Fast cube roots in $\mathbb{F}_{p^2}$ via the algebraic torus

Youssef El Housni

Consensys, Linea, New York, United States
youssef.elhousni@consensys.net

Abstract. Computing cube roots in quadratic extensions of finite fields is a subroutine that arises in elliptic-curve point decompression, hash-to-curve and isogeny-based protocols. While the factorization $p^2 - 1 = (p-1)(p+1)$ suggests a known subgroup decomposition, computing a cube root via separate operations in the two subgroups does not beat a direct $\mathbb{F}_{p^2}$ exponentiation in practice. We propose a carefully engineered algorithm that reduces the $\mathbb{F}_{p^2}$ cube root to a single $\mathbb{F}_p$ addition chain and a single Lucas sequence in the algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$. The concrete formulas apply to primes $p \equiv 1 \pmod 3$ with $v_3(p-1) \leq 2$, which includes all six primes considered here. We prove correctness in all residuosity cases and implement the algorithm in Go. Benchmarks on six primes spanning pairing-based and isogeny-based cryptography show 1.6–2.3× speed-ups over direct (addition chain) exponentiations in $\mathbb{F}_{p^2}$.

Keywords: cube roots · finite fields · quadratic extensions · algebraic torus · Lucas sequences · elliptic curves · isogenies

1 Introduction

Let p be a prime > 3 and let $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)$ for some non-square $\beta \in \mathbb{F}_p$. Computing $z \in \mathbb{F}_{p^2}$ such that $z^3 = x$ for a given $x \in \mathbb{F}_{p^2}$ is a fundamental arithmetic operation that appears in several cryptographic contexts:

- **Hash-to-curve.** Icart's deterministic encoding [25] uses an $\mathbb{F}_p$ cube root to map field elements to curves $y^2 = x^3 + ax + b$. Koshelev [26] generalizes this to curves $y^2 = x^3 + b$ over $\mathbb{F}_p$ relevant to pairing-based cryptography, where the cube root is in $\mathbb{F}_{p^2}$. In that work, Koshelev explicitly states that he "does not know how to express $\sqrt[3]{\cdot} \in \mathbb{F}_{p^2}$ through a few $\mathbb{F}_p$-roots," leaving it as an open problem. Our method resolves this question.
- **Point decompression.** On j-invariant 0 curves $y^2 = x^3 + b$ defined over highly 2-adic fields (as used in SNARKs e.g. BLS12-377 [13] used in Linea zkEVM), storing y and recovering $x = (y^2 - b)^{1/3}$ via a cube root is preferable to the standard approach of storing x and recovering y via a square root, since the latter requires a Tonelli–Shanks square root, which involves an exponentiation followed by a discrete-log correction in the 2-Sylow subgroup of size 2^v, while v is large for SNARK-friendly primes (e.g. $v = 46$ for BLS12-377). Faster variants improve this correction step [35,33], but still have to

handle the high 2-adic component. For pairing-friendly curves of embedding degree 12 (e.g. the BLS12 [5], BN [6] families) the $\mathbb{G}_2$ subgroup lives on a curve over $\mathbb{F}_{p^2}$, so the cube root is in $\mathbb{F}_{p^2}$.
To take this further, Koshelev [27] uses quotient varieties to show that two points can be batch-decompressed using a single cube root, amortizing the cost further. For points living in $\mathbb{G}_2$, one needs efficient cube roots in $\mathbb{F}_{p^2}$.

- **3-isogeny walks.** Three-isogeny walks appear in hash functions and verifiable delay functions, as well as key-encapsulation mechanisms and proofs of isogeny knowledge [19]. In these constructions, the $\mathbb{F}_{p^2}$ cube root is the *bottleneck operation* at every step of the walk. This work speeds up that operation by more than $2\times$.
- **Algebraic VDF analysis.** Biryukov et al. [9] attack algebraic verifiable-delay functions by parallelizing root extraction; for the Sloth++ [11] variant, they exploit that square roots in $\mathbb{F}_{p^2}$ reduce to square roots in $\mathbb{F}_p$. They explicitly note that no analogous reduction is apparent for cube roots (or higher roots) in $\mathbb{F}_{p^2}$. Our work gives such a base-field reduction for cube roots under the congruence conditions considered here, and may therefore be useful both for analyzing cubic-root-based VDF candidates and for choosing parameters that avoid unintended parallel speedups.
- **Solving depressed cubics over $\mathbb{F}_p$.** Cardano's formula solves $t^3+at+b=0$ via the resolvent quadratic $z^2+bz-a^3/27=0$, whose roots involve $\sqrt{-\Delta/108}$ with $\Delta=-4a^3-27b^2$. When Δ is a non-square in $\mathbb{F}_p$, this square root lies in the quadratic extension $\mathbb{F}_{p^2}=\mathbb{F}_p(\sqrt{-\Delta/108})$, and obtaining t then requires extracting a cube root of an element of $\mathbb{F}_{p^2}$, exactly the operation this paper accelerates. A concrete cryptographic instance is the *y-increment* map-to-curve SNARK circuit of [20]: on a $j\neq 0$ curve $y^2=x^3+ax+b$, recovering x from a candidate y-coordinate requires solving the depressed cubic $x^3+ax+d=0$ with $d=b-y^2$, whose discriminant $\Delta=-4a^3-27d^2$ may be a non-square in $\mathbb{F}_p$, leading directly to an $\mathbb{F}_{p^2}$ cube root. Alternatively, in the non-square-discriminant case the cubic has a unique root $t_0\in\mathbb{F}_p$, whose linear factor can be isolated as $T-t_0=\gcd(T^p-T,\,T^3+aT+d)$; the other two roots then follow by solving the remaining quadratic factor.

We focus on the case $p\equiv 1 \pmod 3$, which covers the pairing-friendly curve families targeted in this work (BN and BLS12): their primes are parametrized by a seed t and satisfy this congruence by construction (e.g., $p(t)=36t^4+36t^3+24t^2+6t+1$ for BN), so the non-trivial cube roots of unity lie in $\mathbb{F}_p$. The isogeny-based primes used in [19] also satisfy $p\equiv 1 \pmod 3$: their prime shape forces $p\equiv 1 \pmod 3$, which is required for efficient radical 3-isogeny computation via cube roots over $\mathbb{F}_{p^2}$ [15]. The condition $p\equiv 1 \pmod 3$ ensures $\gcd(3,p+1)=1$, which is crucial for our torus-based approach (Section 3). We also discuss an extension to $p\equiv 2 \pmod 3$ in Section 6 which works entirely on $\mathbb{F}_p$ whenever $p\not\equiv 8 \pmod 9$.

Prior work. At a high level, exponentiation and root extraction in $\mathbb{F}_{p^2}^*$ can be reduced to computations in the subgroup of order $p+1$ (the norm-one ele-

ments) and computations in a quotient of order $p-1$, using the factorization $p^2-1=(p-1)(p+1)$ of the group order. For odd p one has $\gcd(p-1,p+1)=2$, so $\mathbb{F}_{p^2}^*$ is not literally a direct product of the two groups: it is cyclic of order $(p-1)(p+1)$, and the shared 2-part must be handled separately; this caveat aside, the decomposition is an instance of the standard subgroup/quotient principle for abelian groups. Familiar incarnations of the same principle include the Pohlig–Hellman algorithm for discrete logarithms [32] (which decomposes along the prime-power factorization of the group order) and the Adleman–Manders–Miller algorithm for root extraction in finite fields [3]. Another analogue is the CRT-based decomposition of RSA exponentiation; this is not a direct instance because $(\mathbb{Z}/N\mathbb{Z})^*$, for $N=pq$, is itself a direct product. Concretely, the subgroup of order $p+1$ is the algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$, where Lucas-type recurrences provide a particularly efficient exponentiation; the LUC cryptosystem [37] and the analysis of Bleichenbacher, Bosma, and Lenstra [10] develop this point in detail, and the XTR system [28] extends it to the degree-6 algebraic torus $\mathbb{T}_6(\mathbb{F}_p)$ via third-order recurrences. The torus-based compression framework of Rubin and Silverberg [34] systematized the geometric viewpoint of $\mathbb{T}_n$ used in cryptography. Multi-exponentiations $x^{e_0}y^{e_1}$ likewise have a classical treatment via Straus' trick. A direct application of this decomposition to $\mathbb{F}_{p^2}$ cube roots is therefore conceptually straightforward; the engineering question we address is whether it can be made faster than a well-tuned direct $\mathbb{F}_{p^2}$ exponentiation, which is not automatic (see Section 4 and the naive-baseline discussion in Section 3.1).

In practice, the standard approach computes $z=x^e$ in $\mathbb{F}_{p^2}$ for an exponent e that depends on $p^2 \bmod 9$:

- $p^2 \equiv 4 \pmod 9$ (i.e., $p \equiv 2$ or $7 \pmod 9$): $e=(2p^2+1)/9$;
- $p^2 \equiv 7 \pmod 9$ (i.e., $p \equiv 4$ or $5 \pmod 9$): $e=(p^2+2)/9$;
- $p^2 \equiv 1 \pmod 9$ (i.e., $p \equiv 1$ or $8 \pmod 9$): $9 \mid (p^2-1)$ and no single exponent e satisfies $3e \equiv 1 \pmod{(p^2-1)/3}$. A Tonelli–Shanks-like algorithm in $\mathbb{F}_{p^2}$ is needed, decomposing $p^2-1=3^v \cdot m$ and handling the 3^v-part via discrete-log corrections.

In the first two cases the exponent has $\approx 2\log_2 p$ bits and the cost is dominated by the $\mathbb{F}_{p^2}$ exponentiation. The third case adds a Tonelli–Shanks correction loop of $v-1$ iterations. The authors of [19] use, in the companion code [18], a simple binary square-and-multiply method with a hardcoded 759-bit exponent. Since the exponent is fixed, one can derive a short addition chain to speed up the exponentiation a bit.

