J. Algebra Comb. Discrete Appl.
13(2) • 235–252

Received: 24 November 2025
Accepted: 12 March 2026

Journal of Algebra Combinatorics Discrete Structures and Applications

Sieving for large twin smooth integers using single solutions to Prouhet-Tarry-Escott

Research Article

Knud Ahrens

Abstract: In the isogeny-based track of post-quantum cryptography, optimal instances of the signature scheme SQISign rely on primes p such that $p\pm1$ is smooth. In 2021 a new approach to find those numbers was discovered using solutions to the Prouhet-Tarry-Escott (PTE) problem. With these solutions we can sieve for smooth integers A and B with a difference of $|A-B|=C$ fixed by the solution. Then some $2A/C$ and $2B/C$ are smooth integers hopefully enclosing a prime. They took many different PTE solutions and combined them into a tree to process them more efficiently. But for larger numbers there are fewer promising PTE solutions so their advantage over the naive approach (checking a single solution at a time) fades. For a single PTE solution the search can be optimized for the corresponding C and allows to check smoothness only for those integers that are divisible by C. In this work we investigate such optimisations and show a significant speed-up compared to the naive approach - both heuristically and empirically. Along the way we compute the number of roots of a given polynomial modulo prime powers and give an upper bound for the number of roots modulo a composite number.

2020 MSC: 94A60, 11T71

Keywords: Isogeny-based cryptography, Post-quantum cryptography, Twin smooth integers, Prouhet-Tarry-Escott problem, SQISign

1. Introduction

Cryptography based on isogenies of elliptic curves produced some candidates for post-quantum cryptography like CSIDH [6] or the now broken SIDH [14] and B-SIDH [9]. In 2023 SQISign [7, 11] has been submitted to the Post-Quantum Cryptography: Digital Signature Schemes competition held by NIST [16] as a part of their post-quantum standardisation process.

Isogeny-based cryptography usually employs so-called supersingular elliptic curves. They can always be defined over the finite field $\mathbb{F}_{p^2}$ and most cryptographic protocols consider curves with $(p+1)^2$ or

Knud Ahrens (Corresponding Author); Faculty of Computer Science and Mathematics, University of Passau, Passau, Germany (email: ahrens@fim.uni-passau.de).

$(p-1)^2$ points over this field. Isogenies are group homomorphisms of the curves and are often defined via their kernel points. For the purposes of this work the degree of an isogeny is equal to the number of points in its kernel and thus has to divide the group order. Large degrees therefore require fields of large characteristic p or field extensions, both slowing down all computations. Isogenies of small prime degree can be evaluated efficiently using Vélu's formulae, while isogenies of composite degree can be decomposed into several isogenies of prime degree. Therefore, many protocols use fields of characteristic p such that either $p+1$ or $p-1$ is smooth, which allows for efficient evaluation of isogenies of degree dividing $p+1$ or $p-1$, respectively. A smooth integer only has small prime factors below a fixed smoothness bound. If in addition every prime power in the factorisation is below this bound, it is called power-smooth.

A particular feature of SQISign is its compactness; the signature and public key sizes are smaller than in other post-quantum signature schemes. As in B-SIDH, this is achieved by using curves with $(p+1)^2$ points over $\mathbb{F}_{p^2}$ and their so-called quadratic twists with $(p-1)^2$ points to permit more isogenies without increasing the size of p. SQISign uses that a curve and its quadratic twist are isomorphic over $\mathbb{F}_{p^4}$, which allows to work with points (x, y) over $\mathbb{F}_{p^4}$ without going to extensions of $\mathbb{F}_{p^2}$, namely $\mathbb{F}_{p^4}$. This is done by identifying points with $x \in \mathbb{F}_{p^2}$ with points on either the curve or the twisted curve. A prime p is called twin smooth if both $p+1$ and $p-1$ are smooth. To be able to evaluate the isogenies in SQISign efficiently we need the characteristic p to be as close as possible to being twin smooth and SQISign specifies the size of an acceptable non-smooth cofactor[1]. The non-interactive timed commitment scheme SIGNITC [1] needs primes such that p^2-1 contains a large smooth factor and medium-sized non-smooth factor.

Sieving for twin smooth primes using solutions to the Prouhet-Tarry-Escott problem (PTE) can provide suitable primes for protocols like SQISign and SIGNITC. We only use a single PTE solution at a time and customise the sieve accordingly. This is advantageous when only a few suitable solutions are available, which is the case for PTE problems with $n > 6$. The efficiency of our strategy depends on the number of roots of a given polynomial modulo a given positive integer, both related to the PTE solution. We prove an upper bound and develop a heuristic for the number of modular roots in general, which may be of independent interest.

Related work

There are approaches to find twin smooth primes using the extended Euclidean algorithm or polynomials of the form $x^n - 1$, but in 2021 Costello, Meyer and Naehrig [10] found a new way using solutions to the Prouhet-Tarry-Escott (PTE) problem and produced integers with lower smoothness bounds. They used several PTE solutions at once and optimised for a setting with many solutions. There has been a more recent publication [4] including the same authors with another approach. The specifications [7] of SQISign for the NIST competition for post-quantum signatures briefly mention several methods of searching for twin smooth integers.

Let p be a prime and f be a polynomial with integer coefficients. Finding an upper bound for the number of solutions to $f(x) = 0$ modulo powers of p is a classical problem. Usually, we have no information about the factorisation or the roots of f and hence require some conditions on the coefficients, e.g. p does not divide the content of f [18] or p does not divide the content of $f(x) - f(0)$ [15]. In our case, we only require the polynomial to split into linear factors with integral roots.

Structure

The rest of this paper is structured as follows. First we summarise the main idea of Costello, Meyer and Naehrig [10]. Section 3 motivates our optimisation for a single solution and gives the general idea of our approaches. In Section 4 we compute the number of solutions to $f(x) \equiv 0 \bmod p^e$ for primes p and polynomials f with integer roots. We do this by looking at the p-adic distances of the roots of f and using properties similar to p-adic continuity. This allows us to derive an upper bound for the number

[1] *Higher dimensional adaptations of SQISign have different requirements for the characteristic p.*

of solutions modulo a composite number and a heuristic for our new approaches. Next we present our two new strategies how to find twin smooth integers using only one PTE solution in Section 5. The first one tests elements in order of their size and applies a sieve to compute smoothness. The second one uses batch smoothness tests [2, 12] and orders elements by their residue classes. Then in Section 6 we compare our practical results to the naive and the multi-solution tree approach as well as the predicted number of twin smooth integers. A Sage as well as a C implementation are available at https://github.com/kahrens-code/twin-smooth-integers. Finally there is a conclusion including some open questions for future research.

2. Sieving with PTE solutions

In this section we briefly present the Prouhet-Tarry-Escott (PTE) problem and how [10] used it to find twin smooth integers.

2.1. The Prouhet-Tarry-Escott problem

The Prouhet-Tarry-Escott problem of degree n and integer k is a set of equations

$$a_1^j + \cdots + a_n^j = b_1^j + \cdots + b_n^j \quad \text{with} \quad \{a_1, \ldots, a_n\} \cap \{b_1, \ldots, b_n\} = \emptyset$$

for $1 \leq j \leq k$ and $a_i, b_i \in \mathbb{Z}$. A solution $\mathcal{A} = \{a_1, \ldots, a_n\}$, $\mathcal{B} = \{b_1, \ldots, b_n\}$ with $k = n - 1$ is called an ideal solution. There are several known solutions and even some parametric solutions, but for $n = 11$ and $n \geq 13$ no ideal solutions are known [5]. Since we mostly deal with ideal solutions and the polynomials in Equation (1), we deviate from the usual notation of size n and degree k. A statement from Borwein and Ingalls [3] using Newton's identities tells us that for each ideal solution the difference $C = |a(x) - b(x)|$ of the corresponding polynomials

$$a(x) = \prod_{i=1}^{n}(x - a_i), \quad b(x) = \prod_{i=1}^{n}(x - b_i) \tag{1}$$

is an integer constant.

Costello, Meyer and Naehrig [10] realised that $a(x)/C$ and $b(x)/C$ are consecutive integers every time $a(x)$ (or $b(x)$ equivalently) evaluates to a multiple of C and hoped that $2a(x)/C$ and $2b(x)/C$ are smooth with a prime in between. Since the polynomials already have n factors, chances are high that the evaluations are smooth. Ideal solutions with small differences C seem to produce a higher rate of twin smooth integers, but their number is limited (see [10, Table 2]) since the C in parametric solutions often increases fast. Some solutions can be found in a database by Shuwen [17] and there are constructions to find new ones. In Section 3.2 of [10] and on their GitHub they mention some constructions and give several solutions[2].

