More Efficient Algorithms for the NTRU Key Generation using the Field Norm

Thomas Pornin[1] and Thomas Prest[2]

[1] NCC Group, thomas.pornin@nccgroup.com
[2] PQShield Ltd., thomas.prest@pqshield.com

Abstract. NTRU lattices [17] are a class of polynomial rings which allow for compact and efficient representations of the lattice basis, thereby offering very good performance characteristics for the asymmetric algorithms that use them. Signature algorithms based on NTRU lattices have fast signature generation and verification, and relatively small signatures, public keys and private keys.

A few lattice-based cryptographic schemes entail, generally during the key generation, solving the *NTRU equation*:

$$fG - gF = q \mod x^n + 1$$

Here f and g are fixed, the goal is to compute solutions F and G to the equation, and all the polynomials are in $\mathbb{Z}[x]/(x^n + 1)$. The existing methods for solving this equation are quite cumbersome: their time and space complexities are at least cubic and quadratic in the dimension n, and for typical parameters they therefore require several megabytes of RAM and take more than a second on a typical laptop, precluding onboard key generation in embedded systems such as smart cards.

In this work, we present two new algorithms for solving the NTRU equation. Both algorithms make a repeated use of the field norm in tower of fields; it allows them to be faster and more compact than existing algorithms by factors $\tilde{O}(n)$. For lattice-based schemes considered in practice, this reduces both the computation time and RAM usage by factors at least 100, making key pair generation within range of smart card abilities.

1 Introduction

NTRU lattices are a class of trapdoor lattices that were introduced by [17], as the core object in which the NTRUEncrypt asymmetric encryption algorithm is expressed. Given a monic polynomial $\phi \in \mathbb{Z}[x]$ of degree n, the lattice is generated by two "short" polynomials f and g modulo ϕ. The coefficients of f and g are very small integers (in NTRUEncrypt, they are limited to $\{-1, 0, 1\}$). The polynomial f and g are secret, but their ratio:

$$h = g/f \bmod \phi \bmod q \tag{1}$$

for a given, small integer q, is public. The polynomial f is chosen so as to be invertible modulo ϕ and q. q is not necessarily prime.

NTRU lattices offer good performance characteristics; they have been reused in several other asymmetric schemes. Some of these schemes require the lattice trapdoor to be "complete", which means that beyond knowledge of f and g, the private key owner must know two other short polynomials F and G that fulfill the *NTRU equation*:

$$fG - gF = q \tag{2}$$

A complete NTRU trapdoor is required for example in the signature scheme NTRUSign [16], an identity-based encryption scheme [9], the signature scheme Falcon [22], and the hierarchical identity-based encryption scheme LATTE [4].

Finding the *shortest* solution (for a given norm) is a hard problem; however, computing a solution which is short enough for the purpose of running an algorithm based on complete NTRU lattices, is doable. Solving the NTRU equation is then part of the key generation process.

While the NTRU equation looks simple, solving it in an efficient manner is nontrivial. Existing algorithms for finding a solution [16,25] have time and space complexities that are at least cubic and quadratic in the dimension n, respectively. For typical parameter sizes, this translates in practice into requiring several megabytes of RAM and taking around 2 seconds on a typical computer.[3] This precludes implementation in many constrained, embedded systems. One could argue that being able to implement the key generation on an embedded device is not too important since one could simply generate it externally and copy-paste the key in the device, but keeping the private key in a tamper-resistant device for its complete lifecycle is often desirable for security (as there is no external exposure at any time) and compliance (e.g. to the FIPS 140-2 norm [20]).

In this article, we show how we can leverage the field norm in polynomial rings to achieve much improved performance for solving the NTRU equation. It allows us to propose two new algorithms based on the field norm, which provide better (time and space) complexities than existing algorithms by quasilinear factors in n (precisely, at least $O(n/\log n)$). As a by-product, we developed an improved algorithm for computing polynomial resultants, when one of the polynomials is a cyclotomic (see Section 3). Table 1 compares the asymptotic complexity achieved by our new techniques with existing known methods.

We implemented both the classic resultant-based NTRU solver, and our new algorithms, with similar optimization efforts and tools. This allowed direct measures of the performance improvement of our techniques, which corroborated the asymptotic analysis: for a typical degree ($n = 1024$), the new methods are faster *and* smaller than the classic algorithms, both by a factor of 100 or more.

1.1 Techniques

Our algorithms rely on repeatedly applying the project-then-lift paradigm, a well-known paradigm in algorithmic number theory and cryptanalysis which

[3] All the timings in this document are provided for a MacBook Pro laptop (Intel Core i7-6567U @ 3.30 GHz), running Linux in 64-bit mode.

Method	Time complexity	Space complexity
Resultant [16]	$\tilde{O}(n(n^2+B))$	$O(n^2B)$
HNF [25]	$\tilde{O}(n^3B)$	$O(n^2B)$
TowerSolverR (Algorithm 4)	$O((nB)^{\log_2(3)}\log n)$ [K]	$O(n(B+\log n)\log n)$
	$\tilde{O}(nB)$ [SS]	
TowerSolverI (Algorithm 5)	$O((nB)^{\log_2(3)}\log^2 n)$ [K]	$O(n(B+\log n))$
	$\tilde{O}(nB)$ [SS]	

Table 1. Comparison of our new methods for solving the NTRU equation with existing ones. B denotes an upper bound on $\log\|f\|, \log\|g\|$. The tag [K] indicates that Karatsuba's algorithm was used for large integer multiplications, and [SS] indicates the Schönhage-Strassen algorithm was used.

consists of projecting a problem onto a subset in which it becomes easier, before lifting the solution to the original set.

In our case, we rely on using the presence of *towers of fields* and *towers of rings*. As an illustration, let us consider the following tower of fields:

$$\mathbb{K}_\ell \quad / \quad \mathbb{K}_{\ell-1} \quad / \quad \dots \quad / \quad \mathbb{K}_1 \quad / \quad \mathbb{K}_0 = \mathbb{Q}$$

where $\forall i, \mathbb{K}_i = \mathbb{Q}[x]/(x^{2^i}+1)$, and the associated tower of rings (which are the rings of integers of the corresponding fields) with $n = 2^\ell$:

$$\mathbb{Z}[x]/(x^n+1) \supsetneq \mathbb{Z}[x]/(x^{n/2}+1) \supsetneq \dots \supsetneq \mathbb{Z}[x]/(x^2+1) \supsetneq \mathbb{Z}$$

It is well known that the field norm can map any element $f \in \mathbb{Z}[x]/(x^n+1)$ onto a smaller ring of its tower. This fact is exploited in the "overstretched NTRU" attack [1], where problems are mapped to a smaller ring, then solved, at which point the solution is lifted back to the original ring.

However, what is not exploited in these works is the fact that the field norm plays nicely with towers of fields: for a tower of field extensions $\mathbb{L}/\mathbb{K}/\mathbb{J}$ and $f \in \mathbb{L}$, we have $\mathrm{N}_{\mathbb{K}/\mathbb{J}} \circ \mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \mathrm{N}_{\mathbb{L}/\mathbb{J}}(f)$ (where N denotes the field norm). This fact is at the heart of our algorithms.

