AprèsSQI: Extra Fast Verification for SQIsign Using Extension-Field Signing

Maria Corte-Real Santos[1], Jonathan Komada Eriksen[2], Michael Meyer[3], and Krijn Reijnders[4]

[1] University College London
maria.santos.20@ucl.ac.uk
[2] Norwegian University of Science and Technology
jonathan.k.eriksen@ntnu.no
[3] University of Regensburg, Germany
michael@random-oracles.org
[4] Radboud University, Nijmegen, The Netherlands
krijn@cs.ru.nl

Abstract. We optimise the verification of the SQIsign signature scheme. By using field extensions in the signing procedure, we are able to significantly increase the amount of available rational 2-power torsion in verification, which achieves a significant speed-up. This, moreover, allows several other speed-ups on the level of curve arithmetic. We show that the synergy between these high-level and low-level improvements gives significant improvements, making verification 2.07 times faster, or up to 3.41 times when using size-speed trade-offs, compared to the state of the art, without majorly degrading the performance of signing.

Keywords: post-quantum cryptography, isogenies, SQIsign, verification

1 Introduction

Research has shown that large-scale quantum computers will break current public-key cryptography, such as RSA or ECC, whose security relies on the hardness of integer factorization or the discrete logarithm, respectively [36]. Post-quantum cryptography seeks to thwart the threat of quantum computers by developing cryptographic primitives based on alternative mathematical problems that cannot be solved efficiently by quantum computers. In recent years, lattice-based cryptography has developed successful post-quantum schemes for essential primitives such as key encapsulation mechanisms (KEMs) and digital

Work by the researchers formerly known as SICQ.
Author list in alphabetical order; see https://www.ams.org/profession/leaders/CultureStatement04.pdf. This work has been supported by UK EPSRC grant EP/S022503/1 and by the German Federal Ministry of Education and Research (BMBF) under the project 6G-RIC (ID 16KISK033).

Date of this document: 2024-03-05.

signatures that will be standardized by NIST. Lattice-based signatures are able to provide fast signing and verification, but have to resort to larger key and signature sizes than were previously acceptable in pre-quantum signatures. For applications where the amount of data transmitted is crucial, these lattice-based schemes may not be a practical option. NIST is therefore looking for other digital signature schemes with properties such as smaller combined public key and signature sizes to ensure a smooth transition to a post-quantum world [39].

A potential solution to this problem is provided by the sole isogeny-based candidate in NIST's new call for signatures – SQIsign [21] – as it is currently the candidate that comes closest to the data sizes transmitted (i.e. the combined size of the signature and the public key) in pre-quantum elliptic curve signatures [28, 29]. SQIsign is most interesting in scenarios that require small signature sizes and fast verification, particularly in those applications where the performance of signing is not the main concern. A few common examples include long-term signatures, specifically public-key certificates, code updates for small devices, authenticated communication with embedded devices or other microcontrollers that solely run verification, and smart cards. For such use cases it is imperative to bring down the cost of verification as much as possible.

Performance bottlenecks in SQIsign. The bottleneck of verification in SQIsign is the computation of an isogeny of fixed degree 2^e with $e \approx (15/4)\log(p)$, where p denotes the prime one is working over, e.g. $\log(p) \approx 256$ for NIST Level I security. However, the rational 2-power torsion, from here on denoted as the $2^\bullet$-torsion, is limited, since we work with supersingular elliptic curves over $\mathbb{F}_{p^2}$ of order $(p+1)^2$ and $(p-1)^2$. This sets a theoretical limit of $2^{\log p}$ for the $2^\bullet$-torsion. Therefore, the verifier needs to perform several *blocks* of degree $2^\bullet$ to complete the full 2^e-isogeny, where each of these blocks involves costly steps such as computing a $2^\bullet$-torsion basis or isogeny kernel generator. Hence, in general, a smaller number of blocks improves the performance of verification.

On the other hand, the bottleneck in signing is the computation of several T-isogenies for odd smooth $T \approx p^{5/4}$. Current implementations of SQIsign therefore require $T \mid (p-1)(p+1)$, such that $\mathbb{F}_{p^2}$-rational points are available for efficient T-isogeny computations. The performance of this step is dominated by the smoothness of T, i.e., its largest prime factor.

While this additional divisibility requirement theoretically limits the maximal $2^\bullet$-torsion to roughly $p^{3/4}$, current techniques for finding SQIsign-friendly primes suggest that achieving this with acceptable smoothness of T is infeasible [10, 12, 14, 18, 21]. In particular, the NIST submission of SQIsign uses a prime with rational 2^{75}-torsion and 1973 as largest factor of T. Since $e \approx (15/4) \cdot 256 = 960$, this means that the verifier has to perform $\lceil e/75 \rceil = 13$ costly isogeny blocks. Increasing the $2^\bullet$-torsion further is difficult as it decreases the probability of finding a smooth and large enough T for current implementations of SQIsign.

Our contributions. In this work, we deploy a range of techniques to increase the $2^\bullet$-torsion and push the SQIsign verification cost far below the state of the art.

Alongside these technical contributions, we aim to give an accessible description of SQIsign, focusing primarily on verification, which solely uses elliptic curves and isogenies and does not require knowledge of quaternion algebras.

Even though we target faster verification, our main contribution is signing with field extensions. From this, we get a much weaker requirement on the prime p, which in turn enables us to increase the size of the $2^\bullet$-torsion.

Focusing on NIST Level I security, we study the range of possible $2^\bullet$-torsion to its theoretical maximum, and measure how its size correlates to verification time through an implementation that uses an equivalent to the number of field multiplications as cost metric. Compared to the state of the art, increasing the $2^\bullet$-torsion alone makes verification almost 1.7 times faster. Further, we implement the new signing procedure as proof-of-concept in SageMath and show that signing times when signing with field extensions are in the same order of magnitude as when signing only using operations in $\mathbb{F}_{p^2}$.

For verification, in addition to implementing some known general techniques for improvements compared to the reference implementation provided in the NIST submission of SQIsign, we show that increasing the $2^\bullet$-torsion also opens up a range of optimisations that were previously not possible. For instance, large $2^\bullet$-torsion allows for an improved challenge-isogeny computation and improved basis and kernel generation. Furthermore, we show that size-speed trade-offs as first proposed by De Feo, Kohel, Leroux, Petit, and Wesolowski [21] become especially worthwhile for large $2^\bullet$-torsion. When pushing the $2^\bullet$-torsion to its theoretical maximum, this even allows for uncompressed signatures, leading to significant speed-ups at the cost of roughly doubling the signature sizes.

For two specific primes with varying values of $2^\bullet$-torsion, we combine all these speed-ups, and measure the performance of verification. Compared to the implementation of the SQIsign NIST submission [12], we reach a speed-up up to a factor 2.70 at NIST Level I when keeping the signature size of 177 bytes. When using our size-speed trade-offs, we reach a speed-up by a factor 3.11 for signatures of 187 bytes, or a factor 4.46 for uncompressed signatures of 322 bytes. Compared to the state of the art [31], these speed-ups are factors 2.07, 2.38 and 3.41 respectively.

Related work. De Feo, Kohel, Leroux, Petit, and Wesolowski [21] published the first SQIsign implementation, superseded by the work of De Feo, Leroux, Longa, and Wesolowski [22]. Subsequently, Lin, Wang, Xu, and Zhao [31] introduced several improvements for this implementation. The NIST submission of SQIsign [12] features a new implementation that does not rely on any external libraries. Since this is the latest and best documented implementation, we will use it as a baseline for performance comparison, and refer to it as SQIsign (NIST). Since the implementation by Lin, Wang, Xu, and Zhao [31] is not publicly available, we included their main improvement for verification in SQIsign (NIST), and refer to this as SQIsign (LWXZ).

Dartois, Leroux, Robert, and Wesolowski [19] recently introduced SQIsignHD, which massively improves the signing time in SQIsign, in addition to a number of other benefits, but at the cost of a still unknown slowdown in verification. This could make SQIsignHD an interesting candidate for applications that prioritise the combined cost of signing and verification over the sole cost of verification.

Recent work by Eriksen, Panny, Sotáková, and Veroni [24] explored the feasibility of computing the Deuring correspondence (see Section 2.2) for *general* primes p via using higher extension fields. We apply the same techniques and tailor them to *specialised* primes for use in the signing procedure of SQIsign.

Overview. The rest of the paper is organised as follows. Section 2 recalls the necessary background, including a high-level overview of SQIsign. Section 3 describes how using field extensions in signing affects the cost and relaxes requirements on the prime. Section 4 analyses how the size of the $2^{\bullet}$-torsion correlates to verification time. Section 5 presents optimisations enabled by the increased $2^{\bullet}$-torsion, while Section 6 gives further optimisations enabled by increased signature sizes. Finally, Section 7 gives some example parameters, and measures their performance compared to the state of the art.

Availability of software. We make our Python and SageMath software publically available under the MIT licence at

https://github.com/TheSICQ/ApresSQI.

2 Preliminaries

Throughout this paper, p denotes a prime number and $\mathbb{F}_{p^k}$ the finite field with p^k elements, where $k \in \mathbb{Z}_{>0}$.

2.1 Elliptic curves and their endomorphism rings.

We first give the necessary geometric background to understand the SQIsign signature scheme. For a more general exposition we refer to Silverman [38].

Isogenies. An isogeny $\varphi : E_1 \to E_2$ between two elliptic curves E_1, E_2 is a non-constant morphism that sends the identity of E_1 to the identity of E_2. The degree $d = \deg(\varphi)$ of an isogeny is its degree as a rational map. If the degree d of an isogeny φ has the prime factorisation $d = \prod_{i=1}^{n} \ell_i^{e_i}$, we can decompose φ into the composition of e_i isogenies of degree ℓ_i for $i = 1$ to n. For every isogeny $\varphi : E_1 \to E_2$, there is a (unique) *dual* isogeny $\widehat{\varphi} : E_2 \to E_1$ that satisfies $\widehat{\varphi} \circ \varphi = [\deg(\varphi)]$, the multiplication-by-$\deg(\varphi)$ map on E_1. Similarly, $\varphi \circ \widehat{\varphi}$ is the multiplication by $\deg(\varphi)$ on E_2.

A *separable* isogeny is described, up to isomorphism, by its kernel, a group of order d. Given a kernel G of prime order d, we can compute the corresponding isogeny $\phi : E \to E/G$ using Vélu's formulas [41] in $\widetilde{O}(d)$. Bernstein, De Feo, Leroux, and Smith [8] showed that this can be asymptotically reduced to $\widetilde{O}(\sqrt{d})$ using $\sqrt{\text{élu}}$ formulas. In Section 2.5, we return to the topic of computing isogenies and give a more detailed discussion.

Endomorphism rings. An isogeny from a curve E to itself is called an *endomorphism.* For example, for any integer n, the multiplication-by-n map is an endomorphism. Another, not necessarily distinct, example for elliptic curves defined over $\mathbb{F}_q$ is the *Frobenius endomorphism* $\pi : (x, y) \mapsto (x^q, y^q)$.

The set of endomorphisms $\mathrm{End}(E)$ of an elliptic curve E forms a ring under (pointwise) addition and composition of isogenies. The endomorphism ring of $E/\overline{\mathbb{F}}_p$ is either isomorphic to an imaginary quadratic order, or to a maximal order in a quaternion algebra ramified at p and ∞ (which will be defined in Section 2.2). In the latter case, we say that E is *supersingular*, and from this point forward, E will denote a supersingular curve.

Supersingular elliptic curves and their isomorphism classes. We will mostly consider supersingular elliptic curves *up to isomorphism*, and thus work with isomorphism classes of these curves. Throughout, we will exploit the fact that every isomorphism class of supersingular curves has a model over $\mathbb{F}_{p^2}$, such that the p^2-power Frobenius π is equal to the multiplication-by-$(-p)$ map. Such curves E are $\mathbb{F}_{p^2}$-isogenous to curves defined over $\mathbb{F}_p$, and satisfy

$$E(\mathbb{F}_{p^{2k}}) = E\left[p^k - (-1)^k\right] \cong \mathbb{Z}/\left(p^k - (-1)^k\right)\mathbb{Z} \oplus \mathbb{Z}/\left(p^k - (-1)^k\right)\mathbb{Z}, \qquad (1)$$

while their quadratic twist over $\mathbb{F}_{p^{2k}}$, which we will denote E_k^t, satisfies

$$E_k^t(\mathbb{F}_{p^{2k}}) = E\left[p^k + (-1)^k\right] \cong \mathbb{Z}/\left(p^k + (-1)^k\right)\mathbb{Z} \oplus \mathbb{Z}/\left(p^k + (-1)^k\right)\mathbb{Z}. \qquad (2)$$

For such curves, for any positive integer $N \mid p^k \pm 1$, the full N-torsion group $E[N]$ is defined over $\mathbb{F}_{p^{2k}}$, either on the curve itself, or on its twist.

The isogeny problem. The fundamental hard problem underlying the security of all isogeny-based primitives is the following: given two elliptic curves E_1, E_2 defined over $\mathbb{F}_{p^2}$ find an isogeny $\phi : E_1 \to E_2$. The best classical attack against this problem is due to Delfs and Galbraith [23] which runs in time $\widetilde{O}(\sqrt{p})$, and quantum attack due to Biasse, Jao, and Sankar [9] that runs in $\widetilde{O}(\sqrt[4]{p})$. A related problem, which will be useful in the context of SQIsign, is the *endomorphism ring problem*, which asks, given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, to find the endomorphism ring $\mathrm{End}(E)$. Wesolowski [43] showed that this is equivalent to the isogeny problem under reductions of polynomial expected time, assuming the generalised Riemann hypothesis, and further, Page and Wesolowski [33] recently showed that the endomorphism ring problem is equivalent to the problem of computing one endomorphism.

2.2 Quaternion algebras and the Deuring correspondence

We give the necessary arithmetic background to understand the signing procedure of SQIsign at a high level.[5] The heart of the signing procedure in SQIsign

[5]This section is only necessary for Section 2.3 and Section 3, as all other sections are concerned only with SQIsign verification, which will only use well-known isogeny terminology. In contrast, signing heavily relies on the arithmetic of quaternion algebras.

lies in the Deuring correspondence, which connects the geometric world of supersingular curves from Section 2.1 to the arithmetic world of quaternion algebras. For more details on quaternion algebras, we refer to Voight [42].

