, nor should they have to compute any quantity that cannot be derived simply from x-coordinates alone. In particular, neither a user nor an algorithm should have to distinguish between the curve and its quadratic twist—and the curve must be chosen to be twist-secure.
2. **Fast, constant-time execution**: Every Diffie–Hellman key exchange is essentially comprised of four scalar multiplications,[5] so optimizing scalar multiplication $P \mapsto [m]P$ for varying P and m is a very high priority. At the same time, a minimum requirement for protecting against side-channel timing attacks is that every scalar multiplication $P \mapsto [m]P$ must be computed in constant time (and ideally with the same execution pattern), regardless of the values of m and P.

[5] We do not count the cost of authenticating keys, etc., here. Two of the scalar multiplications can be computed in advance in the case of static Diffie–Hellman; in this fixed-base scenario (where the point P is constant but m varies) we could profit from extensive precomputations. For simplicity, in this work we concentrate on the dynamic case (where P and m are variable).

In our concrete implementation, we target a security level of approximately 128 bits (comparable to `Curve25519` [3], `secp256r1` [26], or `brainpoolP256t1` [11]). The reference system with respect to our desired properties is Bernstein's `Curve25519`, which is based on an efficient, uniform differential addition chain applied to a well-chosen pair of curve and twist. `Curve25519` and its twist are presented as Montgomery models. These models not only provide highly efficient group operations, but they are optimized for x-coordinate-only operations; and crucially, they do not distinguish between the curve and its twist. Essentially, well-chosen Montgomery curves offer compactness straight out of the box.

Having chosen Montgomery curves as our platform, we must implement a fast, uniform, and constant-time scalar multiplication on x-coordinates of Montgomery models. To turbocharge our scalar multiplication, we apply a combination of efficiently computable pseudo-endomorphisms and two-dimensional differential addition chains. The use of efficient endomorphisms follows in the tradition of [20,33,15,14], but to the best of our knowledge, this work represents the first use of endomorphism scalar decompositions in the *pure* x-coordinate setting (that is, without additional input to the addition chain).

Our implementation is built on a curve-twist pair $(\mathcal{E}, \mathcal{E}')$ equipped with efficiently computable endomorphisms (ψ, ψ'). The family of $\mathbb{Q}$-curve reductions in [32] offer a combination of fast endomorphisms and compatiblity with fast underlying field arithmetic. Crucially (and unlike earlier endomorphism constructions such as [15,14]), they also offer the possibility of twist-secure group orders over fast fields. One of these curves, with almost-prime order over a 254-bit field, forms the foundation of our construction (see §2). Any other curve from the same family over the same field could be used with only very minor modifications to the formulæ below and the source code for our implementations; we explain our specific curve choice in Appendix B. The endomorphisms ψ and ψ' induce efficient pseudo-endomorphisms ψ_x and ψ'_x on the x-line; we explain their construction and use in §3.

The key idea of this work is to replace single scalar multiplications $(m, x(P)) \mapsto x([m]P)$ with multiscalar multiexponentiations

$$((a,b), x(P)) \longmapsto x([a]P \oplus [b]\psi(P)) = x([a]P \oplus [b]\psi'(P)) \ ,$$

where (a, b) is either a short multiscalar decomposition of a random full-length scalar m (that is, such that $[m]P = [a]P \oplus [b]\psi(P)$ or $[a]P \oplus [b]\psi'(P)$), or a random short multiscalar. The choice of ψ or ψ' formally depends on whether P is on $\mathcal{E}$ or $\mathcal{E}'$, but there is no difference between ψ and ψ' on the level of x-coordinates: they are implemented using exactly the same formulæ. Given that every element of the base field is the x-coordinate of a point on $\mathcal{E}$ or $\mathcal{E}'$, we may view the transformation above as acting purely on field elements, and not curve points.

From a practical point of view, the two crucial differences compared with conventional ECDH over a 254-bit field are

1. The use of 128-bit **multiscalars** (a, b) in $\mathbb{Z}^2$ in place of the 254-bit scalar m in $\mathbb{Z}$. We treat the geometry of multiscalars, the distribution of their corresponding scalar values, and the derivation of constant-bitlength scalar decompositions in §4.
2. The use of **two-dimensional differential addition chains** to compute $x([a]P \oplus [b]\psi(P))$ given only (a, b) and $x(P)$. We detail this process in §5.

We have implemented three different two-dimensional differential addition chains: one due to Montgomery [23] via Stam [34], one due to Bernstein [4], and one due to Azarderakhsh

and Karabina [1]. We provide implementation details and timings for scalar multiplications based on each of our chains in §6. Each offers a different combination of speed, uniformity, and constant-time execution. The differential nature of these chains is essential in the x-coordinate setting, which prevents the effective use of the vector chains traditionally used in the endomorphism literature (such as [35]).

We will be putting detailed Magma code online. A complete mixed-assembly-and-C code is publicly available in eBATS [5] format at

`http://hhisil.yasar.edu.tr/files/hisil20131024compact.tar.gz`

2 The curve

We begin by defining our curve-twist pair $(\mathcal{E}, \mathcal{E}')$. We work over the field $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, where

$$p := 2^{127} - 1 \qquad \text{and} \qquad i^2 = -1 \ .$$

We have chosen this Mersenne prime for its compatibility with a range of fast techniques for modular arithmetic, including Montgomery- and NIST-style approaches. We build efficient $\mathbb{F}_{p^2}$-arithmetic on top of the fast $\mathbb{F}_p$-arithmetic described in [10]. Appendix A provides a complete description of our arithmetic routines.

In what follows, it will be convenient to define the constants

$$u := 1466100457131508421 \ , \quad v := \tfrac{1}{2}(p-1) = 2^{126} - 1 \ , \quad \text{and} \quad w := \tfrac{1}{4}(p+1) = 2^{125} \ .$$

The curve $\mathcal{E}$ and its twist $\mathcal{E}'$. We define $\mathcal{E}$ to be the elliptic curve over $\mathbb{F}_{p^2}$ with affine Montgomery model

$$\mathcal{E} : y^2 = x(x^2 + Ax + 1) \ ,$$

where

$$A = A_0 + A_1 \cdot i \quad \text{with} \quad \begin{cases} A_0 = 451165543445558750850176275933214854 21 \ , \\ A_1 = 2415910908 \ . \end{cases}$$

The element $12/A$ is not a square in $\mathbb{F}_{p^2}$, so the curve over $\mathbb{F}_{p^2}$ defined by

$$\mathcal{E}' : (12/A)y^2 = x(x^2 + Ax + 1)$$

is a model of the quadratic twist of $\mathcal{E}$. The twisting $\mathbb{F}_{p^4}$-isomorphism $\delta : \mathcal{E} \to \mathcal{E}'$ is defined by $\delta : (x, y) \longmapsto (x, (A/12)^{1/2}y)$. The map $\delta_1 : (x, y) \mapsto (x_W, y_W) = (\frac{12}{A}x + 4, \frac{12^2}{A^2}y)$ defines an $\mathbb{F}_{p^2}$-isomorphism between $\mathcal{E}'$ and the Weierstrass model

$$\mathcal{E}_{2,-1,s} : y_W^2 = x_W^3 + 2(9(1+si) - 24)x_W - 8(9(1+si) - 16)$$

of [32, Theorem 1] with $s = i(1 - 8/A^2) = 86878915556079486902897638486322141403$, so $\mathcal{E}$ is a Montgomery model of the quadratic twist of $\mathcal{E}_{2,-1,s}$ (in the notation of [32, §5] we have $\mathcal{E} \cong \mathcal{E}'_{2,-1,s}$ and $\mathcal{E}' \cong \mathcal{E}_{2,-1,s}$). All of these curves have j-invariant

$$j(\mathcal{E}) = j(\mathcal{E}') = j(\mathcal{E}_{2,-1,s}) = 2^8 \frac{(A^2-3)^3}{A^2-4} = 2^6 \frac{(5-3si)^3(1-si)}{(1+s^2)^2} \ .$$

Group orders, structures, and generators. Using the SEA algorithm [28], we find that

$$\#\mathcal{E}(\mathbb{F}_{p^2}) = 4N \qquad \text{and} \qquad \#\mathcal{E}'(\mathbb{F}_{p^2}) = 8N'$$

where

$$N = v^2 + 2u^2 \qquad \text{and} \qquad N' = 2w^2 - u^2$$

are 252-bit and 251-bit primes, respectively. Looking closer, we see that

$$\mathcal{E}(\mathbb{F}_{p^2}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \qquad \text{and} \qquad \mathcal{E}'(\mathbb{F}_{p^2}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/N'\mathbb{Z}\ .$$

Recall that every element of $\mathbb{F}_{p^2}$ is either the x-coordinate of two points in $\mathcal{E}(\mathbb{F}_{p^2})$, the x-coordinate of two points in $\mathcal{E}'(\mathbb{F}_{p^2})$, or the x-coordinate of one point of order two in both $\mathcal{E}(\mathbb{F}_{p^2})$ and $\mathcal{E}'(\mathbb{F}_{p^2})$. The x-coordinates of the points of exact order 2 in $\mathcal{E}(\mathbb{F}_{p^2})$ (and in $\mathcal{E}'(\mathbb{F}_{p^2})$) are 0 and $-\frac{1}{2}A \pm \frac{1}{2}\sqrt{A^2-4}$; the points of exact order 4 in $\mathcal{E}'(\mathbb{F}_{p^2})$ have x-coordinates ± 1. Either of the points with x-coordinate 2 will serve as a generator for the cryptographic subgroup $\mathcal{E}(\mathbb{F}_{p^2})[N]$; either of the points with x-coordinate $2-i$ generate $\mathcal{E}'(\mathbb{F}_{p^2})[N']$.

Curve points, x-coordinates, and random bitstrings. Being Montgomery curves, both $\mathcal{E}$ and $\mathcal{E}'$ are compatible with the Elligator 2 construction [6, §5]. For our curves, [6, Theorem 5] defines efficiently invertible injective maps $\mathbb{F}_{p^2} \to \mathcal{E}(\mathbb{F}_{p^2})$ and $\mathbb{F}_{p^2} \to \mathcal{E}'(\mathbb{F}_{p^2})$. This allows points on $\mathcal{E}$ and/or $\mathcal{E}'$ to be encoded in such a way that they are indistinguishable from uniformly random strings of bitlength 254. Since we work with x-coordinates only in this article, a square root is saved when computing the injection (see [6, §5.5] for more details).

The ECDLP on $\mathcal{E}$ and $\mathcal{E}'$. Suppose we want to solve an instance of the DLP in $\mathcal{E}(\mathbb{F}_{p^2})$ or $\mathcal{E}'(\mathbb{F}_{p^2})$. Applying the Pohlig–Hellman–Silver reduction [24], we almost instantly reduce to the case of solving a DLP instance in either $\mathcal{E}(\mathbb{F}_{p^2})[N]$ or $\mathcal{E}'(\mathbb{F}_{p^2})[N']$. The best known approach to solving such a DLP instance is Pollard's rho algorithm [25], which (properly implemented) can solve DLP instances in $\mathcal{E}(\mathbb{F}_{p^2})[N]$ (resp. $\mathcal{E}'(\mathbb{F}_{p^2})[N']$) in around $\frac{1}{2}\sqrt{\pi N} \sim 2^{125.8}$ (resp. $\frac{1}{2}\sqrt{\pi N'} \sim 2^{125.3}$) group operations on average [9]. One might expect that working over $\mathbb{F}_{p^2}$ would imply a $\sqrt{2}$-factor speedup in the rho method by using Frobenius classes; but this seems not to be the case, since neither $\mathcal{E}$ nor $\mathcal{E}'$ is a subfield curve (cf. [36, §6]).