We first repeatedly use the field norm to project over $\mathbb{Z}$ equations which are originally over $\mathbb{Z}[x]/(x^n+1)$; this is the *descent* phase. It turns out that these equations can be solved much faster over $\mathbb{Z}$. We then use the properties of the field norm to lift our solutions back in $\mathbb{Z}[x]/(x^n+1)$; this is the *lifting* phase. This simple principle allows us to gain a factor at least $\tilde{O}(n)$ over classical algorithms.

We apply a few additional tricks such as memory-laziness, the use of residue number systems, or the fact that in cyclotomic fields, the Galois conjugates of an element in FFT or NTT representation are straightforward to compute. These techniques make our implementation faster and more memory-efficient.

1.2 Applications

Our new algorithms impact at least four existing lattice-based schemes.

NTRUSign. The first scheme which entails solving this equation in the key generation is NTRUSign [16]. In its current form, this scheme is however insecure, but for reasons independent of the key generation.

Falcon. In the signature scheme Falcon [22], the costliest part of the key generation consists of solving an NTRU equation. Without our techniques, it would require about 2^{33} clock cycles on a recent laptop computer, and 3 MBytes of RAM, for the highest security level, limiting its usability for many embedded devices. As we gain a factor 100 in speed *and* memory, this significantly widens the range of the devices on which Falcon can be entirely implemented.

DLP. The setup phase of the identity-based encryption scheme DLP [9] is identical to the key generation of Falcon. The same remark as above applies.

LATTE. Very recently, Campbell and Groves [4] introduced LATTE, a hierarchical identity-based encryption scheme which essentially combines [9] with the Bonsai trees construction of [5]. At each extraction of a user secret key, LATTE needs to solve a *generalized* NTRU equation. More precisely, given $f_1, \ldots, f_k \in \mathbb{Z}[x]/(\phi)$, it needs to compute $F_1, \ldots, F_k \in \mathbb{Z}[x]/(\phi)$ such that

$$\sum f_i F_i = q$$

and k may in practice be equal to 3 or 4 (see [4, slide 23]). Our techniques can be extended in a straightforward way to solve this kind of equation. The impact for LATTE is even more important than for the aforementioned works, as an authority may need to perform many extractions (typically, one per user and per key renewal period). See also [26] for a more complete specification of LATTE.

Our algorithms were also used in the key generation of subsequent works, such as BAT [11], and Falcon variants ModFalcon [7] and Mitaka [10].

1.3 Related Works

The NTRU equation was first introduced and solved in [16].

Another method for solving the NTRU equation was suggested by Stehlé and Steinfeld [25], using the Hermite Normal Form. The most space-efficient algorithm for computing the HNF is due to Micciancio and Warinschi [19]; however, like the method based on resultants, it has quadratic space complexity and quasi-cubic time complexities, and does not solve the RAM usage issue.

The use we make of the field norm is reminiscent of the "overstretched NTRU" attack by [1], except that these works are cryptanalytic and use the field norm once, whereas ours uses it repeatedly and improves cryptographic constructions.

1.4 Roadmap

In Section 2, we introduce our notations, and recall the classic resultant-based algorithm; we also describe some known mathematical tools that we will use in our new algorithm. In Section 3, we present a novel method for computing specific cases of resultants; our new algorithm builds on this method, and is described in Section 4, where we also show how it can be viewed as an optimisation of the classic resultant-based algorithm. Implementation issues are discussed in Section 5.

1.5 Changelog and Acknowledgements

We have corrected some claims regarding the complexity of our algorithms (Lemma 3 and subsequent discussion) and their failure probability (page 22). We are grateful to Thomas Espitau and Eamonn Postlethwaite for pointing out and discussing with us the first and second points, respectively.

Parts of this work were done when the second author was an engineer at Thales, and was supported by the projects PROMETHEUS (European Union Horizon 2020 Research and Innovation Program, grant 780701) and RISQ (French Programme dInvestissement dAvenir, grant P141580).

2 Preliminaries

We denote by $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ the ring of integers and the fields of rational, real and complex numbers. For $a > 0, b > 1$, we denote by $\log_b a$ the logarithm of a in the basis b, with the convention $\log a = \log_2 a$. For an integer $r > 0$, we denote by $\mathbb{Z}_r$ the ring of integers modulo r.

2.1 Polynomial Rings and Fields

Let $\mathbb{Z}[x]$ be the ring of polynomials with integer coefficients (thereafter called *integral polynomials*). Let ϕ be a non-zero monic integral polynomial of degree $n \geq 1$ (i.e. $\phi = x^n + \sum_{i=0}^{n-1} \phi_i x^i$). Euclidean division of any integral polynomial by ϕ is well-defined and yields a unique remainder of degree less than n; we can therefore define $\mathbb{Z}[x]/(\phi)$, the ring of integral polynomials modulo ϕ.

Similarly, we define $\mathbb{Q}[x]/(\phi)$, $\mathbb{C}[x]/(\phi)$ and $\mathbb{Z}_r[x]/(\phi)$. When ϕ is irreducible in $\mathbb{Z}[x]$, it is also irreducible in $\mathbb{Q}[x]$, and $\mathbb{Q}[x]/(\phi)$ is a field. In this article, we will work modulo polynomials ϕ which are irreducible in $\mathbb{Q}[x]$; however, in general, $\mathbb{C}[x]/(\phi)$ and $\mathbb{Z}_r[x]/(\phi)$ are *not* fields.

2.2 Matrices and Vectors

While the point of using polynomial rings to represent lattices is to avoid computations related to matrices and vectors, we will still use such algebraic objects in some proofs.

We will denote matrices in bold uppercase (e.g. $\mathbf{B}$) and vectors in bold lowercase (e.g. $\mathbf{v}$). We use the row convention for vectors.

The p-norm of a vector $\mathbf{v}$ is denoted $\|\mathbf{v}\|_p$, and, by convention, $\|\mathbf{v}\| = \|\mathbf{v}\|_2$. We recall that for $\mathbf{v} \in \mathbb{C}^n$ and $0 < r \leq p \leq \infty$, and with the convention $1/\infty = 0$:

$$\|\mathbf{v}\|_p \leq \|\mathbf{v}\|_r \leq n^{\left(\frac{1}{r}-\frac{1}{p}\right)}\|\mathbf{v}\|_p. \tag{3}$$

For a polynomial $f \in \mathbb{C}[x]/(\phi)$, where ϕ is a monic polynomial of degree n, we denote by $\mathcal{C}_\phi(f)$ the $n \times n$ matrix whose j-th row consists in the coefficients of $x^{j-1}f \bmod \phi$:

$$\mathcal{C}_\phi(f) = \begin{bmatrix} f \bmod \phi \\ xf \bmod \phi \\ \dots \\ x^{n-1}f \bmod \phi \end{bmatrix} \tag{4}$$

When ϕ is clear from context, we will simply note $\mathcal{C}(f)$. One can check that when $\phi = x^n + 1$, the matrix $\mathcal{C}_\phi(f)$ is a skew-circulant matrix.