Quaternion algebras, orders and ideals. Quaternion algebras are four-dimensional $\mathbb{Q}$-algebras, generated by four elements $1, i, j, k$ following certain multiplication rules. For SQIsign, we work in the quaternion algebra *ramified* at p and ∞. When $p \equiv 3 \pmod 4$, one representation of such a quaternion algebra is given by $\mathcal{B}_{p,\infty} = \mathbb{Q} + i\mathbb{Q} + j\mathbb{Q} + k\mathbb{Q}$ with multiplication rules

$$i^2 = -1,\ j^2 = -p,\ ij = -ji = k.$$

For an element $\alpha = x + yi + zj + wk \in \mathcal{B}_{p,\infty}$ with $x, y, z, w \in \mathbb{Q}$, we define its *conjugate* to be $\bar{\alpha} = x - yi - zj - wk$, and its *reduced norm* to be $n(\alpha) = \alpha\bar{\alpha}$.

We are mainly interested in *lattices* in $\mathcal{B}_{p,\infty}$, defined as full-rank $\mathbb{Z}$-modules contained in $\mathcal{B}_{p,\infty}$, i.e., abelian groups of the form

$$\alpha_1\mathbb{Z} + \alpha_2\mathbb{Z} + \alpha_3\mathbb{Z} + \alpha_4\mathbb{Z},$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \mathcal{B}_{p,\infty}$ are linearly independent. If a lattice $\mathcal{O} \subset \mathcal{B}_{p,\infty}$ is also a subring of $\mathcal{B}_{p,\infty}$, i.e., it contains 1 and is closed under multiplication, then $\mathcal{O}$ is called an *order*. Orders that are not strictly contained in any other order are called *maximal* orders. From this point on, we only consider maximal orders.

A lattice I that is closed under multiplication by an order $\mathcal{O}$ on the left is called a *left* (resp. *right*) *$\mathcal{O}$-ideal*. We refer to $\mathcal{O}$ as the left (resp. right) order of I. When $\mathcal{O}$ is the left order of I and $\mathcal{O}'$ the right order of I, we define I to be a *connecting* $(\mathcal{O}, \mathcal{O}')$*-ideal*.[6] A left $\mathcal{O}$-ideal I that is also contained in $\mathcal{O}$ is called an *integral* ideal. From this point on, we only deal with integral left ideals and simply refer to them as ideals.

The *norm* of an ideal I is the greatest common divisor of the reduced norms of the elements of I, whereas the *conjugate* $\bar{I}$ of an ideal I is the ideal consisting of the conjugates of the elements of I. Two ideals I and J are said to be equivalent if $I = J\alpha$ for some $\alpha \in \mathcal{B}_{p,\infty}^{\times}$ and is denoted $I \sim J$. Equivalent ideals have equal left orders and isomorphic right orders.

The Deuring correspondence. Given an elliptic curve E with $\mathrm{End}(E) \cong \mathcal{O}$, there is a one-to-one correspondence between separable isogenies from E and left $\mathcal{O}$-ideals I of norm coprime to p. Given an isogeny φ, we denote the corresponding ideal I_φ, and conversely, given an ideal I, we denote the corresponding isogeny φ_I. The Deuring correspondence acts like a dictionary: a given isogeny $\varphi : E \to E'$ corresponds to an ideal I_φ with left order $\mathcal{O} \cong \mathrm{End}(E)$ and right order $\mathcal{O}' \cong \mathrm{End}(E')$. Furthermore, the degree of φ is equal to the norm of I_φ and the dual isogeny $\widehat{\varphi}$ corresponds to the conjugate $\overline{I_\varphi} = I_{\widehat{\varphi}}$. Equivalent ideals I, J have isomorphic right orders and the corresponding isogenies φ_I, φ_J have isomorphic codomains.

[6]Note that $\mathcal{O}$ and $\mathcal{O}'$ need not be distinct.

The (generalised) KLPT algorithm. The KLPT algorithm, introduced by Kohel, Lauter, Petit, and Tignol [30], is a purely quaternionic algorithm, but has seen a variety of applications in isogeny-based cryptography due to the Deuring correspondence. Given an ideal I, KLPT finds an equivalent ideal J of prescribed norm. The drawback is that the norm of the output J will be comparatively large.

For example, the KLPT algorithm is used to compute isogenies between two curves of known endomorphism ring. Given two maximal orders $\mathcal{O}, \mathcal{O}'$, translating the standard choice[7] of connecting ideal I to its corresponding isogeny is hard. However, by processing I through KLPT first, we can find an equivalent ideal J of smooth norm, allowing us to compute φ_J. This is essential for computing the response in SQIsign.

The original KLPT algorithm only works for $\mathcal{O}_0$-ideals, where $\mathcal{O}_0$ is a maximal order of a special form.[8] This was generalised by De Feo, Kohel, Leroux, Petit, and Wesolowski [21] to work for arbitrary orders $\mathcal{O}$, albeit at the cost of an even larger norm bound for the output. Note that SQIsign utilizes both versions.

2.3 SQIsign

Next, we give a high-level description of signing and verification in SQIsign. SQIsign is a signature scheme based on an underlying Sigma protocol that proves knowledge of a *secret* (non-scalar) endomorphism $\alpha \in \operatorname{End}(E_A)$ for some *public* curve E_A. At its core, the prover shows this knowledge by being able to compute an isogeny φ from E_A to some random curve E_2.

A high-level description of the SQIsign Sigma protocol is given below(see also Figure 1) .

Setup: Fix a prime number p and supersingular elliptic curve $E_0/\mathbb{F}_{p^2}$ with known endomorphism ring.

Key generation: Compute a secret key $\varphi_A : E_0 \to E_A$, giving the prover knowledge of $\operatorname{End}(E_A)$, with corresponding public verification key E_A.

Commit: The prover generates a random commitment isogeny $\varphi_{\text{com}} : E_0 \to E_1$, and sends E_1 to the verifier.

Challenge: The verifier computes a random challenge isogeny $\varphi_{\text{chall}} : E_1 \to E_2$, and sends φ_{chall} to the prover.

Response: The prover uses the knowledge of φ_{com} and φ_{chall} to compute $\operatorname{End}(E_2)$, allowing the prover to compute the response isogeny $\varphi_{\text{resp}} : E_A \to E_2$, by translating an ideal computed using the generalised KLPT algorithm, as described at the end of Section 2.2.

The verifier needs to check that φ_{resp} is an isogeny from E_A to E_2.[9] Assuming the hardness of the endomorphism ring problem, the protocol is sound: if the

[7] $I = N\mathcal{O}\mathcal{O}'$, where N is the smallest integer making I integral.

[8] Specifically, it only works for special, p-extremal orders. An example of such an order when $p \equiv 3 \pmod 4$ is $\operatorname{End}(E_0)$ where $j(E_0) = 1728$.

[9] Additionally, $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ needs to be cyclic. Observe that otherwise, the soundness proof might return a scalar endomorphism.

$$\begin{tikzcd}
& E_0 \arrow[r, "\varphi_{\mathrm{com}}"'] \arrow[dl, dashed, "\varphi_A"] & E_1 \arrow[d, "\varphi_{\mathrm{chall}}"'] \\
E_A \arrow[rr, "\varphi_{\mathrm{resp}}"] & & E_2
\end{tikzcd}$$

Fig. 1: The SQIsign protocol with three phases: commitment φ_{com}, challenge φ_{chall}, and response φ_{resp}.

prover is able to respond to two different challenges φ_{chall}, $\varphi'_{\mathrm{chall}}$ with φ_{resp} and φ'_{resp}, the prover knows an endomorphism of the public key E_A, namely $\widehat{\varphi'_{\mathrm{resp}}} \circ \varphi'_{\mathrm{chall}} \circ \widehat{\varphi_{\mathrm{chall}}} \circ \varphi_{\mathrm{resp}}$. Proving zero-knowledge is harder and relies on the output distribution of the generalised KLPT algorithm. Note that KLPT is needed for computing the response:[10] while setting $\varphi_{\mathrm{resp}} = \varphi_{\mathrm{chall}} \circ \varphi_{\mathrm{com}} \circ \widehat{\varphi_A}$ gives an isogeny from E_A to E_2, this leaks the secret φ_A.[11] For a further discussion on zero-knowledge, we refer to the original SQIsign articles [21, 22].

Remark 1. The best-known attacks against SQIsign are the generic attacks against the endomorphism ring problem. As discussed in Section 2.1, their run time depends only on the size of p and, with high probability, do not recover the original secret isogeny, but rather a different isogeny between the same curves. Therefore, their complexity should be unaffected by the changes we introduce to the SQIsign protocol in Section 3, as for these attacks it does not matter whether the original secret isogeny had kernel points defined over a larger extension field. In short, the changes in this work do not affect the security of SQIsign.

Verification. Using the Fiat–Shamir heuristic, the SQIsign Sigma protocol is transformed into a signature scheme. This means that a signature on the message $\texttt{msg}$ is of the form $\sigma = (\varphi_{\mathrm{resp}}, \texttt{msg}, E_1)$. For efficiency, φ_{resp} is compressed, and E_1 is replaced by a description of φ_{chall}. Thus, given the signature σ and public key E_A, the verifier recomputes the response isogeny $\varphi_{\mathrm{resp}} : E_A \to E_2$ and the (dual of the) challenge isogeny $\widehat{\varphi_{\mathrm{chall}}} : E_2 \to E_1$, and then verifies that the hash $H(\texttt{msg}, E_1)$ indeed generates φ_{chall}.

The isogeny φ_{resp} is of degree 2^e with $e = \lceil \frac{15}{4} \log(p) \rceil + 25$ [12, §7.2.3], where 2^e corresponds to the output size of the generalised KLPT algorithm. The bottleneck in verification is the (re)computation of φ_{resp} in $\lceil e/f \rceil$ steps of size 2^f. Accelerating this will be the focus of this paper.

[10] Alternatively, one can replace the connecting ideal with the shortest equivalent ideal, and translate it by embedding it in an isogeny between higher-dimensional abelian varieties, as shown in SQIsignHD [19]

[11] Further, this is not a valid response, since the composition with $\widehat{\varphi_{\mathrm{chall}}}$ is not cyclic.

2.4 SQIsign-friendly primes

Next, we give more details on the parameter requirements in SQIsign. We refer to the original SQIsign works [12, 21, 22] for a detailed description of their origins.

SQIsign prime requirements. The main bottleneck of SQIsign is the computation of isogenies. Recall from Equations (1) and (2) that, when working with supersingular elliptic curves $E/\mathbb{F}_{p^2}$, we have $E(\mathbb{F}_{p^2}) = E[p \pm 1]$. Thus, to use x-only arithmetic over $\mathbb{F}_{p^2}$, SQIsign restricts to computing isogenies of *smooth* degree $N \mid (p^2 - 1)$. Finding SQIsign-friendly primes reduces to finding primes p, with $p^2 - 1$ divisible by a large, smooth number. More explicitly, for a security level λ, the following parameters are needed:

- A prime p of bitsize $\log_2(p) \approx 2\lambda$ with $p \equiv 3 \mod 4$.
- The torsion group $E[2^f]$ as large as possible, that is $2^f \mid p + 1$.
- A smooth odd factor $T \mid (p^2 - 1)$ of size roughly $p^{5/4}$.
- The degree of φ_{com}, $D_{\text{com}} \mid T$, of size roughly $2^{2\lambda} \approx p$.
- The degree of φ_{chall}, $D_{\text{chall}} \mid 2^f T$, of size roughly $2^\lambda \approx p^{1/2}$.
- Coprimality between D_{com} and D_{chall}.

To achieve NIST Level I, III, and V security, we set the security parameter as $\lambda = 128, 192, 256$, respectively. Concretely, this means that, for each of these security parameters, we have $\log p \approx 256, 384, 512$, and $\log T \approx 320, 480, 640$, with f as large as possible given the above restrictions. The smoothness of T directly impacts the signing time, and the problem of finding primes p with a large enough T that is reasonably smooth is difficult. We refer to recent work on this problem for techniques to find suitable primes [10, 12, 14, 18, 21, 22].

The crucial observation for this work is that T occupies space in $p^2 - 1$ that limits the size of f, hence current SQIsign primes balance the smoothness of T with the size of f.

Remark 2. SQIsign (NIST) further requires $3^g \mid p+1$ such that $D_{\text{chall}} = 2^f \cdot 3^g \geq p^{1/2}$ and $D_{\text{chall}} \mid p + 1$. While this is not a strict requirement in the theoretical sense, it facilitates efficiency of computing φ_{chall}. From this point on, we ensure that this requirement is always fulfilled.

Remark 3. Since the curves E and their twists E^t satisfy

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p \pm 1)\mathbb{Z} \oplus \mathbb{Z}/(p \pm 1)\mathbb{Z},$$

and we work with both simultaneously, choosing T and f is often incorrectly described as choosing divisors of $p^2 - 1$. There is a subtle issue here: even if 2^f divides $p^2 - 1$, $E[2^f]$ may not exist as a subgroup of $\langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle \subseteq E(\mathbb{F}_{p^4})$, where $\rho : E \to E^t$ is the twisting isomorphism. While this does not usually matter in the case of SQIsign (we pick 2^f as a divisor of $p + 1$, and T is odd), this becomes a problem when working over multiple extension fields. In Section 3.2, we make this precise and reconcile it using Theorem 1.

2.5 Computing rational isogenies from irrational generators

Finally, to facilitate signing with field extensions, we recall the techniques for computing $\mathbb{F}_{p^2}$-rational isogenies, i.e., isogenies defined over $\mathbb{F}_{p^2}$, generated by *irrational* kernel points, that is, not defined over $\mathbb{F}_{p^2}$. In the context of this paper, we again stress that such isogenies will only be computed by the signer; the verifier will only work with points in $\mathbb{F}_{p^2}$.

The main computational task of most isogeny-based cryptosystems (including SQIsign) lies in evaluating isogenies given the generators of their kernels. Explicitly, given an elliptic curve $E/\mathbb{F}_q$, a point $K \in E(\mathbb{F}_{q^k})$ such that $\langle K \rangle$ is *defined over* $\mathbb{F}_q$,[12] and a list of points $(P_1, P_2, \ldots, P_n)$ in E, we wish to compute the list of points $(\varphi(P_1), \varphi(P_2), \ldots, \varphi(P_n))$, where φ is the separable isogeny with $\ker \varphi = \langle K \rangle$. Since we work with curves E whose p^2-Frobenius π is equal to the multiplication-by-$(-p)$ map (see Section 2.1), *every* subgroup of E is closed under the action of $\mathrm{Gal}(\bar{\mathbb{F}}_{p^2}/\mathbb{F}_{p^2})$, hence every isogeny from E can be made $\mathbb{F}_{p^2}$-rational, by composing with the appropriate isomorphism.