A natural improvement is *Frobenius splitting*: decompose the exponent as $e=e_0+e_1p$ with $e_0, e_1 \approx p$, then use a multi-exponentiation $x^e = x^{e_0} \cdot \overline{x}^{\,e_1}$ via Straus' trick, where $\overline{x}=x^p$ is the Frobenius conjugate. This halves the effective exponent length but still works entirely in $\mathbb{F}_{p^2}$. While Frobenius splitting is a well-known technique in pairing-based cryptography, to the best of our knowledge it has not been applied to $\mathbb{F}_{p^2}$ cube root computation in existing implementations, including the companion code of [19].

Related work. Adj and Rodríguez-Henríquez [2], Scott [36, Sec. 6.3] and Aardal et al. [1, Alg. 3, $p \equiv 3 \pmod 4$ only] all reduce $\mathbb{F}_{p^2}$ *square roots* to computations in $\mathbb{F}_p$. The key identity is $x^{p+1} = \mathrm{N}(x) \in \mathbb{F}_p$. Since $p+1$ is even, $b = x^{(p+1)/2}$ satisfies $b^2 = \mathrm{N}(x) \in \mathbb{F}_p$, which forces $b \in \mathbb{F}_p$ (or $b \in \mathbb{F}_p \cdot u$). The $\mathbb{F}_{p^2}$ square root thus reduces to: compute b via an $\mathbb{F}_p$ exponentiation (using the Frobenius $x^p = \overline{x}$), take an $\mathbb{F}_p$ square root of $\mathrm{N}(x)$, and combine. For cube roots, one might try $\gamma = x^{(p+1)/3}$, which cubes to $\mathrm{N}(x) \in \mathbb{F}_p$. But γ itself is a general $\mathbb{F}_{p^2}$ element—it does not collapse to $\mathbb{F}_p$—so the Frobenius does not yield an $\mathbb{F}_p$ reduction. More fundamentally, expanding $z^3 = x$ with $z = z_0 + z_1 u$ and eliminating z_1 yields a cubic in z_0 over $\mathbb{F}_p$ whose three roots cannot be expressed through $\mathbb{F}_p$ radicals alone—an instance of the classical *casus irreducibilis*. Our method takes a different route: instead of decomposing an $\mathbb{F}_{p^2}$ exponentiation, we project onto the algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$ and work entirely in $\mathbb{F}_p$ via Lucas sequences. The use of $\mathbb{T}_2(\mathbb{F}_p)$ itself is classical (the LUC line of work [37,10] predates the XTR system [28], which extends the idea to $\mathbb{T}_6(\mathbb{F}_p)$ via third-order recurrences); the systematic geometric treatment of $\mathbb{T}_n$ as a torus is due to Rubin and Silverberg [34], which is the source we cite for the torus viewpoint in Section 2.

A second ingredient of our algorithm is the simultaneous extraction of $n^{1/3}$, n^{-1}, and an auxiliary inverse U^{-1} from a single $\mathbb{F}_p$ addition chain. Combining inversions and roots into one chain has a long history: Montgomery's simultaneous inversion trick [30] computes $1/U$ and $1/V$ from a single $1/(UV)$; Fiat's Batch RSA [22] computes $U^{1/3}$ and $V^{1/5}$ from a single $(U^5V^3)^{1/15}$; Hamburg [23] gives a batched "inverse and square root" variant that ensures the combined exponent is negative and so removes inversions altogether. Our fused chain (Section 3.1) is the specialization of this idea to the exponent set $\{1/3, -1, -1\}$ that arises in our cube-root recovery.

Contributions. As noted above, the subgroup decomposition $\mathbb{F}_{p^2}^* \to \mathbb{T}_2(\mathbb{F}_p) \times \mathbb{F}_p^*$ is classical, and a naive implementation of it does not in itself beat a well-tuned direct $\mathbb{F}_{p^2}$ exponentiation: it requires an $\mathbb{F}_p$ cube root, an $\mathbb{F}_p$ inversion for the torus projection, a torus exponentiation, and a further $\mathbb{F}_p$ inversion for the recovery—roughly 4 $\mathbb{F}_p$ exponentiations to replace the direct method's ≈ 3 (since one $\mathbb{F}_p$ inversion costs about one $\mathbb{F}_p$ exponentiation in our setting). Our contribution is an algorithm that closes this gap and substantially beats direct exponentiation in practice. Concretely:

1. We instantiate the standard subgroup decomposition for the specific exponent $1/3$, giving a complete $\mathbb{F}_{p^2}$ cube-root algorithm that works entirely over $\mathbb{F}_p$: an $\mathbb{F}_p$ cube root of the norm $\mathrm{N}(x)$, a Lucas V-sequence exponentiation on $\mathbb{T}_2(\mathbb{F}_p)$, and a rational recovery step (Algorithm 1).
2. The main technical step is a *fused* $\mathbb{F}_p$ addition chain that simultaneously produces $\sqrt[3]{\mathrm{N}(x)}$, $\mathrm{N}(x)^{-1}$, and a third auxiliary inverse U^{-1}. This is a specialization to the exponent set $\{1/3, -1, -1\}$ of the Montgomery–Fiat–Hamburg line of inversion-merging techniques [30,22,23]. The novelty is in the construction of an auxiliary element $U = -16cn(x_0x_1)^2$ chosen so that an

Okeya–Sakurai-style coordinate recovery for $\mathbb{T}_2(\mathbb{F}_p)$ uses only the inverses already produced by the chain, eliminating the recovery inversion. Writing $\ell = \lfloor \log_2 p \rfloor$ and $\mathbf{M}_p$ for the cost of one multiplication in $\mathbb{F}_p$, the result is a fully inversion-free recovery with a total cost of $\approx 2.8\,\ell\,\mathbf{M}_p$ (Section 4), compared to $\approx 4\,\ell\,\mathbf{M}_p$ for the naive split and $\approx 5.3\,\ell\,\mathbf{M}_p$ for the best direct $\mathbb{F}_{p^2}$ chain.

3. We prove correctness in all residuosity cases and handle the cube-root ambiguity when $p \equiv 1 \pmod 9$ via a cheap verification-and-adjustment step using roots of unity.
4. We implement the method in Go for six primes: three corresponding to widely-deployed pairing-friendly curves and three corresponding to isogeny-walk primes from [19]. As an additional implementation contribution, we also provide a 2-bit windowed Frobenius-splitting multi-exponentiation baseline, which to our knowledge has not been applied to $\mathbb{F}_{p^2}$ cube roots in existing codebases. We benchmark all methods and compare against the direct exponentiation used in practice. The code is available in [24]: `https://github.com/yelhousni/fp2-cbrt`.

Organization. Section 2 recalls the algebraic torus, Lucas sequences, and cube-root residuosity in $\mathbb{F}_p$. Section 3 describes the algorithm, including the fused exponentiation, and proves correctness. Section 4 analyzes the cost. Section 5 presents implementation results. Section 6 extends the algorithm to $p \equiv 2 \pmod 3$. Section 7 concludes.

2 Preliminaries

Throughout, p denotes a prime with $p \equiv 1 \pmod 3$ and $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)$ for some non-square $\beta \in \mathbb{F}_p$. We write $\beta = -c$, where $c = -\beta \in \mathbb{F}_p^*$ is fixed for the field (e.g. $c = 1$ or $c = 5$ in our implementations). We write elements of $\mathbb{F}_{p^2}$ as $x = x_0 + x_1 u$ with $x_0, x_1 \in \mathbb{F}_p$. The Frobenius automorphism $\sigma : x \mapsto x^p$ acts as $\sigma(x_0 + x_1 u) = x_0 - x_1 u$, and we denote $\overline{x} \coloneqq \sigma(x) = x_0 - x_1 u$ the conjugate.

2.1 The $\mathbb{F}_{p^2}/\mathbb{F}_p$ norm

The norm map $\mathrm{N} : \mathbb{F}_{p^2}^* \to \mathbb{F}_p^*$ is defined by

$$\mathrm{N}(x) = x \cdot \overline{x} = x_0^2 - \beta\, x_1^2 = x_0^2 + c\, x_1^2\,, \tag{1}$$

using $\beta = -c$. The norm is surjective and satisfies $\mathrm{N}(x^k) = \mathrm{N}(x)^k$ for all $k \in \mathbb{Z}$.

2.2 The norm-one algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$

The norm-one algebraic torus of $\mathbb{F}_{p^2}$ over $\mathbb{F}_p$ is the subgroup

$$\mathbb{T}_2(\mathbb{F}_p) = \{\alpha \in \mathbb{F}_{p^2}^* : \mathrm{N}(\alpha) = 1\} = \ker(\mathrm{N} : \mathbb{F}_{p^2}^* \to \mathbb{F}_p^*)\,. \tag{2}$$

It has order $|\mathbb{T}_2(\mathbb{F}_p)| = p+1$. Every element α on the torus satisfies $\alpha^p = \overline{\alpha} = \alpha^{-1}$, so α is determined by its *trace*

$$\mathrm{Tr}(\alpha) \coloneqq \alpha + \overline{\alpha} = 2\,\alpha_0 \in \mathbb{F}_p\,. \tag{3}$$

The trace-based exponentiation in $\mathbb{T}_2(\mathbb{F}_p)$ via Lucas V-sequences that we use is classical (see [37,10]); the XTR system [28] extends the same idea to $\mathbb{T}_6(\mathbb{F}_p)$. We follow Rubin and Silverberg [34] for the geometric description of $\mathbb{T}_n$ as an algebraic torus (the source of the $\mathbb{T}_n$ notation used throughout).