The operator $f \in \mathbb{C}[x]/(\phi) \mapsto \mathcal{C}(f)$ is a ring isomorphism onto its image. In particular, for all $f, g \in \mathbb{C}[x]/(\phi)$, we have:

$$\begin{gathered} \mathcal{C}(f+g) = \mathcal{C}(f) + \mathcal{C}(g) \\ \mathcal{C}(fg) = \mathcal{C}(f)\mathcal{C}(g) \end{gathered} \tag{5}$$

2.3 Fast Integer Multiplication

Our techniques, when applied to solving the NTRU equation, imply the use of large integers. Asymptotic computational costs depend on the time complexity of multiplying two such integers. We denote by $\mathcal{M}(b)$ that complexity, when the size in bits of the two integers is bounded by b:

- if we use Karatsuba's algorithm, then $\mathcal{M}(b) = O(b^{\log_2(3)}) \approx O(b^{1.585})$;
- with the Schönhage-Strassen algorithm [23], $\mathcal{M}(b) = \Theta(b \cdot \log b \cdot \log\log b)$.

Karatsuba's algorithm is more efficient for "small" values of b, while Schönhage-Strassen is asymptotically better. When giving time complexities for our improved algorithms, we consider both methods.[4]

It shall be noted that asymptotic complexity is a reasonable estimate of performance only for "large enough" parameters. In our implementations, we found that for typical parameters (degree n up to 1024), the bottleneck was not integer multiplication, but rather Babai's reduction, which entails performing floating-point operations.

[4] For completeness, we note that Harvey and van der Hoeven [15] recently published a $O(b \cdot \log b)$ integer multiplication algorithm. However, its $O(\cdot)$ constant is large enough to place it in the (informal) class of so-called *galactic algorithms*.

2.4 Cyclotomic Polynomials

Most lattice-based cryptographic algorithms that use polynomial rings to represent structured lattices rely on cyclotomic polynomials (some notable exceptions being e.g. [24,3]). Cyclotomic polynomials have some properties that make them ideal for use of the field norm.

Definition 1. *For an integer $m \geq 1$, the m-th cyclotomic polynomial is:*

$$\Phi_m = \prod_{\substack{0<k<m \\ gcd(k,m)=1}} \left(x - e^{2i\pi(k/m)}\right) \tag{6}$$

Cyclotomic polynomials have the following well-known properties:

- They are in $\mathbb{Z}[x]$ and are irreducible in $\mathbb{Q}[x]$.
- The degree of Φ_m is $\varphi(m)$, where φ denotes Euler's function: $\varphi(m) = |\mathbb{Z}_m^\times|$.
- If $n = 2^\ell$, then $\Phi_{2n} = x^n + 1$.
- If p is a prime factor of m, then:

$$\Phi_{mp}(x) = \Phi_m(x^p) \tag{7}$$

Since cyclotomic polynomials are irreducible, $\mathbb{Q}[x]/(\Phi_m)$ is a field for all $m \geq 1$; we will call them *cyclotomic fields*.

2.5 The Field Norm

The field norm is the central tool we use in our algorithms, and the key to their efficiency. In this section, we recall its definition, as well as a few properties.

Definition 2 (Field Norm). *Let $\mathbb{K}$ be a number field, and $\mathbb{L}$ be a Galois extension of $\mathbb{K}$. We denote by $\mathrm{Gal}(\mathbb{L}/\mathbb{K})$ the Galois group of the field extension $\mathbb{L}/\mathbb{K}$. The field norm $\mathrm{N}_{\mathbb{L}/\mathbb{K}} : \mathbb{L} \to \mathbb{K}$ is a map defined for any $f \in \mathbb{L}$ by the product of the Galois conjugates of f:*

$$\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \prod_{\mathrm{g} \in \mathrm{Gal}(\mathbb{L}/\mathbb{K})} \mathrm{g}(f) \tag{8}$$

Equivalently, $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f)$ can be defined as the determinant of the $\mathbb{K}$-linear map $\psi_f : a \in \mathbb{L} \mapsto fa$.

It is clear from the definition that the field norm is a multiplicative morphism. In addition, the field norm is compatible with composition: for a tower of extensions $\mathbb{L}/\mathbb{K}/\mathbb{J}$, it holds that $\mathrm{N}_{\mathbb{L}/\mathbb{K}} \circ \mathrm{N}_{\mathbb{K}/\mathbb{J}}(f) = \mathrm{N}_{\mathbb{L}/\mathbb{J}}(f)$.

For conciseness, $\mathbb{K}$ and $\mathbb{L}$ may be omitted from the subscript when clear from context. For example, when $f \in \mathbb{L}$ and $\mathbb{K}$ is the unique largest proper subfield of $\mathbb{L}$, then we denote $\mathrm{N}(f) = \mathrm{N}_{\mathbb{L}/\mathbb{K}}(f)$. In addition, if $f \in \mathbb{L}$ and $\mathbb{L}$ sits atop a field tower that is clear from context, then we may abusively denote by $\mathrm{N}^i(f)$ the i-times composition of N. For example, if we consider the following field tower:

$$\mathbb{Q}[x]/(x^n+1) \;/\; \mathbb{Q}[x]/(x^{n/2}+1) \;/\; \ldots \;/\; \mathbb{Q}[x]/(x^2+1) \;/\; \mathbb{Q} \tag{9}$$

with $n = 2^\ell$, then $\mathrm{N}^i(f)$ sends $f \in \mathbb{Q}[x]/(x^n+1)$ to $\mathbb{Q}[x]/(x^{n/(2^i)}+1)$.

The Case of Cyclotomic Extensions. For cyclotomic extensions, the field norm can be expressed in a form which is convenient for us. Let $m, n > 0$ be integers such that $n|m$, $\mathbb{L} = \mathbb{Q}[x]/(\Phi_m)$ and $\mathbb{K} = \mathbb{Q}[y]/(\Phi_n)$. The morphism $y \mapsto x^{m/n}$ defines a field extension $\mathbb{L}/\mathbb{K}$. The Galois conjugates $\mathrm{g}_a(f)$ of $f \in \mathbb{L}$ are then of the form

$$\mathrm{g}_a(f)(x) = f(x^a) \tag{10}$$

for the set of $a \in \mathbb{Z}_m$ verifying $a = 1 \bmod n$. This provides a simple and efficient way of computing the norm $\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \prod_a \mathrm{g}_a(f)$, especially in FFT or NTT.

In the particular case where $n = 2^\ell$, $\mathbb{L} = \mathbb{Q}[x]/(\Phi_{2n})$ and $\mathbb{K} = \mathbb{Q}[y]/(\Phi_n)$, the field norm is particularly simple to express. Any $f \in \mathbb{L}$ can be "split" into its coefficients of even and odd degrees:

$$f = f_e(x^2) + x f_o(x^2) \tag{11}$$

with $f_o, f_e \in \mathbb{K}$. Noting $\psi_f : a \in \mathbb{L} \mapsto fa$, we have

$$\mathrm{N}_{\mathbb{L}/\mathbb{K}}(f) = \det{}_{\mathbb{K}}(\psi_f) = \det \begin{bmatrix} f_e & f_o \\ y f_o & f_e \end{bmatrix} = f_e^2 - y f_o^2 \tag{12}$$

2.6 Resultants

Resultants are powerful tools in number theory. Among other applications, they allow to keep track of coefficient growth when computing the (pseudo-)GCD of polynomials in $\mathbb{Z}[x]$, and they play a crucial role in a previous algorithm by [16] which solves the NTRU equation. We will see (in Section 3) that our first application of the field norm is an efficient algorithm to compute resultants between a cyclotomic polynomial ϕ of degree n, and another polynomial of degree less than n.