Computing isogenies of smooth degree. Recall from Section 2.1 that the isogeny factors as a composition of small prime degree isogenies, which we compute using Vélu-style algorithms. For simplicity, for the rest of the section, we therefore assume that $\langle K \rangle$ is a subgroup of order $\ell > 2$, where ℓ is a small prime.

At the heart of these Vélu-style isogeny formulas is evaluating the kernel polynomial. Pick any subset $S \subseteq \langle K \rangle$ such that $\langle K \rangle = S \sqcup -S \sqcup \{\infty\}$. Then the kernel polynomial can be written as

$$f_S(X) = \prod_{P \in S} (X - x(P)). \tag{3}$$

Here, the generator K can be either a rational point, i.e., lying in $E(\mathbb{F}_q)$, or an irrational point, i.e., lying in $E(\mathbb{F}_{q^k})$ for $k > 1$, but whose group $\langle K \rangle$ is defined over $\mathbb{F}_q$. Next, we discuss how to solve the problem efficiently in the latter case.

Irrational generators. For $K \notin E(\mathbb{F}_q)$ of order ℓ, we can speed up the computation of the kernel polynomial using the action of Frobenius. This was used in two recent works [6, 24], though the general idea was used even earlier [40].

As $\langle K \rangle$ is defined over $\mathbb{F}_q$, we know that the q-power Frobenius π acts as an endomorphism on $\langle K \rangle \subseteq E(\mathbb{F}_{p^k})$ and thus maps K to a multiple $[\gamma]K$ for some $\gamma \in \mathbb{Z}$. This fully determines the action on $\langle K \rangle$, i.e., $\pi|_{\langle K \rangle}$ acts as $P \mapsto [\gamma]P$ for all $P \in \langle K \rangle$. For the set S as chosen above, this action descends to an action on its x-coordinates $X_S = \{x(P) \in \mathbb{F}_{q^k} \mid P \in S\}$ and thus partitions X_S into orbits $\{x(P), x([\gamma]P), x([\gamma^2]P), \ldots\}$ of size equal to the order of γ in $(\mathbb{Z}/\ell\mathbb{Z})^\times/\{1, -1\}$.

If we pick one representative $P \in S$ per orbit, and call this set of points S_0, we can compute the kernel polynomial (3) as a product of the minimal polynomials

[12] That is, the group $\langle K \rangle$ is closed under the action of $\mathrm{Gal}(\bar{\mathbb{F}}_q/\mathbb{F}_q)$.

$\mu_{x(P)}$ of the $x(P) \in \mathbb{F}_{q^k}$ for these $P \in S_0$, with each $\mu_{x(P)}$ defined over $\mathbb{F}_q$, as

$$f_S(X) = \prod_{P \in S_0} \mu_{x(P)}(X), \tag{4}$$

where μ_β denotes the minimal polynomial of β over $\mathbb{F}_q$.

To compute $f_S(\alpha)$ for $\alpha \in \mathbb{F}_q$, we only require the smaller polynomial $f_{S_0}(X)$ and compute $\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(f_{S_0}(\alpha))$, as

$$\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(f_{S_0}(\alpha)) = \prod_{\pi \in G} \pi(f_{S_0}(\alpha)) = \prod_{P \in S_0} \prod_{\pi \in G} (\alpha - \pi(x(P))) = \prod_{P \in S_0} \mu_{x(P)}(\alpha),$$

where $G = \mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q)$, as per Banegas, Gilchrist, Le Dévéhat, and Smith [6]. This allows us to compute the image under f_S of x-values of points in $E(\mathbb{F}_q)$, but only works for values in $\mathbb{F}_q$. To evaluate $f_S(\alpha)$ for general $\alpha \in \overline{\mathbb{F}}_p$, i.e. to compute the image of a point in $E(\overline{\mathbb{F}}_p)$, we instead use the larger polynomial $f_S(X)$, which we compute, as in Equation (4), as a product of the minimal polynomials $\mu_{x(P)}$, where we use Shoup's algorithm [37] to compute each $\mu_{x(P)}$ given $x(P)$. Computing $f_S(X)$ requires a total of $O(\ell k) + \widetilde{O}(\ell)$ operations, with k such that each $x(P) \in \mathbb{F}_{q^k}$. Evaluation f_S at α takes $\widetilde{O}(\ell k')$ operations, with k' the smallest value such that $\alpha \in \mathbb{F}_{q^{k'}}$ [24, Section 4.3].

Remark 4. The biggest drawback to using this technique is that $\sqrt{\text{élu}}$ is no longer effective, as we would need to work in the smallest field where both the isogeny generator and the x-value of the point we are evaluating are defined in.

3 Signing with extension fields

By allowing torsion T from extension fields, we enable more flexibility in choosing SQIsign primes p, thus enabling a larger $2^\bullet$-torsion. Such torsion T requires us to compute rational isogenies with kernel points in extension fields $\mathbb{F}_{p^{2k}}$. This section describes how to adapt SQIsign's signing procedure to enable such isogenies, and the increased cost this incurs. In particular, we describe two approaches for T: allowing torsion T from a particular extension field $\mathbb{F}_{p^{2k}}$, or from all extension fields $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$. The first approach means that we can look for T dividing an integer of bit size $\Theta(k \log p)$, and the second approach allows for $\Theta(k^2 \log p)$. In Section 4, we explore how increased $2^\bullet$-torsion affects verification.

Throughout this section, we will reuse the notation from Section 2.4 to describe the various parameters related to SQIsign.

3.1 Changes in the signing procedure

Recall from Section 2.3 that the signing operation in SQIsign requires us to work with both elliptic curves and quaternion algebras, and to translate back and forth between these worlds. Note that the subroutines that work solely with objects in the quaternion algebra $\mathcal{B}_{p,\infty}$, including all operations in KLPT and

Algorithm 1 $\mathsf{IdealToIsogeny}_D(I)$

```
Input: $I$ a left $\mathcal{O}_0$-ideal of norm dividing $D$
Output: $\varphi_I$
 1: Compute $\alpha$ such that $I = \mathcal{O}_0\langle\alpha, \mathrm{nrd}(I)\rangle$
 2: Let $\mathbf{A} = [1, i, \frac{i+j}{2}, \frac{1+k}{2}]$ denote a basis of $\mathcal{O}_0$
 3: Compute $\mathbf{v}_{\bar{\alpha}} := [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\bar{\alpha}} = \bar{\alpha}$
 4: for $\ell^e \mid\mid D$ do
 5:     $\bar{\alpha}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} := x_1\mathbf{I} + x_2(i|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_3(\frac{i+j}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_4(\frac{1+k}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle})$
 6:     Let $a, b, c, d$ be integers such that $\bar{\alpha}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
 7:     $K_{\ell^e} := [a]P_{\ell^e} + [c]Q_{\ell^e}$
 8:     if $\mathrm{ord}(K_{\ell^e}) < \ell^e$ then
 9:         $K_{\ell^e} = [b]P_{\ell^e} + [d]Q_{\ell^e}$
10: Set $\varphi_I$ to be the isogeny generated by the points $K_{\ell^e}$.
11: return $\varphi_I$
```

its derivatives, are indifferent to what extension fields the relevant torsion groups lie in. Hence, a large part of signing is unaffected by torsion from extension fields.

In fact, the only subroutines that are affected by moving to extension fields are those relying on Algorithm 1, $\mathsf{IdealToIsogeny}_D$, which translates $\mathcal{O}_0$-ideals I of norm dividing D to their corresponding isogenies φ_I. $\mathsf{IdealToIsogeny}_D$ is not used during verification, and is used only in the following parts of signing:

Commitment: The signer translates a random ideal of norm D_{com} to its corresponding isogeny, using one execution of $\mathsf{IdealToIsogeny}_{D_{\text{com}}}$.

Response: The signer translates an ideal of norm 2^e to its corresponding isogeny, requiring $2 \cdot \lceil e/f \rceil$ executions of $\mathsf{IdealToIsogeny}_T$.[13]

Remark 5. We will choose parameters such that $2^f \mid p+1$ and $D_{\text{chall}} \mid p+1$, so that $E[2^f]$ and $E[D_{\text{chall}}]$ are defined over $\mathbb{F}_{p^2}$. As a result, the verifier only works in $\mathbb{F}_{p^2}$ and the added complexity of extension fields applies only to the signer.

Adapting ideal-to-isogeny translations to field extensions. To facilitate signing with field extensions, we slightly adapt $\mathsf{IdealToIsogeny}_D$ so that it works with prime powers separately. Note that the additional cost of this is negligible compared to the cost of computing the isogeny from the generators because finding the action of the relevant endomorphisms consists of simple linear algebra. See Algorithm 1 for details.

In Line 5 of Algorithm 1, the notation $\beta|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}$ refers to the action of an endomorphism β on a fixed basis P_{ℓ^e}, Q_{ℓ^e} of $E[\ell^e]$. This action is described by a matrix in $M_2(\mathbb{Z}/\ell^e\mathbb{Z})$. These matrices can be precomputed, hence the only operations in which the field of definition of $E[\ell^e]$ matters are the point additions in Lines 7 and 9, and isogenies generated by each K_{ℓ^e} in Line 10.

[13] The technical details are given by De Feo, Leroux, Longa, and Wesolowski [22].

3.2 Increased torsion availability from extension fields

Next, we detail the two approaches to allow torsion groups from extension fields, which permits more flexibility in choosing the final prime p.

Working with a single field extension of $\mathbb{F}_{p^2}$. Although the choice of solely working in $\mathbb{F}_{p^2}$ occurs naturally,[14] there is no reason *a priori* that this choice is optimal. Instead, we can choose to work in the field $\mathbb{F}_{p^{2k}}$. We emphasise that this does *not* affect signature sizes; the only drawback is that we now perform more expensive $\mathbb{F}_{p^{2k}}$-operations during signing in IdealToIsogeny. The upside, however, is a relaxed prime requirement: we are no longer bound to $E[T] \subseteq \langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle$ and can instead use

$$E[T] \subseteq \langle E(\mathbb{F}_{p^{2k}}), \rho^{-1}(E^t(\mathbb{F}_{p^{2k}}))\rangle.$$

By Equations (1) and (2), we have $E(\mathbb{F}_{p^{2k}}) \cong E[p^k \pm 1]$ and $E^t(\mathbb{F}_{p^{2k}}) \cong E[p^k \mp 1]$, thus we simply get

$$E[T] \subseteq E\left[\frac{p^{2k}-1}{2}\right],$$

since $\langle E[A], E[B]\rangle = E[\mathrm{lcm}(A, B)]$. Hence, by using torsion from $E(\mathbb{F}_{p^{2k}})$, we increase $T \mid (p^2-1)/2$ to $T \mid (p^{2k}-1)/2$. This implies there are $2(k-1)\log p$ more bits available to find T with adequate smoothness.

Working with multiple field extensions of $\mathbb{F}_{p^2}$. Instead of fixing a single higher extension field $\mathbb{F}_{p^{2k}}$, we can also choose to work with multiple field extensions, in particular all fields $\mathbb{F}_{p^{2n}}$, where $1 \leq n \leq k$. The torsion group we can access by this relaxed requirement is described by the following definition.

Definition 1. *Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ and let E_n^t denote an arbitrary quadratic twist of E over $\mathbb{F}_{p^{2n}}$ with respect to the twisting isomorphism $\rho_n : E \to E_n^t$. We define the k-*available torsion* of E to be the group generated by $E(\mathbb{F}_{p^{2n}})$ and $\rho_n^{-1}(E_n^t(\mathbb{F}_{p^{2n}}))$ for $1 \leq n \leq k$.*

Any point P in the k-available torsion can thus be written as a sum

$$P = \sum_{i=1}^{k} (P_i + \rho_n^{-1}(P_i^t))$$

of points $P_i \in E(\mathbb{F}_{p^{2n}})$ and $P_i^t \in E_n^t(\mathbb{F}_{p^{2n}})$. Since the twisting isomorphism keeps the x-coodinate fixed, the computation of this isomorphism can be ignored when using x-only arithmetic, and we simply obtain a sum of points whose x-coordinates lie in $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$. This justifies the name k-available torsion,

[14] It is the smallest field over which every isomorphism class of supersingular elliptic curves has a model.

as we do not have to go beyond $\mathbb{F}_{p^{2k}}$ to do arithmetic with P by working with the summands separately.

The structure of the k-available torsion is completely captured by the following result.

Theorem 1. *Let $p > 2$ be a prime, and let $E/\mathbb{F}_{p^2}$ be a supersingular curve with $\mathrm{tr}(\pi) = \pm 2p$, where π is the Frobenius endomorphism. Then the k-available torsion is precisely the group $E[N]$ with*

$$N = \prod_{n=1}^{k} \varPhi_n(p^2)/2,$$

where $\varPhi_n$ denotes the n-th cyclotomic polynomial.

Lemma 1. *For any integer $m \geq 2$, we have the following identity*

$$\mathrm{lcm}\left(\{m^n - 1\}_{n=1}^{k}\right) = \prod_{n=1}^{k} \varPhi_n(m)$$

where $\varPhi_n$ denotes the n-th cyclotomic polynomial.

Proof. We denote the left-hand side and right-hand side of the equation in the statement of the lemma by LHS and RHS, respectively. We show that any prime power dividing one side, also divides the other.

For any prime ℓ and $e > 0$, if ℓ^e divides the LHS, then, by definition, it divides $m^i - 1 = \prod_{d|i} \varPhi_d(m)$ for some $1 \leq i \leq k$. Hence, it also divides the RHS. Conversely, if ℓ^e divides the RHS, then ℓ^e also divides the LHS. To show this, we need to know when $\varPhi_i(m)$ and $\varPhi_j(m)$ are coprime. We note that

$$\gcd(\varPhi_i(m), \varPhi_j(m)) \mid R$$

where R is the resultant of $\varPhi_i(X)$ and $\varPhi_j(X)$, and a classic result by Apostol [2, Theorem 4.], tells us that

$$\mathrm{Res}(\varPhi_i(X), \varPhi_j(X)) > 1 \Rightarrow i = jm$$

for $i > j$ and some integer m.

Using this, if ℓ^e divides the RHS, then it will also divide the product

$$\prod_{n=1}^{\lfloor k/d \rfloor} \varPhi_{dn}(m),$$

for some integer d, and this product divides the LHS, as it divides $m^{d\lfloor k/d \rfloor} - 1$. $\square$

We can now conclude the proof of Theorem 1.