2.3 Lucas V-sequences

The *Lucas V-sequence* $\{V_n(P,Q)\}_{n\geqslant 0}$ with parameters $P, Q \in \mathbb{F}_p$ is defined by

$$V_0 = 2, \qquad V_1 = P, \qquad V_{n+1} = P\,V_n - Q\,V_{n-1}\,. \tag{4}$$

The binary ladder doubling formulas are:

$$V_{2k} = V_k^2 - 2\,Q^k\,, \qquad V_{2k+1} = V_k \cdot V_{k+1} - P\,Q^k\,. \tag{5}$$

Specialization to the torus ($Q = 1$). Every element $\alpha \in \mathbb{T}_2(\mathbb{F}_p)$ satisfies the characteristic equation $\alpha^2 - \tau\,\alpha + 1 = 0$ where $\tau = \mathrm{Tr}(\alpha)$, so the corresponding Lucas parameters are $P = \tau$ and $Q = 1$. The trace of the n-th power is then $\mathrm{Tr}(\alpha^n) = V_n(\tau, 1)$ for all $n \geqslant 0$ [28]. With $Q = 1$ the doubling formulas simplify to

$$V_{2k} = V_k^2 - 2\,, \qquad V_{2k+1} = V_k \cdot V_{k+1} - \tau\,, \tag{6}$$

since $Q^k = 1$ for all k. In particular, no auxiliary tracking of Q^k is needed.

2.4 Cube roots in $\mathbb{F}_p$

Since $p \equiv 1 \pmod 3$, there are three cube roots of unity in $\mathbb{F}_p$: $\{1, \omega, \omega^2\}$ where ω is a primitive cube root of unity. An element $a \in \mathbb{F}_p^*$ is a cubic residue if and only if $a^{(p-1)/3} = 1$.

For the primes used in this work, an initial cube-root candidate is computed by exponentiating a by a helper exponent h that depends on $p \bmod 9$ (or $p \bmod 27$):

Remark 1. The following four cases cover all primes with $v_3(p-1) \leq 2$, where v_3 denotes the 3-adic valuation. If $v_3(p-1) \geq 3$ (i.e. $27 \mid (p-1)$), the simple exponentiation formula no longer suffices and one must use a discrete-logarithm correction in the 3-Sylow subgroup of $\mathbb{F}_p^*$, analogous to the Tonelli–Shanks algorithm for square roots in the highly 2-adic case. The Adleman–Manders–Miller algorithm [3] gives such a generalization. All pairing-friendly and isogeny primes considered here satisfy $v_3(p-1) \leq 2$.

Case	h	d	r
$p \equiv 7 \pmod 9$	$(p-7)/9$	9	7
$p \equiv 4 \pmod 9$	$(p-4)/9$	9	4
$p \equiv 10 \pmod{27}$	$(p-10)/27$	27	10
$p \equiv 19 \pmod{27}$	$(p-19)/27$	27	19

Table 1. Helper exponent h for $\mathbb{F}_p$ cube root depending on $p \bmod 9$ (or $p \bmod 27$). Divisor d and remainder r.

Given $t = a^h$, set $m_0 = a \cdot t^s$ where $s = 1$ if r is odd and $s = 2$ if r is even. When $p \equiv 4$ or $7 \pmod 9$, this directly gives a cube root of any cubic residue. When $p \equiv 1 \pmod 9$, m_0^3 may differ from a by a cube root of unity; a verification-and-adjustment step using a primitive ninth root of unity returns an actual cube root.

Lemma 1. *With the notation above, if a is a cubic residue then $m_0^3 = a\rho$ for some $\rho \in \{1, \omega, \omega^2\}$. Moreover, $\rho = 1$ when $p \equiv 4$ or $7 \pmod 9$.*

Proof. We have $m_0 = a^{1+sh}$, so $m_0^3 = a^{3(1+sh)}$. Since $h = (p-r)/d$ and $p \equiv 1 \pmod 3$, we compute:

- $(d = 9, r = 7, s = 1)$: $3(1+h) = 1 + (p-1)/3$;
- $(d = 9, r = 4, s = 2)$: $3(1+2h) = 1 + 2(p-1)/3$;
- $(d = 27, r = 19, s = 1)$: $3(1+h) = 1 + (p-1)/9$;
- $(d = 27, r = 10, s = 2)$: $3(1+2h) = 1 + 2(p-1)/9$.

Since a is a cubic residue, $a^{(p-1)/3} = 1$. The first two cases therefore give $m_0^3 = a$. In the last two cases $a^{(p-1)/9}$ has order dividing 3, so the extra factor is one of $1, \omega, \omega^2$. □

3 The Algorithm

Throughout this section we assume $p \equiv 1 \pmod 3$ and $v_3(p-1) \leq 2$. The first condition makes the cube map a bijection on $\mathbb{T}_2(\mathbb{F}_p)$; the second is the scope of the base-field helper formulas in Table 1. Section 6 discusses the extension to $p \equiv 2 \pmod 3$.

The key idea is to avoid working in $\mathbb{F}_{p^2}$ altogether. Given $x = x_0 + x_1 u \in \mathbb{F}_{p^2}^*$, we project x onto the algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$ via the norm-one element $\alpha = x^{1-p}$, compute a cube root of α *on the torus* using a Lucas-sequence exponentiation (which operates entirely in $\mathbb{F}_p$), and then lift the result back to $\mathbb{F}_{p^2}$ using the $\mathbb{F}_p$ cube root of the norm $\mathrm{N}(x)$. The norm cube root, its inverse, and an auxiliary inverse needed for the recovery are all obtained from a single fused $\mathbb{F}_p$ addition chain (Section 3.1). The algorithm proceeds in five steps, summarized in Algorithm 1.

Algorithm 1: Torus-based cube root in $\mathbb{F}_{p^2}$

Input: $x = x_0 + x_1 u \in \mathbb{F}_{p^2}^*$

Output: $z \in \mathbb{F}_{p^2}$ with $z^3 = x$, or $\perp$ if x is not a cubic residue

if $x_1 = 0$ **then**
  return $(x_0^{1/3}, 0)$ using $\mathbb{F}_p$ cube root;
if $x_0 = 0$ **then**
  $a \leftarrow (-x_1 \cdot c^{-1})^{1/3}$ in $\mathbb{F}_p$;
  return $(0, a)$ after verification; `// see Remark 2`
$n \leftarrow x_0^2 + c\,x_1^2$; // $n = \mathrm{N}(x) \in \mathbb{F}_p$
$U \leftarrow -16\,c\,n\,(x_0 x_1)^2$;
$(m, n^{-1}, U^{-1}) \leftarrow$ `FusedExp`(U, n) ; `// forms` $w = U^3 n$`; see §3.4`
$\tau \leftarrow 2(x_0^2 - c\,x_1^2) \cdot n^{-1}$; // $\mathrm{Tr}(x^{1-p}) \in \mathbb{F}_p$
$(\sigma_e,\, \sigma_{e+1}) \leftarrow (V_e(\tau),\, V_{e+1}(\tau))$ where $e = (p+2)/3$;
Recover z from $(m, n^{-1}, U^{-1}, \sigma_e, \sigma_{e+1}, x)$; `// inversion-free; see §3.4`
$z \leftarrow (z_0, z_1)$;
$z \leftarrow$ `VerifyAndAdjust`(z, x);
return z;

Remark 2. When $x_0 = 0$, we have $x = x_1 u$ and seek $z = a\,u$ with $z^3 = a^3 u^3 = a^3 \beta\, u = x_1 u$, so $a = (x_1/\beta)^{1/3} = (-x_1 \cdot c^{-1})^{1/3}$. For $\beta = -1$ (BN254 [7], BLS12-381 [12] used in Ethereum and isogeny primes from [19]) this simplifies to $a = (-x_1)^{1/3}$. For $\beta = -5$ (BLS12-377), the constant $c^{-1} = 5^{-1} \in \mathbb{F}_p$ can be precomputed.

Proposition 1 (Completeness of the edge-case handling). *The base-field cube root checks for $x_1 = 0$ and $x_0 = 0$ in Algorithm 1 are complete: if the $\mathbb{F}_p$ cube root does not exist, then no cube root exists in $\mathbb{F}_{p^2}$ either.*

Proof. Let $z = a + bu \in \mathbb{F}_{p^2}$ with $z^3 = x$. Expanding:

$$z^3 = \left(a^3 + 3ab^2\beta\right) + \left(3a^2 b + b^3\beta\right) u\,.$$

Case $x_1 = 0$. The coefficient of u vanishes if and only if $b = 0$ or $b^2 = -3a^2/\beta = 3a^2/c$. If $b = 0$, then $z^3 = a^3 = x_0$ requires x_0 to be a cube. If $b^2 = 3a^2/c$, then the constant term becomes

$$a^3 + 3a \cdot \left(-3a^2/\beta\right) \cdot \beta = a^3 - 9a^3 = -8a^3 = (-2a)^3\,,$$

which is *always* a perfect cube in $\mathbb{F}_p$. Therefore, if x_0 is not a cube, neither branch can produce it and no cube root exists in $\mathbb{F}_{p^2}$.

Case $x_0 = 0$. The constant term vanishes if and only if $a = 0$ or $a^2 = -3b^2\beta = 3c\,b^2$. The two branches give:

- $a = 0$: $x_1 = b^3\beta$, so $b^3 = x_1/\beta = -x_1/c$;

– $a^2 = 3c\,b^2$: the coefficient of u simplifies to $-8b^3\beta = x_1$, so $b^3 = -x_1/(8\beta) = x_1/(8c)$.

The second value differs from the first by a factor of $-1/8 = (-1/2)^3$, a perfect cube. Hence $-x_1/c$ is a cube in $\mathbb{F}_p$ if and only if $x_1/(8c)$ is. The algorithm checks the first branch; if it fails, the second must fail as well. □

We now describe each step of Algorithm 1 in detail.