Definition 3 (Resultant). *Let f, g be two polynomials in $\mathbb{C}[x]$, of degrees n and m, respectively. We denote their coefficients and roots as follows:*

$$\begin{aligned} f(x) &= \sum_{j=0}^{n} f_j x^j = f_n \prod_{j=0}^{n-1} (x - \alpha_j) \\ g(x) &= \sum_{k=0}^{m} g_k x^k = g_m \prod_{k=0}^{m-1} (x - \beta_k) \end{aligned} \tag{13}$$

The resultant *of f and g is defined by either of these two equivalent definitions:*

1. $\mathrm{Res}(f, g) = f_n^m g_m^n \prod_{j,k} (\alpha_j - \beta_k) = f_n^m \prod_j g(\alpha_j) = (-1)^{mn} g_m^n \prod_k f(\beta_k)$

2. $\mathrm{Res}(f,g) = \det(\mathrm{Syl}(f,g))$*, where* $\mathrm{Syl}(f,g)$ *denotes the Sylvester matrix of f and g:*

$$\mathrm{Syl}(f,g) = \begin{bmatrix} f_n & 0 & \cdots & 0 & g_m & 0 & \cdots & 0 \\ f_{n-1} & f_n & \ddots & \vdots & \vdots & g_m & \ddots & \vdots \\ \vdots & f_{n-1} & \ddots & 0 & \vdots & & \ddots & 0 \\ \vdots & \vdots & \ddots & f_n & g_1 & & & g_m \\ f_0 & & & f_{n-1} & g_0 & \ddots & \vdots & \vdots \\ 0 & \ddots & & \vdots & 0 & \ddots & g_1 & \vdots \\ \vdots & \ddots & f_0 & \vdots & \vdots & \ddots & g_0 & g_1 \\ 0 & \cdots & 0 & f_0 & 0 & \cdots & 0 & g_0 \end{bmatrix} \tag{14}$$

The second definition makes it clear that if f and g are integral polynomials, then their resultant will also be an integer.

The resultant of f and g can be computed with the Euclidean Algorithm on polynomials. The Extended Euclidean Algorithm (also called Extended GCD) furthermore keeps track of intermediate quotients in order to yield Bézout coefficients, i.e. polynomials u and v in $\mathbb{C}[x]$ such that $uf + vg = \mathrm{Res}(f,g)$. When f and g are integral polynomials, the Bézout coefficients will also be integral polynomials.

In addition to these definitions, the following proposition will be useful for providing bounds over the resultant.

Proposition 1. *If g is monic with distinct roots over $\mathbb{C}$, then, for all $f \in \mathbb{C}[x]/(g)$, $\mathrm{Res}(g,f) = \det(\mathcal{C}_g(f))$.*

Proof. For a fixed g, all the matrices $\mathcal{C}_g(f)$ are co-diagonalizable:

$$\mathcal{C}_g(f) = \mathbf{V}^{-1} \times \mathbf{D} \times \mathbf{V}$$

Where $\mathbf{V}$ is the Vandermonde matrix associated to the roots of g, and $\mathbf{D}$ is the diagonal matrix which diagonal terms are the evaluations of f over the roots of g. As a consequence, $\det \mathcal{C}_g(f) = \det \mathbf{D} = \prod_{g(\gamma)=0} f(\gamma) = \mathrm{Res}(g,f)$. □

2.7 Fast Fourier Transform and Number Theoretic Transform

The Fast Fourier Transform, and its variant the Number Theoretic Transform, are powerful tools that allow for efficient computations in polynomial rings. The field norm, in particular, can be very simply and quickly evaluated when the operands use the FFT or NTT representation. Most of the speed-ups obtained by our techniques come from the interaction between the field norm and the FFT/NTT.

Let $\phi \in \mathbb{Q}[x]$ be a monic polynomial of degree n, with n distinct roots $(\gamma_j)_{0 \le j < n}$ over $\mathbb{C}$. For $f \in \mathbb{C}[x]/(\phi)$, its *Fourier Transform* $\hat{f}$ is defined as:

$$\hat{f} = \big(f(\gamma_j)\big)_{0 \le j < n} \tag{15}$$

The Fourier Transform is an isomorphism between $\mathbb{C}[x]/(\phi)$ and $\mathbb{C}^n$. Therefore, for $f, g \in \mathbb{C}[x]/(\phi)$, the Fourier transform of $f + g$ and fg can be computed by term-wise addition and multiplication, respectively, of $\hat{f}$ and $\hat{g}$.

The *Fast Fourier Transform* (or FFT) is a well-known algorithm for computing the Fourier Transform of f in the special case of $\phi = x^n + 1$ with $n = 2^\ell$ [8,13]. The FFT has time complexity $O(n \log n)$ operations in $\mathbb{C}$; the inverse transform can also be computed with similar efficiency. In particular, the FFT allows for computing the product of two polynomials modulo ϕ with complexity $O(n \log n)$. The FFT can be extended to other moduli, especially cyclotomic polynomials.

The *Number Theoretic Transform* (or NTT) is the analog of the Fourier Transform over the finite field $\mathbb{Z}_r$ for a given prime r. The NTT is well-defined as long as ϕ splits over $\mathbb{Z}_r$; when $\phi = x^n + 1$, it suffices that $r = 1 \bmod 2n$. As in the case of the FFT, the NTT can be computed in $O(n \log n)$ elementary operations in $\mathbb{Z}_r$ for some moduli, in particular cyclotomic polynomials.

2.8 Babai's Reduction

Before we show how to solve the NTRU equation, we present one last tool which plays an important role in this process: Babai's reduction, or rather a generalization of it. This reduction transforms a solution of the NTRU equation into another solution with shorter polynomials. We first define the adjoint.

Definition 4 (Adjoint). *Let $\phi \in \mathbb{Q}[x]$ be monic with distinct roots (γ_j) over $\mathbb{C}$. For $f \in \mathbb{C}[x]/(\phi)$, we define its adjoint $f^\star$ as the unique polynomial in $\mathbb{C}[x]/(\phi)$ such that for each γ_j:*

$$f^\star(\gamma_j) = \overline{f(\gamma_j)} \tag{16}$$

where $\bar{\cdot}$ denotes the complex conjugation.

Existence and uniqueness are easily obtained by noticing that, in FFT representation, computing the adjoint is equivalent to replacing each Fourier coefficient with its conjugate.

If $f \in \mathbb{R}[x]/(\phi)$, then $f^\star \in \mathbb{R}[x]/(\phi)$. Indeed, if γ is a root of ϕ, then $\overline{\gamma}$ is also a root of ϕ, and $\overline{f(\gamma)} = f(\overline{\gamma})$; therefore, $f^\star(\overline{\gamma}) = \overline{f^\star(\gamma)}$ for all roots γ of ϕ. This property is achieved only by real polynomials, i.e. polynomials whose complex coefficients are all real numbers.