Proof. From the structure of $E(\mathbb{F}_{p^{2n}})$ (see Remark 3), where E is as in the statement, the k-available torsion can be seen as the group generated by the full torsion groups

$$E[p^n \pm 1]$$

for $1 \leq n \leq k$. Using the fact that

$$\langle E[A], E[B] \rangle = E[\mathrm{lcm}(A, B)],$$

we see that the k-available torsion is $E[N]$ where

$$N = \mathrm{lcm}\left(\{p^n - 1\}_{n=1}^k \cup \{p^n + 1\}_{n=1}^k\right) = \mathrm{lcm}\left(\{p^{2n} - 1\}_{n=1}^k\right)/2,$$

where the last equality only holds for $p > 2$. Applying Lemma 1 with $m = p^2$, we obtain

$$N = \prod_{k=1}^{n} \Phi_k(p^2)/2,$$

$\square$

We find that using all extension fields $\mathbb{F}_{p^{2n}}$, for $1 \leq n \leq k$, increases $T \mid p^2 - 1$ to $T \mid N$, with N as given by Theorem 1. Given that

$$\log(N) = \sum_{n=1}^{k} \log(\Phi_n(p^2)/2) \approx 2 \sum_{n=1}^{k} \phi(n) \log(p),$$

and the fact that $\sum_{n=1}^{k} \phi(n)$ is in the order of $\Theta(k^2)$, where ϕ denotes Euler's totient function, we find that $T \mid N$ gives roughly $k^2 \log(p)$ more bits to find T with adequate smoothness, compared to the $\log(p)$ bits in the classical case of working over $\mathbb{F}_{p^2}$, and $k \log(p)$ bits in the case of working over $\mathbb{F}_{p^{2k}}$. Due to this, we only consider working in multiple field extensions from this point on.

3.3 Cost of signing using extension fields

In SQIsign, operations over $\mathbb{F}_{p^2}$ make up the majority of the cost during signing [22, Section 5.1]. Hence, we can roughly estimate the cost of signing by ignoring purely quaternionic operations, in which case the bottleneck of the signing procedure becomes running $\mathsf{IdealToIsogeny}_T$ as many times as required by the IdealToIsogenyEichler algorithm [22, Algorithm 5] in the response phase. In other words, we estimate the total signing cost from the following parameters:

- f, such that $2^f \mid p + 1$.
- T, the chosen torsion to work with.
- For each $\ell_i^{e_i} \mid T$, the smallest k_i such that $E[\ell_i^{e_i}]$ is defined over $\mathbb{F}_{p^{2k_i}}$.

Since Algorithm 1 works with prime powers separately, we can estimate the cost of a single execution by considering the cost per prime power.

Cost per prime power. For each $\ell_i^{e_i} \mid T$, let k_i denote the smallest integer so that $E[\ell_i^{e_i}] \subseteq E(\mathbb{F}_{p^{2k_i}})$, and let $M(k_i)$ denote the cost of operations in $\mathbb{F}_{p^{2k_i}}$ in terms of $\mathbb{F}_{p^2}$-operations. Computing the generator $K_{\ell_i^{e_i}}$ consists of a few point additions in $E[\ell_i^{e_i}]$, hence is $O(M(k) \cdot e \log \ell)$, while the cost of computing the isogeny generated by $K_{\ell_i^{e_i}}$ comes from computing e isogenies of degree ℓ at a cost of $O(\ell k) + \tilde{O}(\ell)$, using the techniques from Section 2.5.

To compute the whole isogeny, we need to push the remaining generators $K_{\ell_j^{e_j}}$, through this isogeny. To minimize the total cost, we pick the greedy strategy of always computing the smaller ℓ first. This bounds the cost of evaluating K_{ℓ^e} in *other* isogenies by $O(M(k) \cdot \ell)$.

Total cost of signing. Based on the analysis above, we let

$$\textsc{Cost}_p(\ell_i^{e_i}) = c_1 M(k_i) e \log \ell + c_2 e_i \ell_i k_i + c_3 e_i \ell_i \log(\ell_i) + c_4 M(k_i) \ell$$

where k_i, and $M(k)$ are as before, and c_i are constants corresponding to the differences in the asymptotic complexities. Since we can estimate the total cost of executing $\mathsf{IdealToIsogeny}_T$ by summing the cost of each maximal prime power divisor of T, and observing that signing consists of executing $\mathsf{IdealToIsogeny}_{D_{\mathsf{com}}}$ one time, and $\mathsf{IdealToIsogeny}_T$ a total of $2 \cdot \lceil e/f \rceil$ times, we get a rough cost estimate of signing as

$$\textsc{SigningCost}_p(T) = (2 \cdot \lceil e/f \rceil + 1) \cdot \sum_{\ell_i^{e_i} \mid T} \textsc{Cost}_p(\ell_i^{e_i}).$$

In Section 7, we use this function to pick p and T minimising this cost. While this cost metric is very rough, we show that our implementation roughly matches the times predicted by this function. Further, we show that this cost metric suggests that going to extension fields gives signing times within the same order of magnitude as staying over $\mathbb{F}_{p^2}$, even when considering the additional benefit of using $\sqrt{\text{élu}}$ to compute isogenies in the latter case.

4 Effect of increased $2^\bullet$-torsion on verification

In Section 3, we showed that signing with extension fields gives us more flexibility in choosing the prime p, and, in particular, allows us to find primes with rational 2^f-torsion for larger f. In this section, we analyse how such an increase in $2^\bullet$-torsion affects the cost of $\mathsf{SQIsign}$ verification, e.g., computing φ_{resp} and $\widehat{\varphi_{\text{chall}}}$, in terms of $\mathbb{F}_p$-multiplications,[15] taking the $\mathsf{SQIsign}$ (NIST) implementation (with no further optimisations) as the baseline for comparison.

[15] As standard, we denote multiplications by $\mathbf{M}$, squarings by $\mathbf{S}$, and additions by $\mathbf{a}$.

4.1 Detailed description of verification

Before giving an in-depth analysis of verification performance, we give a detailed description of how verification is executed. Recall that a SQIsign signature σ for a message `msg` created by a signer with secret signing key $\varphi_A : E_0 \to E_A$ proves knowledge of an endomorphism on E_A by describing an isogeny $\varphi_{\text{resp}} : E_A \to E_2$ of degree 2^e(see Figure 1). A given message `msg` is hashed on E_1 to a point K_{chall} of order D_{chall}, hence represents an isogeny $\varphi_{\text{chall}} : E_1 \to E_2$. A signature is valid if the composition of φ_{resp} with $\widehat{\varphi_{\text{chall}}}$ is cyclic of degree $2^e \cdot D_{\text{chall}}$.

Thus, to verify a signature σ, the verifier must **(a)** recompute φ_{resp}, **(b)** compute the dual of φ_{chall}, to confirm that both are well-formed, and finally **(c)** recompute the hash of the message `msg` to confirm the validity of the signature.

In SQIsign, the size of the sample space for φ_{chall} impacts soundness, a key security property for signature schemes. In SQIsign (NIST), to obtain negligible soundness error (in the security parameter λ) the message is hashed to an isogeny of degree $D_{\text{chall}} = 2^f \cdot 3^g$ so that the size of cyclic isogenies of degree D_{chall} is larger than 2^λ. In contrast, when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$.

The signature σ consists of a compressed description of the isogenies φ_{resp} and $\widehat{\varphi_{\text{chall}}}$. For $f < \lambda$ and $D_{\text{chall}} = 2^f \cdot 3^g$ it is of the form

$$\sigma = (b, s^{(1)}, \ldots, s^{(n)}, r, b_2, s_2, b_3, s_3)$$

with $s^{(j)}, s_2 \in \mathbb{Z}/2^f\mathbb{Z}$, $s_3 \in \mathbb{Z}/3^g\mathbb{Z}$, $r \in \mathbb{Z}/2^f 3^g\mathbb{Z}$, and $b, b_2, b_3 \in \{0,1\}$. If $f \geq \lambda$, we set $D_{\text{chall}} = 2^f$ and have $s_2 \in \mathbb{Z}/2^\lambda\mathbb{Z}$ and $r \in \mathbb{Z}/2^f\mathbb{Z}$, while b_3, s_3 are omitted. Algorithmically, the verification process mostly requires three subroutines.

FindBasis: Given a curve E, find a deterministic basis (P, Q) of $E[2^f]$.
FindKernel: Given a curve E with basis (P, Q) for $E[2^f]$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$, compute the kernel generator $K = P + [s]Q$.
ComputeIsogeny: Given a curve E and a kernel generator K, compute the isogeny $\varphi : E \to E/\langle K \rangle$ and $\varphi(Q)$ for some $Q \in E$.

Below we detail each of the three verification steps **(a)**-**(c)**.

Step (a). Computing φ_{resp} is split up into $n-1$ *blocks* $\varphi^{(j)} : E^{(j)} \to E^{(j+1)}$ of size 2^f, and a last block of size 2^{f_0}, where $f_0 = e - (n-1) \cdot f$. For every $\varphi^{(j)}$, the kernel $\langle K^{(j)} \rangle$ is given by the generator $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$ for a deterministic basis $(P^{(j)}, Q^{(j)})$ of $E^{(j)}[2^f]$.

In the first block, after sampling $(P^{(1)}, Q^{(1)})$ via FindBasis, the bit b indicates whether $P^{(1)}$ and $Q^{(1)}$ have to be swapped before running FindKernel. For the following blocks, the verifier pushes $Q^{(j)}$ through the isogeny $\varphi^{(j)}$ to get a point $Q^{(j+1)} \leftarrow \varphi^{(j)}(Q^{(j)})$ on $E^{(j+1)}$ of order 2^f above $(0,0)$.[16] Hence, for $j > 1$ FindBasis only needs to find a suitable point $P^{(j)}$ to complete the basis $(P^{(j)}, Q^{(j)})$. Furthermore, $K^{(j)}$ is never above $(0,0)$ for $j > 1$, which ensures cyclicity when composing $\varphi^{(j)}$ with $\varphi^{(i-1)}$. In all cases we use $s^{(j)}$ from σ to compute the kernel generator $K^{(j)}$ via FindKernel and $\varphi^{(j)}$ via ComputeIsogeny.

[16]A point P is said to be *above* a point R if $[k]P = R$ for some $k \in \mathbb{N}$.

The last block of degree 2^{f_0} uses $Q^{(n)} \leftarrow [2^{f-f_0}]\varphi^{(n-1)}(Q^{(n-1)})$ and samples another point $P^{(n)}$ as basis of $E^{(n)}[2^{f_0}]$. In the following, we will often assume $f_0 = f$ for the sake of simplicity.[17] An algorithmic description of a single block of SQIsign (NIST) is given in Algorithm 2 in Appendix B.

Step (b). Computing $\widehat{\varphi_{\text{chall}}}$ requires a single isogeny of smooth degree $D_{\text{chall}} \approx 2^{\lambda}$. For the primes given in SQIsign (NIST), we have $E_2[D_{\text{chall}}] \subseteq E_2(\mathbb{F}_{p^2})$. Thus, we compute φ_{chall} by deterministically computing a basis (P, Q) for $E_2[D_{\text{chall}}]$ and finding the kernel $\langle K \rangle$ for $\widehat{\varphi_{\text{chall}}} : E_2 \to E_1$. For $f < \lambda$, we have $D_{\text{chall}} = 2^f \cdot 3^g$, and split this process into two parts.

Given the basis (P, Q) for $E_2[D_{\text{chall}}]$, we compute $(P_2, Q_2) = ([3^g]P, [3^g]Q)$ as basis of $E_2[2^f]$, and use $K_2 = P_2 + [s_2]Q_2$, where b_2 indicates whether P_2 and Q_2 have to be swapped prior to computing K_2. We compute $\varphi_2 : E_2 \to E_2'$ with kernel $\langle K_2 \rangle$, and $P_3 = [2^f]\varphi_2(P)$ and $Q_3 = [2^f]\varphi_3(Q)$ form a basis of $E_2'[3^g]$. Then b_3 indicates a potential swap of P_3 and Q_3, while $K_3 = P_3 + [s_3]Q_3$ is the kernel generator of the isogeny $\varphi_3 : E_2' \to E_1$. Thus, we have $\widehat{\varphi_{\text{chall}}} = \varphi_3 \circ \varphi_2$. If $f \geq \lambda$, we require only the first step.

We furthermore verify that the composition of φ_{resp} and $\widehat{\varphi_{\text{chall}}}$ is cyclic, by checking that the first 2-isogeny step of φ_2 does not revert the last 2-isogeny step of $\varphi^{(n)}$. This guarantees that $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is non-backtracking, hence cyclic.

Step (c). On E_1, the verifier uses the point $Q' \leftarrow \widehat{\varphi_{\text{chall}}}(Q')$, where Q' is some (deterministically generated) point, linearly independent from the generator of $\widehat{\varphi_{\text{chall}}}$, and r (given in σ) to compute $[r]Q'$, and checks if $[r]Q'$ matches the hashed point $K_{\text{chall}} = H(\texttt{msg}, E_1)$ with hash function H.

4.2 Impact of large f on verification

The techniques of Section 3 extend the possible range of f to any size below $\log(p)$. This gives two benefits to the cost of verification, especially when $f \geq \lambda$.

Number of blocks in φ_{resp}. The larger f is, the fewer blocks of size 2^f are performed in **Step (a)**. Per block, the dominating part of the cost are FindBasis and FindKernel as we first need to complete the torsion basis $(P^{(j)}, Q^{(j)})$ for $E^{(j)}[2^f]$ (given $Q^{(j)}$ if $j > 1$), followed by computing $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$. By minimizing the number of blocks n, we minimize the amount of times we perform FindBasis and FindKernel, and the cost of each individual FindKernel only mildly increases, as $s^{(j)}$ increases in size. The overall cost of ComputeIsogeny, that is, performing the n isogenies of degree 2^f given their kernels $K^{(j)}$, only moderately increases with growing f.

We further note that larger f requires fewer T-isogeny computations for the signer, hence signing performance also benefits from smaller n.

[17] In contrast to earlier versions, SQIsign (NIST) fixes $f_0 = f$. However, our analysis benefits from allowing $f_0 < f$.

Challenge isogeny. When $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^{\lambda}$, which has two main benefits.

- The cost of FindBasis for this step is reduced as finding a basis for $E[2^{\lambda}]$ is much easier than a basis search for $E[2^f \cdot 3^g]$.
- The cost for ComputeIsogeny for φ_{chall} decreases as we only have to compute a chain of 2-isogenies instead of additional 3-isogenies.