They also show heuristically that this approach has a higher chance of success for a given smoothness bound or rather allows for a lower smoothness bound for a given probability to find twin smooth integers compared to previous attempts, e.g. using the extended Euclidean algorithm.

[2] *We found two further solutions with $n = 7$ in addition to the eight mentioned in [10, Table 2]. One was found in [17] and the other is a parametric solution from [8] for $m = -3$. They are listed on GitHub.*

2.2. Sieving with multiple PTE solutions

As we have just seen, we can use solutions of the PTE problem to find twin smooth integers. So we need an ideal PTE solution and then for different $\ell \in \mathbb{Z}$ check if $a(\ell)$ and $b(\ell)$ are smooth and if $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. For ease of notation an ideal PTE solution is only called a solution in the following. We now present the tree approach from [10].

We start by creating a look-up-table of smooth numbers in the search interval $I \subset \mathbb{Z}$ basically using a sieve of Eratosthenes. The smoothness of $a(\ell)$ and $b(\ell)$ is then tested using their factors. Let the set of roots of $a(x)$ and $b(x)$ be $Z = \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$. Note that in our case these roots are small compared to $\ell \in I$. Then for each integer ℓ in the search interval I we look up the values $\{\ell - z \mid z \in Z\}$. If all of them are smooth, then surely their product is smooth and we have two smooth integers $a(\ell)$ and $b(\ell)$ that differ by C. For those smooth integers we finally compute $a(\ell) \bmod C$ (or $b(\ell) \bmod C$) to see if $a(\ell)/C$ and $b(\ell)/C$ are integers. If they are, we have found twin smooth integers and can check if $a(\ell)/C + b(\ell)/C$ is a prime.

If there are multiple PTE solutions of the same degree n we may speed up the search as follows. All PTE solutions can be normalised in a way such that 0 is in the zero set Z of the corresponding polynomials and from now on all solutions are taken to be normalised. So if $\ell - 0$ is not smooth, then no solution gives smooth integers $a(\ell), b(\ell)$. Similarly, if there are different (normalised) solutions their corresponding polynomials probably share some zeros. If $\ell - z$ is not smooth, we can skip all solutions that share this zero z for this ℓ. Based on this Costello, Meyer and Naehrig [10] presented a tree structure based on a hitting set problem with the most common zeros as vertices to test many solutions at the same time with the minimal number of look-ups in the smoothness list produced by the sieve. When smooth integers $a(\ell)$ and $b(\ell)$ are found, the residues modulo their corresponding C are calculated to see if they give twin smooth integers.

For $n = 6$ there are 2438 solutions with $C \leq 2^{50}$, so if all of them can be checked by only a few look-ups this is a significant speed-up compared to the naive approach testing every solution separately. More on their algorithm and a detailed example can be found in their paper [10] alongside a link to their code on GitHub.

3. Motivation

Since different solutions have different C, the tree approach forbids to do the modulo calculation first, but for a single solution this is possible. Costello, Meyer and Naehrig [10] left it as an open question to explore this path. In this section we see why this is promising.

For primes in the range of 256 bit $n = 6$ seems to work well, but solutions of higher degree n allow for much lower smoothness bounds for larger primes at a given probability to find twin smooth integers [10, Table 3]. Or more explicitly: when we need a twin smooth integer in the 512 bit range with smoothness bound below 2^{25} we probably need to test 2^{50} values before we get a hit when using $n = 6$ compared to only 2^{30} for $n = 8$. According to [10, Table 3] a search for twin smooth primes in the range of 512 bit and beyond therefore favours solutions of higher degree and [10, Table 2] shows that only a few solutions with high degree n and small C are known. Therefore checking them individually is feasible.

The idea is to calculate the set R of all elements r in $[0, C)$ that solve $a(r) \equiv b(r) \equiv 0 \bmod C$. Obviously $a(r + C) \equiv a(r) \bmod C$ and thus all relevant $\ell \in I$, i.e. those solving $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$, are of the form $\ell = r + kC$. Hence, instead of looking at all ℓ in the search interval I it is sufficient to look at $\{r + kC \mid r \in R, k \in \mathbb{Z}\} \cap I$. This subset has size about $(|R|/C)|I|$ and can be significantly smaller than I. The heuristic in Section 4.3 and Table 3 indicate that this fraction decreases for larger C. Another improvement is that the modulo computation only needs to be done once per PTE solution and not for every smooth integer $a(\ell)$ (or $b(\ell)$) anew.

To see more clearly why we want a larger degree n and a small difference C (which is specific to every PTE solution) we have to understand how they influence our search. On the one hand a higher number n of factors improves the chances of $a(\ell)$ and $b(\ell)$ being smooth, on the other hand it decreases the number of possible values for ℓ.

Remark 3.1. *We can approximate p with*

$$p = \frac{a(\ell) + b(\ell)}{C} \approx \frac{2a(\ell)}{C} \approx \frac{2b(\ell)}{C} \approx \frac{2\ell^n}{C}.$$

Let $x - \delta \lesssim \log_2 p \lesssim x + \delta$ be the range for an "x bit prime" p. Then the search space I for ℓ is roughly restricted to

$$\frac{x - \delta - 1 + \log_2 C}{n} \lesssim \log_2 \ell \lesssim \frac{x + \delta - 1 + \log_2 C}{n}.$$

We see that C does not really change the size of the search interval, it is merely a shift, but higher values for n decrease the search space. This is why some entries in [10, Table 3] are greyed out, but this is more relevant when searching for smaller primes.

As we mentioned above, solutions with larger C for a fixed n in general have a smaller fraction $|R|/C$ of $\ell \in I$ that satisfy $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. This makes our approach faster compared to the naive one, but not more efficient in absolute terms. A smaller density $|R|/C$ makes it less likely to find a twin smooth integer, or a twin smooth prime accordingly, in a given interval I. Increasing I could help but the interval could be restricted (cf. Remark 3.1) such that enlargement would result in p being too small or too large.

Thus PTE solutions with $n > 6$ and small C have a better chance of finding (twin smooth) primes suitable for SQISign of size at least 512 bit. Since $p + 1$ and $p - 1$ are of the form $2a(\ell)/C$ and $2b(\ell)/C$, their prime factors are smaller than ℓ. With Remark 3.1 we can approximate this bound as $\sqrt[n]{pC}$. This can be beneficial for SIGNITC as it limits the size of the non-smooth factors of $p^2 - 1$.

4. Modular roots

In this section we compute the number of roots of a given polynomial modulo prime powers and an upper bound for the number of roots modulo a composite number. For this section let p be a prime and $f(x) = \prod_{i=1}^{n}(x - z_i)$ a polynomial with integer roots z_i and degree n. For the rest of this section we will distinguish between "roots" $z \in \{z_i \mid 1 \leq i \leq n\}$ of f and "solutions" x to $f(x) \equiv 0 \bmod p^e$. Since all solutions also have to solve the equation modulo p, we only need to look at integers that are congruent to one of the roots z_i.

Definition 4.1 (Solutions above z)**.** *A solution x to $f(x) \equiv 0 \bmod p^e$ is* above *z, if it is congruent to z modulo p.*

The set of all integers that are congruent to z modulo p is $z + p\mathbb{Z}$. For elements $s \in p\mathbb{Z}$ we choose the unique representation in base p as $s = \sum_{j \geq 1} d_j p^j$ with only finitely many non-zero integers $0 \leq d_j < p$. The p-adic valuation ν_p gives the highest power of p that divides the argument and $\nu_p(0)$ is infinite by definition. For two numbers x and y their p-adic distance $|x - y|_p = p^{-\nu_p(x-y)}$ is small if the p-adic valuation $\nu_p(x - y)$ is large.

Counting the number of solutions modulo p^e for arbitrary $z_i \in \mathbb{Z}$ is not trivial. If we have $z_i \not\equiv z_j \bmod p$ for $i \neq j$, then we have n solutions to $f(x) \equiv 0 \bmod p$ since $\mathbb{Z}/p\mathbb{Z}$ is a field. Applying Hensel lifting we can generalise this to n solutions to $f(x) \equiv 0 \bmod p^e$ for all $e \in \mathbb{N}_{>0}$. But even if we could assume that $z_i \neq z_j$ for $i \neq j$ they could still be congruent.