The embedding degrees of $\mathcal{E}$ and $\mathcal{E}'$ with respect to N and N' are $\frac{1}{50}(N-1)$ and $\frac{1}{2}(N'-1)$, respectively, so ECDLP instances in $\mathcal{E}(\mathbb{F}_{p^2})[N]$ and $\mathcal{E}(\mathbb{F}_{p^2})[N']$ are not vulnerable to the Menezes–Okamoto–Vanstone [21] or Frey–Rück [13] attacks. The trace of $\mathcal{E}$ is $p^2+1-4N \neq \pm 1$, so neither $\mathcal{E}$ nor $\mathcal{E}'$ are amenable to the Smart–Satoh–Araki–Semaev attack [30,27,29].

While our curves are defined over a quadratic extension field, this does not seem to reduce the expected difficulty of the ECDLP when compared to elliptic curves over similar-sized prime fields. Taking the Weil restriction of $\mathcal{E}$ (or $\mathcal{E}'$) to $\mathbb{F}_p$ as in the Gaudry–Hess–Smart attack [17], for example, produces a simple abelian surface over $\mathbb{F}_p$; and the best known attacks on DLP instances on simple abelian surfaces over $\mathbb{F}_p$ offer no advantage over simply attacking the ECDLP on the original curve [31,16]. (See [14, §9] for further discussion.)

Superficially, $\mathcal{E}$ is what we would normally call twist-secure (in the sense of Bernstein [3] and Fouque–Réal–Lercier–Valette [12]), since its twist $\mathcal{E}'$ has a similar security level. Indeed, $\mathcal{E}$ (and the whole class of curves from which it was drawn) was designed with this notion of twist-security in mind. However, twist-security is more subtle in the context of endomorphism-based scalar decompositions; we will return to this subject in §4 below.

The endomorphism ring. Let $\pi_{\mathcal{E}}$ denote the Frobenius endomorphism of $\mathcal{E}$. The curve $\mathcal{E}$ is ordinary (its trace $t_{\mathcal{E}}$ is prime to p), so its endomorphism ring is an order in the quadratic field $K := \mathbb{Q}(\pi_{\mathcal{E}})$. (The endomorphism ring of an ordinary curve and its twist are always isomorphic, so what holds below for $\mathcal{E}$ also holds for $\mathcal{E}'$.)

The `safecurves` specification [8] suggests that the discriminant of $\mathbb{Z}[\pi_{\mathcal{E}}]$ should have at least 100 bits. This discriminant is $t_{\mathcal{E}}^2 - 4p^2 = (2u)^2 D_K$, where

$$D_K = -8 \cdot 5 \cdot 397 \cdot 10528961 \cdot 6898209116497 \cdot 1150304667927101$$

is the fundamental discriminant of K. Hence, $t_{\mathcal{E}}^2 - 4p^2$ is a 253-bit integer, easily meeting the `safecurves` requirement. Note that this is within two bits of the discriminants for comparable curves such as `Curve25519` [3] and `brainpoolP256t1` [11].

One might argue that the discriminant condition should really be applied to the full endomorphism ring of $\mathcal{E}$, rather than $\mathbb{Z}[\pi_{\mathcal{E}}]$—especially since efficient endomorphisms play an important role in the use of $\mathcal{E}$. We will see below that $\mathcal{E}$ has an endomorphism ψ such that $\psi^2 = -[2]\pi_{\mathcal{E}}$. The conductor of $\mathbb{Z}[\pi_{\mathcal{E}}]$ in $\mathbb{Z}[\psi]$ is $2u$, so the discriminant of $\mathbb{Z}[\psi]$ is D_K, which implies that $\mathbb{Z}[\psi]$ is the maximal order in K; hence, $\mathrm{End}(\mathcal{E}) = \mathbb{Z}[\psi]$. This smaller discriminant D_K also meets the `safecurves` requirement. We note that well-chosen GLS curves can also have large endomorphism discriminants, but GLV curves have tiny endomorphism discriminants by construction: for example, the endomorphism ring of the curve `secp256k1` [26] (at the heart of the Bitcoin system) has discriminant -3.

`Brainpool` [11] requires the ideal class number of K to be larger than 10^6; this property is never satisfied by GLV curves, which have tiny class numbers (typically ≤ 2) by construction. But $\mathcal{E}$ easily meets this requirement: the class number of $\mathrm{End}(\mathcal{E})$ is

$$h(\mathrm{End}(\mathcal{E})) = h(D_K) = 2^7 \cdot 31 \cdot 37517 \cdot 146099 \cdot 505117 \sim 10^{19} \ .$$

3 The endomorphism on $\mathcal{E}$ and pseudo-endomorphisms on the x-line

Theorem 1 of [32] defines an efficient endomorphism

$$\psi_{2,-1,s} : (x_W, y_W) \longmapsto \left(\frac{-x_W^p}{2} - \frac{9(1-si)}{x_W^p - 4} \ , \ \frac{y_W^p}{\sqrt{-2}} \left(\frac{-1}{2} + \frac{9(1-si)}{(x_W^p - 4)^2} \right) \right)$$

of degree $2p$ on the Weierstrass model $\mathcal{E}_{2,-1,s}$, with kernel $\langle (4,0) \rangle$. To avoid an ambiguity in the sign of the endomorphism, we must fix a choice of $\sqrt{-2}$ in $\mathbb{F}_{p^2}$. We choose the "small" root:

$$\sqrt{-2} := 2^{64} \cdot i \ . \tag{1}$$

Applying the isomorphisms δ and δ_1, we define efficient $\mathbb{F}_{p^2}$-endomorphisms

$$\psi := (\delta_1 \delta)^{-1} \psi_{2,-1,s} \delta_1 \delta \qquad \text{and} \qquad \psi' := \delta \psi \delta^{-1} = \delta_1^{-1} \psi_{2,-1,s} \delta_1$$

of degree $2p$ on $\mathcal{E}$ and $\mathcal{E}'$, respectively, each with kernel $\langle (0,0) \rangle$. More explicitly: if we let

$$n(x) := \tfrac{A^p}{A} \left(x^2 + Ax + 1 \right) \ , \quad d(x) := -2x \ , \quad m(x) := n'(x)d(x) - n(x)d'(x),$$

$$r(x) := \tfrac{A^p}{A}(x^2 - 1) \ , \quad \text{and} \quad s(x) := \frac{n(x)^p}{d(x)^p},$$

then ψ and ψ' are defined (using the same value of $\sqrt{-2}$ fixed in Eq. (1)) by

$$\psi : (x,y) \mapsto \left(s(x) \ , \ -\frac{12^v}{A^v} \frac{y^p}{\sqrt{-2}} \frac{m(x)^p}{d(x)^{2p}} \right) \quad \text{and} \quad \psi' : (x,y) \mapsto \left(s(x) \ , \ -\sqrt{-2} \frac{12^{2v}}{A^{2v}} \frac{y^p r(x)^p}{d(x)^{2p}} \right) \ .$$

The action of the endomorphisms on curve points. Theorem 1 of [32] tells us that

$$\psi^2 = -[2]\pi_{\mathcal{E}} \qquad \text{and} \qquad (\psi')^2 = [2]\pi_{\mathcal{E}'} \ , \tag{2}$$

where $\pi_{\mathcal{E}}$ and $\pi_{\mathcal{E}'}$ are the p^2-power Frobenius endomorphisms of $\mathcal{E}$ and $\mathcal{E}'$, respectively, and

$$P(\psi) = P(\psi') = 0 \ , \qquad \text{where} \qquad P(T) = T^2 - 4uT + 2p \ . \tag{3}$$

If we restrict to the cryptographic subgroup $\mathcal{E}(\mathbb{F}_{p^2})[N]$, then ψ must act as multiplication by an integer eigenvalue λ, one of the two roots of $P(T)$ modulo N. Similarly, ψ' acts on $\mathcal{E}'(\mathbb{F}_{p^2})[N']$ as multiplication by one of the roots λ' of $P(T)$ modulo N'. The correct eigenvalues are

$$\lambda \equiv -\frac{v}{u} \pmod{N} \qquad \text{and} \qquad \lambda' \equiv -\frac{2w}{u} \pmod{N'} \ .$$

Equation (2) implies that $\lambda^2 \equiv -2 \pmod{N}$ and $\lambda'^2 \equiv 2 \pmod{N'}$. (Note that choosing the other square root of -2 in Eq. (1) negates ψ, ψ', λ, λ', and u.)

To complete our picture of the action of ψ on $\mathcal{E}(\mathbb{F}_{p^2})$ and ψ' on $\mathcal{E}'(\mathbb{F}_{p^2})$, we describe its action on the points of order 2 and 4 listed above. Choosing square roots appropriately:

$$\begin{array}{rcll} (0,0) & \longmapsto & 0 & \text{under } \psi \text{ and } \psi' \ , \\ \left(-\frac{1}{2}A \pm \frac{1}{2}\sqrt{A^2-4}, 0\right) & \longmapsto & (0,0) & \text{under } \psi \text{ and } \psi' \ , \\ \left(1, \pm\frac{1}{2}\sqrt{A(A+2)/3}\right) & \longmapsto & \left(-\frac{1}{2}A - \frac{1}{2}\sqrt{A^2-4}, 0\right) & \text{under } \psi' \ , \\ \left(-1, \pm\frac{1}{2}\sqrt{-A(A+2)/3}\right) & \longmapsto & \left(-\frac{1}{2}A + \frac{1}{2}\sqrt{A^2-4}, 0\right) & \text{under } \psi' \ . \end{array} \tag{4}$$

Pseudo-endomorphisms on the x-line. One advantage of the Montgomery model is that it allows a particularly efficient arithmetic using only the x-coordinate. Technically speaking, this corresponds to viewing the x-line $\mathbb{P}^1$ as the Kummer variety of $\mathcal{E}$: that is, $\mathbb{P}^1 \cong \mathcal{E}/\langle \pm 1 \rangle$.