4.3 Implementation and benchmark of cost in $\mathbb{F}_p$-multiplications

To measure the influence of the size of f on the performance, we implemented SQIsign verification for the NIST Level I security parameter set in Python, closely following SQIsign (NIST). As is standard in isogeny-based schemes, we use x-only arithmetic and represent points and curve coefficients projectively. The benchmark counts $\mathbb{F}_p$-operations and uses a cost metric that allows us to estimate the runtime of real-world implementations for 256-bit primes $p^{(f)}$, where $p^{(f)}$ denotes a prime such that 2^f divides $p^{(f)} + 1$. We benchmark primes $p^{(f)}$ for all values $50 \leq f \leq 250$. These results serve as a baseline to which we compare the optimisations that we introduce in Sections 5 and 6.

We briefly outline how SQIsign (NIST) implements the three subroutines FindBasis, FindKernel, and ComputeIsogeny.

FindBasis. We search for points of order 2^f by sampling x-coordinates in a specified order,[18] and check if the corresponding point P lies on E (and not on its twist E^t). We then compute $P \leftarrow [\frac{p+1}{2^f}]P$ and verify that $[2^{f-1}]P \neq \infty$. Given two points $P, Q \in E$ of order 2^f, we verify linear independence by checking that $[2^{f-1}]P \neq [2^{f-1}]Q$, and discard and re-sample the second point otherwise.

FindKernel. Given a basis (P, Q), FindKernel computes $K = P + [s]Q$ via the `3ptLadder` algorithm as used in SIKE [27]. In addition to the x-coordinates x_P and x_Q of P and Q, it requires the x-coordinate x_{P-Q} of $P - Q$. Hence, after running FindBasis, we further compute x_{P-Q} as described in SQIsign (NIST) [12].

ComputeIsogeny. Given a kernel generator K of order 2^f, ComputeIsogeny follows the approach of SIKE [27], and computes the 2^f-isogeny $\varphi^{(j)}$ as a chain of 4-isogenies for efficiency reasons. If f is odd, we further compute a single 2-isogeny. Following SQIsign (NIST), ComputeIsogeny proceeds as follows:

1. Compute $R = [2^{f-2}]K$ and the corresponding 4-isogeny φ with kernel $\langle R \rangle$. Note that the point $(0, 0)$ might be contained in $\langle R \rangle$ for the first block in φ_{resp}, which requires a special 4-isogeny formula. Thus, we check if this is the case and call the suitable 4-isogeny function. We set $K \leftarrow \varphi(K)$.

[18] SQIsign (NIST) fixes the sequence $x_k = 1 + k \cdot i$ with $i \in \mathbb{F}_{p^2}$ such that $i^2 = -1$ and picks the smallest k for which we find a suitable point.

2. If f is odd, we compute $R = [2^{f-3}]K$, the 2-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$.
3. Compute the remaining isogeny of degree $2^{f'}$ with even f' as a chain of 4-isogenies, where $(0, 0)$ is guaranteed not to lie in any of the kernels.

In the last step, SQIsign (NIST) uses *optimal strategies* as in SIKE [27] to compute a chain of 4-isogenies. Naive multiplicative strategies would compute $R = [2^{f'-2j}]K$, the 4-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$ for $j = 1, \ldots, f'/2$. However, this strategy is dominated by costly doublings. Instead, we can save intermediate multiples of K during the computation of $R = [2^{f'-2j}]K$, and push them through isogenies to save multiplicative effort in following iterations. Optimal strategies that determine which multiples are pushed through isogenies and minimise the cost can be found efficiently [20, 27].

We note that for $f < \lambda$ the computation of $\widehat{\varphi_{\text{chall}}}$ requires small adaptations to these algorithms to allow for finding a basis of $E[D_{\text{chall}}]$ and computing 3-isogenies. Most notably, SQIsign (NIST) does *not* use optimised formulas or optimal strategies for 3-isogenies from SIKE [27], but uses a multiplicative strategy and general odd-degree isogeny formulas [16, 32]. We slightly deviate from SQIsign (NIST) by implementing optimised 3-isogeny formulas, but note that the performance difference is minor and in favor of SQIsign (NIST).

Cost metric. In implementations, $\mathbb{F}_{p^2}$-operations usually call underlying $\mathbb{F}_p$-operations. We follow this approach and use the total number of $\mathbb{F}_p$-operations in our benchmarks. As cost metric, we express these operations in terms of $\mathbb{F}_p$-multiplications, with $\mathbf{S} = 0.8 \cdot \mathbf{M}$, ignoring $\mathbb{F}_p$-additions and subtractions due to their small impact on performance. $\mathbb{F}_p$-inversions, $\mathbb{F}_p$-square roots, and Legendre symbols over $\mathbb{F}_p$ require exponentiations by an exponent in the range of p, hence we count their cost as $\log p$ $\mathbb{F}_p$-multiplications. In contrast to measuring clock cycles of an optimised implementation, our cost metric eliminates the dependence on the level of optimisation of finite field arithmetic and the specific device running SQIsign, hence, can be considered more general.

Benchmark results. Figure 2 shows the verification cost for the NIST Level I-sized primes $p^{(f)}$ for $50 \leq f \leq 250$, fixing $e = 975$, using our cost metric. For more efficient benchmarking, we sample random public key curves and signatures σ of the correct form instead of signatures generated by the SQIsign signing procedure.

The graph shows the improvement for $f \geq 128$. Furthermore, we can detect when the number of blocks n decreases solely from the graph (e.g. $f = 122, 140, 163, 195, 244$). The cost of sampling a 2^f-torsion basis is highly variable between different runs for the same prime, which is visible from the oscillations of the graph. The performance for odd f is worse in general due to the inefficient integration of the 2-isogeny, which explains the zigzag-shaped graph.

From the above observations, we conclude that $f \geq \lambda$ is significantly faster for verification, with local optima found at $f = 195$ and $f = 244$, due to those being (almost) exact divisors of the signing length $e = 975$.

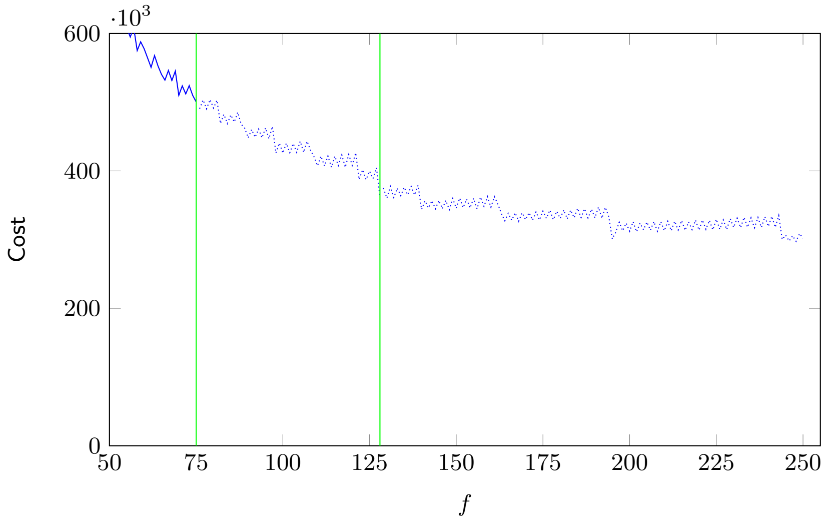

Fig. 2: Cost in $\mathbb{F}_p$-multiplications for verification at NIST Level I security, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. The green vertical lines mark $f = 75$ as used in SQIsign (NIST) for signing without extension fields, and $f = \lambda = 128$, beyond which we can set $D_{\text{chall}} = 2^\lambda$. The dotted graph beyond $f = 75$ is only accessible when signing with extension fields.

Remark 6. The average cost of FindBasis differs significantly between primes p even if they share the same 2^f-torsion. This happens because SQIsign (NIST) finds basis points from a pre-determined sequence $[x_1, x_2, x_3, \ldots]$ with $x_j \in \mathbb{F}_{p^2}$. As we will see in Section 5, these x_j values can not be considered random: some values x_j are certain to be above a point of order 2^f, while others are certain not to be, for any supersingular curve over p.

5 Optimisations for verification

In this section, we show how the improvements from Section 3 that increase f beyond λ together with the analysis in Section 4 allow several other optimisations that improve the verification time of SQIsign in practice. Whereas the techniques in Section 3 allow us to decrease the *number* of blocks, in this section, we focus on the operations occurring *within blocks*. We optimise the cost of FindBasis, FindKernel and ComputeIsogeny.

We first analyse the properties of points that have full 2^f-torsion, and use these properties to improve FindBasis and FindKernel for general f. We then describe several techniques specifically for $f \geq \lambda$. Altogether, these optimisations significantly change the implementation of verification in comparison to SQIsign (NIST). We remark that the implementation of the signing procedure must be altered accordingly, as exhibited by our implementation.

Notation. As we mostly focus on the subroutines *within* a specific block $E^{(j)} \to E^{(j+1)}$, we will omit the superscripts in $E^{(j)}, K^{(j)}, P^{(j)}, \ldots$ and write $E, K, P, \ldots$ to simplify notation.

For reference throughout this section, the pseudocode for a single block in the verification procedure of SQIsign (NIST) and of our optimised variant is in Appendix B as Algorithm 2 and Algorithm 3, respectively.

5.1 Basis generation for full 2-power torsion

We first give a general result on points having full 2^f-torsion that we will use throughout this section. This theorem generalises previous results [17, 31] and will set the scene for easier and more efficient basis generation for $E[2^f]$.

Theorem 2. *Let $E : y^2 = (x - \lambda_1)(x - \lambda_2)(x - \lambda_3)$ be an elliptic curve over $\mathbb{F}_{p^2}$ with $E[2^f] \subseteq E(\mathbb{F}_{p^2})$ the full 2-power torsion. Let $L_i = (\lambda_i, 0)$ denote the points of order 2 and $[2]E$ denote the image of E under $[2] : P \mapsto P + P$ so that $E \setminus [2]E$ are the points with full 2^f-torsion. Then*

$$Q \in [2]E \text{ if and only if } x_Q - \lambda_i \text{ is square for } i = 1, 2, 3.$$

More specifically, for $Q \in E \setminus [2]E$, Q is above L_i if and only if $x_Q - \lambda_i$ is square and $x_Q - \lambda_j$ is non-square for $j \neq i$.

Proof. It is well-known that $Q = (x, y) \in [2]E$ if and only if $x - \lambda_1$, $x - \lambda_2$ and $x - \lambda_3$ are all three squares [26, Ch. 1, Thm. 4.1]. Thus, for $Q \in E \setminus [2]E$, one of these three values must be a square, and the others non-squares (as their product must be y^2, hence square). We proceed similarly as the proof of [31, Thm. 3]. Namely, let P_1, P_2 and P_3 denote points of order 2^f above $L_1 = (\lambda_1, 0)$, $L_2 = (\lambda_2, 0)$ and $L_3 = (\lambda_3, 0)$, respectively, of order 2. A point $Q \in E \setminus [2]E$ must lie above one of the L_i. Therefore, the reduced Tate pairing of degree 2^f of P_i and Q gives a primitive 2^f-th root of unity if and only if Q is not above L_i. Let $\zeta_i = e_{2^f}(P_i, Q)$, then by [25, Thm. IX.9] we have

$$\zeta_i^{2^{f-1}} = e_2(L_i, Q).$$

We can compute $e_2(L_i, Q)$ by evaluating a Miller function f_{2,L_i} in Q, where $\operatorname{div} f_{2,L_i} = 2(L_i) - 2(\mathcal{O})$. The simplest option is the line that doubles L_i, that is, $f_{2,L_i}(x, y) = x - \lambda_i$, hence

$$e_2(L_i, Q) = (x_Q - \lambda_i)^{\frac{p^2-1}{2}}.$$

Applying Euler's criterion to this last term, we get that if $x_Q - \lambda_i$ is square, then ζ_i is not a primitive 2^f-th root and hence Q must be above L_i, whereas if $x_Q - \lambda_i$ is non-square, then ζ_i is a primitive 2^f-th root and hence Q is not above L_i. □

Note that for Montgomery curves $y^2 = x^3 + Ax^2 + x = x(x - \alpha)(x - 1/\alpha)$, the theorem above tells us that non-squareness of x_Q for $Q \in E(\mathbb{F}_{p^2})$ is enough to imply Q has full 2^f-torsion and is not above $(0, 0)$ [31, Thm. 3].

Finding points with 2^f-torsion above $(0,0)$. We describe two methods to efficiently sample Q above $(0,0)$, based on Theorem 2.

1. **Direct x sampling.** By deterministically sampling $x_Q \in \mathbb{F}_p$, we ensure that x_Q is square in $\mathbb{F}_{p^2}$. Hence, if Q lies on E and $x_Q - \alpha \in \mathbb{F}_{p^2}$ is non-square, where α is a root of $x^2 + Ax + 1$, then Theorem 2 ensures that $Q \in E \setminus [2]E$ and above $(0,0)$.
2. **Smart x sampling.** We can improve this using the fact that α is always square [3, 15]. Hence, if we find $z \in \mathbb{F}_{p^2}$ such that z is square and $z-1$ is non-square, we can choose $x_Q = z\alpha$ square and in turn $x_Q - \alpha = (z-1)\alpha$ non-square. Again, by Theorem 2 if Q is on E, this ensures Q is above $(0,0)$ and contains full 2^f-torsion. Hence, we prepare a list $[z_1, z_2, \dots]$ of such values z for a given prime, and try $x_j = z_j\alpha$ until x_j is on E.

Both methods require computing α, dominated by one $\mathbb{F}_{p^2}$-square root. Direct sampling computes a Legendre symbol of $x^3 + Ax^2 + x$ per x to check if the corresponding point lies on E. If so, we check if $x - \alpha$ is non-square via the Legendre symbol. On average, this requires four samplings of x and six Legendre symbols to find a suitable x_Q with $Q \in E(\mathbb{F}_{p^2})$, and, given that we can choose x_Q to be small, we can use fast scalar multiplication on x_Q (see Appendix A).

In addition to computing α, smart sampling requires the Legendre symbol computation of $x^3 + Ax^2 + x$ per x. On average, we require two samplings of an x to find a suitable x_Q, hence saving four Legendre symbols in comparison to direct sampling. However, we can no longer choose x_Q small, which means that improved scalar multiplication for small x_Q is not available.

Finding points with 2^f-torsion *not* above $(0,0)$. As shown in [31], we find a point P with full 2^f-torsion *not* above $(0,0)$ by selecting a point on the curve with non-square x-coordinate. Non-squareness depends only on p, not on E, so a list of small non-square values can be precomputed. In this way, finding such a point P simply becomes finding the first value x_P in this list such that the point $(x_P, -)$ lies on $E(\mathbb{F}_{p^2})$, that is, $x_P^3 + Ax_P^2 + x_P$ is square. On average, this requires two samplings of x, hence two Legendre symbol computations.