3.1 Step 1: Norm and fused exponentiation – `FusedExp` (line 8)

Given $x = x_0 + x_1 u$, we compute the $\mathbb{F}_{p^2}/\mathbb{F}_p$ norm $n = \mathrm{N}(x) = x_0^2 + c\,x_1^2 \in \mathbb{F}_p$ and $U = -16cn(x_0x_1)^2$. The routine `FusedExp`(U, n) then uses a single $\mathbb{F}_p$ addition chain to return $m = n^{1/3}$, n^{-1}, and U^{-1}, the latter being needed for the inversion-free recovery of Section 3.4.

Correctness. Assume that x is a cubic residue. Then there exists $z^* \in \mathbb{F}_{p^2}^*$ with $(z^*)^3 = x$. Taking norms gives $\mathrm{N}(z^*)^3 = \mathrm{N}(x) = n$, so n is a cube in $\mathbb{F}_p$, and $m = n^{1/3}$ exists. Lemma 1 gives an initial candidate m_0; when $p \equiv 1 \pmod 9$, `FusedExp` adjusts m_0 by a ninth root of unity before returning m, so that $m^3 = n$.

The fused computation. A naïve approach would perform three separate $\mathbb{F}_p$ operations: a cube root, an inversion of the norm, and an inversion of the auxiliary U. Folding several inversions (and roots) into a single addition chain is itself a classical idea: Montgomery's simultaneous-inversion trick [30] combines two inversions into one, Fiat's Batch RSA [22] combines roots with coprime exponents, and Hamburg [23] gave a batched "inverse and square root" variant that removes inversions by choosing a negative combined exponent. We specialize this idea to the exponent set $\{1/3, -1, -1\}$ that arises in our recovery and to the helper-exponent structure of $\mathbb{F}_p$ cube roots (Table 1), extending Scott's "progenitor" trick for square roots [36].

On inputs (U, n), `FusedExp` first forms $w = U^3 \cdot n$. Let $h = (p - r)/d$ be the helper exponent from Section 2.4, and let $t = w^h$ be the result of the addition-chain exponentiation. Set $\chi = w \cdot t^s$. When $p \not\equiv 1 \pmod 9$, χ is already a cube root of w; otherwise it is the unadjusted candidate from Lemma 1. The following proposition shows how to obtain w^{-1} from the same chain in all cases.

Proposition 2. *Let $\chi = w \cdot t^s$ and let $a = r - 2$, $b = d - a \cdot s$. Then $\chi^a \cdot t^b = w^{-1}$.*

Proof. We have $\chi = w^{1+sh}$ and $t = w^h$, so

$$\chi^a \cdot t^b = w^{a(1+sh)+bh} = w^{a+(as+b)h}\,.$$

Now $as + b = (r-2)s + d - (r-2)s = d$, so the exponent becomes $a + dh = (r-2) + d \cdot \frac{p-r}{d} = p - 2$. By Fermat's little theorem, $w^{p-2} = w^{-1}$. □

Since `FusedExp` receives U and n separately and $w = U^3 n$, it can recover the three desired values:

$$m_0 = \chi \cdot U^{-1}\,, \qquad n^{-1} = U^3 \cdot w^{-1}\,, \qquad U^{-1} = U^2 \cdot n \cdot w^{-1}\,.$$

When $p \equiv 1 \pmod 9$, the routine verifies $m_0^3 = n$ and, if necessary, adjusts m_0 by a precomputed primitive ninth root of unity. Thus, for cubic n, `FusedExp` returns the adjusted value m with $m^3 = n$; this base-field adjustment is distinct from the final verification of the recovered $\mathbb{F}_{p^2}$ candidate.

Short extraction chains. The four cases in Table 1 give

$p \bmod 27$	(a, b)	cost of $\chi^a t^b$
$7, 16, 25$	$(5, 4)$	$4\,\mathbf{S}_p + 2\,\mathbf{M}_p$
$4, 13, 22$	$(2, 5)$	$3\,\mathbf{S}_p + 2\,\mathbf{M}_p$
10	$(8, 11)$	$6\,\mathbf{S}_p + 3\,\mathbf{M}_p$
19	$(17, 10)$	$7\,\mathbf{S}_p + 3\,\mathbf{M}_p$

For example, $\chi^5 t^4 = (\chi^2)^2 \chi\,(t^2)^2$. Thus the extraction cost is at most $7\,\mathbf{S}_p + 3\,\mathbf{M}_p = 8.6\,\mathbf{M}_p$ under the convention of Section 4, which is small compared with the main addition chain for $t = w^h$.

3.2 Step 2: Torus trace (line 9)

The element $x^{1-p} = x/\overline{x} \in \mathbb{T}_2(\mathbb{F}_p)$ lies on the torus. We compute its trace $\tau = \mathrm{Tr}(x^{1-p})$ directly from x_0, x_1 without performing the $\mathbb{F}_{p^2}$ exponentiation.

Proposition 3. *For $x = x_0 + x_1 u \in \mathbb{F}_{p^2}^*$ with $x_1 \neq 0$,*

$$\mathrm{Tr}(x^{1-p}) = \frac{2(x_0^2 + \beta\, x_1^2)}{\mathrm{N}(x)} = \frac{2(x_0^2 - c\, x_1^2)}{x_0^2 + c\, x_1^2}\,.$$

Proof. Since $x^p = \overline{x}$, we have $x^{1-p} = x/\overline{x}$ and $x^{p-1} = \overline{x}/x$. Therefore

$$\mathrm{Tr}(x^{1-p}) = x^{1-p} + x^{p-1} = \frac{x}{\overline{x}} + \frac{\overline{x}}{x} = \frac{\overline{x}^2 + x^2}{x \cdot \overline{x}} = \frac{x^2 + \overline{x}^2}{\mathrm{N}(x)}\,.$$

Now $x^2 = (x_0^2 + \beta\, x_1^2) + 2x_0 x_1 u$ and $\overline{x}^2 = (x_0^2 + \beta\, x_1^2) - 2x_0 x_1 u$, so $x^2 + \overline{x}^2 = 2(x_0^2 + \beta\, x_1^2)$. □

Note that τ is computed using x_0^2 and x_1^2 (already available from the norm computation) and n^{-1} (from the fused exponentiation), at a cost of 1 multiplication and $O(1)$ additions.

3.3 Step 3: Lucas chain — cube root on the torus (line 10)

Since $|\mathbb{T}_2(\mathbb{F}_p)| = p+1$ and $p \equiv 1 \pmod 3$ implies $p+1 \equiv 2 \pmod 3$, the cube map $\alpha \mapsto \alpha^3$ is a bijection on $\mathbb{T}_2(\mathbb{F}_p)$. Its inverse is exponentiation by the canonical representative

$$e = \frac{p+2}{3}, \qquad 0 < e < p+1,$$

because $3e = p+2 \equiv 1 \pmod{p+1}$.

Now, if $z^3 = x$ then $z^{3(1-p)} = x^{1-p}$, i.e., $(z^{1-p})^3 = x^{1-p}$. Since $z^{1-p} \in \mathbb{T}_2(\mathbb{F}_p)$, the trace $\sigma = \mathrm{Tr}(z^{1-p})$ satisfies $V_3(\sigma) = \tau$ and is recovered as

$$\sigma = V_e(\tau)\,, \qquad e = \frac{p+2}{3}\,.$$

This is computed via the Lucas-sequence binary ladder (equations (6)) in $\lfloor \log_2 e \rfloor$ steps. Since $e \le p$, this is at most $\lfloor \log_2 p \rfloor$ steps.

3.4 Step 4: Recovery of z (line 11)

We now have $m = \mathrm{N}(x)^{1/3} = \mathrm{N}(z)$ and $\sigma = \mathrm{Tr}(z^{1-p})$. We can recover $z = z_0 + z_1 u$ as follows.

Proposition 4. *Let $z = z_0 + z_1 u$ with $z^3 = x$ and $z_1 \neq 0$. Write $m = \mathrm{N}(z) = \mathrm{N}(x)^{1/3}$ and $\sigma = \mathrm{Tr}(z^{1-p})$. Then*

$$z_0 = \frac{x_0}{m\,(\sigma - 1)}\,, \qquad z_1 = \frac{x_1}{m\,(\sigma + 1)}\,.$$

Proof. From $z^3 = x$ and $\mathrm{N}(z) = m$, we derive $z^2 = x \cdot \overline{z}/m$. Writing this out component-wise:

$$\begin{aligned} z_0^2 + \beta\, z_1^2 &= (x_0 z_0 - \beta\, x_1 z_1)/m\,, \\ 2 z_0 z_1 &= (x_1 z_0 - x_0 z_1)/m\,. \end{aligned}$$

Meanwhile, from $\mathrm{Tr}(z^{1-p}) = \sigma$ and Proposition 3 applied to z:

$$\sigma = \frac{2(z_0^2 + \beta\, z_1^2)}{m}\,.$$

Combined with $z_0^2 - \beta\, z_1^2 = m$, we obtain $z_0^2 = m(\sigma+2)/4$ and $z_1^2 = m(\sigma - 2)/(4\beta)$. Substituting back into the component equations of $z^2 = x\overline{z}/m$ and simplifying yields the stated formulas.