4.1. Assigning roots

In order to count the number of solutions, it is useful to assign them to the different roots z_i. Intuitively, we would expect that a small $|f(x) - 0|_p$ implies a small $|x - z_i|_p$ for some $1 \leq i \leq n$.

Therefore, we assign the solutions x to the roots z_i that are p-adically close to them. The idea then is to compute how many solutions each root contributes.

First we only look at one root z of f at a time. Let $m_e(z)$ be the number of roots z_i of f that are congruent to z modulo p^e. Then $1 \leq m_e(z) \leq n$ and note that $m_0(z) = n$ for all roots of f. Recall that $\nu_p(z - z) = \infty$ and let $m_\infty(z)$ be the number of roots that are equal to z.

Definition 4.2 (Solutions close to z)**.** *A solution x to $f(x) \equiv 0 \bmod p^e$ is* close to z *(with respect to the p-adic distance) if $\nu_p(x - z) > k$ for $\sum_{\ell=1}^{k} m_\ell(z) < e \leq \sum_{\ell=1}^{k+1} m_\ell(z)$, i.e. if $x \in z + p^{k+1}\mathbb{Z}$.*

Note that the solutions close to z are a subset of the solutions above z and we will prove that every solution is close to at least one root. If two roots are not congruent modulo p, then they do not interact in the sense that a solution can not be above (and especially not close to) both of them. To get an intuition for the notation we give two examples.

Example 4.3. *Let $f(x) = x(x-2)(x-4)$ and $p = 2$. Then $m_1(z) = 3$ for all roots $z \in \{0, 2, 4\}$, $m_2(2) = 1$, $m_2(0) = m_2(4) = 2$ and $m_e(z) = 1$ for $e > 2$ and all $z \in \{0, 2, 4\}$. For $0 < e \leq 3$ the solutions to $f(x) \bmod p^e$ are all the even numbers less than p^e and they are close to all three roots. For $e > m_1(2) = 3$ the solutions close to 2 are $\{2 + ip^{e-2} \mid 0 \leq i < p^2\}$ and they are not close to 0 or 4. For $m_1(0) = 3 < e \leq 5 = m_1(0) + m_2(0)$ the solutions not close to 2 are $\{ip^2 \mid 0 \leq i < p^{e-2}\}$ and they are close to both 0 and 4. For $e > 5$ every solution is only close to one root. For $z \in \{0, 4\}$ they are $\{z + ip^{e-3} \mid 0 \leq i < p^3\}$. This is illustrated in Table 1. Modulo powers of 3 the three roots are distinct and not congruent. Hence, there are always only $1 + 1 + 1 = 3$ solutions.*

Table 1. Solutions of $f(x) = x(x-2)(x-4)$ modulo different powers of 2. All solutions are above all roots. The columns show which solutions are close to which roots.

mod	$z_1 = 0$	$z_2 = 4$	$z_3 = 2$	number
2^1	0			1
2^2	$0 + 2^1\mathbb{Z}$			2
2^3	$0 + 2^1\mathbb{Z}$			4
2^4	$0 + 2^2\mathbb{Z}$		$2 + 2^2\mathbb{Z}$	$4 + 4$
2^5	$0 + 2^2\mathbb{Z}$		$2 + 2^3\mathbb{Z}$	$8 + 4$
2^6	$0 + 2^3\mathbb{Z}$	$4 + 2^3\mathbb{Z}$	$2 + 2^4\mathbb{Z}$	$8 + 8 + 4$
2^7	$0 + 2^4\mathbb{Z}$	$4 + 2^4\mathbb{Z}$	$2 + 2^5\mathbb{Z}$	$8 + 8 + 4$
$2^{e>5}$	$0 + 2^{e-3}\mathbb{Z}$	$4 + 2^{e-3}\mathbb{Z}$	$2 + 2^{e-2}\mathbb{Z}$	$8 + 8 + 4$

Example 4.4. *For the polynomial $g(x) = x^2(x-2)$ we have a similar picture as in Example 4.3 when looking at it modulo powers of 2. The only difference is that $m_e(0) = 2$ for all $e > 1$ and hence the number of solutions grows indefinitely for increasing e. Table 2 shows the solutions modulo powers of 2 and 3. In the second case we see that the root $z_3 = 2$ is not congruent to any other root and thus can only produce a single solution.*

These examples show that the definition of "close" seems reasonable for assigning roots to the solutions. But we have to prove that we can assign every solution to a root in this fashion.

Theorem 4.5. *Let p be a prime, $e \in \mathbb{N}_{>0}$ and $f(x) = \prod_{i=1}^{n}(x - z_i)$ a polynomial with integer roots z_i and degree n. Then every solution to $f(x) \equiv 0 \bmod p^e$ is close to at least one root $z \in \{z_i \mid 1 \leq i \leq n\}$.*

Proof. Let x be an arbitrary solution to $f(x) \equiv 0 \bmod p^e$. It also solves the equation modulo $p^{e'}$ for all $e' < e$ and especially modulo p. Therefore, it is above a root z of f and can be written as $x = z + s$ for some $s \in p\mathbb{Z}$. Since z is congruent to $m_1(z)$ roots modulo p, the solution x is above $m_1(z)$ roots. We can choose any of them as z and they all have the same $m_1(z)$. Since s is in $p\mathbb{Z}$, we already have

Table 2. Solutions of $g(x) = x^2(x-2)$ modulo different powers of 2 and 3. The columns show which solutions are close to which roots.

mod	$z_1 = 0$	$z_2 = 0$	$z_3 = 2$	number
2^1	0			1
2^2	$0 + 2^1\mathbb{Z}$			2
2^3	$0 + 2^1\mathbb{Z}$			4
2^4	$0 + 2^2\mathbb{Z}$	$2 + 2^2\mathbb{Z}$		$4 + 4$
2^5	$0 + 2^2\mathbb{Z}$	$2 + 2^3\mathbb{Z}$		$8 + 4$
2^6	$0 + 2^3\mathbb{Z}$	$2 + 2^4\mathbb{Z}$		$8 + 4$
2^7	$0 + 2^3\mathbb{Z}$	$2 + 2^5\mathbb{Z}$		$16 + 4$
$2^{e>3}$	$0 + 2^{\lceil \frac{e-1}{2} \rceil}\mathbb{Z}$	$2 + 2^{e-2}\mathbb{Z}$		$2^{e-\lceil \frac{e-1}{2} \rceil} + 4$

mod	$z_1 = 0$	$z_2 = 0$	$z_3 = 2$	number
3^1	0		2	$1 + 1$
3^2	$0 + 3^1\mathbb{Z}$		2	$3 + 1$
3^3	$0 + 3^2\mathbb{Z}$		2	$3 + 1$
3^4	$0 + 3^2\mathbb{Z}$		2	$9 + 1$
3^5	$0 + 3^3\mathbb{Z}$		2	$9 + 1$
3^6	$0 + 3^3\mathbb{Z}$		2	$27 + 1$
3^7	$0 + 3^4\mathbb{Z}$		2	$27 + 1$
3^e	$0 + 3^{\lceil \frac{e}{2} \rceil}\mathbb{Z}$		2	$3^{e-\lceil \frac{e}{2} \rceil} + 1$

$\nu_p(x - z) = \nu_p(s) \geq 1$ and x is close to z as solution modulo $p^{e'}$ for $0 < e' \leq m_1(z)$, i.e. for $k = 0$ in $\sum_{\ell=1}^{k} m_\ell(z) < e' \leq \sum_{\ell=1}^{k+1} m_\ell(z)$. In fact, it is close to all $m_1(z)$ roots it is above for $0 < e' \leq m_1(z)$.

Now we show that we can choose z such that x is close to z as solution modulo $p^{e'}$ for $\sum_{\ell=1}^{k} m_\ell(z) < e' \leq \sum_{\ell=1}^{k+1} m_\ell(z)$ (case k), if it is close to z for $\sum_{\ell=1}^{k-1} m_\ell(z) < e' \leq \sum_{\ell=1}^{k} m_\ell(z)$ (case $k-1$) and $e > \sum_{\ell=1}^{k} m_\ell(z)$. By induction, this proves that x is close to a root z. For simplicity we write just m_ℓ instead of $m_\ell(z)$ and use $v_i = \nu_p(z - z_i)$ as shorthand. Without loss of generality let $z = z_1$ and $v_i \geq v_j$ for $1 \leq i \leq j \leq n$. Then we have $z = z_i$ for $1 \leq i \leq m_\infty$, $v_i = j$ for $m_{j+1} < i \leq m_j$ and $v_i = 0$ for $m_1 < i \leq m_0 = n$. For $i > m_\infty$ we define δ_i via $p^{v_i}\delta_i = z - z_i$, then $\nu_p(\delta_i) = 0$.