5.2 General improvements to verification

In this section, we describe improvements to SQIsign verification and present new optimisations, decreasing the cost of the three main subroutines of verification.

Known techniques from literature. There are several state-of-the-art techniques in the literature on efficient implementations of elliptic curve or isogeny-based schemes that allow for general improvements to verification, but are not included in SQIsign (NIST). We implemented such methods, e.g., to improve scalar multiplication $P \mapsto [n]P$ and square roots. The details are described in Appendix A. In particular, we use that $P \mapsto [n]P$ is faster when x_P is small.

Improving the subroutine FindBasis. In SQIsign (NIST), to find a complete basis for $E[2^f]$ we are given a point $Q \in E[2^f]$ lying above $(0, 0)$ and need to find another point $P \in E(\mathbb{F}_{p^2})$ of order 2^f not lying above $(0, 0)$. We sample P directly using x_P non-square, as described above and demonstrated by [31], and in particular can choose x_P small. We then compute $P \leftarrow [\frac{p+1}{2^f}]P$ via fast scalar multiplication to complete the torsion basis (P, Q).

Improved strategies for ComputeIsogeny. Recall that ComputeIsogeny follows three steps in SQIsign (NIST): it first computes a 4-isogeny that may contain $(0, 0)$ in the kernel, and a 2-isogeny if f is odd, before entering an optimal strategy for computing the remaining chain of 4-isogenies. However, the first two steps include many costly doublings. We improve this by adding these first two steps in the optimal strategy. If f is even, this is straightforward, with a simple check for $(0, 0)$ in the kernel in the first step. For odd f, we add the additional 2-isogeny in this first step.[19] For simplicity of the implementation, we determine optimal strategies as in SIKE [27], thus we assume that only 4-isogenies are used.

Note that techniques for strategies with variable isogeny degrees are available from the literature on CSIDH implementations [13]. However, the performance difference is very small, hence our simplified approach appears to be preferable.

In addition to optimising 4-isogeny chains, we implemented optimised 3-isogeny chains from SIKE [27] for the computation of $\widehat{\varphi_{\text{chall}}}$ when $f < 128$.

5.3 To push, or not to push[20]

In SQIsign (NIST), the point Q is pushed through φ so that we easily get the basis point above $(0, 0)$ on the image curve, and we can then use Theorem 2 to sample the second basis point P. Instead of pushing Q, one can also use Theorem 2 to efficiently sample this basis point Q above $(0, 0)$. Although pushing Q seems very efficient, for larger f we are pushing Q through increasingly larger isogeny chains, whereas sampling becomes increasingly more efficient as multiplication cost by $\frac{p+1}{2^f}$ decreases. Furthermore, sampling *both* P and Q allows us to use those points as an *implicit basis* for $E[2^f]$, even if their orders are multiples of 2^f, as described in more detail below. We observe experimentally that this makes sampling Q, instead of pushing Q, more efficient for $f > 128$.

Using implicit bases. Using Theorem 2, it is possible to find points P and Q efficiently so that both have full 2^f-torsion. The pair (P, Q) is not an *explicit basis* for $E[2^f]$, as the orders of these points are likely to be multiples of 2^f. However, instead of multiplying both points by the cofactor to find an explicit basis, we can use these points implicitly, as if they were a basis for $E[2^f]$. This allows us to compute $K = P + [s]Q$ first, and only then multiply K by the

[19]In particular, we compute $R' = [2^{f-3}]K$ and $R = [2]R'$, a 4-isogeny with kernel $\langle R \rangle$, push R' through, and compute a 2-isogeny with kernel $\langle R' \rangle$.

[20]–that is, the $\boldsymbol{Q}$.

cofactor. This saves a full scalar multiplication by the cofactor $\frac{p+1}{2^f}$. We refer to such a pair (P, Q) as an *implicit basis* of $E[2^f]$. Algorithmically, implicit bases combine FindBasis and FindKernel into a single routine FindBasisAndKernel.

5.4 Improved challenge for $f \geq \lambda$

Recall from Section 4.2 that when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$. This decreases the cost of FindBasis for the challenge computation considerably, as we can now use Theorem 2 to find a basis for $E[2^\lambda]$.

Improving FindBasis for the challenge isogeny when $f \geq \lambda$. We use Theorem 2 twice, first to find P not above $(0, 0)$ having full 2^f-torsion and then to find Q above $(0, 0)$ having full 2^f-torsion. We choose x_P and x_Q small such that faster scalar multiplication is available. We find the basis for $E[2^\lambda]$ by $P \leftarrow [\frac{p+1}{2^f}]P$ followed by $f - \lambda$ doublings, and $Q \leftarrow [\frac{p+1}{2^f}]Q$ followed by $f - \lambda$ doublings.[21] Alternatively, if Q is pushed through isogenies, we can reuse $Q \leftarrow \varphi^{(n)}(Q^{(n)}) \in E[2^f]$ from the computation of the last step of φ_{resp}, so that we get a basis point for $E[2^\lambda]$ by $f - \lambda$ doublings of Q. Reusing this point Q also guarantees cyclicity of $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$.

Remark 7. For SQIsign without extension fields, obtaining $f \geq \lambda$ seems infeasible, hence the degree D of φ_{chall} is $2^f \cdot 3^g$. Nevertheless, some optimizations are possible in the computation of φ_{chall} in this case. FindBasis for $E[2^f \cdot 3^g]$ benefits from similar techniques as previously used in SIDH/SIKE, as we can apply known methods to improve generating a torsion basis for $E[3^g]$ coming from 3-descent [17, § 3.3]. Such methods are an analogue to generating a basis for $E[2^f]$ as described in Theorem 2 and [31, Thm. 3].

6 Size-speed trade-offs in SQIsign signatures

The increase in f also enables several size-speed trade-offs by adding further information in the signature or by using uncompressed signatures. Some trade-offs were already present in earlier versions of SQIsign [21], however, by using large f and the improvements from Section 5, they become especially worthwhile.

We take a slightly different stance from previous work on SQIsign as for many use cases the main road block to using SQIsign is the efficiency of verification in cycles. In contrast, in several applications the precise size of a signature is less important as long as it is below a certain threshold.[22] For example, many applications can handle the combined public key and signature size of RSA-2048 of 528 bytes, while SQIsign (NIST) features a combined size of only 241 bytes. In this section, we take the 528 bytes of RSA-2048 as a baseline, and explore size-speed trade-offs for SQIsign verification with data sizes up to this range.

We note that the larger signatures in this section encode the same information as standard SQIsign signatures, hence have no impact on the security.

[21] Algorithmically, this is faster than a single scalar multiplication by $2^{f-\lambda} \cdot \frac{p+1}{2^f}$.
[22] See https://blog.cloudflare.com/sizing-up-post-quantum-signatures/.

6.1 Adding seeds for the torsion basis in the signature

We revisit an idea that was previously present in SQIsign verification [21] (but no longer in [12] or [22]), and highlight its particular merits whenever $f \geq \lambda$, as enabled by signing with extension fields. So far, we have assumed that completing or sampling a basis for $E[2^f]$ is done by deterministically sampling points. Recall from Section 5.1 that sampling x_P resp. x_Q (when not pushing Q) on average requires the computation of several Legendre symbols resp. square roots. We instead suggest using a seed to find x_P (when pushing Q) or x_P and x_Q (otherwise), which we include in the signature, so that the verifier saves all of the above cost for finding x_P, resp. x_Q. Finding these seeds adds negligible overhead for the signer, while verification performance improves. Signer and verifier are assumed to agree upon all precomputed values.

Seeding a point *not* above $(0,0)$. For x_P *not* above $(0,0)$, we fix a large enough $k > 0$ and precompute the 2^k smallest values $u_j \in \mathbb{F}_p$ such that $u_j + i \in \mathbb{F}_{p^2}$ is non-square (where i is the same as in Section 5). During signing, we pick the smallest u_j such that $x_P = u_j + i$ is the x-coordinate of a point $P \in E(\mathbb{F}_{p^2})$, and add the index j to the signature as a seed for x_P. Theorem 2 ensures that any $P \in E(\mathbb{F}_{p^2})$ for non-square x_P is a point with full 2^f-torsion not above $(0,0)$. This furthermore has the advantage of fast scalar multiplication for x_P as the x-coordinate is very small.

Seeding a point *above* $(0,0)$. As noted above, when f is large, it is faster to deterministically compute a point of order 2^f above $(0,0)$ than to push Q through φ. We propose a similar seed here for fixed large enough $k > 0$, using Theorem 2 and the "direct sampling" approach from Section 5.1. During signing, we pick the smallest $j \leq 2^k$ such that $x_Q = j$ is the x-coordinate of a point $Q \in E(\mathbb{F}_{p^2})$ and $x_Q - \alpha$ is non-square. We add $x_Q = j$ to the signature as seed.

Note that when using both seeding techniques, we do not explicitly compute $[\frac{p+1}{2^f}]P$ or $[\frac{p+1}{2^f}]Q$, but rather use the seeded points P and Q as an implicit basis, as described in Section 5.3.

Size of seeds. Per seeded point, we add k bits to the signature size. Thus, we must balance k such that signatures are not becoming too large, while failure probabilities for not finding a suitable seed are small enough. In particular, seeding x_P resp. x_Q via direct sampling has a failure probability of $\frac{1}{2}$ resp. $\frac{3}{4}$ per precomputed value. For the sake of simplicity, we set $k = 8$ for both seeds, such that every seed can be encoded as a separate byte.[23] This means that the failure rate for seeding Q is $(\frac{3}{4})^{256} \approx 2^{-106.25}$ for our choice, while for P it is 2^{-256}. Theoretically it is still possible that seeding failures occur. In such a case, we simply recompute KLPT. We furthermore include similar seeds for the torsion basis on E_A and E_2, giving a size increase of $(n+1) \cdot 2$ bytes.

[23]Note that for equal failing rates the number of possible seeds for P can be chosen smaller than for Q, hence slightly decreasing the additional data sizes.

The synergy with large f now becomes apparent. The larger f gets, the fewer blocks n are required, hence adding fewer seeds overall. For $f = 75$, the seeds require an additional 28 bytes when seeding both P and Q. For $f = 122, 140, 163, 195, 244$ this drops to 18, 16, 14, 12, and 10 additional bytes, respectively, to the overall signature size of 177 bytes for NIST Level I security.

Remark 8. Instead of using direct sampling for Q with failure probability $\frac{3}{4}$, we can reduce it to $\frac{1}{2}$ via "smart sampling" (see Section 5.1). However, this requires the verifier to compute α via a square root to set $x_Q = z\alpha$ with seeded z. We thus prefer direct sampling for seeded Q, which incurs no such extra cost.

6.2 Uncompressed signatures

In cases where f is very large, and hence the number of blocks is small, in certain cases it is worthwhile to replace the value s in the signature by the full x-coordinate of $K = P + [s]Q$. In essence, this is the uncompressed version of the SQIsign signature σ, and we thus refer to this variant as *uncompressed* SQIsign.

Speed of uncompressed signatures. Adding the precise kernel point K removes the need for both FindBasis and FindKernel, leaving ComputeIsogeny as the sole remaining cost. This speed-up is significant, and leaves little room for improvement beyond optimizing the cost of computing isogenies. The cost of verification in this case is relatively constant, as computing an 2^e-isogeny given the kernels is only slightly affected by the size of f, as is visible in the black dashed line in Figure 3. This makes uncompressed SQIsign an attractive alternative in cases where the signature size, up to a certain bound, is less relevant.

Size of uncompressed signatures. Per step, this increases the size from $\log(s) \approx f$ to $2 \cdot \log(p)$ bits, which is still relatively size efficient when f is close to $\log(p)$. For recomputing φ_{chall}, we take a slightly different approach than before. We add the Montgomery coefficient of E_1 to the signature, and seeds for a basis of $E[2^f]$. From this, the verifier can compute the kernel generator of φ_{chall}, and verify that the j-invariant of its codomain matches E_2. Hence this adds $2 \cdot \log(p)$ bits for E_1 and two bytes for seeds to the signature, for a total of $(n + 1) \cdot (\log p/4) + 2$ bytes.

For $f = 244$, this approach less than doubles the signature size from 177 bytes to 322 bytes for NIST Level I security, for $f = 145$, the signature becomes approximately 514 bytes, while for the current NIST Level I prime with $f = 75$, the size would become 898 bytes. When adding the public key size of 64 bytes, especially the first two cases still appear to be reasonable alternatives to RSA-2048's combined data size of 528 bytes.

Remark 9. Uncompressed signatures significantly simplify verification, as many functionalities required for compressed signatures are not necessary. Hence, this allows for a much more compact code base, which might be important for use cases featuring embedded devices with strict memory requirements.

7 Primes and Performance

In this section we show the performance of verification for varying f, using the optimisations from the previous sections. Further, we find specific primes with suitable f for $n = 4$ and $n = 7$, and report their signing performance using our SageMath implementation, comparing it with the current SQIsign (NIST) prime.

7.1 Performance of optimised verification

To compare the verification performance of our optimised variants with compressed signatures to SQIsign (NIST) and SQIsign (LWXZ),[24] we run benchmarks in the same setting as in Section 4.3. In particular, Figure 3 shows the cost of verification for the NIST Level I primes $p^{(f)}$ for $50 \leq f \leq 250$. As before, we sample random public key curves and signatures σ of the correct form instead of using signatures generated by the SQIsign signing procedure.

For the sake of simplicity, Figure 3 displays only the fastest compressed ApresSQI variant, namely the version that does not push Q through isogenies and uses seeds to sample P and Q. This variant significantly outperforms both SQIsign (NIST) and SQIsign (LWXZ) already at $f = 75$, at the cost of slightly larger signatures. A detailed description and comparison of all four compressed variants is in Appendix C, which shows that our unseeded variants achieve similar large speed-ups with no increase in signature size. Lastly, the uncompressed variant achieves the fastest speed, although at a significant increase in signature size.

7.2 Finding specific primes

We now give two example primes, one prime optimal for 4-block verification, as well as the best we found for 7-block verification. The "quality" of a prime p is measured using the cost metric $\textsc{SigningCost}_p$ defined in Section 3.3.