The x-line is not a group: if P and Q are points on $\mathcal{E}$, then $x(P)$ and $x(Q)$ determine the pair $\{x(P \oplus Q), x(P \ominus Q)\}$, but not the individual elements $x(P \oplus Q)$ and $x(P \ominus Q)$. However, the x-line inherits part of the endomorphism structure of $\mathcal{E}$: every endomorphism ϕ of $\mathcal{E}$ induces a pseudo-endomorphism[6] $\phi_x : x \mapsto \phi_x(x)$ of $\mathbb{P}^1$, which determines ϕ up to sign; and if ϕ_1 and ϕ_2 are two endomorphisms of $\mathcal{E}$, then

$$(\phi_1)_x(\phi_2)_x = (\phi_2)_x(\phi_1)_x = (\phi_1\phi_2)_x = (\phi_2\phi_1)_x \ .$$

In addition[7] to pseudo-doubling (`DBL`), pseudo-addition (`ADD`), and combined pseudo-doubling and pseudo-addition (`DBLADD`) on $\mathbb{P}^1$, we need expressions for both ψ_x and $(\psi \pm 1)_x$ to initialise the addition chains in Section 5. Moving to projective coordinates, write $x = X/Z$ and $y = Y/Z$; then the negation map on $\mathcal{E}$ is $[-1] : (X : Y : Z) \mapsto (X : -Y : Z)$, and the double cover $\mathcal{E} \to \mathcal{E}/\langle [\pm 1] \rangle \cong \mathbb{P}^1$ is $(X : Y : Z) \mapsto (X : Z)$. The pseudo-doubling on $\mathbb{P}^1$ is

$$[2]_x((X : Z)) = \left((X+Z)^2(X-Z)^2 : (4XZ)\left((X-Z)^2 + \tfrac{A+2}{4} \cdot 4XZ\right)\right) \ . \tag{5}$$

Our endomorphism ψ induces the pseudo-endomorphism

$$\psi_x((X : Z)) = \left(A^p\left((X-Z)^2 - \tfrac{A+2}{2}(-2XZ)\right)^p : A(-2XZ)^p\right) \ .$$

Composing ψ_x with itself and comparing with Eq. (5), we confirm that $\psi_x\psi_x = -[2]_x(\pi_{\mathcal{E}})_x$.

[6] "Pseudo-endomorphisms" are, strictly speaking, endomorphisms of the x-line $\mathbb{P}^1$. We use the term pseudo-endomorphism here to avoid confusion with endomorphisms of elliptic curves, and also to fit with the use of terms like "pseudo-addition" for operations on the x-line.

[7] Montgomery's formulæ for these operations are available in explicit form on the EFD [7].

Proposition 1. *With the notation above, and with the value of* $\sqrt{-2}$ *chosen as in Eq.* (1),

$$\begin{aligned}(\psi \pm 1)_x(x) &= (\psi' \pm 1)_x(x) \\ &= \frac{2s^2nd^{4p} - x(xn)^p m^{2p} A^{p-1} \mp 2(1/A)^{(p-1)/2} m^p s(xn)^{(p+1)/2} d^{2p} A^{p-1}\sqrt{-2}}{2s(x-s)^2 d^{4p} A^{p-1}}. \quad (6)\end{aligned}$$

Proof. If P and Q are points on a Montgomery curve $By^2 = x(x^2 + Ax + 1)$, then

$$x(P \pm Q) = \frac{B\left(x(P)y(Q) \mp x(Q)y(P)\right)^2}{x(P)x(Q)\left(x(P) - x(Q)\right)^2}\ .$$

Taking $P = (x, y)$ to be a generic point on $\mathcal{E}$ (where $B = 1$), setting $Q = \psi(P)$, and using $y^2 = -\frac{A^p}{2A}dn$ to eliminate y yields the expression for $(\psi \pm 1)_x$ above. The same process for $\mathcal{E}'$ (with $B = \frac{12}{A}$), eliminating y with $\frac{12}{A}y^2 = -\frac{A^p}{2A}dn$, yields the same expression for $(\psi' \pm 1)_x$. □

Deriving explicit formulæ to compute the expression(s) in (6) is straightforward, so we omit them for space considerations (but see our code online). If $P \in \mathcal{E}$, then on input of $x(P)$, the combined computation of the three projective elements $(X_{\lambda-1} : Z_{\lambda-1})$, $(X_\lambda : Z_\lambda)$, $(X_{\lambda+1} : Z_{\lambda+1})$, which respectively correspond to the three affine elements $x([\lambda-1]P)$, $x([\lambda]P)$, $x([\lambda+1]P)$, incurs 15 multiplications, 129 squarings and 10 additions in $\mathbb{F}_{p^2}$. The bottleneck of this computation is raising dn to the power of $(p+1)/2 = 2^{126}$, which incurs 126 squarings. We note that squarings are significantly faster than multiplications in $\mathbb{F}_{p^2}$ (see Appendix A).

4 Scalar decompositions

We want to evaluate scalar multiplications $[m]P$ as $[a]P \oplus [b]\psi(P)$, where

$$m \equiv a + b\lambda \pmod{N}$$

and the multiscalar (a, b) has a significantly shorter bitlength[8] than m. For our applications we impose two extra requirements on multiscalars (a, b), so as to add a measure of side-channel resistance:

1. both a and b must be **positive**, to avoid branching and to simplify our algorithms; and
2. the multiscalar (a, b) must have **constant bitlength** (that is, bitlength independent of m as m varies over $\mathbb{Z}$), so that multiexponentiation can run in constant time.

In some protocols—notably Diffie–Hellman—we are not interested in the particular values of our random scalars, as long as those values remain secret. In this case, rather than starting with m in $\mathbb{Z}/N\mathbb{Z}$ (or $\mathbb{Z}/N'\mathbb{Z}$) and finding a short, positive, constant-bitlength decomposition of m, it would be easier to randomly sample some short, positive, constant-bitlength multiscalar (a, b) from scratch. The sample space must be chosen to ensure that the corresponding distribution of values $a + b\lambda$ in $\mathbb{Z}/N\mathbb{Z}$ does not make the discrete logarithm problem of finding $a + b\lambda$ appreciably easier than if we started with a random m.

[8] The *bitlength* of a scalar m is $\lceil \log_2 |m| \rceil$; the bitlength of a multiscalar (a, b) is $\lceil \log_2 \|(a, b)\|_\infty \rceil$.

The lattices of zero decompositions. The problems of finding good decompositions and sampling good multiscalars are best addressed using the geometric structure of the spaces of decompositions for $\mathcal{E}$ and $\mathcal{E}'$. The multiscalars (a, b) such that $a + b\lambda \equiv 0 \pmod{N}$ or $a + b\lambda' \equiv 0 \pmod{N'}$ form lattices

$$\mathcal{L} = \langle (N, 0), (-\lambda, 1) \rangle \qquad \text{and} \qquad \mathcal{L}' = \big\langle (N', 0), (-\lambda', 1) \big\rangle \ ,$$

respectively, with $a + b\lambda \equiv c + d\lambda \pmod{N}$ if and only if $(a, b) - (c, d)$ is in $\mathcal{L}$ (similarly, $a + b\lambda' \equiv c + d\lambda' \pmod{N'}$ if and only if $(a, b) - (c, d)$ is in $\mathcal{L}'$).

The sets of decompositions of m for $\mathcal{E}(\mathbb{F}_p)[N]$ and $\mathcal{E}(\mathbb{F}_{p^2})[N']$ therefore form lattice cosets

$$(m, 0) + \mathcal{L} \qquad \text{and} \qquad (m, 0) + \mathcal{L}' \ ,$$

respectively; we can find short decompositions of m for $\mathcal{E}(\mathbb{F}_p)[N]$ (resp. $\mathcal{E}(\mathbb{F}_{p^2})[N']$) by subtracting vectors near $(m, 0)$ in $\mathcal{L}$ (resp. $\mathcal{L}'$) from $(m, 0)$. To find these vectors, we need $\|\cdot\|_\infty$-reduced[9] bases for $\mathcal{L}$ and $\mathcal{L}'$.

Proposition 2 (Definition of $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}'_1, \mathbf{e}'_2$). *Up to order and sign, the shortest possible bases for $\mathcal{L}$ and $\mathcal{L}'$ (with respect to $\|\cdot\|_\infty$) are given by*

$$\begin{aligned} \mathcal{L} &= \langle\ \mathbf{e}_1 := (v, u)\ ,\ \mathbf{e}_2 := (-2u, v)\ \rangle \qquad\qquad \textit{and} \\ \mathcal{L}' &= \langle\ \mathbf{e}'_1 := (u, w)\ ,\ \mathbf{e}'_2 := (2u - 2w, 2w - u)\ \rangle\ . \end{aligned}$$

Proof. The proof of [32, Proposition 2] constructs sublattices $\langle \tilde{\mathbf{e}}_1 := -2(v, u), \tilde{\mathbf{e}}_2 := -2(2u, v) \rangle$ of index 4 in $\mathcal{L}$ and $\langle \tilde{\mathbf{e}}'_1 := 2(2w, -u), \tilde{\mathbf{e}}'_2 := 4(u, w) \rangle$ of index 8 in $\mathcal{L}'$. We easily verify that $\mathbf{e}_1 = -\frac{1}{2}\tilde{\mathbf{e}}_2$ and $\mathbf{e}_2 = -\frac{1}{2}\tilde{\mathbf{e}}_1$ are both in $\mathcal{L}$; since $\langle \tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2 \rangle$ has index 4 in $\langle \mathbf{e}_1, \mathbf{e}_2 \rangle$, we must have $\mathcal{L} = \langle \mathbf{e}_1, \mathbf{e}_2 \rangle$. Similarly, both $\mathbf{e}'_1 = \frac{1}{4}\tilde{\mathbf{e}}'_2$ and $\mathbf{e}'_2 = \frac{1}{2}(\tilde{\mathbf{e}}'_2 - \tilde{\mathbf{e}}'_1)$ are in $\mathcal{L}'$, and thus form a basis for $\mathcal{L}'$. According to [19, Definition 3], an ordered lattice basis $[\mathbf{b}_1, \mathbf{b}_2]$ is $\|\cdot\|_\infty$-reduced if

$$\|\mathbf{b}_1\|_\infty \leq \|\mathbf{b}_2\|_\infty \leq \|\mathbf{b}_1 - \mathbf{b}_2\|_\infty \leq \|\mathbf{b}_1 + \mathbf{b}_2\|_\infty \ .$$

These conditions are satisfied for $[\mathbf{b}_1, \mathbf{b}_2] = [\mathbf{e}_2, -\mathbf{e}_1]$ and $[\mathbf{e}'_1, \mathbf{e}'_2]$, so $(\|\mathbf{e}_2\|_\infty, \|\mathbf{e}_1\|_\infty)$ (resp. $(\|\mathbf{e}'_1\|_\infty, \|\mathbf{e}'_2\|_\infty)$) are the successive minima of $\mathcal{L}$ (resp. $\mathcal{L}'$) by [19, Theorem 5].[10] □

In view of Proposition 2, the fundamental parallelograms of $\mathcal{L}$ and $\mathcal{L}'$ are the regions of the (a, b)-plane defined by

$$\begin{aligned} \mathcal{A} &:= \big\{(a, b) \in \mathbb{R}^2\ :\ 0 \leq vb - ua < N \ \text{ and } \ 0 \leq 2ub + va < N\big\}\ , \\ \mathcal{A}' &:= \big\{(a, b) \in \mathbb{R}^2\ :\ 0 \leq ub - wa < N' \ \text{ and } \ 0 \leq (2u - 2w)b - (2w - u)a < N'\big\}\ . \end{aligned}$$

Every integer m has precisely one decomposition for $\mathcal{E}(\mathbb{F}_{p^2})[N]$ (resp. $\mathcal{E}'(\mathbb{F}_{p^2})[N']$) in any translate of $\mathcal{A}$ by $\mathcal{L}$ (resp. $\mathcal{A}'$ by $\mathcal{L}'$).