The adjoint allows us to define Reduce (Algorithm 1), which is a straightforward generalization of Babai's nearest plane algorithm [2] over $\mathbb{Z}[x]/(\phi)$-modules. For inputs $f, g, F, G \in \mathbb{Z}[x]/(\phi)$, the Reduce algorithm computes F' and G' of close to minimal size such that $fG - gF = fG' - gF'$. We note that variants of this algorithm already appear in earlier works, for example [16].

Algorithm 1 $\mathsf{Reduce}_\phi(f, g, F, G)$

```
Require: $f, g, F, G \in \mathbb{Z}[x]/(\phi)$
Ensure: $F', G' \in \mathbb{Z}[x]/(\phi)$ such that $fG' - gF' = fG - gF \bmod \phi$
do
    $k \leftarrow \left\lfloor \frac{Ff^\star + Gg^\star}{ff^\star + gg^\star} \right\rceil$
    $(F, G) \leftarrow (F - kf, G - kg)$
while $k \neq 0$
return $F, G$
```

Several iterations may be needed, especially if k is computed in low precision. Indeed, in practice the coefficients of the polynomials F, G can be extremely large before reduction, and it is therefore more efficient to compute k with a low precision (e.g. using `double` values in the C programming language) over approximations of the polynomial coefficients: this allows the use of the FFT representation, where polynomial multiplications and adjoints are easily computed. Each iteration then yields an approximate k value with small coefficients (with scaling). Of course, using floating-point arithmetic means that one could be stuck in an infinite loop, but this is easily thwarted by exiting the algorithm as soon as the norm of (F, G) stops decreasing.

We note that computation of k involves a division of polynomials modulo ϕ. In FFT representation, division is simply applied term by term. Since ϕ is irreducible over $\mathbb{Q}[x]$, no division by 0 occurs here. In practice, though, use of approximate values in low precision may (rarely) yield situations where we end up dividing by 0. As will be explained in Section 5.1, occasional failures can easily be tolerated in the context of key pair generation for a cryptographic algorithm.

In this article, we use Algorithm 1 in several places, each time with polynomials f, g, F, G that fulfill the NTRU equation (2). In this case, informally, Algorithm 1 computes F' and G' such that the norm of (F', G') is about $O(\sqrt{n})$ larger than (f, g).

2.9 Solving The NTRU Equation With Resultants

We now present the first known method for solving the NTRU equation (2): given f and g in $\mathbb{Z}[x]/(\phi)$, find F and G in $\mathbb{Z}[x]/(\phi)$ such that $fG - gF = q$, where q is a given relatively small integer. Our techniques are best demonstrated by showing how they apply to, and speed up, this classic NTRU solving algorithm.

This method works for any monic ϕ irreducible over $\mathbb{Q}[x]$; it was introduced in [16] and an implementation can be found in [9]. It relies on Bézout equations over $\mathbb{Z}[x]$:

- First, we compute Bézout coefficients $s, s', t, t' \in \mathbb{Z}[x]$, and integers R_f and R_g, such that:

$$\begin{aligned} sf + s'\phi &= R_f \\ tg + t'\phi &= R_g \end{aligned} \tag{17}$$

Since ϕ is irreducible over $\mathbb{Q}[x]$, it is guaranteed that we can enforce the condition $R_f, R_g \in \mathbb{Z}$. The s' and t' polynomials do not actually need to be computed; only s and t are used thereafter.

- We compute the GCD δ of R_f and R_g, along with Bézout coefficients $u, v \in \mathbb{Z}$ such that:
$$uR_f + vR_g = \delta \tag{18}$$
- If δ is a divisor of q, we can then combine (17) and (18), yielding a solution to (2):
$$\left(\frac{uq}{\delta}s\right)f + \left(\frac{vq}{\delta}t\right)g = q \mod \phi \tag{19}$$

Since $\mathbb{Q}$ is a field and ϕ is irreducible over $\mathbb{Q}[x]$, finding solutions $s, t \in \mathbb{Q}[x], R_f = R_g = 1$ to (17) is doable via the extended GCD. Because $\mathbb{Z}$ is not a field but only an integral domain, we cannot straightforwardly apply the extended GCD on $\mathbb{Z}[x]$. However, by scaling the solutions in $\mathbb{Q}[x]$, one may obtain solutions in $\mathbb{Z}[x]$ which verify $R_f = \operatorname{Res}(\phi, f)$ and $R_g = \operatorname{Res}(\phi, g)$ (see e.g. [12, Corollary 6.21]).[5]

In practice, one may get $|R_f|$ and $|R_g|$ to be smaller than $|\operatorname{Res}(f, \phi)|$ and $|\operatorname{Res}(g, \phi)|$, but usually not by much. In the general case, these bounds are tight (e.g. for $f = 1 - kx, \phi = 1 + x$).

Using this method, combined with Babai's reduction to obtain a short solution (F, G), yields Algorithm 2, originally described in [16].

Algorithm 2 ResultantSolver$_{\phi,q}(f, g)$

```
Require: $f, g \in \mathbb{Z}[x]/(\phi)$
Ensure: Polynomials $F, G$ such that (2) is verified
Compute $R_f \in \mathbb{Z}$ and $s \in \mathbb{Z}[x]$ such that $sf = R_f \bmod \phi$
Compute $R_g \in \mathbb{Z}$ and $t \in \mathbb{Z}[x]$ such that $tg = R_g \bmod \phi$
Compute $u, v \in \mathbb{Z}$ such that $uR_f + vR_g = \mathrm{GCD}(R_f, R_g)$
if $\delta = \mathrm{GCD}(R_f, R_g)$ is not a divisor of $q$ then
    abort
$(F, G) \leftarrow (-(vq/\delta)s, (uq/\delta)t)$        ▷ At this point, $fG - gF = q$ already
Reduce$(f, g, F, G)$
return $F, G$
```

Correctness. One can show that if ϕ is irreducible over $\mathbb{Q}[x]$, then Algorithm 2 fails if and only the NTRU equation does not have a solution for the inputs (f, g). This will also be true for our new algorithms. Of course, handling such cases is important (and has been studied in [25]), but since our algorithms are "optimal" in this regard (they fail if and only if there is no solution at all), we consider this to be outside the scope of this document.

[5] Several techniques (on-the-fly rescaling, computation modulo small primes, etc.) have been proposed to make the extended GCD more efficient (for an overview, see e.g. [12, Chapter 6]), but they result in the same bounds over R_f, R_g.

Lemma 1 (Complexity of ResultantSolver for $\phi = x^n + 1$ and $q = 1$). *Let $\phi = x^n + 1$, $q = 1$, $\deg(f), \deg(g) < n$ and the euclidean norms of f, g be bounded by some value: $\log\|f\|, \log\|g\| \leq B$. Algorithm 2 (ResultantSolver) runs in space $O(n^2 B)$ and time $O(n(n^2 + B)(\log n + \log B)^2)$.*

Proof. We perform a step-by-step analysis of Algorithm 2.