For the inductive argument, assume that x is close to z as solution modulo $p^{e'}$ for $\sum_{\ell=1}^{k-1} m_\ell < e' \leq \sum_{\ell=1}^{k} m_\ell$ and $e > \sum_{\ell=1}^{k} m_\ell$. Then $x = z + s$ with $k \leq \nu_p(x - z) = \nu_p(s)$ and $s = \sum_{j \geq k} d_j p^j$. This gives us

$$\begin{aligned}
f(z+s) &= \prod_{i=1}^{n}(z - z_i + s) = \prod_{i=1}^{m_\infty}(0+s) \prod_{i=m_\infty+1}^{m_0}\left(p^{v_i}\delta_i + s\right) \\
&= \prod_{i=1}^{m_\infty} p^k\left(0 + \sum_{j\geq k} d_j p^{j-k}\right) \prod_{i=m_\infty+1}^{m_k} p^k\left(p^{v_i-k}\delta_i + \sum_{j\geq k} d_j p^{j-k}\right) \\
&\quad \cdot \prod_{\ell=1}^{k} \prod_{i=m_{k-\ell+1}+1}^{m_{k-\ell}} p^{k-\ell}\left(\delta_i + \sum_{j\geq k} d_j p^{j-(k-\ell)}\right) \\
&= p^{km_k} \prod_{i=1}^{m_\infty}\left(0 + \sum_{j\geq k} d_j p^{j-k}\right) \prod_{i=m_\infty+1}^{m_k}\left(p^{v_i-k}\delta_i + \sum_{j\geq k} d_j p^{j-k}\right) \\
&\quad \cdot \prod_{\ell=1}^{k} p^{(k-\ell)(m_{k-\ell}-m_{k-\ell+1})} \prod_{i=m_{k-\ell+1}+1}^{m_{k-\ell}}\left(\delta_i + \sum_{j\geq k} d_j p^{j-k+\ell}\right) \\
&\equiv p^M \prod_{i=1}^{m_{k+1}}(d_k) \prod_{i=m_{k+1}+1}^{m_k}(\delta_i + d_k) \prod_{i=m_k+1}^{m_0}(\delta_i) \mod p^{M+1}
\end{aligned}$$

where $M = km_k + \sum_{\ell=1}^{k}(k-\ell)(m_{k-\ell} - m_{k-\ell+1}) = \sum_{\ell=1}^{k} m_\ell$. Modulo p^{M+1} the above only vanishes if $d_k = 0$ or $d_k \equiv -\delta_{i'} \bmod p$ for some $m_{k+1} < i' \leq m_k$. One of these conditions has to be satisfied, since $f(z+s) = f(x) \equiv 0 \bmod p^e$ and $e \geq M+1$. In the first case, we directly get $\nu_p(s) \geq k+1 > k$ and hence x is close to z for $\sum_{\ell=1}^{k} m_\ell(z) < e' \leq \sum_{\ell=1}^{k+1} m_\ell(z)$. In the second case, we have $v_{i'} = \nu_p(z - z_{i'}) = k$ and

with $\delta_{i'}p^k = p^{v_{i'}}\delta_{i'} = z - z_{i'}$ we can write

$$k+1 \leq \nu_p\big(\sum_{j>k} d_j p^j\big) = \nu_p\big(p^{v_{i'}}\delta_{i'} - \delta_{i'}p^k + \sum_{j>k} d_j p^j\big) = \nu_p(z - z_{i'} + s) = \nu_p(x - z_{i'}).$$

Now we use that $m_\ell(z) = m_\ell(z')$ for $\ell \leq \nu_p(z - z')$. Since $\nu_p(z - z_{i'}) = k$ we get $\sum_{\ell=1}^{k} m_\ell(z) = \sum_{\ell=1}^{k} m_\ell(z_{i'})$ and x is close to $z_{i'}$ for $\sum_{\ell=1}^{k} m_\ell(z_{i'}) < e' \leq \sum_{\ell=1}^{k+1} m_\ell(z_{i'})$. Replacing z by $z_{i'}$ does not change anything for $e' \leq \sum_{\ell=1}^{k} m_\ell$, so without loss of generality assume that x is close to z, or rather that we initially chose the "correct" root to be z. Note that x is actually close to all $m_{k+1}(z)$ roots that are congruent to z modulo p^{k+1} for $0 < e' \leq \sum_{\ell=1}^{k+1} m_\ell(z)$. □

4.2. Number of solutions

In this subsection we use the results of the previous subsection to calculate the number of solutions $x \in \mathbb{Z}/p^e\mathbb{Z}$ of $f(x) = 0$.

For the solutions $z + s \in \mathbb{Z}/p^e\mathbb{Z}$ above z modulo p^e we have s of the form $s = \sum_{i=1}^{e-1} d_i p^i$ and since $0 \leq d_i < p$ we can have at most p^{e-1} of them. Theorem 4.5 shows that it is sufficient to consider the solutions close to the roots. For $0 < e \leq m_1(z)$ all p^{e-1} solutions above z are close to z, since $\nu_p(s) \geq 1 > k = 0$ for all s. Starting from $e = m_1(z) + 1$ we can set $d_1 = 0$ for all solutions close to z and by definition we can set $d_k = 0$ for all $e > \sum_{\ell=1}^{k} m_\ell(z)$. This gives us the following lemma:

Lemma 4.6. *In $\mathbb{Z}/p^e\mathbb{Z}$ with $\sum_{\ell=1}^{k-1} m_\ell(z) < e \leq \sum_{\ell=1}^{k} m_\ell(z)$ the equation $f(x) = 0$ has exactly p^{e-k} solutions close to a root z.*

If z is a single root, i.e. $m_\infty(z) = 1$, then there exists an $e_{\text{fin}} = \max\{e \in \mathbb{N} \mid m_e(z) > 1\}$. Let $M = \sum_{\ell=1}^{e_{\text{fin}}} m_\ell(z)$, then there are $p^{M-e_{\text{fin}}}$ solutions close to z modulo p^M. Since $m_i(z) = 1$ for $i > e_{\text{fin}}$ we get that there are $p^{(M+k)-(e_{\text{fin}}+k)} = p^{M-e_{\text{fin}}}$ solutions close to z modulo p^{M+k} for all $k \in \mathbb{N}$.

Remark 4.7. *For a root z with $m_\infty(z) = 1$ there is an upper bound of $p^{M-e_{\text{fin}}}$ solutions close to it. Both M and e_{fin} depend on z and we can compute $M - e_{\text{fin}}$ directly as*

$$M - e_{\text{fin}} = \sum_{\ell=1}^{e_{\text{fin}}} m_\ell(z) - e_{\text{fin}} = \sum_{\ell=1}^{e_{\text{fin}}} \ell(m_\ell(z) - m_{\ell+1}(z)) = \sum_{z_i \neq z} \nu_p(z - z_i).$$

If z is not a single root there is no such bound. Since $m_\ell(z) \leq n$ the most solutions close to z exist if all z_i are equal, so for $m_\ell(z) = n$ for all $\ell \in \mathbb{N}$. Then $\sum_{\ell=1}^{k} m_\ell(z) = nk$ and the number of solutions close to z modulo p^e is

$$p^{e-\lceil \frac{e}{n} \rceil} \leq p^{e\frac{n-1}{n}}.$$

Up to now we only considered a single root z. If we want the total number of solutions of $f(x) = 0$ in $\mathbb{Z}/p^e\mathbb{Z}$ we have to consider double counting. For $\nu_p(z_i - z_j) = k'$ we only distinguished between solutions close to z_i and solutions close to z_j when we set $d_{k'} = 0$, i.e. modulo p^e for $e > \sum_{\ell=1}^{k'} m_\ell(z_i) = \sum_{\ell=1}^{k'} m_\ell(z_j)$. So for $\sum_{\ell=1}^{k-1} m_\ell(z) < e \leq \sum_{\ell=1}^{k} m_\ell(z)$ the p^{e-k} solutions modulo p^e close to the root z mentioned in Lemma 4.6 are close to $m_k(z_i)$ roots in total.