[9] That is, reduced with respect to Kaib's generalized Gauss reduction algorithm [19] for the infinity norm.
[10] For the Euclidean norm, the bases $[\mathbf{e}_1, \mathbf{e}_2]$ and $[\mathbf{e}'_1, 2\mathbf{e}'_1 - \mathbf{e}'_2]$ are $\|\cdot\|_2$-reduced, but $[\mathbf{e}'_1, \mathbf{e}'_2]$ is not.

Short, constant-bitlength scalar decompositions. Returning to the problem of finding short decompositions of m: let (α, β) be the (unique) solution in $\mathbb{Q}^2$ to the system $\alpha\mathbf{e}_1 + \alpha\mathbf{e}_2 = (m, 0)$. Since $\mathbf{e}_1, \mathbf{e}_2$ is reduced, the closest vector to $(m, 0)$ in $\mathcal{L}$ is one of the four vectors $\lfloor\alpha\rfloor\mathbf{e}_1 + \lfloor\beta\rfloor\mathbf{e}_2$, $\lfloor\alpha\rfloor\mathbf{e}_1 + \lceil\beta\rceil\mathbf{e}_2$, $\lceil\alpha\rceil\mathbf{e}_1 + \lfloor\beta\rfloor\mathbf{e}_2$, or $\lceil\alpha\rceil\mathbf{e}_1 + \lceil\beta\rceil\mathbf{e}_2$ by [19, Theorem 19]. Following Babai [2], we subtract $\lfloor\alpha\rceil\mathbf{e}_1 + \lfloor\beta\rceil\mathbf{e}_2$ from $(m, 0)$ to get a decomposition $(\tilde{a}, \tilde{b})$ of m; by the triangle inequality, $\|(\tilde{a}, \tilde{b})\|_\infty \leq \frac{1}{2}(\|\mathbf{e}_1\|_\infty + \|\mathbf{e}_2\|_\infty)$. This decomposition is approximately the shortest possible, in the sense that the true shortest decomposition is at most $\pm\mathbf{e}_1 \pm \mathbf{e}_2$ away. Observe that $\|\mathbf{e}_1\|_\infty = \|\mathbf{e}_2\|_\infty = 2^{126} - 1$, so $(\tilde{a}, \tilde{b})$ has bitlength at most 126.

However, $\tilde{a}$ or $\tilde{b}$ may be negative (violating the positivity requirement), or have fewer than 126 bits (violating the constant bitlength requirement). Indeed, $m \mapsto (\tilde{a}, \tilde{b})$ maps $\mathbb{Z}$ onto $(\mathcal{A} - \frac{1}{2}(\mathbf{e}_1 + \mathbf{e}_2)) \cap \mathbb{Z}^2$. This region of the (a, b)-plane, "centred" on $(0, 0)$, contains multiscalars of every bitlength between 0 and 126—and the majority of them have at least one negative component. We can achieve positivity and constant bitlength by adding a carefully chosen offset vector from $\mathcal{L}$, translating $(\mathcal{A} - \frac{1}{2}(\mathbf{e}_1 + \mathbf{e}_2)) \cap \mathbb{Z}^2$ into a region of the (a, b)-plane where every multiscalar is positive and has the same bitlength. Adding $3\mathbf{e}_1$ or $3\mathbf{e}_2$ ensures that the first or second component always has precisely 128 bits, respectively; but adding $3(\mathbf{e}_1 + \mathbf{e}_2)$ gives us a constant bitlength of 128 bits in both. Theorem 3 makes this all completely explicit.

Theorem 3. *Given an integer m, let (a, b) be the multiscalar defined by*

$$a := m + (3 - \lfloor\alpha\rceil)\, v - 2\,(3 - \lfloor\beta\rceil)\, u \qquad \textit{and} \qquad b := (3 - \lfloor\alpha\rceil)\, u + (3 - \lfloor\beta\rceil)\, v \ ,$$

where α and β are the rational numbers

$$\alpha := (v/N)m \qquad \textit{and} \qquad \beta := -(u/N)m \ .$$

Then $2^{127} < a, b < 2^{128}$, and $m \equiv a + b\lambda \pmod N$. In particular, (a, b) is a positive decomposition of m, of bitlength exactly 128, for any m.

Proof. We have $m \equiv a + b\lambda \pmod N$ because $(a, b) = (\tilde{a}, \tilde{b}) + 3(\mathbf{e}_1 + \mathbf{e}_2) \equiv (m, 0) \pmod{\mathcal{L}}$, where $(\tilde{a}, \tilde{b})$ is the translate of $(m, 0)$ by the Babai roundoff $\lfloor\alpha\rceil\mathbf{e}_1 + (\lfloor\beta\rceil\mathbf{e}_2$ described above. Now $(\tilde{a}, \tilde{b})$ lies in $\mathcal{A} - \frac{1}{2}(\mathbf{e}_1 + \mathbf{e}_2)$, so (a, b) lies in $\mathcal{A} + \frac{5}{2}(\mathbf{e}_1, \mathbf{e}_2)$; our claim on the bitlength of (a, b) follows because the four "corners" of this domain all have 128-bit components. □

Random multiscalars. As we remarked above, in a pure Diffie–Hellman implementation it is more convenient to simply sample random multiscalars than to decompose randomly sampled scalars. Proposition 4 shows that random multiscalars of at most 127 bits correspond to reasonably well-distributed values in $\mathbb{Z}/N\mathbb{Z}$ and in $\mathbb{Z}/N'\mathbb{Z}$, in the sense that none of the values occur more than one more or one fewer times than the average, and the exceptional values are in $O(\sqrt{N})$. Such multiscalars can be trivially turned into constant-bitlength positive 128-bit multiscalars—compatible with our implementation—by (for example) completing a pair of 127-bit strings with a 1 in the 128-th bit position of each component.

Proposition 4. *Let $\mathcal{B} = [0, p]^2$; we identify $\mathcal{B}$ with the set of all pairs of strings of 127 bits.*

1. *The map $\mathcal{B} \to \mathbb{Z}/N\mathbb{Z}$ defined by $(a, b) \mapsto a + b\lambda \pmod N$ is 4-to-1, except for $4(p - 6u + 4) \approx 4\sqrt{2N}$ values in $\mathbb{Z}/N\mathbb{Z}$ with 5 preimages in $\mathcal{B}$, and $8(u^2 - 3u + 2) \approx \frac{1}{5}\sqrt{N}$ values in $\mathbb{Z}/N\mathbb{Z}$ with only 3 preimages in $\mathcal{B}$.*

2. *The map* $\mathcal{B} \to \mathbb{Z}/N'\mathbb{Z}$ *defined by* $(a,b) \mapsto a + b\lambda' \pmod{N'}$ *is 8-to-1, except for* $8u^2 \approx \frac{2}{7}\sqrt{N'}$ *values with 9 preimages in* $\mathcal{B}$.

Proof (Sketch). For (1): the map $(a,b) \mapsto a + b\lambda \pmod{N}$ defines a bijection between each translate of $\mathcal{A} \cap \mathbb{Z}^2$ by $\mathcal{L}$ and $\mathbb{Z}/N\mathbb{Z}$. Hence, every m in $\mathbb{Z}/N\mathbb{Z}$ has a unique preimage (a_0, b_0) in $\mathcal{A} \cap \mathbb{Z}^2$, so it suffices to count $((a_0,b_0) + \mathcal{L}) \cap \mathcal{B}$ for each (a_0,b_0) in $\mathcal{A} \cap \mathbb{Z}^2$. Cover $\mathbb{Z}^2$ with translates of $\mathcal{A}$ by $\mathcal{L}$; the only points in $\mathbb{Z}^2$ that are on the boundaries of tiles are the points in $\mathcal{L}$. Dissecting $\mathcal{B}$ along the edges of translates of $\mathcal{A}$ and reassembling the pieces, we see that $8v - 24u + 20 < 4p$ multiscalars in $\mathcal{B}$ occur with multiplicity five, $8u^2 - 24u + 16 < p/9$ with multiplicity three, and every other multiscalar occurs with multiplicity four. There are therefore $4N + (8v - 24u + 20) - (8u^2 - 24u + 16) = (p+1)^2$ preimages in total, as expected. The proof of (2) is similar to (1), but counting $((a,b) + \mathcal{L}') \cap \mathcal{B}$ as (a,b) ranges over $\mathcal{A}'$. □

Twist-security with endomorphisms. We saw in §2 that DLPs on $\mathcal{E}$ and its twist $\mathcal{E}'$ have essentially the same difficulty, while Proposition 4 shows that the real DLP instances presented to an adversary by 127-bit multiscalar multiplications are not biased into a significantly more attackable range. But there is an additional subtlety when we consider the fault attacks considered in [3] and [12]: If we try to compute $[m]P$ for P on $\mathcal{E}$, but an adversary sneaks in a point P' on the twist $\mathcal{E}'$ instead, then in the classical context the adversary can derive m after solving the discrete logarithm $[m \mod N']P'$ in $\mathcal{E}'(\mathbb{F}_{p^2})$. But in the endomorphism context, we compute $[m]P$ as $[a]P \oplus [b]\psi(P)$, and the attacker sees $[a + b\lambda' \mod N']P'$, which is *not* $[m \mod N']P'$ (or even $[a + b\lambda \mod N']P'$); we should ensure that the values $(a + b\lambda' \mod N')$ are not concentrated in a small subset of $\mathbb{Z}/N'\mathbb{Z}$ when (a,b) is a decomposition for $\mathcal{E}(\mathbb{F}_{p^2})[N]$. This can be achieved by a similar argument to that of Proposition 4: the map $\mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N'\mathbb{Z}$ defined by $m \mapsto (a,b) \mapsto a + b\lambda' \pmod{N'}$ is a good approximation of a 2-to-1 mapping.

5 Two-dimensional differential addition chains

Addition chains are used to compute scalar multiplications using a sequence of group operations (or pseudo-group operations). A *one-dimensional* addition chain computes $[m]P$ for a given integer m and point P; a *two-dimensional* addition chain computes $[a]P \oplus [b]Q$ for a given multiscalar (a,b) and points P and Q. In a *differential* addition chain, the computation of any `ADD` $P \oplus Q$ is always preceded (at some earlier stage in the chain) by the computation of its associated difference $P \ominus Q$. The simplest differential addition chain is the original one-dimensional "Montgomery ladder" [22], which computes scalar multiplications $[m]P$ for a single exponent m and point P. Every `ADD` in the Montgomery ladder is in the form $[i]P \oplus [i+1]P$, so every associated difference is equal to P. Several two-dimensional differential addition chains have been proposed, targeting multi-exponentiations in elliptic curves and other primitives; we suggest [4] and [34] for overviews.