Optimal 4-block primes. For 4-block primes, taking $e = 975$ as a baseline, we need f bigger than 244. In other words, we are searching for primes of the form

$$p = 2^{244}N - 1$$

where $N \in [2^4, 2^{12}]$ (accepting primes between 250 and 256 bits). This search space is quickly exhausted. For each prime of this form, we find the optimal torsion T to use, minimising $\textsc{SigningCost}_p(T)$. The prime with the lowest total cost in this metric, which we denote p_4, is

$$p_4 = 2^{246} \cdot 3 \cdot 67 - 1$$

[24]Our implementation of SQIsign (LWXZ) [31] is identical to SQIsign (NIST) except for the improved sampling of P described in Section 5.1.

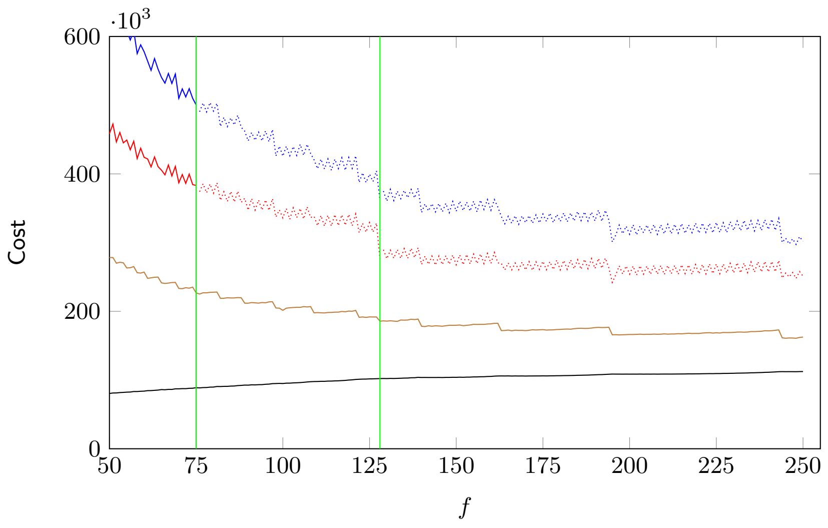

Fig. 3: Extended version of Figure 2 showing the cost in $\mathbb{F}_p$-multiplications for verification at NIST Level I security, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. In addition to SQIsign (NIST) in blue, it shows the performance of SQIsign (LWXZ) in red, our fastest compressed ApresSQI variant in brown, and uncompressed ApresSQI in black.

Balanced primes. Additionally, we look for primes that get above the significant $f > 128$ line, while minimizing $\text{SigningCost}_p(T)$. To do this, we adopt the "sieve-and-boost" technique used to find the current SQIsign primes [12, §5.2.1]. However, instead of looking for divisors of $p^2 - 1$, we follow Theorem 1 and look for divisors of

$$\prod_{n=1}^{k} \Phi_n(p^2)/2$$

to find a list of good candidate primes. This list is then sorted with respect to their signing cost according to SigningCost_p. The prime with the lowest signing cost we could find, which we call p_7, is

$$p_7 = 2^{145} \cdot 3^9 \cdot 59^3 \cdot 311^3 \cdot 317^3 \cdot 503^3 - 1.$$

Remark 10. This method of searching for primes is optimised for looking for divisors of $p^2 - 1$, hence it might be suboptimal in the case of allowing torsion in higher extension fields. We leave it as future work to find methods which further take advantage of added flexibility in the prime search.

7.3 Performance for specific primes

We now compare the performance of the specific primes p_4, p_7, as well as the current NIST Level I prime p_{1973} used in SQIsign (NIST).

Table 1: Comparison between estimated cost of signing for three different primes.

p	largest $\ell \mid T$	largest $\mathbb{F}_{p^{2k}}$	$\textsc{SigningCost}_p(T)$	Adj. Cost	Timing
p_{1973}	1973	$k = 1$	8371.7	1956.5	11m, 32s
p_7	997	$k = 23$	4137.9	-	9m, 20s
p_4	2293	$k = 53$	9632.7	-	15m, 52s

Signing performance. We give a summary of the estimated signing costs in Table 1. For p_{1973}, we include the metric "Adjusted Cost", which we compute as SigningCost with the isogeny computations scaling as $\sqrt{\ell}\log\ell$ to (rather optimistically) account for the benefit of $\sqrt{\text{élu}}$. Further, we ran our proof-of-concept SageMath implementation on the three primes, using SageMath 9.8, on a laptop with an Intel-Core i5-1038NG7 processor, averaged over five runs. An optimised C implementation will be orders of magnitude faster; we use these timings simply for comparison.

We note that the SigningCost-metric correctly predicts the ordering of the primes, though the performance difference is smaller than predicted. A possible explanation for this is that the SigningCost-metric ignores all overhead, such as quaternion operations, which roughly adds similar amounts of cost per prime.

Our implementation uses $\sqrt{\text{élu}}$ whenever the kernel generator is defined over $\mathbb{F}_{p^2}$ and ℓ is bigger than a certain crossover point. This mainly benefits p_{1973}, as this prime only uses kernel generators defined over $\mathbb{F}_{p^2}$. The crossover point is experimentally found to be around $\ell > 300$ in our implementation, which is not optimal, compared to an optimised C implementation.[25] Nevertheless, we believe that these timings, together with the cost metrics, provide sufficient evidence that extension field signing in an optimised implementation stays in the same order of magnitude for signing time as staying over $\mathbb{F}_{p^2}$.

Verification performance. In Table 2, we summarise the performance of verification for p_{1973}, p_7, and p_4, both in terms of speed, and signature sizes.

Two highlights of this work lie in using p_7, both with and without seeds, having (almost) the same signature sizes as the current SQIsign signatures, but achieving a speed-up of factor 2.37 resp. 2.80 in comparison to SQIsign (NIST) and 1.82 resp. 2.15 in comparison to SQIsign (LWXZ), using p_{1973}. Another interesting alternative is using uncompressed p_4, at the cost of roughly double signature sizes, giving a speed-up of factor 4.46 in comparison to SQIsign (NIST) and 3.41 in comparison to SQIsign (LWXZ).

Remark 11. We analyse and optimise the cost of verification with respect to $\mathbb{F}_p$-operations. However, primes of the form $p = 2^f \cdot c - 1$ are considered to be particularly suitable for fast optimised finite field arithmetic, especially when f is large [4]. Hence, we expect primes like p_4 to improve significantly more in

[25]For instance, work by Adj, Chi-Domínguez, and Rodríguez-Henríquez [1] gives the crossover point at $\ell > 89$, although for isogenies defined over $\mathbb{F}_p$.

Table 2: Comparison between verification cost for different variants and different primes, with cost given in terms of 10^3 $\mathbb{F}_p$-multiplications, using $\mathbf{S} = 0.8 \cdot \mathbf{M}$.

p	f	Implementation	Variant	Verif. cost	Sig. size
p_{1973}	75	SQIsign (NIST) [12]	-	500.4	177 B
		SQIsign (LWXZ) [31]	-	383.1	177 B
		ApresSQI	unseeded	276.1	177 B
		ApresSQI	seeded	226.8	195 B
p_7	145	ApresSQI	unseeded	211.0	177 B
		ApresSQI	seeded	178.6	193 B
		ApresSQI	uncompressed	103.7	514 B
p_4	246	ApresSQI	unseeded	185.2	177 B
		ApresSQI	seeded	160.8	187 B
		ApresSQI	uncompressed	112.2	322 B

comparison to p_{1973} in low-level field arithmetic, leading to a larger speed-up than predicted in Table 2. Furthermore, other low-level improvements, such as fast non-constant time GCD for inversions or Legendre symbols, will improve the performance of primes in terms of cycles, which is unaccounted for by our cost metric.

Acknowledgements. We thank Craig Costello for helpful suggestions and comments on an earlier version of this work. We thank the anonymous Eurocrypt 2024 reviewers for their constructive feedback.

References

[1] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. “Karatsuba-based square-root Vélu’s formulas applied to two isogeny-based protocols”. In: *J. Cryptogr. Eng.* 13.1 (2023), pp. 89–106. DOI: 10.1007/S13389-022-00293-Y. URL: https://doi.org/10.1007/s13389-022-00293-y.

[2] Tom M. Apostol. “Resultants of cyclotomic polynomials”. In: *Proceedings of the American Mathematical Society* 24.3 (1970), pp. 457–462.

[3] Roland Auer and Jaap Top. “Legendre Elliptic Curves over Finite Fields”. In: *Journal of Number Theory* 95.2 (2002), pp. 303–312. ISSN: 0022-314X. DOI: https://doi.org/10.1006/jnth.2001.2760. URL: https://www.sciencedirect.com/science/article/pii/S0022314X0192760X.

[4] Jean-Claude Bajard and Sylvain Duquesne. “Montgomery-friendly primes and applications to cryptography”. In: *J. Cryptogr. Eng.* 11.4 (2021), pp. 399–415. DOI: 10.1007/s13389-021-00260-z. URL: https://doi.org/10.1007/s13389-021-00260-z.

[5] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. “CTIDH: faster constant-time CSIDH”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.4 (2021), pp. 351–387. DOI: 10.46586/tches.v2021.i4.351-387. URL: https://doi.org/10.46586/tches.v2021.i4.351-387.

[6] Gustavo Banegas, Valerie Gilchrist, Anaëlle Le Dévéhat, and Benjamin Smith. “Fast and Frobenius: Rational Isogeny Evaluation over Finite Fields”. In: *LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America.* Springer. 2023, pp. 129–148.

[7] Daniel J. Bernstein. *Differential addition chains.* 2006. URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf.

[8] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. “Faster computation of isogenies of large prime degree”. In: *Open Book Series* 4.1 (2020), pp. 39–55.

[9] Jean-François Biasse, David Jao, and Anirudh Sankar. “A quantum algorithm for computing isogenies between supersingular elliptic curves”. In: *International Conference on Cryptology in India.* Springer. 2014, pp. 428–442.

[10] Giacomo Bruno, Maria Corte-Real Santos, Craig Costello, Jonathan Komada Eriksen, Michael Meyer, Michael Naehrig, and Bruno Sterner. “Cryptographic Smooth Neighbors”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 1439. URL: https://eprint.iacr.org/2022/1439.

[11] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. “Stronger and Faster Side-Channel Protections for CSIDH”. In: *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings.* Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. Lecture Notes in Computer Science. Springer, 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9. URL: https://doi.org/10.1007/978-3-030-30530-7_9.

[12] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign: Algorithm specifications and supporting documentation.* National Institute of Standards and Technology. 2023. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/sqisign-spec-web.pdf.

[13] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. “Optimal strategies for CSIDH”. In: *Adv. Math. Commun.* 16.2 (2022), pp. 383–411. DOI: 10.3934/amc.2020116. URL: https://doi.org/10.3934/amc.2020116.

[14] Craig Costello. "B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II.* Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 440–463. DOI: 10.1007/978-3-030-64834-3_15. URL: https://doi.org/10.1007/978-3-030-64834-3_15.

[15] Craig Costello. "Computing Supersingular Isogenies on Kummer Surfaces". In: *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III.* Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 428–456. DOI: 10.1007/978-3-030-03332-3_16. URL: https://doi.org/10.1007/978-3-030-03332-3_16.

[16] Craig Costello and Hüseyin Hisil. "A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II.* Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 303–329. DOI: 10.1007/978-3-319-70697-9_11. URL: https://doi.org/10.1007/978-3-319-70697-9_11.

[17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I.* Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24. URL: https://doi.org/10.1007/978-3-319-56620-7_24.

[18] Craig Costello, Michael Meyer, and Michael Naehrig. "Sieving for Twin Smooth Integers with Solutions to the Prouhet-Tarry-Escott Problem". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 272–301. DOI: 10.1007/978-3-030-77870-5_10. URL: https://doi.org/10.1007/978-3-030-77870-5_10.

[19] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. "SQISignHD: New Dimensions in Cryptography". In: *IACR Cryptol. ePrint Arch.* (2023), p. 436. URL: https://eprint.iacr.org/2023/436.

[20] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *J. Math. Cryptol.* 8.3 (2014), pp. 209–247. DOI: 10.1515/jmc-2012-0015. URL: https://doi.org/10.1515/jmc-2012-0015.

[21] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3. URL: https://doi.org/10.1007/978-3-030-64837-4_3.

[22] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. "New Algorithms for the Deuring Correspondence - Towards Practical and Secure SQISign Signatures". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 659–690. DOI: 10.1007/978-3-031-30589-4_23. URL: https://doi.org/10.1007/978-3-031-30589-4_23.

[23] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *Designs, Codes and Cryptography* 78 (2016), pp. 425–440.

[24] Jonathan Komada Eriksen, Lorenz Panny, Jana Sotáková, and Mattia Veroni. "Deuring for the People: Supersingular Elliptic Curves with Prescribed Endomorphism Ring in General Characteristic". In: *IACR Cryptol. ePrint Arch.* (2023), p. 106. URL: https://eprint.iacr.org/2023/106.

[25] Steven D. Galbraith. *Advances in Elliptic Curve Cryptography, Chapter IX*. Ed. by Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. Vol. 317. Cambridge University Press, 2005.

[26] Dale Husemöller. *Elliptic Curves, 2nd edition*. Springer, 2004.

[27] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. *SIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[28] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *Int. J. Inf. Sec.* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002. URL: https://doi.org/10.1007/s102070100002.

[29] Simon Josefsson and Ilari Liusvaara. “Edwards-Curve Digital Signature Algorithm (EdDSA)”. In: *RFC* 8032 (2017), pp. 1–60. DOI: 10.17487/RFC8032. URL: https://doi.org/10.17487/RFC8032.

[30] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. “On the quaternion-isogeny path problem”. In: *LMS Journal of Computation and Mathematics* 17.A (2014), pp. 418–432.

[31] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. *A Faster Software Implementation of SQISign*. Cryptology ePrint Archive, Paper 2023/753. 2023. URL: https://eprint.iacr.org/2023/753.

[32] Michael Meyer and Steffen Reith. “A Faster Way to the CSIDH”. In: *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. Lecture Notes in Computer Science. Springer, 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8. URL: https://doi.org/10.1007/978-3-030-05378-9_8.

[33] Aurel Page and Benjamin Wesolowski. “The supersingular Endomorphism Ring and One Endomorphism problems are equivalent”. In: *CoRR* abs/2309.10432 (2023). DOI: 10.48550/arXiv.2309.10432. arXiv: 2309.10432. URL: https://doi.org/10.48550/arXiv.2309.10432.

[34] Joost Renes and Benjamin Smith. “qDSA: small and secure digital signatures with curve-based Diffie–Hellman key pairs”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 273–302.