S1. We have $|R_f| \leq |\operatorname{Res}(f, \phi)| = |\det \mathcal{C}_\phi(f)|$ since ϕ is monic with distinct roots. In addition:

$$\begin{array}{lcc} |\det \mathcal{C}_\phi(f)| \leq & \|f\|_2^n & \text{(upper bound)} \\ |\det \mathcal{C}_\phi(f)| \approx & \sqrt{2\pi n}\left[\frac{\|f\|_2}{e}\right]^n & \text{(heuristic)} \end{array} \tag{20}$$

For any square matrix $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, we have $|\det(\mathbf{B})| = \prod_i \|\tilde{\mathbf{b}}_i\| \leq \prod_i \|\mathbf{b}_i\|$, where $\tilde{\mathbf{b}}_i$ denotes the orthogonalization of $\mathbf{b}_i$ with respect to the previous rows. In our case:

- The upper bound uses the fact that each row of $\mathcal{C}_\phi(f)$ has a norm $\leq \|f\|_2$.
- For the heuristic approximation, we modelize each row of $\mathcal{C}_\phi(f)$ as a random vector of size $\|f\|_2$, so that the orthogonalization of the i-th row has a norm $\sqrt{\frac{n+1-i}{n}}\|f\|_2$. Of course, in our case the vectors are not independent, however this heuristic gives good approximations in practice.

The proven upper bound yields $\log|R_f| = O(nB)$. We even have $\log|R_f| = \Theta(nB)$ with the heuristic.

To finish the study of this step, we bound $\log\|p\|_\infty$. Since $\mathcal{C}(p) = R_f \mathcal{C}(f)^{-1}$, a straigthforward application of Cramer's rule yields $\|p\|_\infty \leq \|f\|^{n-1}$, so $\log\|p\|_\infty = O(nB)$.

S2. The analysis is identical and yields $\log|R_g|, \log\|s\|_\infty = O(nB)$.

S3. The extended GCD algorithm finds u, v such that $|u| < |R_g|$ and $|v| < |R_f|$. Since $\log|R_f|$ and $\log|R_g|$ both are in $O(nB)$, the same asymptotic bound applies to $\log|u|$ and $\log|v|$.

S6. From the previous items, we have $\log\|F\|_\infty, \log\|G\|_\infty = O(nB)$.

S7. This step can be performed with a space overhead $O(n)$ using Algorithm 1 with precision $O(1)$.

Overall, the logarithms of $|u|, |v|$ and of each coefficient of p, s, F, G are in $O(nB)$, so the total space complexity is in $O(n^2 B)$.

The time complexity is now easy to analyze. The costliest steps are S1 and S2, and according to [12, Corollary 6.39] they can be performed in time $O(n(n^2 + B)(\log n + \log B)^2)$, which concludes the proof. □

3 Improved Algorithm for Computing Resultants

Our first application of the field norm is an improved algorithm for computing polynomial resultants, which we present in this section. This algorithm, by itself, is not sufficient to significantly reduce the CPU and RAM costs of the classic NTRU equation solving algorithm (ResultantSolver, described in Section 2.9); however, it is an important step toward the construction of our improved solver. Moreover, this algorithm may prove useful to other applications that use resultants but not necessarily NTRU lattices.

Let $\phi = \Phi_{pm}$ be the pm-th cyclotomic polynomial for some integers p and m, $n = \varphi(m)$ its degree and $\mathbb{K} = \mathbb{Q}[x]/(\phi)$.

Let $f \in \mathbb{K}$. It is well-known that the field norm $\mathrm{N}_{\mathbb{K}/\mathbb{Q}}(f)$ is equal to the resultant $\mathrm{Res}(\phi, f)$, however we re-explain the intuition here for cyclotomic polynomials. We recall that the resultant of f with ϕ can be computed as:

$$\mathrm{Res}(\phi, f) = \prod_{j=0}^{n-1} f(\gamma_j) \tag{21}$$

where the γ_j values are the roots of ϕ over $\mathbb{C}$.

As noted previously, $\mathbb{K}$ is a field extension of $\mathbb{Q}[x]/(\Phi_m)$ by the morphism $y \mapsto x^p$. We can thus group the n roots of ϕ into n/p sets $\{\gamma_j\zeta^k\}$, each set using a base root γ_j multiplied by ζ_k, for $k = 0$ to $p-1$, and $\zeta = e^{2i\pi/p}$ a primitive p-th root of unity. It is easily shown that these n/p sets are a partition of the n roots of ϕ. To ease notation, we number here the base roots from 1 to n/p. This yields the following:

$$\mathrm{Res}(\phi, f) = \prod_{j=1}^{n/p} \prod_{k=0}^{p-1} f(\gamma_j\zeta^k) = \prod_{j=1}^{n/p} \mathrm{N}(f)(\gamma_j) \tag{22}$$

Note that the γ_j (the base roots of our sets) are exactly the roots of Φ_m. Thus:

$$\mathrm{Res}(\Phi_{pm}, f) = \mathrm{Res}(\Phi_m, \mathrm{N}(f)) \tag{23}$$

In other words, we can divide the degree of ϕ by p, by replacing f with $\mathrm{N}(f)$.

We may note that, in FFT representation, computing the field norm is no more than multiplying together the Fourier coefficients along the groups into which the roots of ϕ are partitioned.

As we saw in Section 2.6, the resultant $\mathrm{Res}(\phi, f)$ can also be defined as the determinant of the Sylvester matrix of ϕ and f. As such, it can be expressed as a polynomial expression of the coefficients of f and ϕ. The result above then expresses an equality of the expressions of $\mathrm{Res}(\Phi_{pm}, f)$ and $\mathrm{Res}(\Phi_m, \mathrm{N}(f))$ that will hold over any other field where Φ_{pm} has n roots. In particular, if f is an integral polynomial and r is a prime such that pm divides $r-1$, then we can perform all computations modulo r; notably, we can use the NTT. In NTT representation, just like in FFT representation, the field norm of f is computed by simply multiplying the NTT coefficients together along the partition groups of the roots of ϕ.

An important special case is $n = 2^\ell$. The cyclotomic polynomial is then $\phi = x^n + 1 = \Phi_{2n}$, and $p = 2$. We can furthermore apply the process repeatedly: we replace $\mathrm{Res}(x^n + 1, f)$ with $\mathrm{Res}(x^{n/2} + 1, \mathrm{N}(f))$, then $\mathrm{Res}(x^{n/4} + 1, \mathrm{N}(\mathrm{N}(f)))$, and so on. We thus obtain (Algorithm 3) a very simple, recursive algorithm for computing resultants of f with $x^n + 1$.

Algorithm 3 $\mathsf{TowerResultant}_n(f)$

```
Require: $f \in \mathbb{Z}[x]/(x^n + 1)$ with $n = 2^\ell$
Ensure: $R_f = \mathrm{Res}(x^n + 1, f)$
if $n = 1$ then
    return $f_0$
return $\mathsf{TowerResultant}_{n/2}(\mathrm{N}(f))$
```

If f is an integral polynomial, then algorithm TowerResultant can be computed modulo any prime r such that n divides $r - 1$; this yields the resultant modulo r. Using sufficiently many such small primes allows rebuilding the resultant value with the Chinese Remainder Theorem. Modulo each small prime r, the NTT can be used, with cost $O(n \log n)$ operations in $\mathbb{Z}_r$, followed by $n - 1$ multiplications in $\mathbb{Z}_r$. The total cost will then depend on how many small primes we need, i.e. what is the maximum size of the resultant $\mathrm{Res}(x^n + 1, f)$. The following lemma gives us a bound on that size:

Lemma 2. *It holds that* $\log |\mathrm{Res}(x^n + 1, f)| \leq n \log \|f\|_1$.