In any two-dimensional differential chain computing $[a]P \oplus [b]Q$ for general P and Q, the input consists of the multiscalar (a,b) and the three points P, Q, and $P \ominus Q$. The initial difference $P \ominus Q$ (or equivalently, the initial sum $P \oplus Q$) is essential to kickstart the chain on P and Q, since otherwise (by definition) $P \oplus Q$ cannot appear in the chain. As we noted in §1, computing this initial difference is an inconvenient obstruction to pure x-coordinate only multiexponentiations on general input: the pseudo-group operations `ADD`, `DBL`, and `DBLADD` can all be made to work on x-coordinates (the `ADD` and `DBLADD` operations make use of the

associated differences that are available in a differential chain), but in general it is impossible to compute the initial difference $x(P \ominus Q)$ in terms of $x(P)$ and $x(Q)$.

For our application, we want to compute $x([a]P \oplus [b]\psi(P))$ given inputs (a, b) and $x(P)$. Crucially, we can compute $x(P \ominus \psi(P))$ as $(\psi - 1)_x(x(P))$ using Proposition 1; this allows us to compute $x([a]P \oplus [b]\psi(P))$ using two-dimensional differential addition chains with input (a, b), $x(P)$, $\psi_x(x(P))$, and $(\psi - 1)_x(x(P))$.

We have implemented one one-dimensional differential addition chain (LADDER) and three two-dimensional differential addition chains (PRAC, AK, and DJB). We briefly describe each chain, with its relative benefits and drawbacks, below.

(Montgomery) LADDER chains. We implemented the full-length one-dimensional Montgomery ladder, as a reference to assess the speedup that our techniques offer over conventional scalar multiplication (It is also used as a subroutine within PRAC). The ladder can be made constant-time by adding a suitable multiple of N to the input scalar.

PRAC chains. Montgomery [23] proposed a number of algorithms for generating differential addition chains that are often much shorter than his eponymous ladder. His one-dimensional "PRAC" routine contains an easily-implemented two-dimensional subroutine, which computes the double-exponentiation $[a]P \oplus [b]Q$ very efficiently. The downside for our purposes is that the chain is not *uniform*: different inputs (a, b) give rise to different execution patterns, rendering the routine vulnerable to a number of side-channel attacks. Our implementation of this chain follows Algorithm 3.25 of [34][11]: given a multiscalar (a, b) and points P, Q, and $P - Q$, this algorithm computes $d = \gcd(a, b)$ and $R = [\frac{a}{d}]P \oplus [\frac{b}{d}]Q$. To finish computing $[a]P \oplus [b]Q$, we write $d = 2^i e$ with $i \geq q$ and e odd, then compute $S = [2^i]R$ with i consecutive `DBL`s, before finally computing $[e]S$ with a one-dimensional LADDER chain[12].

AK chains. Azarderakhsh and Karabina [1] recently constructed a two-dimensional differential addition chain which offers some middle ground in the trade-off between uniform execution and efficiency. While it is less efficient than PRAC, their chain has the advantage that all but one of the iterations consist of a single `DBLADD`; this uniformity may be enough to thwart some simple side-channel attacks. The single iteration which does *not* use a `DBLADD` requires a separate `DBL` and `ADD`, and this slightly slower step can appear at different stages of the algorithm. The location of this longer step could leak some information to a side-channel adversary under some circumstances, but we can protect against this by replacing all of the `DBLADD`s with separate `DBL` and `ADD`s, incurring a very minor performance penalty. A more serious drawback for this chain is its variable length: the total number of iterations depends on the input multiscalar. This destroys any hope of achieving a runtime that is independent of the input. Nevertheless, depending on the physical threat model, this chain may still be a suitable alternative. Our implementation of this chain follows Algorithm 1 in [1].

DJB chains. Bernstein gives the fastest known two-dimensional differential chain that is both fixed length and uniform [4, §4]. This chain is slightly slower than the PRAC and AK chains, but it offers stronger resistance against many side-channel attacks.[13] If the multiscalar

[11] We implemented the binary version of Montgomery's two-dimensional PRAC chain, neglecting the ternary steps in [23, Table 4] (see also [34, Table 3.1]). Isncluding these ternary steps could be significantly faster than our implementation, though it would require fast explicit formulæ for tripling on Montgomery curves.

[12] In practice d is very small, so there is little benefit in using a more complicated chain for this final step.

[13] It would be interesting to implement our techniques with Bernstein's non-uniform two-dimensional *extended-gcd* differential addition chain [4]. This chain can outperform two-dimensional PRAC, though it "takes more time to compute and is not easy to analyse".

(a, b) has bitlength ℓ, then this chain requires precisely $\ell - 1$ iterations, each of which computes one `ADD` and one `DBLADD`. In our case, Theorem 3 allows us to fix the number of iterations at 127. The execution pattern of the multiexponentiation is therefore independent of the input, and will run in constant time. It takes some work to organise the description in [4] into a concrete algorithm; we give an algorithm specific to our chosen curve in Appendix C.

Operation counts. Table 1 profiles the number of high-level operations required by each of our addition chain implementations on $\mathcal{E}$. We used the decomposition in Theorem 3 to guarantee positive constant-bitlength multiscalars. In situations where side-channel resistance is not a priority, and the AK or PRAC chain is preferable, variable-length decompositions could be used: these would give lower operation counts and slightly faster average timings.

Table 1. Pseudo-group operation counts per scalar multiplication on the x-line of $\mathcal{E}$ for the 2-dimensional DJB, AK and PRAC chains (using endomorphism decompositions) and the 1-dimensional LADDER. The counts for LADDER and DJB are exact; those for PRAC and AK are averages, with corresponding standard deviations, over 10^6 random trials (random scalars and points). In addition to the operations listed here, each chain requires a final $\mathbb{F}_{p^2}$-inversion to convert the result into affine form.

chain	dim.	endomorphisms	#DBL		#ADD		#DBLADD	
		ψ_x, $(\psi \pm 1)_x$	av.	std. dev.	av.	std. dev.	av.	std. dev.
LADDER	1	—	1	—	—	—	253	—
DJB	2	affine	1	—	128	—	127	—
AK	2	affine	1	—	1	—	179.6	6.7
PRAC	2	projective	0.2	0.4	113.8	11.6	73.4	11.1

The LADDER and DJB chains offer some slightly faster high-level operations. In these chains, then "difference elements" fed into the `ADD`s are fixed; if these points are affine, then this saves one $\mathbb{F}_{p^2}$-multiplication for every `ADD`. In LADDER, the difference is always the affine $x(P)$, so these savings come for free. In DJB, the difference is always one of the four values $x(P)$, $\psi_x(x(P))$, or $(\psi \pm 1)_x(x(P))$, so a shared inversion is used to convert $\psi_x(x(P))$ and $(\psi \pm 1)_x(x(P))$ from projective to affine coordinates. While this costs one $\mathbb{F}_{p^2}$-inversion and six-$\mathbb{F}_{p^2}$ multiplications, it saves 253 $\mathbb{F}_{p^2}$-inversions inside the loop.

6 Timings

Table 2 lists cycle counts for our implementations run on an Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz with hyper-threading turned off, over-clocking ("turbo-boost") disabled, and all-but-one of the cores switched off in BIOS. The implementations were compiled with gcc 4.6.3 and tested on a 64-bit Linux environment. All cycle counts were obtained using the SUPERCOP toolkit [5].

The most meaningful performance comparison we can draw is with Bernstein's `Curve25519` software. Like our software, `Curve25519` works entirely on the x-line, from start to finish; using the uniform one-dimensional Montgomery ladder, it runs in constant time. Thus, fair performance comparisons can only be made between his implementation and the two of ours that are also both uniform and constant-time: LADDER and DJB. Benchmarked on our hardware with all settings as above, `Curve25519` scalar multiplications ran in 182,000 cycles on average. Looking at Table 2, we see that using the one-dimensional LADDER on the

Table 2. Performance timings for four different implementations of compact, x-coordinate-only scalar multiplications targeting the 128-bit security level. Timings are given for the one-dimensional Montgomery ladder, as well as the two-dimensional (DJB, AK and PRAC) chains that benefit from the application of an endomorphism and subsequent short scalar decompositions.

addition chain	dimension	uniform?	constant time?	cycles
LADDER	1	✓	✓	152,000
DJB	2	✓	✓	145,000
AK	2	✓	✗	130,000
PRAC	2	✗	✗	110,000

x-line of $\mathcal{E}$ gives a factor 1.20 speed up over `Curve25519`, while combining an endomorphism with the two-dimensional DJB chain on the x-line of $\mathcal{E}$ gives a factor 1.26 speed up over `Curve25519`.

While there are several other implementations targeting the 128-bit security level that give faster performance numbers than ours, we reiterate that our aim was to push the boundary in the arena of implementations that are purely x-coordinate-only.

Hamburg [18] has also documented a fast software implementation employing x-coordinate-only Montgomery arithmetic. However, it is difficult to compare Hamburg's software with ours: his is not available to be benchmarked, and his figures were obtained on the Sandy Bridge architecture (they were also manually scaled back to compensate for turbo-boost being enabled). Nevertheless, Hamburg's own comparison with `Curve25519` suggests that a fair comparison between our constant-time implementations and his would be close.

Acknowledgements The authors would like to thank Joppe W. Bos for independently benchmarking our code on his computer. The second author acknowledges that the notes of Appendix A grew from discussions with Joppe W. Bos on an earlier work [10].

References

1. Reza Azarderakhsh and Koray Karabina. A new double point multiplication algorithm and its application to binary elliptic curves with endomorphisms. *To appear in IEEE Transactions on Computers*, 2013. URL: `http://cacr.uwaterloo.ca/techreports/2012/cacr2012-24.pdf`.
2. László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
3. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
4. Daniel J. Bernstein. Differential addition chains. `http://cr.yp.to/ecdh/diffchain-20060219.pdf`, February 2006.
5. Daniel J. Bernstein and Tanja Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems, accessed 28 September, 2013. `http://bench.cr.yp.to`.
6. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. 2013. `http://eprint.iacr.org/`.
7. Daniel J. Bernstein and Tanja Lange. Explicit-formulas database, accessed 10 October, 2013. `http://www.hyperelliptic.org/EFD/`.
8. Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, accessed 16 October, 2013. `http://safecurves.cr.yp.to`.
9. Daniel J Bernstein, Tanja Lange, and Peter Schwabe. On the correct use of the negation map in the Pollard rho method. In *Public Key Cryptography–PKC 2011*, pages 128–146. Springer, 2011.