[35] Michael Scott. “A note on the calculation of some functions in finite fields: Tricks of the trade”. In: *Cryptology ePrint Archive* (2020).

[36] Peter W. Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332.

[37] Victor Shoup. “Efficient computation of minimal polynomials in algebraic extensions of finite fields”. In: *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*. 1999, pp. 53–58.

[38] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Springer, 2009.

[39] National Institute of Standards and Technology (NIST). *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. 2022. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[40] Kiminori Tsukazaki. “Explicit isogenies of elliptic curves”. PhD thesis. University of Warwick, 2013.

[41] Jacques Vélu. “Isogénies entre courbes elliptiques”. In: *Comptes-Rendus de l’Académie des Sciences* 273 (1971), pp. 238–241.

[42] John Voight. *Quaternion algebras*. Springer Nature, 2021.

[43] Benjamin Wesolowski. “The supersingular isogeny path and endomorphism ring problems are equivalent”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 1100–1111.

A Curve arithmetic

In this section we describe in detail the known techniques from literature that allow for general improvements to verification, but that are not included in SQIsign (NIST).

We use `xDBL`(x_P) to denote x-only point doubling of a point P and similarly `xADD`(x_P, x_Q, x_{P-Q}) to denote x-only differential addition of points P and Q. We use `xMUL`(x_P, m) to denote x-only scalar multiplication of a point P by the scalar m.

A.1 Faster scalar multiplications

We describe three improvements to the performance of `xMUL`, that can be applied in different situations during verification.

1. **Affine A.** Throughout verification and specifically in FindBasis and FindKernel, we work with the Montgomery coordinate A in projective form. However, some operations, such as computing the point difference x_{P-Q} given x_P and x_Q require A in affine form. Having an affine A allows an additional speed-up, as `xDBL` requires one $\mathbb{F}_{p^2}$-multiplication less in this case. Thus, `xMUL` with affine A is cheaper by 3 $\mathbf{M}$ per bit of the scalar.
2. **Affine points.** Using batched inversion, whenever we require A in affine form we can get x_P and x_Q in affine form for almost no extra cost. An `xMUL` with affine x_P or x_Q saves another $\mathbb{F}_{p^2}$-multiplication, hence again 3 $\mathbf{M}$ per bit of the scalar.
3. **Small x-coordinate.** For a point P with $x_P = a + bi$ with small a and b, we can replace an $\mathbb{F}_{p^2}$-multiplication by x_P with $a + b$ additions. This, in turn, saves almost 3 $\mathbf{M}$ per bit of the scalar in any `xMUL` of x_P.

As we can force P and Q to have $b \in \{0, 1\}$ and small a when sampling them in FindBasis, these points are affine and have small x-coordinates. Together with the affine A, this saves almost 9 $\mathbf{M}$ per bit for such scalar multiplications, saving roughly 27% per `xMUL`. We call such an `xMUL` a *fast* `xMUL`. Whenever `xMUL` uses 2 of these optimisations, we call it *semi-fast*.

Whenever possible, we use differential addition chains [7] to improve scalar multiplications by certain system parameters, such as $\frac{p+1}{2^f}$. In particular, we will only need to multiply by a few, predetermined scalars, and therefore we follow the method described by Cervantes-Vázquez, Chenu, Chi-Domínguez, Feo, Rodríguez-Henríquez, and Smith [11, §4.2]. Our optimal differential addition chains were precomputed using the CTIDH software [5].

A.2 Faster square roots

We apply several techniques from the literature to further optimise low-level arithmetic in all of verification. The most significant of these is implementing faster square roots in $\mathbb{F}_{p^2}$ [35, §5.3], which decreases the cost of finding square roots to two $\mathbb{F}_p$-exponentiations and a few multiplications.

A.3 Projective point difference

The implementation of SQIsign (NIST) switches between affine and projective representations for x_P, x_Q and A within each block. It does so to be able to derive the point difference x_{P-Q} from x_P and x_Q in order to complete the basis P, Q in terms of x-coordinates. However, it is possible to compute the point difference entirely projectively using Proposition 3 from [34]. This allows us to stay projective during the SQIsign (NIST) verification until we reach E_2, where we do normalization of the curve. This saves costly inversions during verification and has the additional benefit of improved elegance for SQIsign (NIST).

However, in our variant of verification, we make no use of projective point difference, as the improvements of Section 5 seem to outperform this already.

B Algorithms

The bottleneck of SQIsign verification is the computation of an isogeny of fixed degree 2^e, which is computed as $\lceil e/f \rceil$ isogenies of degree 2^f, where $f \leq e$. Each such 2^f-isogeny is called a *block*. In this section, we present algorithms for the computation of a single block in verification of SQIsign (NIST) (see Algorithm 2) and the computation using the improvements described in Sections 5 and 6 (see Algorithm 3).

Algorithm 2 Single block in verification of SQIsign (NIST)

```
Input: Affine coeff. $A \in \mathbb{F}_p$, a basis $x_P, x_Q, x_{P-Q}$ for $E_A[2^f]$ with $Q$ above $(0,0)$ and
    $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel
Output: Affine coeff. $A' \in \mathbb{F}_p$ as the codomain of $E_A \to E_{A'}$ of degree $2^f$, with a
    basis $x_P, x_Q, x_{P-Q}$ for $E'_A[2^f]$ with $Q$ above $(0,0)$
1: $K \leftarrow$ 3ptLadder($x_P, x_Q, x_{P-Q}, s, A$)
2: $A_{\text{proj.}}, x_Q \leftarrow$ FourIsogenyChain($K, x_Q, A$)
3: $A, x_Q \leftarrow$ ProjectiveToAffine($A_{\text{proj.}}, x_Q$)
4: $x_P, x_{P-Q} \leftarrow$ CompleteBasis$_{2^f}$($x_Q, A$)
5: return $A, x_P, x_Q, x_{P-Q}$
```

C Performance of optimised verification

The optimisations for compressed variants from Section 5 and Section 6 allow for several variants of verification, depending on using seeds and pushing Q through isogenies. We summarise the four resulting approaches and measure their performance.

Algorithm 3 Single block in verification using improvements of Section 5

```
Input: Projective coeff. $A \in \mathbb{F}_p$, a seed $(n, m)$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel
Output: Affine coeff. $A' \in \mathbb{F}_p$ as the codomain of $E_A \to E_{A'}$ of degree $2^f$
1: $x_P \leftarrow$ SmallNonSquare$(m)$, $x_Q \leftarrow n$
2: $x_{P-Q} \leftarrow$ PointDifference$(x_P, x_Q, A)$          ▷ implicit basis $x_P$, $x_Q$, $x_{P-Q}$
3: $K \leftarrow$ 3ptLadder$(x_P, x_Q, x_{P-Q}, s, A)$
4: $K \leftarrow$ xMUL$(x_K, \frac{p+1}{2^f}, A)$          ▷ semi-fast xMUL
5: $A_{\text{proj.}} \leftarrow$ FourIsogenyChain$(K, A)$
6: $A \leftarrow$ ProjectiveToAffine$(A_{\text{proj.}})$
7: return $A$
```

C.1 Four approaches for verification

To obtain our measurements, we combine our optimisations to give four different approaches to perform SQIsign verification, specifically optimised for $f \geq \lambda$. Firstly, we either push Q through φ in every block, or sample Q. Secondly, we either sample the basis or seed it.

Pushing Q, sampling P without seed. This variant is closest to the original SQIsign (NIST) and SQIsign (LWXZ) implementations. It is the optimal version for non-seeded verification for $f \leq 128$, using the general optimisations from Section 5.2 and the challenge optimisations from Section 5.4.

Not pushing Q, sampling both P and Q without seed. This variant competes with the previous version in terms of signature size. Due to Section 5.3, sampling a new Q is more efficient than pushing Q for large f.[26] This is the optimal version for non-seeded verification for $f > 128$, and additionally uses the optimisations from Section 5.3.

Pushing Q, sampling P with seed. This variant only adds seeds to the signature to describe x_P. As such, it lies between the other three variants in terms of both signature size and speed. The signature is 1 byte per block larger than the unseeded variants, and 1 byte per block smaller than the variant where x_Q is seeded too. In terms of speed, it is faster than the variants where P is unseeded, but slower than the variant where Q is seeded too. It uses the optimisations from Sections 5.2 and 5.4, but cannot benefit from the kernel computation via implicit bases from Section 5.3.

Not pushing Q and sampling both P and Q with seed. This is the fastest compressed version that we present in this work. Although it adds 2 bytes per block, the small number of blocks n for large f makes the total increase

[26]Based on benchmarking results, we sample Q with $x = n\alpha$ for $f < 200$ and directly for $f \geq 200$.

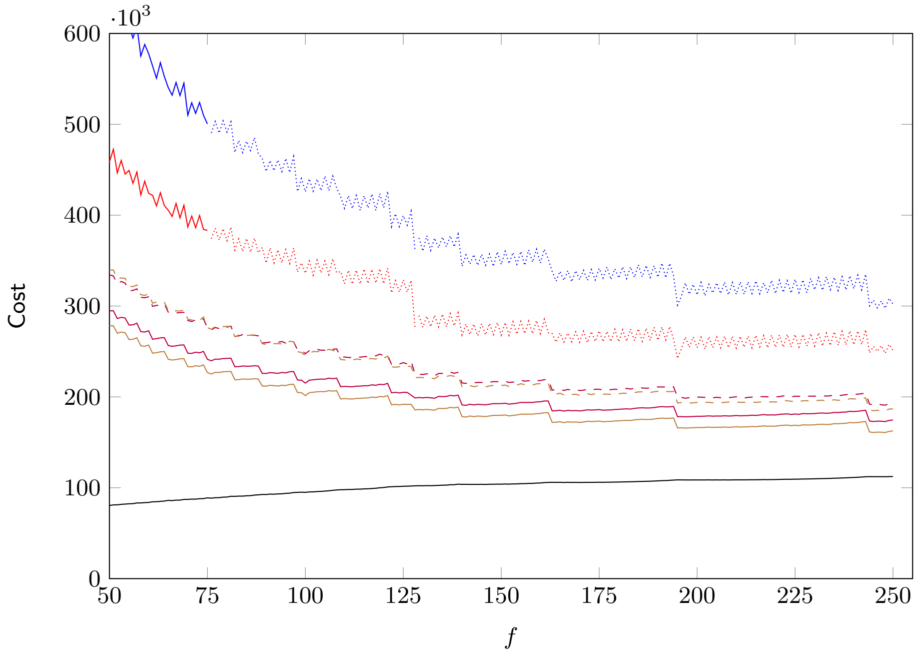

Fig. 4: Extended version of Figure 3 showing the cost in $\mathbb{F}_p$-multiplications for verification at NIST-I security level, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. In addition to SQIsign (NIST) in blue, and SQIsign (LWXZ) in red, it shows all `AprèsSQI` variants: In purple is the performance of `AprèsSQI` when pushing Q, with dashed blue when not seeding P. In brown is the performance of `AprèsSQI` when not pushing Q, with dashed brown when not seeding P, Q. The performance of uncompressed `AprèsSQI` is shown in black.

in signature size small. All the optimisations from Section 5 now apply: we additionally have fast `xMUL` for Q, as well as the optimised implicit basis method to compute the kernel and optimised challenge. An algorithmic description of a single block in this version is given in Algorithm 3.

C.2 Performance benchmark

We benchmarked these four approaches according to our cost metric by taking the average over 1024 random signatures. The results are given in Figure 4 showing the significant increase in performance compared to SQIsign (NIST) and SQIsign (LWXZ), as well as the additional performance gained from seeding. For comparison, we also show the performance when using uncompressed signatures, serving as a lower bound for the cost.

Table 3: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_7

k	N
1	$3^7, 53^2, 59^3, 61, 79, 283, 311^3, 317^3, 349, 503^2, 859, 997$
3	$13, 109, 223, 331$
4	17
5	$11, 31, 71, 241, 271$
6	157
7	$7^2, 29, 43, 239$
8	113
9	19^2
10	$5^4, 41$
11	$23, 67$
12	193
13	131
15	181
18	$37, 73$
23	47

D Detailed information on primes

We give more details on the specific primes used in Section 7.

D.1 Details on p_7

The prime p_7 is used for a verification with $n = 7$ blocks. It achieves $f = 145$, with T given as below.

$$
\begin{aligned}
p_7 &= \texttt{0x309c04bcaedbb0134cca8373e439ffffffffffffffffffffffffffffffffffff}\\
T &= 3^7 \cdot 5^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19^2 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53^2 \cdot 59^3 \cdot 61 \cdot 67\\
&\quad \cdot 71 \cdot 73 \cdot 79 \cdot 109 \cdot 113 \cdot 131 \cdot 157 \cdot 181 \cdot 193 \cdot 223 \cdot 239 \cdot 241 \cdot 271 \cdot 283 \cdot 311^3\\
&\quad \cdot 317^3 \cdot 331 \cdot 349 \cdot 503^2 \cdot 859 \cdot 997\\
\textsc{SigningCost}_{p_7}(T) &= 4137.91235
\end{aligned}
$$

The field of definition for the various torsion groups we work with can be found in Table 3.

D.2 Details on p_4

The prime p_4 is used for a verification with $n = 4$ blocks. It achieves $f = 246$, with T given as below.

$$p_4 = \texttt{0x323fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff}$$

$$\begin{aligned} T = {} & 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71 \\ & \cdot 73 \cdot 79 \cdot 83 \cdot 89 \cdot 97 \cdot 101 \cdot 103 \cdot 107 \cdot 109 \cdot 113 \cdot 127 \cdot 149 \cdot 151 \cdot 157 \cdot 163 \cdot 181 \\ & \cdot 197 \cdot 211 \cdot 229 \cdot 241 \cdot 271 \cdot 317 \cdot 397 \cdot 577 \cdot 593 \cdot 641 \cdot 661 \cdot 757 \cdot 1069 \cdot 2293 \end{aligned}$$

$$\textsc{SigningCost}_{p_4}(T) = 9632.7307$$

The field of definition for the various torsion groups can be found in Table 4.

Table 4: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_4

k	N
1	$67, 73, 757$
2	$317, 2293$
3	$37, 127, 1069$
4	593
5	$11, 31, 71, 661$
6	13
7	43
8	$17, 113$
9	$3^3, 19, 181, 577$
10	$5^2, 61, 641$
11	$23, 89$
14	$29, 197$
18	397
19	229
20	41
21	7^2
23	47
25	151
26	53
27	$109, 163, 271$
29	59
30	241
35	211
37	149
39	$79, 157$
41	83
48	97
50	101
51	103
53	107