Proof. We recall that $\hat{p}$ denote the Fourier transform of p over the roots of ϕ (here, $\phi = x^n + 1$). We have:

$$\|\widehat{\mathrm{N}(f)}\|_\infty = \|(f(\gamma)f(-\gamma))_{\{\gamma\}}\|_\infty \leq \|(f(\gamma))_{\{\gamma\}}\|_\infty \|(f(-\gamma))_{\{\gamma\}}\|_\infty = \|\hat{f}\|_\infty^2 \tag{24}$$

where γ runs over all the roots of $x^n + 1$. Therefore, for any $0 \leq j \leq \log n$, we have $\|\widehat{\mathrm{N}^j(f)}\|_\infty \leq \|\hat{f}\|_\infty^{2^j}$. Using classical properties of p-norms, it follows that:

$$\|\,\mathrm{N}^j(f)\|_2 = \frac{1}{\sqrt{n/2^j}} \|\widehat{\mathrm{N}^j(f)}\|_2 \leq \|\widehat{\mathrm{N}^j(f)}\|_\infty \leq \|\hat{f}\|_\infty^{2^j} \leq \|f\|_1^{2^j} \tag{25}$$

When $2^j = n$, corresponding to the end of the recursion in algorithm TowerResultant, the result follows. □

Therefore, if the L^1-norm of f is bounded by 2^C, the resultant can be expressed over at most nC bits. The number of required small primes for the computation, using the NTT, is then $O(nC)$, yielding a total time cost of $O(n^2 C \log n)$; space complexity is $O(nC)$ (the size needed to represent the result). This improved resultant computation can be applied to the classic NTRU equation solver:

- $\mathrm{Res}(\phi, f)$ is computed modulo many small primes r, as explained above.
- The Bézout coefficient s such that $sf = \mathrm{Res}(\phi, f)$ is also computed modulo each r; since $\mathbb{Z}_r$ is a field, f can be inverted modulo ϕ, allowing for computing s efficiently, in particular with the NTT.

- When enough small primes have been used, the complete resultant $\mathrm{Res}(\phi, f)$ can be rebuilt, as well as the Bézout coefficient s, by applying the CRT on all individual monomials.

While this new algorithm reduces the time cost of ResultantSolver, it does not help with the space complexity: indeed, the coefficients of s have all about the same size as the resultant, and there are n of them. We will now show how the field norm yields new recursive formulas for solving the NTRU equation, that allow for a dramatic improvement in space complexity.

4 Improved Algorithms for Solving the NTRU Equation

This section presents new techniques and algorithms for solving the NTRU equation (2). These algorithms result from the recursive application of the field norm to the classic NTRU solver itself, building on the improvements made to resultants and described in the previous section.

We will first present the outline and the intuition of our techniques in Section 4.1. In Section 4.2, we present a recursive algorithm based on our observations, and in Section 4.3, we present a slightly slower but more memory-efficient iterative algorithm. Finally, in Section 4.4, we will provide analyses for the time and memory requirements of both algorithms.

4.1 Outline

Let $m, p > 0$ be integers, $\mathbb{L} = \mathbb{Q}[x]/(\Phi_{pm}), \mathbb{K} = \mathbb{Q}[y]/(\Phi_m)$ and $\mathrm{N} = \mathrm{N}_{\mathbb{L}/\mathbb{K}}$. Suppose that we have a given integer q and two polynomials $f, g \in \mathbb{Z}[x]/(\Phi_{pm})$, and we want to find $F, G \in \mathbb{Z}[x]/(\Phi_{pm})$ such that:

$$fG - gF = q \tag{26}$$

On the other hand, suppose that for $\mathrm{N}(f), \mathrm{N}(g)$, which are in the smaller ring $\mathbb{Z}[y]/(\Phi_m)$, we already know $F', G' \in \mathbb{Z}[y]/(\Phi_m)$ such that:

$$\mathrm{N}(f)G' - \mathrm{N}(g)F' = q \tag{27}$$

We claim that we can use the solutions F', G' to (27) to deduce solutions F, G for (26). Indeed, we recall that $\mathrm{N}(f) = \prod_{\mathrm{g} \in \mathrm{Gal}(\mathbb{L}/\mathbb{K})} \mathrm{g}(f) = ff^{\times}$, where $f^{\times} = \prod_{\mathrm{g} \in \mathrm{Gal}(\mathbb{L}/\mathbb{K})^{\times}} \mathrm{g}(f)$ denotes the product of all the Galois conjugates of f except itself, and we have a similar equality for g. (27) is then equivalent to:

$$ff^{\times}G'(x^p) - gg^{\times}F'(x^p) = q \tag{28}$$

which is an equality in the larger ring $\mathbb{Z}[x]/(\Phi_{pm})$. From this last equation, it follows that $F = g^{\times}F'(x^p)$ and $G = f^{\times}G'(x^p)$ are valid solutions for the NTRU equation.

From these observations, we can now give the outline of our algorithms for solving the NTRU equation: (i) use the field norm to project it to a smaller subring, (ii) solve the equation in the smaller ring, (iii) use (28) to lift the solutions back in the original ring. However, and contrary to the "overstretched NTRU" attack [1], we do not perform the projection and lifting steps once, but repeatedly. More precisely:

- we project f, g onto a smaller subring until we reach the ring of integers $\mathbb{Z}$; we call it the *descent* phase;
- once we obtain solutions in $\mathbb{Z}$, we lift them repeatedly until we are back to the original ring; we call this the *lifting* phase.

The multiple projections and liftings are key to the efficiency of our algorithms: performing them once would only yield gains in a $O(1)$ factor, but we will show that their repetition allows to gain factors larger than $\tilde{O}(n)$ in theory, and in practice a factor 100 for a typical value $n = 1024$.

The flow of our two algorithms is summarized in Fig. 1. The descent phase is represented in the middle column, and the lifting phase is represented in the right column.

$$\begin{tikzcd}
\mathbb{Z}[x]/(x^n+1) \arrow[d, phantom, "\rotatebox{-90}{$\supsetneq$}"] & \ni & f, g \arrow[r] \arrow[d] & F, G \\
\mathbb{Z}[x]/(x^{n/2}+1) \arrow[d, phantom, "\rotatebox{-90}{$\supsetneq$}"] & \ni & \mathrm{N}(f), \mathrm{N}(g) \arrow[r] \arrow[d] & F^{[1]}, G^{[1]} \arrow[u] \\
\mathbb{Z}[x]/(x^{n/4}+1) \arrow[d, phantom, "\rotatebox{-90}{$\supsetneq$}"] & \ni & \mathrm{N}^2(f), \mathrm{N}^2(g) \arrow[r] \arrow[d] & F^{[2]}, G^{[2]} \arrow[u] \\
\vdots \arrow[d, phantom, "\rotatebox{-90}{$\supsetneq$}"] & \vdots & \vdots \arrow[d] & \vdots \arrow[u] \\
\mathbb{Z} & \ni & \mathrm{N}^\ell(f), \mathrm{N}^\ell(g) \arrow[r] & F^{[\ell]}, G^{[\ell]} \arrow[u]
\end{tikzcd} \quad (29)$$

Fig. 1. Outline of Algorithms 4 and 5 for solving (2).