Theorem 4.8. *Let $f(x) = \prod_{i=1}^{n}(x - z_i)$ with $z_i \in \mathbb{Z}$, p a prime and $m_\ell(z) = |\{i \in \mathbb{N} \mid z_i \equiv z \bmod p^\ell, 1 \leq i \leq n\}|$. Then the total number of solutions to $f(x) \equiv 0$ modulo p^e is*

$$\sum_{i=1}^{n} \frac{p^{e-k_i}}{m_{k_i}(z_i)}$$

for k_i such that $\sum_{\ell=1}^{k_i-1} m_\ell(z_i) < e \leq \sum_{\ell=1}^{k_i} m_\ell(z_i)$.

Modulo p there can be at most n solutions. This requires $z_i \not\equiv z_j \bmod p$ for $i \neq j$. But the highest possible number of solutions close to a given z is $p^{e\frac{n-1}{n}}$ and this requires all the z_i to be equal.

Corollary 4.9. *The total number of solutions to $f(x) \equiv 0 \bmod p^e$ is strictly smaller than $np^{e\frac{n-1}{n}}$.*

4.3. Modulo composite numbers

Let $|R|$ be the number of solutions to $f(x) \equiv 0 \bmod C$ for a composite number C. To find a bound for $|R|$ it is sufficient to find a bound on the number of solutions per prime power factor $p_i^{e_i}$ of C. By [13] we know that a random C has $\log\log C$ different prime factors, considered asymptotically. For each factor $p_i^{e_i}$ of C there are less than $np_i^{e_i\frac{n-1}{n}}$ solutions and using the Chinese Remainder Theorem the bound for the number of solutions modulo C is just the product.

Theorem 4.10. *Let $C = \prod_{i=1}^{t} p_i^{e_i}$ be the prime factorisation of $C \in \mathbb{N}_{>1}$ and $f(x) = \prod_{i=1}^{n}(x - z_i)$ with $z_i \in \mathbb{Z}$. Then the number of solutions to $f(x) \equiv 0$ modulo C is*

$$|R| < \prod_{i=1}^{t}\left(np_i^{e_i\frac{n-1}{n}}\right) = n^t\left(\prod_{i=1}^{t}p_i^{e_i}\right)^{\frac{n-1}{n}} = n^t C^{\frac{n-1}{n}} \approx n^{\log\log C} C^{\frac{n-1}{n}} \tag{2}$$

and the ratio is bounded by

$$\frac{|R|}{C} \leq \frac{n^t C^{\frac{n-1}{n}}}{C} = n^t C^{-\frac{1}{n}} \approx \frac{n^{\log\log C}}{\sqrt[n]{C}} \underset{C\to\infty}{\longrightarrow} 0.$$

Due to the weak bound in Corollary 4.9, the number of solutions is significantly smaller in practice. But as a heuristic for the following sections this decreasing behaviour of $|R|/C$ is sufficient.

Remark 4.11. *For our PTE solutions with $n = 7$ we find that roughly half the prime power factors of C are square-free and therefore admit at most $n = 7$ solutions in contrast to $np^{\frac{n-1}{n}}$. For these PTE solutions the bound in Equation (2) is too large by a factor of size roughly $|R|$. More empirical values can be found in Table 3.*

5. Optimising for single PTE solutions

As we have seen, we need to find the set of relevant residues

$$R = \{r \in \mathbb{N} \mid 0 \leq r < C,\ a(r) \equiv b(r) \equiv 0 \bmod C\}.$$

Calculating $a(r) \bmod C$ (or $b(r)$) directly for all $0 \leq r < C$ is too slow for the sizes of C we are dealing with. Since we know the factorisation, it is useful to compute the individual factors $r - a_i$ (or $r - b_i$) and multiply them reducing modulo C after every step. Otherwise the evaluations $a(r), b(r)$ can become large. Additionally, we use the Chinese Remainder Theorem to work with the prime power factors of C instead.

Also the search interval I is usually too large to process it at once and we have two options how to divide it. We can split it up into smaller chunks of fixed size (e.g. 2^{20} elements) and for each one find all representatives of all residue classes in that chunk. The other option is to fix one residue class for each iteration and to find all representatives of that residue class in the search interval. In the following we refer to these two strategies as "fixed interval" or "fixed residue" approach, respectively.

When using only one solution, we have seen that it suffices to check smoothness for $a(\ell)$ and $b(\ell)$ with $\ell \in \{r + kC \mid r \in R, k \in \mathbb{Z}\} \cap I$. Let Z be the set of roots of the polynomials a and b. Since we verify smoothness of $a(\ell)$ by looking at the factors $\ell - a_i$, we have to calculate the smoothness of

$\{r + kC - z \mid r \in R, z \in Z, k \in \mathbb{Z}\} \cap I$. For polynomials of degree n and assuming that all roots are distinct, the size of this set is roughly $2n|R|/C$ the size of I. If one of the factors $\ell - a_i$ is not smooth, we can skip the other factors. Hence, in practice we need to test the smoothness of less than $|I| \cdot 2n|R|/C$ elements.

Depending on how we divide the search interval, we can apply different strategies for smoothness testing. Batch smoothness tests for arbitrary sets like those from Bernstein [2] or Franke, Kleinjung, Morain and Wirth [12] depend on the size of the elements and the size of the set. For large intervals with large elements they are therefore slower than sieves. We could also use a polynomial sieve to check the set $f(I')$ for a smaller interval I' instead of the interval I for a polynomial $f \in \mathbb{Z}[X]$, e.g. $f(x) = a(Cx + r)$ or $f(x) = Cx + r - z$. The problem is that the polynomial sieve requires us to solve $f(x) = 0 \bmod q$ for every prime power q, i.e. in every loop. So, for the fixed interval approach we use a regular sieve and for small $|R|/C$ we switch to the fixed residue approach and apply the batch smoothness tests.

5.1. Fixed interval size

Of our two new approaches this one is closer to the original paper [10] and is only suitable for solutions with $|R|$ small enough to fit in the RAM and $|R|/C$ not too small. Let $J \subset I$ be the chunk of the search interval for one iteration and for simplicity assume that $C > |J|$ where $|J|$ is fixed.

5.1.1. Computing congruences

We start by computing the set of relevant residues R. Since the resulting residue classes modulo C have no predictable order, we write them in a list and order it afterwards. The list can get long and this pre-computation might take some time, but it is essential for the efficiency of this approach and it has to be done only once. This step limits the size of R for which this approach is applicable.

Looking for representatives in J for all $r \in R$ is slow if we have $|R| > |J|$. Therefore, we need the ordered list $r_1 < \cdots < r_{|R|}$ of residues to find the smallest representative in the chunk and then take the next one until we leave the chunk. This way we still only need to check about $2n|R|/C$ of the whole chunk. Here we have to keep in mind that we could wrap around from $r_{|R|} + kC$ to $r_1 + (k+1)C$.

5.1.2. Smoothness testing

Now that we know R, we can check the smoothness of the elements in $\{r + kC - z \mid r \in R, z \in Z, k \in \mathbb{Z}\} \cap J$ for each chunk J. Since sieving is so efficient on J, the fastest option (at least for large $|R|/C$) is to compute the smoothness for all $\ell \in J$ with a sieve.

We use a sieve of Eratosthenes with the powers of all primes below the smoothness bound. Algorithm 1 shows the basic smoothness sieve to create a look-up-table of smooth integers in a chunk J. It returns a list of zeros and ones that correspond to the smoothness of the elements of J. It runs through all relevant powers of all primes below the bound, so there is a fixed number of loops that have to be executed at least once. But above this threshold it is practically linear in the size of the chunk and only depends slightly on the size of the elements. The only operations are additions and some multiplications, which are easy and fast to compute.

In step 4 of Algorithm 1 we can choose a smaller bound for p^e to improve efficiency. This would result in a stronger notion of smoothness like power-smooth for p^e smaller than the smoothness bound. The loop for p^e has roughly $|J|/p^e$ iterations. Therefore, the smallest p^e have the biggest computational impact, while contributing only p to the product. For example we have at least 2^{19} multiplications for 2 in a chunk of size 2^{20}. If we can tolerate a non-smooth factor, then we can safely omit a factor of 2 and write it off as non-smooth part. The logarithm allows us to replace multiplications with additions and taking only integer approximations makes the logarithm efficient. Since SQISign and SIGNITC tolerate such a non-smooth factor, the modification to use rounded logarithms and to ignore the smallest primes or at least their small powers is a good option to speed up this part of the calculations.