Alternatively, the formulas can be verified directly: if $z_0 = x_0/(m(\sigma-1))$ and $z_1 = x_1/(m(\sigma+1))$, one can check that $z_0^2 - \beta\, z_1^2 = m$ and $2(z_0^2 + \beta\, z_1^2)/m = \sigma$ by substitution. □

Inversion-free recovery. The two divisions in Proposition 4 can be evaluated without any $\mathbb{F}_p$ inversion by exploiting (i) the three outputs m, n^{-1}, U^{-1} from the fused exponentiation of Section 3.1, and (ii) the adjacent Lucas value $V_{e+1}(\tau)$, which the binary ladder computes alongside $V_e(\tau)$ at no extra cost. Write $\alpha = x^{1-p} = A + Bu$ where $A = \tau/2$ and $B = 2x_0x_1/n$. The Lucas ladder returns both $\sigma_e = V_e(\tau) = \mathrm{Tr}(\gamma)$ and $\sigma_{e+1} = V_{e+1}(\tau) = \mathrm{Tr}(\alpha\gamma)$, where $\gamma = \alpha^e = z^{1-p} \in \mathbb{T}_2(\mathbb{F}_p)$ is the torus cube root. This is the torus analogue of Okeya–Sakurai coordinate recovery: the adjacent traces determine the hidden coordinate of γ. If $\gamma = C + Du$, define

$$W_0 = \sigma_{e+1} - A\,\sigma_e = -2cBD\,, \qquad W_1 = B\,\sigma_e = 2BC\,.$$

The preceding edge-case checks ensure $B \neq 0$. Thus γ would be recovered by one division by $2cB$. To avoid this inversion, use $U = -16cn(x_0x_1)^2 = -4cn^3B^2$. From U^{-1} computed in the fused step, form $\delta^{-1} = n^3 \cdot U^{-1}$ and $k = 2B \cdot \delta^{-1} = -1/(2cB)$, then:

$$z = x \cdot \overline{\gamma_k} \cdot m^{-1}\,, \qquad \gamma_k = (-c\,W_1k,\ W_0k) = \gamma\,, \qquad m^{-1} = m^2 \cdot n^{-1}\,.$$

This recovery uses only base-field operations; exact counts are given in Section 4.

3.5 Step 5: Verification and adjustment – `VerifyAndAdjust` (line 13)

The value m returned by `FusedExp` has already been adjusted so that $m^3 = n$. The final `VerifyAndAdjust` independently verifies the recovered $\mathbb{F}_{p^2}$ candidate and rejects non-residues. For a cubic input, the preceding adjustment and the correctness argument give $z^3 = x$; the remaining cases below also make the check robust to an unadjusted candidate satisfying $z^3 = x\omega^j$ for some $j \in \{0, 1, 2\}$. A primitive ninth root of unity $\zeta \in \mathbb{F}_p$ (with the convention $\zeta^3 = \omega^2$, and hence $(\zeta^2)^3 = \omega$) allows correction:

- If $z^3 = x$: return z.
- If $z^3 \cdot \omega^2 = x$: return $z \cdot \zeta$.
- If $z^3 \cdot \omega = x$: return $z \cdot \zeta^2$.
- Otherwise: x is not a cubic residue; return $\perp$.

Remark 3. Since $\omega, \zeta \in \mathbb{F}_p$, multiplication of z by either constant costs only 2 base-field multiplications rather than 3 for a general $\mathbb{F}_{p^2}$ multiplication. Similarly, the check $z^3 \cdot \omega^j = x$ involves multiplication by a base-field constant.

Remark 4. When $p \not\equiv 1 \pmod 9$ (i.e. $p \equiv 4$ or $7 \pmod 9$ in the cases of Table 1), the $\mathbb{F}_p$ cube root formula always yields $z^3 = x$ directly and no adjustment is needed. This is the case for BLS12-377, where $p \equiv 7 \pmod 9$, and also for primes with $p \equiv 4 \pmod 9$.

Step	$\mathbf{M}_p$	$\mathbf{S}_p$	$\mathbf{C}_p$	$\mathbf{A}_p$	$\mathbf{M}_{p^2}$	$\mathbf{S}_{p^2}$
Prepare n, U, w	4	4	2	1	0	0
`FusedExp` chain and extraction	$M_h + M_x + 5$	$S_h + S_x$	0	0	0	0
Base-field adjustment	ε	ε	εC_m	0	0	0
Torus trace	1	0	0	2	0	0
Lucas ladder	L	$L+1$	0	$2L+1$	0	0
Recovery	15	2	2	5	0	0
Verification/adjustment	0	0	C_z	0	1	1
Total	$M_h + M_x + L + 25 + \varepsilon$	$S_h + S_x + L + 7 + \varepsilon$	$4 + \varepsilon C_m + C_z$	$2L + 9$	1	1

Table 2. Unweighted operation counts for the main path of Algorithm 1.

4 Cost Analysis

We first give unweighted operation counts. Write $\ell = \lfloor \log_2 p \rfloor$ and $L = \lfloor \log_2 e \rfloor$, and let (S_h, M_h) denote the numbers of base-field squarings and multiplications in the addition chain for w^h, and let (S_x, M_x) denote the corresponding counts for extracting w^{-1} from (χ, t); the four possible values of (S_x, M_x) are listed in Section 3.1. We distinguish multiplication by an input-independent constant ($\mathbf{C}_p$) and field addition, subtraction, or doubling ($\mathbf{A}_p$) from general multiplication. Negations, assignments, and equality tests are not counted; optional defensive checks in the implementation are also excluded. Let $\varepsilon = 1$ when $p \equiv 1 \pmod 9$ and 0 otherwise, let $C_m \le 3$ count the constant multiplications needed to adjust the base-field root, and let $C_z \le 6$ count those needed by the final adjustment.

Preparation and trace. Computing $x_0^2, x_1^2, n, x_0x_1, U, U^2, U^3, w$ costs $4\,\mathbf{M}_p + 4\,\mathbf{S}_p + 2\,\mathbf{C}_p + 1\,\mathbf{A}_p$. Here a multiplication by $-16c$ is counted as one $\mathbf{C}_p$; for the constants used in our implementations it is realized more cheaply by doublings and additions. After `FusedExp`, the trace reuses cx_1^2 and costs $1\,\mathbf{M}_p + 2\,\mathbf{A}_p$.

Fused exponentiation – `FusedExp`. The long chain for $t = w^h$ costs $S_h\,\mathbf{S}_p + M_h\,\mathbf{M}_p$. Forming χ, extracting $w^{-1} = \chi^a t^b$, and deriving U^{-1}, m, n^{-1} cost $S_x\,\mathbf{S}_p + (M_x + 5)\,\mathbf{M}_p$; when $s = 2$, the value t^2 used to form χ is reused by the extraction chain. For $p \equiv 1 \pmod 9$, checking and adjusting m adds $1\,\mathbf{S}_p + 1\,\mathbf{M}_p + C_m\,\mathbf{C}_p$.

Lucas V_e. The binary ladder, including initialization and the adjacent output V_{e+1}, costs exactly $L\,\mathbf{M}_p + (L+1)\,\mathbf{S}_p + (2L+1)\,\mathbf{A}_p$.

Remark 5 (Differential addition chains). Since $e = (p+2)/3$ is fixed, one could replace the binary ladder with a shorter differential addition chain [8], typically saving 5–10% of the Lucas steps. However, differential chains compute only V_e, not the pair (V_e, V_{e+1}) that the binary ladder produces naturally. This makes them incompatible with the inversion-free recovery of Section 3.4, which requires V_{e+1}. In our experiments, a PRAC-based [8] implementation with Montgomery-trick recovery (costing an extra one $\mathbb{F}_p$ inversion) was slower in wall-clock time than the binary ladder with inversion-free recovery, despite the shorter chain.

Recovery (inversion-free). Computing δ^{-1}, the hidden coordinate of γ, m^{-1}, the conjugate product, and the final scaling costs $15\,\mathbf{M}_p + 2\,\mathbf{S}_p + 2\,\mathbf{C}_p + 5\,\mathbf{A}_p$.

The two $\mathbf{C}_p$ operations are multiplications by the fixed quadratic non-residue parameter.

Verification. Computing z^3 costs $1\,\mathbf{S}_{p^2} + 1\,\mathbf{M}_{p^2}$. If the first comparison fails, the root-of-unity checks and selected adjustment require at most $C_z = 6$ base-field constant multiplications.

For comparison with the earlier estimates, define the weighted cost

$$W = \#\mathbf{M}_p + 0.8\,\#\mathbf{S}_p + 3\,\#\mathbf{M}_{p^2} + 2\,\#\mathbf{S}_{p^2},$$

leaving $\mathbf{A}_p$ and $\mathbf{C}_p$ visible because their costs depend strongly on the constants and implementation. Table 2 gives

$$W = M_h + 0.8S_h + M_x + 0.8S_x + 1.8L + 35.6 + 1.8\varepsilon.$$

Since the long addition chain costs approximately $1.0(\ell - \log_2 d)\,\mathbf{M}_p$ and $L \approx \ell$, this recovers the summary $W \approx (2.8\ell + 36)\,\mathbf{M}_p$, in addition to the separately reported $\mathbf{A}_p$ and $\mathbf{C}_p$ operations.

As concrete examples:

- For BLS12-377 ($\ell = 377$, $d = 9$), the addition chain for $w^{(p-7)/9}$ has $(S_h, M_h) = (368, 75)$, $(S_x, M_x) = (4, 2)$, and $L = 375$. The total is $477\,\mathbf{M}_p$, $754\,\mathbf{S}_p$, $4\,\mathbf{C}_p$, $759\,\mathbf{A}_p$, $1\,\mathbf{M}_{p^2}$, and $1\,\mathbf{S}_{p^2}$ when the final adjustment is not needed, giving $W \approx 1085\,\mathbf{M}_p$.
- For `i-p381` ($\ell = 381$, $d = 9$), the addition chain has $(S_h, M_h) = (369, 37)$, $(S_x, M_x) = (3, 2)$, and $L = 379$. The total is $443\,\mathbf{M}_p$, $758\,\mathbf{S}_p$, $4\,\mathbf{C}_p$, $767\,\mathbf{A}_p$, $1\,\mathbf{M}_{p^2}$, and $1\,\mathbf{S}_{p^2}$, giving $W \approx 1054\,\mathbf{M}_p$.