10. Joppe W. Bos, Craig Costello, Huseyin Hisil, and Kristin Lauter. Fast cryptography in genus 2. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 194–210. Springer Berlin Heidelberg, 2013.
11. ECC Brainpool. ECC Brainpool standard curves and curve generation, October, 2005. `www.secg.org/collateral/sec2_final.pdf`.
12. Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. Fault attack on elliptic curve Montgomery ladder implementation. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *FDTC*, pages 92–98. IEEE Computer Society, 2008.
13. Gerhard Frey, Michael Müller, and H-G Rück. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *Information Theory, IEEE Transactions on*, 45(5):1717–1719, 1999.
14. Steven D. Galbraith, Xibin Lin, and Michael Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011.
15. Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
16. Pierrick Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symb. Comput.*, 44(12):1690–1702, 2009.
17. Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.
18. Mike Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/`.
19. Michael Kaib. The Gauss lattice basis reduction algorithm succeeds with any norm. In Lother Budach, editor, *Fundamentals of Computation Theory*, volume 529 of *Lecture Notes in Computer Science*, pages 275–286. Springer Berlin Heidelberg, 1991.
20. Neal Koblitz. CM-curves with good cryptographic properties. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287. Springer, 1991.
21. Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.
22. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
23. Peter L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains. *Available at ftp.cwi.nl:/pub/pmontgom/lucas.ps.gz*, 349, 1992.
24. Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *Information Theory, IEEE Transactions on*, 24(1):106–110, 1978.
25. John M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.
26. Certicom Research. Standards for Efficient Cryptography 2 (SEC 2), January, 2010. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`.
27. Takakazu Satoh and Kiyomichi Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curve. *Commentarii Mathematici Universitatis Sancti Pauli*, 47(1):81–92, 1998.
28. René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.
29. Igor Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of Computation*, 67(221):353–356, 1998.
30. Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, 1999.
31. Nigel P. Smart. How secure are elliptic curves over composite extension fields? In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 30–39. Springer, 2001.
32. Benjamin Smith. Families of fast elliptic curves from $\mathbb{Q}$-curves. In *ASIACRYPT*, 2013, *to appear*. `http://eprint.iacr.org/2013/312`.
33. Jerome A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.
34. Martijn Stam. *Speeding up subgroup cryptosystems*. PhD thesis, Technische Universiteit Eindhoven, 2003.
35. Ernst G. Straus. Addition chains of vectors. *American Mathematical Monthly*, 70(806-808):16, 1964.
36. Michael J. Wiener and Robert J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In Stafford E. Tavares and Henk Meijer, editors, *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 1998.

A Efficient arithmetic in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$

We access lower level integer arithmetic for efficient addition, subtraction, multiplication and squaring operations in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ where $p = 2^{127} - 1$, see §2. At this level, elements of $\mathbb{F}_p$ are represented by integer values in the usual way, with the exception that the representation of 0 is not unique: 0 is allowed to be represented in "semi-reduced" form by the integers 0 and p. Semi-reduced values can be used in any chain of operations without causing an exception, since all of our algorithms are designed to accept inputs and produce outputs in the interval $[0, p]$. The implementor should reduce each output into the range $[0, p)$ at the very end of the target computation, in order to satisfy unique representation field elements. This type of arithmetic has already been exploited in earlier works, such as [10], but a thorough exposition has not yet appeared.

We will be frequently referring back to the divisibility lemma of integers.

Lemma 5. *Let $u, v \in \mathbb{Z}$ with $v > 0$. Then there* exist unique $q, r \in \mathbb{Z}$ *such that $u = r + qv$ and $0 \leq r < v$. In particular, $q = \lfloor u/v \rfloor$ and $r = u - \lfloor u/v \rfloor v$ where $\lfloor . \rfloor$ is the floor function.*

In what follows, the "mod 2^{128}" and "mod 2^{256}" operators, are included (even though they are often unnecessary) to reinforce the fact that all arithmetic operations are being performed on an unsigned integer arithmetic circuit over a 128-bit data type. We let k_i denote the i^{th} significant bit of an integer k and use $(k_i, \ldots, k_j)$ to denote the integer formed by the bit-string that starts with k_i, continues with bits in order of increasing significance, and ends with k_j (with $0 \leq i \leq j \leq 127$). This is a Big-endian type representation. Although it is possible to provide much shorter arguments for sections A.1-5, we prefer to keep the notes in longer format in order to assist easier verification.

It should be noted that all of the techniques in this section avoid branching. This is highly desirable for an efficient implementation, especially on an architecture with pipelining capability.

A.1 Semi-reduced addition modulo p

The operation $(a + b) \bmod p$ is replaced by Algorithm 1.

Input: $a, b \in \mathbb{Z}$ such that $0 \leq a, b \leq p$.
Output: $f \in \mathbb{Z}$ such that $f \equiv (a + b) \pmod{p}$ and $0 \leq f \leq p$.
1 $c := (a + b) \bmod 2^{128}$;
2 $d := (c_0, c_1, \ldots, c_{126})$, $e := (c_{127})$;
3 $f := (d + e) \bmod 2^{128}$;
4 **return** f;

Algorithm 1: Semi-reduced addition modulo p

- **Line-1:** Notice that $0 \leq c = a + b \leq 2p < 2^{128}$.
- **Line-2:** Use Lemma 5 to write $c = d + 2^{127}e$ for integers $0 \leq d < 2^{127}$ and e. There are two cases to investigate:

- Case 1: Assume that $a + b \leq p$. The bounds on c and d imply that $\lfloor 0/2^{127} \rfloor \leq \lfloor c/2^{127} \rfloor = \lfloor (d + 2^{127}e)/2^{127} \rfloor = \lfloor d/2^{127} \rfloor + \lfloor 2^{127}e/2^{127} \rfloor = e \leq \lfloor p/2^{127} \rfloor$, so $e = 0$. Thus $a + b \equiv d + 2^{127}e \equiv d + 2^{127} \cdot 0 \equiv d + 0 \equiv \underline{d + e} \pmod{p}$.
- Case 2: Assume that $a + b > p$. Then $p < c \leq 2p$. The bounds on c and d imply that $\lfloor (p+1)/2^{127} \rfloor \leq e \leq \lfloor 2p/2^{127} \rfloor$, so $e = 1$. The bounds on c also imply that $p - 2^{127} < c - 2^{127} \leq 2p - 2^{127}$ and we have $d = c - 2^{127}e = c - 2^{127}$, so $0 \leq d < p$. Thus $a + b \equiv d + 2^{127}e \equiv d + 2^{127} \cdot 1 \equiv d + 1 \equiv \underline{d + e} \pmod{p}$.

– **Line-3:** A semi-reduced output is given by $f := (d+e) \bmod 2^{128}$, observing that $0 \leq f \leq p$.

A.2 Semi-reduced subtraction modulo p

The operation $(a - b) \bmod p$ is replaced by Algorithm 2.

Input: $a, b \in \mathbb{Z}$ such that $0 \leq a, b \leq p$.
Output: $f \in \mathbb{Z}$ such that $f \equiv (a - b) \pmod{p}$ and $0 \leq f \leq p$.
1 $c := (a - b) \bmod 2^{128}$;
2 $d := (c_0, c_1, \ldots, c_{126})$, $e := (c_{127})$;
3 $f := (d - e) \bmod 2^{128}$;
4 **return** f;

Algorithm 2: Semi-reduced subtraction modulo p

– **Line-1:** Notice that $0 \leq c < 2^{128}$.

– **Line-2:** Use Lemma 5 to write $c = d + 2^{127}e$ for integers $0 \leq d < 2^{127}$ and e. There are two cases to investigate:
- Case 1: Assume that $a \geq b$. Then $0 \leq c = a - b \leq p$. The bounds on c and d imply that $\lfloor 0/2^{127} \rfloor \leq \lfloor c/2^{127} \rfloor = \lfloor (d + 2^{127}e)/2^{127} \rfloor = e \leq \lfloor p/2^{127} \rfloor$, so $e = 0$. Thus $a - b \equiv d + 2^{127}e \equiv \underline{d - e} \pmod{p}$.
- Case 2: Assume that $a < b$. Then $c = 2^{128} + a - b$ and $-p \leq a - b < 0$. So, $2^{127} < c < 2^{128}$. The bounds on c and d imply that $\lfloor (2^{127} + 1)/2^{127} \rfloor \leq e \leq \lfloor (2^{128} - 1)/2^{127} \rfloor$, so $e = 1$. The bounds on c also imply that $2^{127} - 2^{127} < c - 2^{127} < 2^{128} - 2^{127}$, and we have $d = c - 2^{127}e = c - 2^{127}$. So, $0 < d \leq p$ and $d \geq e$. Thus $a - b \equiv (2^{128} + a - b) - 2^{128} \equiv c - 2^{128} \equiv d + 2^{127}e - 2^{128} \equiv \underline{d - e} \pmod{p}$.

Line-3: A semi-reduced output is given by $f := (d - e) \bmod 2^{128}$, observing that $0 \leq f \leq p$.

A.3 Semi-reduced multiplication modulo p

The operation $(ab) \bmod p$ is replaced by Algorithm 3.

– **Line-1:** Notice that $0 \leq c = ab \leq p^2 < 2^{256}$.

– **Line-2:** Use Lemma 5 to write $c = d + 2^{127}e$ for integers $0 \leq d < 2^{127}$ and e. The bounds on c and d imply that $\lfloor 0/2^{127} \rfloor \leq \lfloor c/2^{127} \rfloor = \lfloor (d + 2^{127}e)/2^{127} \rfloor = e \leq \lfloor p^2/2^{127} \rfloor$, so $0 \leq e < p$.

– **Line-3:** Noting that $ab \equiv d + 2^{127}e \equiv d + (2^{127} - 1)e + e \equiv d + pe + e \equiv d + e \pmod{p}$, that $0 \leq d, e \leq p$, and that $0 \leq d + e \leq 2p$, a semi-reduced output is obtained by Algorithm 1 applied on the operands d and e.

Input: $a, b \in \mathbb{Z}$ such that $0 \leq a, b \leq p$.

Output: $f \in \mathbb{Z}$ such that $f \equiv (ab) \pmod{p}$ and $0 \leq f \leq p$.

1 $c := (ab) \bmod 2^{256}$;
2 $d := (c_0, c_1, \ldots, c_{126})$, $e := (c_{127}, c_{128}, \ldots, c_{253})$;
3 $f :=$ semi-add(d, e);
4 **return** f;

Algorithm 3: Semi-reduced multiplication modulo p

A.4 Lazy semi-reduction modulo p following a double-word addition

The lazy reduction $(a\hat{b} + \hat{a}b) \bmod p$ is replaced by Algorithm 4.

Input: $a, \hat{a}, b, \hat{b} \in \mathbb{Z}$ such that $0 \leq a, \hat{a}, b, \hat{b} \leq p$.

Output: $h \in \mathbb{Z}$ such that $h \equiv (a\hat{b} + \hat{a}b) \pmod{p}$ and $0 \leq h \leq p$.