Algorithm 1 `SmoothnessSieve`

Require: Set of smooth primes P_{smooth}, chunk $J = (j_1, \dots, j_s)$
Ensure: Look-up-table of smooth integers in J

Set $L = (l_1, \dots, l_s) = (1, \dots, 1)$
for all $p \in P_{\text{smooth}}$ **do**
  Set $e = 1$
  while $p^e < j_s$ **do**
    Find smallest $i \in \{1, \dots, s\}$ such that $p^e \mid j_i$ or set $i = s + 1$
    while $i \leq s$ **do**
      $l_i \leftarrow p \cdot l_i$
      $i \leftarrow i + p^e$
    $e \leftarrow e + 1$
for $i = 1, \dots, s$ **do**
  if $l_i = j_i$ **then** $l_i \leftarrow 1$ ▷ j_i *is smooth*
  else $l_i \leftarrow 0$ ▷ j_i *is not smooth*
return L

5.2. Fixed residue class

As mentioned before, we only need to check the smoothness for a fraction $2n|R|/C$ of the interval I. The approach from Section 5.1 can become inefficient in two cases. Firstly, because R is too large to use an ordered list of residues or, secondly, because $|R|/C$ gets too small. For decreasing $|R|/C$, the number of representatives per chunk also decreases and computing the smoothness for all elements in the chunk becomes disproportional. In these cases we can look at the representatives of one residue class at a time, i.e. the set $\{r + kC \mid k \in \mathbb{Z}\} \cap I$ for each $r \in R$. In contrast to the other approach, we do not compute the whole set of relevant residues R, but a single $r \in R$ at a time (one for each iteration).

5.2.1. Smoothness testing

We need to check the smoothness of the elements in $\{r + kC - z \mid z \in Z, k \in \mathbb{Z}\} \cap I$. Using a sieve to compute the smoothness of the whole search interval I is too slow.

Franke, Kleinjung, Morain and Wirth [12] present an algorithm that computes the smooth part of each integer m_i in an arbitrary set S. They use product and remainder trees to efficiently compute the product P of all primes below a smoothness bound B and $\widetilde{m}_i = P \bmod m_i$. Then they repeatedly apply the gcd to find the smooth parts. Algorithm 2 shows how this can be used to test smoothness. For efficiency, the product of all elements in S should be smaller than P. Larger sets can be divided and the algorithm gets applied to the subsets. In this case, the computation of P can be thought of as pre-computation that has to be done only once. Bernstein [2] uses repeated squaring of P and only one gcd instead. This approach is used in Algorithm 3. He claims that these batch smoothness tests have a better asymptotical complexity than individual smoothness tests and that in practice Algorithm 3 is faster than Algorithm 2.

Similar to the sieve, small primes have a disproportional impact on the running time of these test. Since they can appear with high powers, they can cause many repetitions of the while-loop in Algorithm 2 or a large number e of squarings in Algorithm 3. We can omit them and write them off as non-smooth part or use trial division for these small primes to improve the algorithms.

These tests can not efficiently check all elements of $\{r + kC - z \mid z \in Z, k \in \mathbb{Z}\} \cap I$ at once. One of the main arguments of the original tree approach [10] is that we usually do not have to compute the smoothness of all factors $\ell - a_i$ of $a(\ell)$. Since we need all of them to be smooth, we can skip ℓ if at least one $\ell - a_i$ is not smooth. Therefore, most of the time we only need to check a few factors $\ell - z$ of $a(\ell)$ and $b(\ell)$ instead of up to $2n$ to reject an element ℓ. So we start with $\ell \in I_r = \{r + kC \mid k \in \mathbb{Z}\} \cap I$ and only check the other factors $\ell - z$ for $z \in Z \setminus \{0\}$ if $\ell = \ell - 0$ is smooth.

Algorithm 2 Batch Smoothness FrKlMoWi [12]

Require: Smoothness bound B, set of integers $S = (m_1, \ldots, m_s)$
Ensure: Look-up-table of smooth integers in S
Compute product of smooth primes $P = \prod_{p<B} p$ using a product tree
Compute $M = \prod_{i=1}^{s} m_i$ using a product tree and save intermediate steps
Use intermediate steps to compute $\widetilde{m}_i = P \bmod m_i$ using a remainder tree
Set $L = (l_1, \ldots, l_s) = S$
for $i = 1, \ldots, s$ **do**
    while $\widetilde{m}_i > 1$ **do**
        $\widetilde{m}_i \leftarrow \gcd(\widetilde{m}_i, l_i)$
        $l_i \leftarrow l_i / \widetilde{m}_i$
    if $l_i = 1$ or $\widetilde{m}_i = 0$ **then** $l_i \leftarrow 1$ ▷ m_i *is smooth*
    else $l_i \leftarrow 0$ ▷ m_i *is not smooth*
return L

Algorithm 3 Batch Smoothness Bernstein [2]

Require: Smoothness bound B, set of integers $S = (x_1, \ldots, x_s)$
Ensure: Look-up-table of smooth integers in S
Compute product of smooth primes $P = \prod_{p<B} p$ using a product tree
Compute $M = \prod_{i=1}^{s} x_i$ using a product tree and save intermediate steps
Use intermediate steps to compute $\widetilde{x}_i = P \bmod x_i$ using a remainder tree
Set $L = (l_1, \ldots, l_s)$
for $i = 1, \ldots, s$ **do**
    Find smallest integer e such that $2^{2^e} \geq x_i$
    Compute $y_i = \widetilde{x}_i^{2^e} \bmod x_i$ via repeated squaring
    if $y_i = 0$ **then** $l_i \leftarrow 1$ ▷ x_i *is smooth*
    else $l_i \leftarrow 0$ ▷ x_i *is not smooth*
return L

6. Practical results

The target size of the integers $a(\ell)/C$ and $b(\ell)/C$ is 2^{240} to 2^{256} for a 256 bit prime, 2^{370} to 2^{384} for a 384 bit prime and 2^{500} to 2^{512} for a 512 bit prime. The smoothness bounds are taken from [10, Table 3] for a probability of 2^{-30}. The start values depend on the degree n of the PTE solution, the size of C and the target size (cf. Remark 3.1).

We test three different approaches. Let us briefly recall them for clarification:

Naive Fixed Interval Use a sieve to compute smoothness of all elements in a chunk J of fixed size. For all $\ell \in J$ look up the smoothness of the factors $\ell - a_i$ and $\ell - b_i$. If $a(\ell)$ and $b(\ell)$ are smooth, test if $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$.

New Fixed Interval Pre-compute all residue classes such that the representatives satisfy $a(r) \equiv b(r) \equiv 0 \bmod C$. Use a sieve to compute smoothness of all elements in a chunk of fixed size. For all representatives in that chunk look up the smoothness of the factors of $a(r)$ and $b(r)$.

Fixed Residue Compute a single residue class such that the representatives satisfy $a(r) \equiv b(r) \equiv 0 \bmod C$. Use batch smoothness tests to first compute the smoothness of the representatives r and then the other factors of $a(r)$ and $b(r)$ for the the smooth representatives.

The implementations and parameter sets used for the following tests are available at https://github.com/kahrens-code/twin-smooth-integers. An overview of the parameters can be found in Table 3.