4.1 Comparison with prior methods

We compare against two alternative approaches for computing $z^3 = x$ in $\mathbb{F}_{p^2}$:

- **Direct addition chain in $\mathbb{F}_{p^2}$.** Compute x^e where e is the appropriate exponent for the given $p^2 \bmod 9$. The exponent has $\approx 2\ell$ bits. With a short addition chain (≈ 1.2 operations per bit), one uses $\approx 2.4\,\ell$ steps. Most steps are squarings ($\mathbf{S}_{p^2} = 2\,\mathbf{M}_p$) with occasional multiplications ($\mathbf{M}_{p^2} = 3\,\mathbf{M}_p$), giving an average cost of $\approx 2.2\,\mathbf{M}_p$ per step and a total of $\approx 5.3\,\ell\,\mathbf{M}_p$. With naïve square-and-multiply (1.5 operations per bit), the cost would be $\approx 6.5\,\ell\,\mathbf{M}_p$. For BLS12-377, the concrete addition chain uses $747\,\mathbf{S}_{p^2} + 138\,\mathbf{M}_{p^2}$. Including the final cube verification, the complete count is $748\,\mathbf{S}_{p^2} + 139\,\mathbf{M}_{p^2} = 1913\,\mathbf{M}_p$.
- **Frobenius splitting.** Decompose $e = e_0 + e_1 p$ with $e_0, e_1 \approx p$ and use Straus' trick: $x^e = x^{e_0} \cdot \overline{x}^{\,e_1}$. This halves the effective exponent length to $\approx \ell$ bits.

  1-bit windows. Each of the ℓ steps requires one $\mathbf{S}_{p^2} = 2\,\mathbf{M}_p$, and on average $\approx 75\%$ of the steps require an additional $\mathbf{M}_{p^2} = 3\,\mathbf{M}_p$ (since each bit position has probability $3/4$ that at least one of the two exponent bits is 1), giving a cost of $\approx \ell\,(2 + 0.75 \times 3) = 4.25\,\ell\,\mathbf{M}_p$. The precomputed table has 3 entries $(\overline{x},\, x,\, x \cdot \overline{x})$.

2-bit windows. Processing two bits at a time from each exponent reduces the number of iterations to $\lceil \ell/2 \rceil$, each performing $2\,\mathbf{S}_{p^2} = 4\,\mathbf{M}_p$. The 2-bit window indices (w_0, w_1) with $w_0, w_1 \in \{0,1,2,3\}$ yield 16 combinations, of which only $(0,0)$ requires no multiplication, so $\approx 15/16$ of the steps require a $\mathbf{M}_{p^2} = 3\,\mathbf{M}_p$. The precomputed table has 15 entries ($x^i \cdot \overline{x}^j$ for $(i,j) \in \{0,\ldots,3\}^2 \setminus \{(0,0)\}$), built from 8 multiplications. The total cost is $\approx \frac{\ell}{2}(4 + \frac{15}{16} \times 3) \approx 3.4\,\ell\,\mathbf{M}_p$, a 20% improvement over 1-bit windows at the expense of a larger table.

Table 3 summarizes the asymptotic and concrete costs. The structural advantage is that all torus operations are in $\mathbb{F}_p$, avoiding the 2–3× overhead of $\mathbf{S}_{p^2}$ and $\mathbf{M}_{p^2}$.

Method	Asymptotic	$\ell = 377$	Ratio
Direct (addition chain in $\mathbb{F}_{p^2}$)	$\approx 5.3\,\ell$	1913	1×
Frobenius 1-bit	$\approx 4.25\,\ell$	1602	1.19×
Frobenius 2-bit	$\approx 3.4\,\ell$	1282	1.49×
Torus (binary ladder)	$\approx 2.8\,\ell + 36$	1085	1.76×

Table 3. Theoretical weighted-cost comparison for $\mathbb{F}_{p^2}$ cube root using the definition of W above; $\mathbf{A}_p$ and $\mathbf{C}_p$ are excluded. The $\ell = 377$ column uses exact addition-chain counts for Direct and Torus; the Frobenius entries instantiate the displayed asymptotic estimates. Ratios are speed-ups relative to the direct method.

5 Implementation and Benchmarks

We implemented the $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ arithmetic together with Algorithm 1 for six primes (Section 5.1) in the Go language. The addition chains for the helper exponent h are generated by `addchain` [29] for five of the six primes. For `i-p765`, the 765-bit helper exponent exceeds the capacity of `addchain`'s internal allocator, so the helper exponentiation falls back to generic square-and-multiply. An optimized addition chain for this prime would further improve the torus timings. The code is available in [24]: `https://github.com/yelhousni/fp2-cbrt`.

5.1 Target parameters

We target 6 primes spanning pairing-based and isogeny-based cryptographic applications (see Table 4).

Pairing-friendly primes. These are defined by a polynomial family $p(t)$ evaluated at a specific seed t_0:

– p-p254 (BN254):
  $p(t) = 36t^4 + 36t^3 + 24t^2 + 6t + 1$,
  $t_0 =$ 0x44E992B44326A040.
– p-p381 (BLS12-381):
  $p(t) = (t-1)^2(t^4 - t^2 + 1)/3 + t$,
  $t_0 = -$0xD201000000010000.
– p-p377 (BLS12-377):
  $p(t) = (t-1)^2(t^4 - t^2 + 1)/3 + t$,
  $t_0 =$ 0x8508C00000000001.

Isogeny-friendly primes. These are from [19], of the form $p = 2^{3a} \cdot f - 1$ with $\gcd(3, f) = 1$:

– i-p381: $p = 2^{372} \cdot 437 - 1$.
– i-p575: $p = 2^{567} \cdot 139 - 1$.
– i-p765: $p = 2^{756} \cdot 257 - 1$.

Prime	$\log_2 p$	$p \bmod 9$	$\beta = -c$	s	Adjust
p-p254	254	1 (19 mod 27)	-1	1	yes
p-p381	381	1 (10 mod 27)	-1	2	yes
p-p377	377	7	-5	1	no
i-p381	381	4	-1	2	no
i-p575	575	4	-1	2	no
i-p765	765	4	-1	2	no

Table 4. Parameters for the six target primes. The "Adjust" column indicates whether the base-field cube-root candidate may require multiplication by a ninth root of unity.

5.2 Benchmark results

All benchmarks are run on an AMD EPYC 9R14 (x86_64) with Go 1.24, using `go test -bench` with `-count=4` (we report the median). The four methods compared are:

– **Direct**: exponentiation in $\mathbb{F}_{p^2}$ using short addition chains generated by `addchain` [29].
– **Frob. 1-bit**: Frobenius-splitting multi-exponentiation in $\mathbb{F}_{p^2}$ using 1-bit Straus and a 3-entry table.
– **Frob. 2-bit**: same with 2-bit windowed Straus (15-entry table).
– **Torus**: Algorithm 1 using the binary ladder with inversion-free recovery (Section 3.4).

Prime	Direct (μs)	Frob. 1-bit (μs)	Frob. 2-bit (μs)	Torus (μs)	Torus speed-up vs. Direct	Torus speed-up vs. Frob. 2-bit
p-p254	33.0	27.8	23.0	18.6	1.77×	1.24×
p-p377	113.4	86.1	74.5	52.3	2.17×	1.43×
p-p381	95.6	80.0	65.9	53.1	1.80×	1.24×
i-p381	66.9	67.0	54.2	39.0	1.72×	1.39×
i-p575	354.8	366.6	290.1	153.6	2.31×	1.89×
i-p765	1162.8	1226.3	967.0	732.0	1.59×	1.32×

Table 5. Benchmark results for $\mathbb{F}_{p^2}$ cube root on AMD EPYC 9R14 (x86_64), Go 1.24.

The *Direct* column uses optimized addition chains generated by `addchain` [29], giving a strong baseline with zero heap allocations. Against this baseline, the torus method achieves 1.6–2.3× speed-ups across all six primes. For the isogeny primes, these translate to substantial absolute savings (28–431 μs per cube root), which is particularly significant since the cube root is the bottleneck operation in every step of the 3-isogeny walk [19].

Frobenius splitting, while a well-known technique, had not been applied to $\mathbb{F}_{p^2}$ cube roots in practice. Our 2-bit windowed implementation provides 1.2–1.5× speed-ups over the direct method and is itself 1.2–1.3× faster than the 1-bit variant, consistent with the theoretical ∼20% improvement predicted in Section 4. Note that for the largest primes (i-p575, i-p765), the 1-bit Frobenius method is actually *slower* than the direct approach, as the overhead of two half-length multi-exponentiations with a 3-entry table exceeds the cost of a single full-length addition chain. The torus method remains 1.2–1.9× faster than even the 2-bit Frobenius, with the largest gap on i-p575 (1.89×) where the pure-$\mathbb{F}_p$ Lucas ladder benefits most from the wider superscalar pipeline of the x86 microarchitecture. A summary of benchmark results is presented in Table 5.

5.3 Constant-time considerations

Cube roots in $\mathbb{F}_{p^2}$ arise in several cryptographic contexts with different secrecy requirements. We briefly survey the main applications and discuss the constant-time properties of the methods presented in this paper.

Applications and their secrets. In the CGL hash function [17] instantiated with 3-isogenies (CGLHash_3), the input path is the *public* message being hashed, so the cube root input is public and constant-time execution is not required. In isogeny-based key-encapsulation mechanisms such as QFESTA [31] and group-action-based protocols such as CSIDH/CTIDH [16,4,14], the isogeny path encodes the *secret key*: the curve coefficient a_3 fed to the cube root depends on the secret walk taken so far. In that setting the cube root must run in constant time

with respect to its input. In hash-to-curve constructions (e.g. RFC 9380 [21]), the field element being mapped may derive from a secret (e.g. a password in OPAQUE); constant-time behavior is required as a defensive measure. In point decompression, the compressed coordinate is typically public, so constant-time execution is not needed.