1 $c := (a\hat{b} + \hat{a}b) \bmod 2^{256}$;
2 $d := (c_0, c_1, \ldots, c_{126})$, $e := (c_{127}, c_{128}, \ldots, c_{253})$, $f := (c_{254})$;
3 $g := (e + f) \bmod 2^{128}$;
4 $h :=$ semi-add(d, g);
5 **return** h;

Algorithm 4: Lazy semi-reduction modulo p following a double-word addition

- **Line-1:** Notice that $0 \leq c = a\hat{b} + \hat{a}b \leq 2p^2 < 2^{256}$.
- **Line-2:** Use Lemma 5 to write $c = d + 2^{127}(e + 2^{127}f)$ for integers $0 \leq d, e < 2^{127}$ and f. There are two cases to investigate:
  - Case 1: Assume that $a\hat{b} + \hat{a}b < (p+1)^2$. Then $0 \leq c < (p+1)^2$. The bounds on c, d, and e imply that $\lfloor 0/(2^{127})^2 \rfloor \leq \lfloor c/(2^{127})^2 \rfloor = \lfloor (d + 2^{127}e + (2^{127})^2 f)/(2^{127})^2 \rfloor = f \leq \lfloor ((p+1)^2 - 1)/(2^{127})^2 \rfloor$, so $f = 0$. Thus $a\hat{b} + \hat{a}b \equiv d + 2^{127}(e + 2^{127}f) \equiv d + 2^{127}(e + 2^{127} \cdot 0) \equiv d + 2^{127}(e + 0) \equiv \underline{d + 2^{127}(e + f)} \pmod{p}$ and $0 \leq e + f < p$.
  - Case 2: Assume that $a\hat{b} + \hat{a}b \geq (p+1)^2$. Then $(p+1)^2 \leq c \leq 2p^2$. The bounds on c, d, and e imply that $\lfloor (p+1)^2/(2^{127})^2 \rfloor \leq f \leq \lfloor 2p^2/(2^{127})^2 \rfloor$, so $f = 1$. The bounds on c also imply that $(p+1)^2 - (2^{127})^2 \leq c - (2^{127})^2 \leq 2p^2 - (2^{127})^2$, and we have $d + 2^{127}e = c - (2^{127})^2 f = c - (2^{127})^2$. So, $0 \leq d + 2^{127}e \leq ((p-1)^2 - 2)$. The bounds on $d + 2^{127}e$ imply that $\lfloor 0/2^{127} \rfloor \leq \lfloor (d + 2^{127}e)/2^{127} \rfloor \leq \lfloor ((p-1)^2 - 2)/2^{127} \rfloor$, so $0 \leq e < (p-2)$. Thus $a\hat{b} + \hat{a}b \equiv d + 2^{127}(e + 2^{127}f) \equiv d + 2^{127}(e + 2^{127} \cdot 1) \equiv d + 2^{127}(e + 1) \equiv \underline{d + 2^{127}(e + f)} \pmod{p}$ and $0 \leq e + f < p$.
- **Line-3:** Set $g := (e + f) \bmod 2^{128}$ where $0 \leq g \leq p$.
- **Line-4:** Noting that $d + 2^{127}(e + 2^{127}f) \equiv d + 2^{127}g \equiv d + g \pmod{p}$, that $0 \leq d, g \leq p$, and that $0 \leq d + g \leq 2p$, a semi-reduced output is obtained by Algorithm 1 applied on the operands d and g.

A.5 Lazy semi-reduction modulo p following a double-word subtraction

The lazy reduction $(ab - \hat{a}\hat{b}) \bmod p$ is replaced by Algorithm 5.

Input: $a, \hat{a}, b, \hat{b} \in \mathbb{Z}$ such that $0 \leq a, \hat{a}, b, \hat{b} \leq p$.

Output: $h \in \mathbb{Z}$ such that $h \equiv (ab - \hat{a}\hat{b}) \pmod{p}$ and $0 \leq h \leq p$.

1 $c := (ab - \hat{a}\hat{b}) \bmod 2^{256}$;
2 $d := (c_0, c_1, \ldots, c_{126})$, $e := (c_{127}, c_{128}, \ldots, c_{253})$, $f := (c_{254})$, $g := (c_{255})$;
3 $h := (e - f) \bmod 2^{128}$;
4 $j :=$ semi-add(d, g);
5 **return** j;

Algorithm 5: Lazy semi-reduction modulo p following a double-word subtraction

- **Line-1:** Notice that $0 \leq c < 2^{256}$.
- **Line-2:** Use Lemma 5 to write $c = d + 2^{127}(e + 2^{127}(f + 2g))$ for integers $0 \leq d, e < 2^{127}$, $0 \leq f < 2$, and g. There are two cases to investigate:
  - Case 1: Assume that $ab \geq \hat{a}\hat{b}$. Then $0 \leq c = ab - \hat{a}\hat{b} \leq p^2$. The bounds on c, d, e and f imply that $\lfloor 0/(2^{127})^2 \rfloor \leq \lfloor c/(2^{127})^2 \rfloor = \lfloor (d + 2^{127}e + (2^{127})^2(f + 2g))/(2^{127})^2 \rfloor = f + 2g \leq \lfloor p^2/(2^{127})^2 \rfloor$; that is $f + 2g = 0$. So, $f = g = 0$. Thus $d + 2^{127}(e + 2^{127}(f + 2g)) \equiv d + 2^{127}(e + 2^{127} \cdot 0) \equiv d + 2^{127}(e - 0) \equiv \underline{d + 2^{127}(e - f)} \pmod{p}$.
  - Case 2: Assume that $ab < \hat{a}\hat{b}$. Then $c = 2^{256} + ab - \hat{a}\hat{b}$ and $-p^2 \leq ab - \hat{a}\hat{b} < 0$. So, $2^{256} - p^2 \leq c < 2^{256}$. As in the previous case, the bounds on c, d, e and f imply that $\lfloor (2^{256} - p^2)/(2^{127})^2 \rfloor \leq f + 2g \leq \lfloor (2^{256} - 1)/(2^{127})^2 \rfloor$, so $f + 2g = 3$ and $f = g = 1$. The bounds on c also imply that $2^{256} - p^2 - 3(2^{127})^2 = 2^{128} - 1 \leq c - 3(2^{127})^2 < 2^{256} - 3(2^{127})^2$ and we also have $d + 2^{127}e = c - (2^{127})^2(f + 2g) = c - 3(2^{127})^2$. So, $2^{128} - 1 \leq d + 2^{127}e$. The bounds on $d + 2^{127}e$ imply that $\lfloor (2^{128} - 1)/2^{127} \rfloor \leq \lfloor (d + 2^{127}e)/2^{127} \rfloor < \lfloor (2^{256} - 3(2^{127})^2)/2^{127} \rfloor = 2^{127}$, so $1 \leq e < 2^{127}$ and $e \geq f$. Thus

$$\begin{aligned} ab - \hat{a}\hat{b} &\equiv (2^{256} + ab - \hat{a}\hat{b}) - 2^{256} \;=\; c - 2^{256} \;\equiv\; c - 4 \\ &\equiv d + 2^{127}(e + 2^{127}(f + 2g)) - 4 \\ &\equiv d + 2^{127}(e + 2^{127}(1 + 2 \cdot 1)) - 4 \\ &\equiv d + 2^{127}(e - 1) \;\equiv\; \underline{d + 2^{127}(e - f)} \pmod{p}. \end{aligned}$$

- **Line-3:** Set $h := (e - f) \bmod 2^{128}$ where $0 \leq h \leq p$.
- **Line-4:** Noting that $d + 2^{127}(e + 2^{127}(f + 2g)) \equiv d + 2^{127}h \equiv d + h \pmod{p}$, that $0 \leq d, h \leq p$, and that $0 \leq d + h \leq 2p$, a semi-reduced output is obtained by Algorithm 1 applied on the operands d and h.

A.6 Addition and subtraction in $\mathbb{F}_{p^2}$

Let $a, \hat{a}, b, \hat{b} \in \mathbb{Z}$ and $0 \leq a, \hat{a}, b, \hat{b} \leq p$. We use the obvious method which computes $(a + \hat{a}i) + (b + \hat{b}i)$ as $((a + b) \bmod p) + ((\hat{a} + \hat{b}) \bmod p)i$. Both modular additions are replaced by Algorithm 1. Analogous comments apply for the case of subtraction which uses Algorithm 2.

A.7 Multiplication in $\mathbb{F}_{p^2}$

Let $a, \hat{a}, b, \hat{b} \in \mathbb{Z}$ and $0 \leq a, \hat{a}, b, \hat{b} \leq p$. On the target architecture, we experienced the best performance for computing $(a + \hat{a}i)(b + \hat{b}i)$ by coupling a Karatsuba-based operation scheduling

with two lazy reductions. This computes the product as

$$\Big((ab - \hat{a}\hat{b}) \bmod p\Big) + \Big(\big[(a+\hat{a})(b+\hat{b}) - ab - \hat{a}\hat{b}\big] \bmod p\Big)i.$$

The routine starts with two integer additions $t_0 := a + \hat{a}$ and $t_1 := b + \hat{b}$ satisfying $0 \leq t_0, t_1 < (2^{128} - 1)$. The routine continues with the 3 integer multiplications $t_2 := t_0 t_1$, $t_3 := ab$ and $t_4 := \hat{a}\hat{b}$ satisfying $0 \leq t_2 \leq (2^{128} - 2)^2 < 2^{256}$ and $0 \leq t_3, t_4 \leq (2^{127} - 1)^2 < 2^{254}$. Since $t_2 > t_3$ and $(t_2 - t_3) > t_4$, the integer value $t_5 := (t_2 - t_3) - t_4$ is positive and satisfies both $0 \leq t_5 \leq 2p^2 < 2^{255}$ and $t_5 = a\hat{b} + \hat{a}b$. The reduction of t_5 is performed as in Algorithm 4. The reduction of $t_6 := (t_3 - t_4) \bmod 2^{256}$ is performed as in Algorithm 5.

A.8 Squaring in $\mathbb{F}_{p^2}$

Let $a, \hat{a}, b, \hat{b} \in \mathbb{Z}$ and $0 \leq a, \hat{a}, b, \hat{b} \leq p$. On the target architecture, we experienced that a lazy semi-reduction strategy gives the same timings as the (non-lazy) semi-reduction strategy for computing $(a + \hat{a}i)^2 = \big((a - \hat{a})(a + \hat{a})\big) + \big(2a\hat{a}\big)i$.

A.9 Other operations in $\mathbb{F}_{p^2}$

Many other $\mathbb{F}_{p^2}$ operations can be efficiently performed by $\mathbb{F}_p$ arithmetic only. For instance, negation can be performed as $-a = (0 - a) + (0 - \hat{a})i$, p-th powering as $a^p = a + (0 - \hat{a})i$, and inversion as $a^{-1} = a(a^2 + \hat{a}^2)^{p-2} + (0 - \hat{a}(a^2 + \hat{a}^2)^{p-2})i$ – our $\mathbb{F}_{p^2}$-inversion implementation incurs 128 $\mathbb{F}_p$-squarings, 12 $\mathbb{F}_p$-multiplications and 2 $\mathbb{F}_p$-additions/subtractions.

B How was this curve chosen?

The particular curve-twist pair described in Section 2 was chosen from the family of degree-2 $\mathbb{Q}$-curve reductions described in [32, Proposition 3]. This family (over $\mathbb{F}_{p^2}$) is parameterised by a free parameter s in $\mathbb{F}_p$, and covers at least $p - 3$ different isomorphism classes for any given p. Every curve in the family is equipped with an efficient endomorphism, and the arithmetic properties of the family make it possible to find twist-secure curves in the family.