Table 3. Properties of PTE solutions with $n = 7$ and $C \leq 2^{60}$.

parameters		sizes		relevant elements in chunk	
C	5,145,940,800	C	33 bit	estimated	20760
$\lvert R\rvert$	101,879,232	$\lvert R\rvert$	27 bit	computed (max)	20847
ratio	1.98×10^{-2}	R	0.8 GB	size of chunk	1,048,576
C	13,967,553,600	C	34 bit	estimated	17912
$\lvert R\rvert$	238,592,172	$\lvert R\rvert$	28 bit	computed (max)	17969
ratio	1.71×10^{-2}	R	1.8 GB	size of chunk	1,048,576
C	293,318,625,600	C	39 bit	estimated	6599
$\lvert R\rvert$	1,845,811,968	$\lvert R\rvert$	31 bit	computed (max)	6693
ratio	6.29×10^{-3}	R	14 GB	size of chunk	1,048,576
C	1,037,896,675,200	C	40 bit	estimated	3048
$\lvert R\rvert$	3,017,192,640	$\lvert R\rvert$	32 bit	computed (max)	3147
ratio	2.91×10^{-3}	R	22 GB	size of chunk	1,048,576
C	5,771,633,313,600	C	43 bit	estimated	303
$\lvert R\rvert$	1,670,145,204	$\lvert R\rvert$	31 bit	computed (max)	378
ratio	2.89×10^{-4}	R	12 GB	size of chunk	1,048,576
C	1,440,534,083,788,800	C	51 bit	estimated	128
$\lvert R\rvert$	4,753,277,448,192	$\lvert R\rvert$	43 bit	computed (max)	
ratio	3.30×10^{-3}	R	35 TB	size of chunk	1,048,576
C	11,471,328,290,822,400	C	54 bit	estimated	16
$\lvert R\rvert$	176,264,555,376	$\lvert R\rvert$	38 bit	computed (max)	
ratio	1.54×10^{-5}	R	1.3 TB	size of chunk	1,048,576
C	15,256,916,548,473,600	C	54 bit	estimated	8
$\lvert R\rvert$	113,569,873,872	$\lvert R\rvert$	37 bit	computed (max)	
ratio	7.44×10^{-6}	R	0.8 TB	size of chunk	1,048,576
C	106,911,286,818,336,000	C	57 bit	estimated	13
$\lvert R\rvert$	1,307,456,866,800	$\lvert R\rvert$	41 bit	computed (max)	
ratio	1.22×10^{-5}	R	10 TB	size of chunk	1,048,576
C	314,073,647,406,288,000	C	59 bit	estimated	2
$\lvert R\rvert$	457,609,903,380	$\lvert R\rvert$	39 bit	computed (max)	
ratio	1.46×10^{-6}	R	3.3 TB	size of chunk	1,048,576

6.1. Comparing fixed interval results

First, we compare the new to the naive approach. In our tests the chunks are of size 2^{20} and we use power-smoothness for more efficient smoothness sieving. In Table 4 we only compare timings without smoothness computation. These tests were done on an Intel Core i7-6700 @3.40 GHz with 32 GB memory and 13 MB cache using the "128 bit" C implementation.

As expected, the naive algorithm has similar running times for different PTE solutions, but the new algorithm depends heavily on the ratio of relevant residues $|R|/C$. In the tests for Table 4 the new approach is always significantly faster than the naive one. But the application of the new approach is limited. Table 3 shows that even for $C \leq 2^{60}$ the size of R can become so large, that pre-computing of all the residue classes is too slow or requires too much memory.

In tests with the C implementation using GMP the new approach was slower than the naive one. Our explanation is that the naive approach mostly uses small (standard) integers ($< 2^{20}$) and only if all factors are smooth it needs larger (GMP) integers to check $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. However, the new approach does the modulo computation first and hence mostly uses large (GMP) integers.

Table 4. Running times of the different algorithms implemented in C using the native `__int128` for a search interval of size 2^{20} starting at the start value. All PTE solutions have $n = 7$. More detail in Section 6.1.

parameters of PTE solutions		size of p [bit]	smooth. bound (2^x)	start of chunk (2^x)	running time [ms]	
					naive	new
C	5,145,940,800	256	17	40	2.436	0.089
$\lvert R\rvert$	101,879,232	384	25	60	2.056	0.073
ratio	1.98×10^{-2}	512	27	80	1.024	0.043
C	13,967,553,600	256	17	40	2.556	0.075
$\lvert R\rvert$	238,592,172	384	25	60	2.071	0.082
ratio	1.71×10^{-2}	512	27	80	1.024	0.043
C	293,318,625,600	256	17	40	2.265	0.033
$\lvert R\rvert$	1,845,811,968	384	25	60	0.837	0.022
ratio	6.29×10^{-3}	512	27	80	0.691	0.021
C	1,037,896,675,200	256	17	40	2.553	0.020
$\lvert R\rvert$	3,017,192,640	384	25	60	0.865	0.011
ratio	2.91×10^{-3}	512	27	80	0.717	0.010
C	5,771,633,313,600	256	17	40	2.807	0.007
$\lvert R\rvert$	1,670,145,204	384	25	60	0.866	0.005
ratio	2.89×10^{-4}	512	27	80	0.714	0.005

Smoothness sieves

The running time of the entire algorithm is dominated by the smoothness sieve. It takes several milliseconds to several hundred milliseconds and hence the difference between the naive and the new approach is only relevant for large scale tests. Table 5 shows times for different types of smoothness sieves. As in [10], we used power-smoothness in all smoothness sieves. This improves the efficiency and makes the comparison easier. The first one is the basic one described in Algorithm 1. The second sieve just applies rounded logarithms. The third omits the prime 2 and its powers and the fourth omits all prime powers smaller than 1024. These tests were performed on a (slower) Intel Core i3-6006U @2.00 GHz running 64-bit Windows 10 with 8 GB memory using a C implementation with GMP. So for our applications, the prime truncated logarithmic sieve omitting the prime 2 and its powers seems to be the fastest smoothness sieve.

Table 5. Different smoothness sieves for one chunk of size 2^{20} implemented in C using GMP.

size of p	256 bit	384 bit	512 bit
size of elements	42 bit	60 bit	78 bit
smoothness bound	2^{17}	2^{25}	2^{27}
smoothness sieve	359 ms	638 ms	1147 ms
logarithmic sieve	109 ms	312 ms	819 ms
prime trunc log sieve	97 ms	297 ms	809 ms
power trunc log sieve	112 ms	316 ms	822 ms

6.2. Comparing fixed residue results

For simplicity we choose batch sizes that are powers of 2 and we assume that the elements in one batch have roughly the same size. To have a meaningful comparison between the fixed residue and the fixed interval approach we look at the time it takes to check 1,000,000 representatives.

We use the naive fixed interval approach, because it is faster than the new one when using GMP and it is always applicable. For its smoothness sieve we use power-smoothness and a logarithmic sieve omitting the prime 2 and its powers. The comparison for five PTE solutions with $n = 7$ is shown in Table 6. The tests were done on an Intel Core i3-6006U @2.00 GHz running 64-bit Windows 10 with 8 GB memory using a C implementation with GMP. The batch smoothness test [12] is the one presented in Algorithm 2 and [2] corresponds to Algorithm 3. The worst case is choosing the residue class $[0] = [C]$, because every representative is smooth in this case. The best case is the fastest residue in our test and the average case shows the average over five random residues.

Table 6. Time it takes to compute the smoothness of 1,000,000 representatives using a sieve (cf. Section 6.1) or batch smoothness tests by [12] and [2] (cf. Section 6.2). All implemented in C using GMP.

size of p		256 bit			384 bit			512 bit		
size of elements		42 bit			60 bit			78 bit		
smoothness bound		2^{17}			2^{25}			2^{27}		
PTE $n = 7$	case	[12]	[2]	sieve	[12]	[2]	sieve	[12]	[2]	sieve
C 33 bit	worst	29 s	34 s		156 s	159 s		147 s	146 s	
$\|R\|$ 27 bit	best	3 s	4 s	5 s	30 s	31 s	15 s	32 s	33 s	42 s
ratio 1.98×10^{-2}	average	5 s	6 s		43 s	43 s		40 s	43 s	
C 34 bit	worst	31 s	36 s		165 s	164 s		148 s	157 s	
$\|R\|$ 28 bit	best	3 s	3 s	5 s	31 s	31 s	17 s	33 s	34 s	44 s
ratio 1.71×10^{-2}	average	5 s	5 s		45 s	47 s		42 s	43 s	
C 39 bit	worst	32 s	40 s		181 s	186 s		181 s	189 s	
$\|R\|$ 31 bit	best	3 s	3 s	13 s	31 s	32 s	44 s	33 s	34 s	123 s
ratio 6.29×10^{-3}	average	4 s	4 s		38 s	39 s		46 s	47 s	
C 40 bit	worst	33 s	38 s		179 s	184 s		188 s	201 s	
$\|R\|$ 32 bit	best	3 s	4 s	31 s	30 s	31 s	95 s	33 s	33 s	254 s
ratio 2.91×10^{-3}	average	4 s	4 s		43 s	44 s		40 s	42 s	
C 43 bit	worst	34 s	40 s		178 s	192 s		206 s	214 s	
$\|R\|$ 31 bit	best	3 s	4 s	289 s	30 s	31 s	936 s	32 s	33 s	2572 s
ratio 2.89×10^{-4}	average	4 s	4 s		41 s	42 s		36 s	38 s	

As expected, we see that the times for the naive approach increase proportionally to the inverse of the ratio $|R|/C$. The fixed residue approach does not depend on this ratio and only the worst case times increase for larger C. Since the batch smoothness tests depend on the product of all smooth primes, it seems reasonable that the gap between a smoothness bound of 2^{17} and 2^{25} is larger than the gap between 2^{25} and 2^{27}. Note that the smoothness sieve in the naive approach only uses power-smoothness, whereas the batch smoothness tests in the fixed residue approach compute regular smoothness. If we would sieve for regular smoothness, the naive approach would be even slower and less related to the tree approach [10]. Simulating power-smoothness in the batch smoothness tests would be slower than using regular smoothness.