Analysis of the four methods. The arithmetic kernels of all four cube root methods studied in this paper (Direct addition-chain, Frobenius 1-bit, Frobenius 2-bit, and the Torus binary ladder) execute a *fixed sequence of field operations* determined solely by the prime p. In each case:

- The exponent (or chain) is a public constant derived from p, not from the input.
- The loop structure, table indices, and conditional swaps depend only on fixed exponent bits, never on the input value.

Thus, the main addition and Lucas chains are constant-time with respect to the cube root input, assuming the underlying field arithmetic is itself constant-time. This statement does not by itself cover the exceptional-case handling, the post-chain checks in `FusedExp`, or the final verification in Algorithm 1.

Input-dependent branches. Algorithm 1, as written, first branches on $x_1 = 0$ and $x_0 = 0$ and returns early on these exceptional inputs. Although the addition chain inside `FusedExp` is fixed, the routine is not constant-time as written: it tests whether $m_0^3 = n$ and, for $p \equiv 1 \pmod 9$, branches to select a ninth-root-of-unity correction; the implementations for $p \equiv 4 \pmod 9$ also branch to reject a failed defensive cube check. The final `VerifyAndAdjust` step (line 13) similarly branches on whether $z^3 = x$ and, when needed, selects a ninth-root-of-unity correction. Therefore, the complete pseudocode is not constant-time as written. When the input is secret, the exceptional cases must be handled using masked inputs and constant-time conditional selection rather than early returns. All correction candidates should be computed and selected with masks, and equality and residuosity tests must likewise produce and consume masks without input-dependent early exit. These requirements are in addition to requiring constant-time underlying field routines, including multiplication, squaring, and any inversion or base-field root extraction. At the protocol level:

- In $\mathrm{CGLHash}_3$ both the cube root input and the trit $\mathsf{path}_i \in \{0, 1, 2\}$ are public, so these branches are harmless.
- In QFESTA/CSIDH, the trit is secret. As noted in [19], the standard approach is to compute all three cube roots $[\sqrt[3]{-a_3},\ \zeta_3\sqrt[3]{-a_3},\ \zeta_3^2\sqrt[3]{-a_3}]$ and select the correct one via a constant-time conditional move keyed on the secret trit. The two exceptional-input cases above must also be handled without secret-dependent control flow.

6 Extension to $p \equiv 2 \pmod 3$

Throughout the paper we have assumed $p \equiv 1 \pmod 3$, which ensures $\gcd(3, p+1) = 1$ and makes the cube map a bijection on $\mathbb{T}_2(\mathbb{F}_p)$. We now discuss how to adapt the torus approach when $p \equiv 2 \pmod 3$, where $3 \mid (p+1)$ and the cube map on $\mathbb{T}_2(\mathbb{F}_p)$ is no longer bijective.

6.1 Structural differences

When $p \equiv 2 \pmod 3$, two things change:

1. $\mathbb{F}_p$ **cube roots are trivial.** Since $p \equiv 2 \pmod 3$, we have $\gcd(3, p-1) = 1$, so every element of $\mathbb{F}_p^*$ is a cube and the cube root is unique: $a^{1/3} = a^{(2p-1)/3}$. The exponent $(2p-1)/3$ has $\approx \log_2 p$ bits.
2. **The cube map on $\mathbb{T}_2(\mathbb{F}_p)$ has a non-trivial kernel.** The torus $\mathbb{T}_2(\mathbb{F}_p)$ has order $p+1$ with $3 \mid (p+1)$. Write $p+1 = 3^v \cdot \mu$ with $\gcd(3, \mu) = 1$ and $v \geqslant 1$. The cube map $\alpha \mapsto \alpha^3$ on $\mathbb{T}_2(\mathbb{F}_p)$ has kernel $\mu_3 \subset \mathbb{T}_2(\mathbb{F}_p)$, the subgroup of cube roots of unity of order 3 (these lie in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$ since $3 \nmid (p-1)$). Therefore the cube map is 3-to-1 on cubes and 0-to-1 on non-cubes.

An element $x \in \mathbb{F}_{p^2}^*$ is a cube if and only if $x^{(p^2-1)/3} = 1$. Since $p^2 - 1 = (p-1)(p+1)$ and $\gcd(3, p-1) = 1$, this simplifies to $(x^{1-p})^{(p+1)/3} = 1$, i.e., the torus element $\alpha = x^{1-p}$ has order dividing $(p+1)/3$.

6.2 The case $v = 1$

When $v = 1$, we have $p + 1 = 3\mu$ with $\gcd(3, \mu) = 1$. The adaptation of Algorithm 1 is straightforward and requires no Tonelli–Shanks-like correction.

The key observation is that $e = 3^{-1} \bmod \mu$ satisfies $3e = 1 + j\mu$ for some integer j. If x is a cube and $\alpha = x^{1-p} \in \mathbb{T}_2(\mathbb{F}_p)$, then α has order dividing μ (by the residuosity condition above), so $\alpha^\mu = 1$ and:

$$\alpha^{3e} = \alpha^{1+j\mu} = \alpha \cdot (\alpha^\mu)^j = \alpha\,.$$

Therefore $\gamma_0 = \alpha^e$ exactly satisfies $\gamma_0^3 = \alpha$ on $\mathbb{T}_2(\mathbb{F}_p)$—no ambiguity arises. The adapted algorithm is:

1. Compute $n = \mathrm{N}(x)$, the (unique) $\mathbb{F}_p$ cube root $m_0 = n^{1/3} = n^{(2p-1)/3}$, and the auxiliary inverses n^{-1} and U^{-1} (directly or by an adapted fused chain).
2. Compute $\tau = \mathrm{Tr}(x^{1-p})$ as in Proposition 3.
3. Compute $\sigma = V_e(\tau)$ where $e = 3^{-1} \bmod \mu$.
4. Recover z from (m_0, σ) as in Proposition 4.
5. Verify $z^3 = x$; if not, x is not a cube (return $\perp$).

Compared to the $p \equiv 1 \pmod 3$ case:

- The $\mathbb{F}_p$ cube root is unique, so no ω/ζ-adjustment step is needed.
- The Lucas exponent $e = 3^{-1} \bmod \mu$ has $\approx \log_2(\mu) \approx \log_2(p) - 1.58$ bits, essentially the same length as $(p+2)/3$ in the $p \equiv 1 \pmod 3$ case.
- If verification fails, x is provably a non-cube; there is no ambiguity between "wrong cube root" and "non-residue".

Cost. The $\mathbb{F}_p$ exponentiation computes $n^{(2p-1)/3}$ (the cube root); the required inverses can be computed directly or fused with an adapted addition-chain construction. The helper exponent $(2p-1)/3$ has $\approx \log_2 p$ bits, comparable to the helper exponents in the $p \equiv 1 \pmod 3$ case. The Lucas chain costs $\approx 1.8\,\mathbf{M}_p$ per bit for $\approx \log_2 p$ bits. In total, the cost is comparable to Algorithm 1, and the speed-up over the direct $\mathbb{F}_{p^2}$-exponentiation method (exponent $\approx 2\log_2 p$ bits at $\approx 3\,\mathbf{M}_p$ per bit) is of the same order.

6.3 The general case $v \geqslant 2$

When $v \geqslant 2$, the μ-part cube root $\gamma_0 = \alpha^e$ (with $e = 3^{-1} \bmod \mu$) may no longer satisfy $\gamma_0^3 = \alpha$ exactly. Instead, $\gamma_0^3 = \alpha \cdot \eta$ where $\eta = \alpha^{j\mu} \in \mathbb{T}_2(\mathbb{F}_p)$ has order dividing 3^{v-1}. A Tonelli–Shanks-like correction is then needed.

Algorithm 2: Torus cube root correction for the 3^v-Sylow ($v \geqslant 2$)

Input: $\gamma_0 \in \mathbb{T}_2(\mathbb{F}_p)$ with $\gamma_0^3 = \alpha \cdot \eta$, η of order $\mid 3^{v-1}$; precomputed $g_3 \in \mathbb{T}_2(\mathbb{F}_p)$ of order 3^v

Output: $\gamma \in \mathbb{T}_2(\mathbb{F}_p)$ with $\gamma^3 = \alpha$

$\gamma \leftarrow \gamma_0$; $t \leftarrow \eta$; $g \leftarrow g_3$; $r \leftarrow v$;
while $t \neq 1$ **do**
  find smallest $i \geqslant 1$ such that $t^{3^i} = 1$;
  $b \leftarrow g^{3^{r-i-1}}$; // b has order 3^{i+1}
  find $j \in \{1, 2\}$ such that $(t \cdot b^{3j})^{3^{i-1}} = 1$;
  $\gamma \leftarrow \gamma \cdot b^j$; $t \leftarrow t \cdot b^{3j}$; $g \leftarrow b^3$; $r \leftarrow i$;
return γ;

The loop runs at most $v-1$ iterations. Each iteration requires computing powers of torus elements, which can be done via Lucas sequences at a cost of $O(\log p)$ per iteration. In practice, v is small (typically $v \leqslant 3$ for cryptographic primes), so the correction overhead is modest.

Precomputation. The correction requires a generator $g_3 \in \mathbb{T}_2(\mathbb{F}_p)$ of order 3^v. This can be found by choosing a random $g \in \mathbb{F}_{p^2}^*$, computing $g^{(p-1)\cdot\mu} \in \mathbb{T}_2(\mathbb{F}_p)$ (which has order dividing 3^v), and checking that it has exact order 3^v. Since g_3 depends only on (p, β), it is computed once and stored as a curve constant.