We chose our curve $\mathcal{E}$ from this family as follows. First, we fixed $p = 2^{127} - 1$; a Mersenne prime, this p facilitates very fast modular arithmetic. Next, we chose a tiny nonsquare to define $\mathbb{F}_{p^2} = \mathbb{F}_p(\sqrt{-1})$; this makes $\mathbb{F}_{p^2}$-arithmetic slightly easier, and our formulæ much simpler.

We then needed to find a twist-secure curve $\mathcal{E}$ in the family with a rational transformation into Montgomery form. This means that $\mathcal{E}$ had to have order divisible by 4, so optimal group order for $\mathcal{E}$ was therefore $\#\mathcal{E} = 4N$, with N prime; it follows (from $p^2 \equiv 1 \bmod 4$) that the optimal group order for $\mathcal{E}'$ is then $\#\mathcal{E}' = 8N'$, with N' is prime. Our curve search therefore discarded parameter values corresponding to pairs that did not have these optimal cofactors.

To optimise performance, we searched for parameter values s in $\mathbb{F}_p$ that gave rise to Montgomery representations whose constant A in $\mathbb{F}_{p^2}$ had "small" coefficients: that is, for $A = A_0 + A_1 i$ with A_0 and A_1 in $\mathbb{F}_p$, we wanted the integer representation of A_0 and A_1 to be small. However, the construction in [32, §5] requires the constant A to satisfy $8/A^2 = 1 + si$, from which it plainly follows that

$$A_0^4 + 2A_0^2A_1^2 + A_1^4 + 8(A_1^2 - A_0^2) = 0 \tag{7}$$

in $\mathbb{F}_p$. This means that, for any fixed value of A_1 in $\mathbb{F}_p$, there are at most four corresponding possibilities for $A_0 \in \mathbb{F}_p$, which can be found as the roots of a quartic equation over $\mathbb{F}_p$ (and vice versa). Keeping in mind that we need to search over many values of A to find a twist-secure pair $(\mathcal{E}, \mathcal{E}')$, the upshot of (7) is that we cannot expect to find a cryptographically suitable curve with both A_0 and A_1 being small. The $\mathbb{F}_{p^2}$-arithmetic described in Appendix A places no preference on which of these two coefficients is small, so we flipped a coin and conducted our search choosing A_1 to be small[14]: we restricted our search to A_1 values with integer representations less than 2^{32}, so that A_1 would only occupy one word on 32-bit and 64-bit platforms. Our search also prioritised A_1 whose integer representations had low signed Hamming-weight, in the hope that multiplication by A_1 might possibly be faster when computed via sequence of additions and shifts. When the integer representation of A_1 had a signed Hamming-weight of 1,2 or 3, we did not find any curve-twist pairs with optimal cofactors; however, when further loosening the search to include those A_1 with a corresponding signed Hamming-weight of 4, we found 10 such pairs. Among these, there are 3 whose A_1 had an integer representation of precisely 32 bits – the curve in Section 2 corresponds to the *smallest* such A_1. Although the low signed Hamming-weight of A_1 did not end up being useful in our implementation, its small size did give rise to a minor but noticeable speedup.

The takeaway message is that the construction in [32, §5] is flexible enough to find a vast number of twist-secure curves over any quadratic extension field, for which all of the discussion in this paper can either be directly applied or easily modified (regardless of how the parameter search is designed). Such curve-twist pairs can be readily found in a *verifiably random* manner, following, for instance, the method described in [11, §5].

For example: let $\mathcal{H}$ be a hash function, and let $\pi(i)$ be the string consisting of the first i digits of π (without the decimal), i.e. $\pi(1) =$ "3", $\pi(2) =$ "31", $\pi(3) =$ "314", and so on. Starting with $i = 1$, we conducted some other searches by taking $s := \mathcal{H}(\pi(i)) \bmod p$ until we found the first curve-twist pair $\mathcal{E}$ and $\mathcal{E}'$ with optimal cofactors 4 and 8. With $\mathcal{H} = \mathrm{SHA} - 1$, the first pair was found at $i = 19244$; and, with $\mathcal{H} = \mathrm{SHA} - 256$, the first pair was found at $i = 41004$. Our discussion and software can easily be modified to handle such cases.

C Bernstein's uniform two-dimensional differential addition chain

Algorithm 6 is a concrete adaptation of Bernstein's addition chain [4, §4] to our curve $\mathcal{E}$, following the multiscalar decomposition described in Section 4. We use the usual formulæ (see [7]) for pseudo-doubling, pseudo-addition, and for the combination of the two, writing their inputs and outputs as follows. For pseudo-doubling, we write

$$x([2]R) = \mathtt{DBL}(x(R)) \ ;$$

for pseudo-addition, we write

$$x(T \oplus U) = \mathtt{ADD}(x(T), x(U), x(T \ominus U)) \ ;$$

and for their combined computation, we write

$$x([2]R), x(R \oplus S) = \mathtt{DBLADD}(x(R), x(S), x(R \ominus S)) \ .$$

[14] We also included the restriction that the integer representation of A_1 be congruent to 2 modulo 4, since it is actually the constant $(A + 2)/4$ which is used in Montgomery's formulæ [22, p. 261].

The main iterations in the chain compute a DBLADD alongside a standalone ADD, so we denote combined pseudo-doubling and pseudo-addition by

$$x([2]R), x(R \oplus S), x(T \oplus U) = \texttt{DBLDBLADD}(x(R), x(S), x(R \ominus S), x(T), x(U), x(T \ominus U)) \ .$$

Input: $a, b \in \mathbb{Z}^+$ (both 128 bits - see Theorem 3), and
$x(P), x(Q), x(Q \ominus P), x(Q \oplus P)$ (four affine elements on the x-line, where $Q = \psi(P)$ on $\mathcal{E}$)
Output: $x([a]P \oplus [b]\psi(P))$

1 initialization: $(a)_2 = (a_{127}, \ldots, a_0) \in \{0,1\}^{128}$, $(b)_2 = (b_{127}, \ldots, b_0) \in \{0,1\}^{128}$.
2 $z_0, z_1, z_2, z_3 \leftarrow ()$. /* z's start as empty bit-sequences */
3 **if** $a_0 \oplus b_0 = 1$ **then** $ind_{final} \leftarrow 2$ **else** $ind_{final} \leftarrow\sim b_0$ **end**
4 $add_{first} \leftarrow a_0$. /* $add_{first} \in \{0,1\}$ */
5 **for** $i \leftarrow 0$ **to** 126 **do** /* $z_0, \ldots z_3 \in \{0,1\}^{127}$ at end of loop */
6 $\quad \hat{a} = a_i \oplus a_{i+1}, \quad \hat{b} = b_i \oplus b_{i+1}, \quad \hat{ab} = \hat{a} \oplus \hat{b}$.
7 $\quad z_0 \leftarrow \hat{ab}||z_0, \quad z_1 \leftarrow \hat{a}||z_1, \quad z_2 \leftarrow (a_{i+1} \oplus b_{i+1})||z_2, \quad z_3 \leftarrow add_{first}||z_3$.
8 $\quad add_{first} \leftarrow \hat{a} \oplus ((\sim \hat{ab}) \otimes add_{first})$.
9 **end**
10 $T_0 = x(Q \oplus P), \quad T_1 = \texttt{DBL}(T_0)$
11 **if** $add_{first} = 1$ **then** $T_2 \leftarrow \texttt{ADD}(x(Q), T_0, x(P))$ **else** $T_2 \leftarrow \texttt{ADD}(x(P), T_0, x(Q))$ **end**
12 **for** $i \leftarrow 0$ **to** 126 **do** /* main loop */
13 $\quad$ **switch** $[z_{0,i}, z_{1,i}, z_{2,i}, z_{3,i}]$ **do** /* $z_j = (z_{j,0}, \ldots, z_{j,126}) \in \{0,1\}^{127}$, $j = 0, \ldots 3$ */
14 $\quad\quad$ **case** $[0,0,0,0]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_1, T_0, x(Q \oplus P), T_2, T_1, x(Q))$.
15 $\quad\quad$ **case** $[0,0,0,1]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_1, T_0, x(Q \oplus P), T_2, T_1, x(P))$.
16 $\quad\quad$ **case** $[0,0,1,0]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_1, T_0, x(Q \ominus P), T_2, T_1, x(Q))$.
17 $\quad\quad$ **case** $[0,0,1,1]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_1, T_0, x(Q \ominus P), T_2, T_1, x(P))$.
18 $\quad\quad$ **case** $[0,1,0,0]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_0, T_1, x(Q \oplus P), T_2, T_0, x(Q))$.
19 $\quad\quad$ **case** $[0,1,0,1]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_0, T_1, x(Q \oplus P), T_2, T_0, x(P))$.
20 $\quad\quad$ **case** $[0,1,1,0]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_0, T_1, x(Q \ominus P), T_2, T_0, x(Q))$.
21 $\quad\quad$ **case** $[0,1,1,1]$: $T_1, T_0, T_2 \leftarrow \texttt{DBLADDADD}(T_0, T_1, x(Q \ominus P), T_2, T_0, x(P))$.
22 $\quad\quad$ **case** $[1,0,0,0]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_1, x(Q), T_0, T_1, x(Q \oplus P))$.
23 $\quad\quad$ **case** $[1,0,0,1]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_0, x(P), T_0, T_1, x(Q \oplus P))$.
24 $\quad\quad$ **case** $[1,0,1,0]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_1, x(Q), T_0, T_1, x(Q \ominus P))$.
25 $\quad\quad$ **case** $[1,0,1,1]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_0, x(P), T_0, T_1, x(Q \ominus P))$.
26 $\quad\quad$ **case** $[1,1,0,0]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_0, x(Q), T_0, T_1, x(Q \oplus P))$.
27 $\quad\quad$ **case** $[1,1,0,1]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_1, x(P), T_0, T_1, x(Q \oplus P))$.
28 $\quad\quad$ **case** $[1,1,1,0]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_0, x(Q), T_0, T_1, x(Q \ominus P))$.
29 $\quad\quad$ **case** $[1,1,1,1]$: $T_1, T_2, T_0 \leftarrow \texttt{DBLADDADD}(T_2, T_1, x(P), T_0, T_1, x(Q \ominus P))$.
30 $\quad$ **end**
31 **end**
32 **return** $T_{ind_{final}}$.

Algorithm 6: Bernstein's uniform 2-D chain, tailored to the curve in Section 2.

The chain is determined in its entirety using only bit operations before any arithmetic is done on the x-line (the symbols $\oplus$ and $\ominus$ denote bit operations in Lines 3-9 of Algorithm 2, but curve operations everywhere else).

Observe that the associated differences in pseudo-additions are always one of the four affine input points $x(P)$, $x(Q)$, $x(P \ominus Q)$, or $x(P \oplus Q)$. On the other hand, the three running values $T_0 = (X_0 \colon Z_0)$, $T_1 = (X_1 \colon Z_1)$ and $T_2 = (X_2 \colon Z_2)$ are projective. Thus, the final step (which chooses one of the three running values to output) will involve an $\mathbb{F}_{p^2}$-inversion to output $T_{ind_{final}}$ in affine form.