These results show that for $n = 7$ the fixed residue approach is more efficient than the naive fixed interval approach in most cases and probably similar to the original tree approach [10]. For $n = 9$ there are only two solutions with $C < 2^{60}$ and for $n > 9$ there is at most one known solution with $C < 2^{100}$ [10, Table 2]. Hence, the tree approach can at most be twice as efficient as the naive one. Judging from

our experiments and the heuristic in Section 4.3, for the solutions with larger C we expect a significant improvement over the naive approach and in total even an improvement over the tree approach. So for finding large primes with small smoothness bounds we expect the fixed residue approach to be most efficient.

6.3. Larger tests

We also performed some larger tests in order to find twin smooth primes. Due to the allowed non-smooth part in the smoothness sieve, many of the found primes have neighbors with large factors. The reasonably twin smooth primes are listed on GitHub[3]. The best one we found is of size 372 bit and its neighbors are 2825227-smooth (22 bit).

Table 7. Results of larger test.

target size	n	smooth. bound	tested integers	twin smooth	notes
384	7	2^{25}	2^{40}	869	
384	8	2^{22}	2^{45}	160	complete interval, fixed interval
512	7	2^{27}	2^{36}	0	combined from 3 PTE solutions
512	7	2^{27}	2^{36}	0	complete interval, fixed residue
512	8	2^{31}	2^{30}	1	

Our tests however do not seem to agree with the heuristics given in [10, Table 3] regarding the distribution of twin smooth integers of size 512 bit. In the last three rows of Table 7 we would expect 64, 64 and 2048 twin smooth integers, respectively. For integers of size 384 bit the heuristics match our tests, but the twin smooth integers are distributed very unevenly. The heuristic assumes that smooth numbers are evenly distributed. So the prediction might be true on a larger scale and our test were just too small to be representative.

7. Conclusion and outlook

We have seen that the tree approach for fixed interval sizes presented by Costello, Meyer and Naehrig [10] is well suited to find twin smooth integers with ideal PTE solutions of degree $n = 6$. But for primes in the range of 512 bit and above solutions of higher degree $n \geq 7$ are favorable. Since there are fewer solutions of higher degree and with small C, our fixed residue approach from Section 5.2 can be faster. Our benchmarks and the bounds on $|R|$ and $|R|/C$ led to believe that it is faster for $n \geq 9$ and similar to the tree approach for $n = 7$. The formula for the number of zeros of a given polynomial modulo a prime power using p-adic distances could be of independent interest.

But there are still some open research questions. One problem to address is running larger searches or testing PTE solutions with $n \geq 9$. Another one is to optimize (and parallelize) the code for a specific set of parameters or for the requirements of SQISign or SIGNITC. Computing the smoothness dominates all approaches. Hence, improvements to the smoothness tests would significantly increase the efficiency. Optimizations for the fixed residue approach could be explored by varying the batch sizes or by changing the way we iterate through the different factors $\ell - z$. One new approach would be to use a filtration. So we would first test all $\ell - z_1$ and for the smooth ones test $\ell - z_2$. Then for the smooth ones iteratively

[3] *https://github.com/kahrens-code/twin-smooth-integers*

test the remaining factors. This can be combined with another new approach using a hybrid strategy. Two examples would be to use a polynomial sieve for $\ell = \ell - 0$ or to use Brent's version of Pollard's rho to test the other factors $\ell - z$ on demand.

Finally, the PTE solutions themselves are an interesting field. On the one hand, there is still the question whether there exist ideal solutions for $n = 11$ and $n \geq 13$. On the other hand, there is the problem of finding solutions with small C. Even when systematic ways to construct ideal solutions are known, they produce large differences C most of the time. Investigating this area could especially facilitate the search for larger twin smooth numbers.

Acknowledgments

I would like to thank a few people, who supported me during the work on this paper: Jens Zumbrägel for his general advice and helpful comments on my work, John Abbott for our discussions about implementation, Miriam Sophie Kaesberg for our conversation about counting roots and finally Michael Naehrig for making the PTE solutions for $n = 7$ and $n = 8$ available. Also thanks to the anonymous reviewers for their constructive feedback on earlier versions of this paper.

References

[1] K. Ahrens, SIGNITC: Supersingular isogeny graph non-interactive timed commitments, Cryptology ePrint Archive Paper 2024/1225 (2024).
[2] D. J. Bernstein, How to find smooth parts of integers, (2004).
[3] P. Borwein and C. Ingalls, The Prouhet-Tarry-Escott problem revisited, Enseign. Math. (2) 40(1-2) (1994) 3–27.
[4] G. Bruno, M. Corte-Real Santos, C. Costello, J. K. Eriksen, M. Meyer, M. Naehrig, and B. Sterner, Cryptographic smooth neighbors. In J. Guo and R. Steinfeld, editors, Advances in Cryptology – ASIACRYPT 2023 Springer Nature Singapore (2023) 190–221.
[5] T. Caley, The Prouhet-Tarry-Escott problem, PhD thesis University of Waterloo (2012).
[6] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, CSIDH: An efficient post-quantum commutative group action, In T. Peyrin and S. Galbraith, editors, Advances in Cryptology – ASIACRYPT 2018 pages 395–427 Cham. Springer International Publishing (2018).
[7] J. Chavez-Saab, M. Corte-Real Santos, L. De Feo, J. Komada Eriksen, B. Hess, D. Kohel, A. Leroux, P. Longa, M. Meyer, L. Panny, S. Patranabis, C. Petit, F. Rodríguez Henríquez, S. Schaeffler, and B. Wesolowski, SQIsign algorithm specifications and supporting documentation, Project Homepage (2023).
[8] J. Chernick, Ideal solutions of the Tarry-Escott Problem, Amer. Math. Monthly 44(10) (1937) 626–633.
[9] C. Costello, B-SIDH: Supersingular isogeny diffie-hellman using twisted torsion, In S. Moriai and H. Wang, editors, Advances in Cryptology – ASIACRYPT 2020, Cham. Springer International Publishing (2020) 440–463.
[10] C. Costello, M. Meyer, and M. Naehrig. Sieving for twin smooth integers with solutions to the Prouhet-Tarry-Escott problem. In A. Canteaut and F.-X. Standaert, editors, Advances in Cryptology – EUROCRYPT 2021 Cham. Springer International Publishing (2021) 272–301 .
[11] L. De Feo, D. Kohel, A. Leroux, C. Petit, and B. Wesolowski, SQISign: compact post-quantum signatures from quaternions and isogenies, In S. Moriai and H. Wang, editors, Advances in Cryptology – ASIACRYPT 2020 Cham. Springer International Publishing (2020) 64–93.
[12] J. Franke, T. Kleinjung, F. Morain, and T. Wirth, Proving the primality of very large numbers with fastECPP. In D. Buell, editor, Algorithmic Number Theory Berlin Heidelberg Springer Berlin

Heidelberg (2004) 194–207.
[13] G. H. Hardy and S. Ramanujan. The normal number of prime factors of a number n, Quart. J. 48: (1917) 76–92.
[14] D. Jao and L. De Feo, Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies, In B.-Y. Yang, editor, Post-Quantum Cryptography Springer Berlin Heidelberg (2011) 19–34.
[15] J. H. Loxton and R. C. Vaughan, The estimation of complete exponential sums, Canad. Math. Bull. 28(4):(1985) 440–454.
[16] National Institute of Standards and Technology. Post-quantum cryptography: additional digital signature schemes, (2023).
[17] C. Shuwen. The Prouhet-Tarry-Escott problem. Equal Sums of Like Powers, (2022).
[18] C. L. Stewart, On the number of solutions of polynomial congruences and the equations, J. Amer. Math. Soc. 4:(1991) 793–835.