Trace representation. A subtlety arises because the Lucas-sequence representation stores only the trace $\mathrm{Tr}(\gamma) \in \mathbb{F}_p$, not the full element $\gamma \in \mathbb{F}_{p^2}$. The correction in Algorithm 2 requires multiplying torus elements, which cannot be done from traces alone. Two approaches are possible:

- **Decompression.** Recover the full element from its trace: given $s = \mathrm{Tr}(\gamma)$, the coordinates are $\gamma_0 = s/2$ and $\gamma_1 = \sqrt{(s^2-4)/(4\beta)}$, costing one $\mathbb{F}_p$ square root. The correction then proceeds in full $\mathbb{F}_{p^2}$ arithmetic.

– **Direct $\mathbb{F}_{p^2}$ exponentiation of the μ-part.** Instead of using Lucas sequences, compute $\gamma_0 = \alpha^e$ directly in $\mathbb{F}_{p^2}$ (via a multi-exponentiation using the Frobenius), then apply Algorithm 2 in $\mathbb{F}_{p^2}$. This loses the per-bit cost advantage of Lucas sequences but avoids the square root.

For $v = 1$, no correction is needed and the trace representation suffices. For $v \geqslant 2$, the choice between the two approaches depends on the relative costs of the $\mathbb{F}_p$ square root and the Lucas-vs-$\mathbb{F}_{p^2}$ exponentiation trade-off.

Overall, our algorithm works without correction whenever $v_3(p+1) \leq 1$ (i.e., $p \not\equiv 8 \pmod 9$), which unifies the $p \equiv 1 \pmod 3$ case (Section 3) and the $v = 1$ subcase of $p \equiv 2 \pmod 3$ (Section 6).

7 Conclusion

We presented a new algorithm for computing cube roots in $\mathbb{F}_{p^2}$ that reduces all operations to the base field $\mathbb{F}_p$ via the algebraic torus $\mathbb{T}_2(\mathbb{F}_p)$ and Lucas V-sequences. A fused $\mathbb{F}_p$ exponentiation simultaneously produces $\sqrt[3]{\mathrm{N}(x)}$, $\mathrm{N}(x)^{-1}$, and an auxiliary inverse U^{-1}, enabling a fully inversion-free recovery that uses only $\mathbb{F}_p$ multiplications.

We implemented the algorithm for six pairing-friendly and isogeny-friendly primes. It is 1.6–2.3× faster than direct addition-chain exponentiation in $\mathbb{F}_{p^2}$, and 1.2–1.9× faster than our new baseline, a 2-bit windowed Frobenius-splitting multi-exponentiation.

More generally, the torus component of the approach extends to odd k-th roots in $\mathbb{F}_{p^2}$ whenever $\gcd(k, p+1) = 1$: the inverse of the k-th power map on $\mathbb{T}_2(\mathbb{F}_p)$ is evaluated by a Lucas sequence with exponent $e = k^{-1} \bmod (p+1)$. Efficient fused exponentiation and recovery formulas, however, depend on k and require separate analysis.

Future work. An implementation of the $p \equiv 2 \pmod 3$ variant (Section 6) for concrete primes would quantify the practical gains in that setting and is a natural next step. Another direction is to derive and evaluate the fused exponentiation and recovery steps for odd roots beyond the cube-root case.

Acknowledgments. We thank Benjamin Smith for suggesting the Okeya–Sakurai-style coordinate recovery viewpoint, which led us to rework the recovery formula and fuse the recovery inverse with the existing fused exponentiation. We also thank Dmitrii Koshelev for carefully reviewing the paper and for his helpful remarks.

References

1. Aardal, M.A., Adj, G., Alblooshi, A., Aranha, D.F., Martinez, I.A.C., Chávez-Saab, J., Filho, D.L.G., Reijnders, K., Rodríguez-Henríquez, F.: Optimized one-dimensional SQIsign verification on intel and cortex-M4. IACR TCHES **2025**(1), 497–522 (2025). https://doi.org/10.46586/tches.v2025.i1.497-522

2. Adj, G., Rodríguez-Henríquez, F.: Square root computation over even extension fields. IEEE Transactions on Computers **63**(11), 2829–2841 (2014). https://doi.org/10.1109/TC.2013.145
3. Adleman, L.M., Manders, K., Miller, G.L.: On taking roots in finite fields. 18th Annual Symposium on Foundations of Computer Science (FOCS) pp. 175–178 (1977)
4. Banegas, G., Bernstein, D.J., Campos, F., Chou, T., Lange, T., Meyer, M., Smith, B., Sotáková, J.: CTIDH: faster constant-time CSIDH. IACR TCHES **2021**(4), 351–387 (2021). https://doi.org/10.46586/tches.v2021.i4.351-387, `https://tches.iacr.org/index.php/TCHES/article/view/9069`
5. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 257–267. Springer, Berlin, Heidelberg (Sep 2003). https://doi.org/10.1007/3-540-36413-7_19
6. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Berlin, Heidelberg (Aug 2006). https://doi.org/10.1007/11693383_22
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 781–796. USENIX Association (Aug 2014), `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson`
8. Bernstein, D.J., Cottaar, J., Lange, T.: Searching for differential addition chains. Research in Number Theory **11**(45) (2025). https://doi.org/10.1007/s40993-024-00604-8
9. Biryukov, A., Fisch, B., Herold, G., Khovratovich, D., Leurent, G., Naya-Plasencia, M., Wesolowski, B.: Cryptanalysis of algebraic verifiable delay functions. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part III. LNCS, vol. 14922, pp. 457–490. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68382-4_14
10. Bleichenbacher, D., Bosma, W., Lenstra, A.K.: Some remarks on Lucas-based cryptosystems. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 386–396. Springer, Berlin, Heidelberg (Aug 1995). https://doi.org/10.1007/3-540-44750-4_-31
11. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_25
12. Bowe, S.: BLS12-381: New zk-SNARK elliptic curve construction. Zcash blog (March 11 2017), `https://blog.z.cash/new-snark-curve/`
13. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00050
14. Campos, F., Hellenbrand, A., Meyer, M., Reijnders, K.: dCTIDH: Fast & deterministic CTIDH. IACR TCHES **2025**(3), 516–541 (2025). https://doi.org/10.46586/tches.v2025.i3.516-541
15. Castryck, W., Decru, T., Vercauteren, F.: Radical isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 493–519. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_17

16. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
17. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. Journal of Cryptology **22**(1), 93–113 (Jan 2009). https://doi.org/10.1007/s00145-007-9002-x
18. Chi-Domínguez, J.J., Ochoa-Jiménez, E., Pontaza-Rodas, R.N.: Let us walk on the 3-isogeny graph: efficient, fast, and simple (2025), `https://github.com/Crypto-TII/pqc-engineering-ssec-23`
19. Chi-Domínguez, J.J., Ochoa-Jiménez, E., Rodas, R.N.P.: Let us walk on the 3-isogeny graph: efficient, fast, and simple. IACR TCHES **2025**(4), 644–666 (2025). https://doi.org/10.46586/tches.v2025.i4.644-666
20. El Housni, Y., Bünz, B.: On the security of constraint-friendly map-to-curve relations. Cryptology ePrint Archive, Report 2026/590 (2026), `https://eprint.iacr.org/2026/590`
21. Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R.S., Wood, C.A.: Hashing to elliptic curves. RFC 9380 (2023), `https://www.rfc-editor.org/rfc/rfc9380`
22. Fiat, A.: Batch RSA. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 175–185. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_17
23. Hamburg, M.: Decaf: Eliminating cofactors through point compression. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 705–723. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_34
24. Housni, Y.E.: Fast cube roots in fp2 via the algebraic torus in go (2026), `https://github.com/yelhousni/fp2-cbrt`
25. Icart, T.: How to hash into elliptic curves. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 303–316. Springer, Berlin, Heidelberg (Aug 2009). https://doi.org/10.1007/978-3-642-03356-8_18
26. Koshelev, D.: Some remarks on how to hash faster onto elliptic curves. Journal of Computer Virology and Hacking Techniques **20**, 593–605 (2024). https://doi.org/10.1007/s11416-024-00514-4
27. Koshelev, D.: Batch point compression in the context of advanced pairing-based protocols. Applicable Algebra in Engineering, Communication and Computing **36**, 611–629 (2025). https://doi.org/10.1007/s00200-023-00625-3
28. Lenstra, A.K., Verheul, E.R.: The XTR public key system. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 1–19. Springer, Berlin, Heidelberg (Aug 2000). https://doi.org/10.1007/3-540-44598-6_1
29. McLoughlin, M.: addchain: Cryptographic addition chain generation in go (2024), `https://github.com/mmcloughlin/addchain`
30. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (1987). https://doi.org/10.2307/2007888
31. Nakagawa, K., Onuki, H.: QFESTA: Efficient algorithms and parameters for FESTA using quaternion algebras. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part V. LNCS, vol. 14924, pp. 75–106. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68388-6_4
32. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). IEEE Transactions on Information Theory **24**(1), 106–110 (1978). https://doi.org/10.1109/TIT.1978.1055817

33. Pornin, T.: Optimized discrete logarithm computation for faster square roots in finite fields. Cryptology ePrint Archive, Report 2023/828 (2023), https://eprint.iacr.org/2023/828
34. Rubin, K., Silverberg, A.: Torus-based cryptography. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 349–365. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_21
35. Sarkar, P.: Computing square roots faster than the tonelli-shanks/bernstein algorithm. Cryptology ePrint Archive, Report 2020/1407 (2020), https://eprint.iacr.org/2020/1407
36. Scott, M.: A note on the calculation of some functions in finite fields: Tricks of the trade. Cryptology ePrint Archive, Report 2020/1497 (2020), https://eprint.iacr.org/2020/1497
37. Smith, P., Skinner, C.: A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In: Pieprzyk, J., Safavi-Naini, R. (eds.) ASIACRYPT'94. LNCS, vol. 917, pp. 357–364. Springer, Berlin, Heidelberg (Nov / Dec 1995). https://doi.org/10.1007/BFb0000447