4.2 A recursive algorithm

In the special case of $\phi = x^n + 1$ with $n = 2^\ell$, we can apply these formulas with $p = 2$, and then do so again on $\phi' = x^{n/2} + 1$, repeatedly. This yields TowerSolverR (Algorithm 4).

Algorithm 4 $\mathsf{TowerSolverR}_{n,q}(f, g)$

```
Require: $f, g \in \mathbb{Z}[x]/(x^n + 1)$ with $n$ a power of two
Ensure: Polynomials $F, G$ such that (2) is verified
if $n = 1$ then
    Compute $u, v \in \mathbb{Z}$ such that $uf - vg = \mathrm{GCD}(f, g)$
    if $\delta = \mathrm{GCD}(f, g)$ is not a divisor of $q$ then
        abort
    $(F, G) \leftarrow (vq/\delta, uq/\delta)$
    return $(F, G)$
else
    $f' \leftarrow \mathrm{N}(f)$                                  ▷ $f', g', F', G' \in \mathbb{Z}[x]/(x^{n/2} + 1)$
    $g' \leftarrow \mathrm{N}(g)$
    $(F', G') \leftarrow \mathsf{TowerSolverR}_{n/2,q}(f', g')$
    $F \leftarrow g^{\times}(x)F'(x^2)$                            ▷ $F, G \in \mathbb{Z}[x]/(x^n + 1)$
    $G \leftarrow f^{\times}(x)G'(x^2)$
    $\mathsf{Reduce}(f, g, F, G)$
return $(F, G)$
```

The informal explanation of why algorithm TowerSolverR uses much less space than the classic solver (ResultantSolver) is that, at each recursion step, the size of individual coefficients roughly doubles, but the degree is halved, so there are only half as many coefficients to store. The algorithm relies on Babai's reduction (Reduce) to bring back the coefficients of the newly computed (F, G) to about the same size as the coefficients of (f, g) for this recursion level. A formal space complexity analysis is given in Lemma 3.

Correctness. If it outputs a solution (termination is addressed below), the correctness of Algorithm 4 is immediate. Indeed, correctness is clear at the deepest recursion level, and if the algorithm is correct for $(f, g) \in \mathbb{Z}[x]/(x^{n/2} + 1)$, then (27) and (28) assure us that it will be correct for $(f, g) \in \mathbb{Z}[x]/(x^n + 1)$.

Other Cyclotomic Polynomials. Algorithm TowerSolverR can be extended to arbitrary cyclotomic polynomials. Each iteration corresponds to a case where $\mathbb{Q}[x]/(\Phi_m)$ is considered as an extension of $\mathbb{Q}[x]/(\Phi_{m'})$, where m' divides m. The degree is divided by m/m', while average coefficient size grows by a factor approximately m/m'. The exact order in which successive divisions are applied on the degree is a matter of choice.

For instance, if we consider $\phi = \Phi_{2304} = x^{768} - x^{384} + 1$, then the algorithm may first apply a division of the degree by 3, yielding sub-polynomials modulo $x^{256} - x^{128} + 1$, then doing seven degree halving steps to bring the NTRU solving problem down to modulus $x^2 - x + 1$.

On the other hand, an implementation could first perform the seven degree halvings, down to modulus $x^6 - x^3 + 1$, and only then perform the division by 3. Both options have similar algorithmic complexity in time and space, but one may be more efficient than the other, depending on specific implementation context.

Finally, we would like to mention the polynomials of the form $x^p - x - 1$ for a prime p, as used in NTRU Prime [3]. The deliberate lack of nontrivial subfield when working with these polynomials makes it seemingly hard to apply our techniques there in a straightforward way, but a recent work [18] suggest that it might be possible.

4.3 An Iterative Algorithm

Each recursion involves computing $\mathrm{N}(f)$ and $\mathrm{N}(g)$, then saving them while the algorithm is invoked again on these two polynomials. However, all the $\mathrm{N}^i(f), \mathrm{N}^i(g)$ can be recomputed from f, g. Therefore, we may adopt a *memory-lazy* strategy and avoid storing the intermediate $\mathrm{N}^i(f), \mathrm{N}^i(g)$, instead recomputing them when needed. This yields a slower but more space-efficient iterative algorithm, described in algorithm TowerSolverI.

Algorithm 5 TowerSolverI$_{n,q}(f, g)$

```
Require: $f, g \in \mathbb{Z}[x]/(x^n + 1)$ with $n$ a power of two
Ensure: Polynomials $F, G$ such that (2) is verified
$(f', g') \leftarrow (f, g)$
for $i \leftarrow 1, \dots, \log n$ do
    $(f', g') \leftarrow (\mathrm{N}(f'), \mathrm{N}(g'))$        ▷ At that point, $f'$ and $g'$ have degree 0
Compute $u, v \in \mathbb{Z}$ such that $uf' - vg' = \mathrm{GCD}(f, g)$
if $\delta = \mathrm{GCD}(f, g)$ is not a divisor of $q$ then
    abort
$(F, G) \leftarrow (vq/\delta, uq/\delta)$
for $i \leftarrow \log n, \dots, 1$ do
    $(f', g') \leftarrow (f, g)$
    for $j \leftarrow 1, \dots, i - 1$ do
        $(f', g') \leftarrow (\mathrm{N}(f'), \mathrm{N}(g'))$
    $(F, G) \leftarrow (g'^{\times} F, f'^{\times} G)$
    Reduce$(f', g', F, G)$
return $(F, G)$
```

Compared to Algorithm 4, Algorithm 5 therefore performs a balanced tradeoff by a factor $\ell = \log n$ between speed and memory.

4.4 Complexity analysis

We now formally study the complexities of TowerSolverR and TowerSolverI. For simplicity, we set $q = 1$ in our analysis: the NTRU equation for $q > 1$ is easily solved by first solving it for $q = 1$ and scaling the output (F, G) by a factor q.

Lemma 3 (Space complexity analysis). *Let $q = 1$ and the euclidean norms of f, g be bounded:* $\log \|f\|, \log \|g\| \leq B$. *We also note $\ell = \log n$. Finally, let:*

$$\beta = \left\| \left(\frac{f^\star}{ff^\star + gg^\star}, \frac{g^\star}{ff^\star + gg^\star} \right) \right\|. \tag{30}$$

If $\beta = O(n\|(f, g)\|)$, then Algorithm 4 (TowerSolverR) and Algorithm 5 (TowerSolverI) run in space $O(n\ell(B + \ell))$ and $O(n(B + \ell))$, respectively.

Proof. We start with Algorithm 4 (TowerSolverR). It is clear that we have the following tower of recursive calls:

$$\mathsf{TowerSolverR}_{n,q}(f, g) \to \mathsf{TowerSolverR}_{n/2,q}(\mathrm{N}(f), \mathrm{N}(g)) \to \dots$$
$$\dots \to \mathsf{TowerSolverR}_{1,q}(\mathrm{N}^\ell(f), \mathrm{N}^\ell(g))$$

We now bound the space needed by internal variables.

1. From (3) and (25), each $\mathrm{N}^i(f), \mathrm{N}^i(g)$ takes $O(n(B + \ell))$ bits.
2. We now bound the (euclidean) norm of (F, G). First, we consider its norm *after reduction*. Noting $V = \mathrm{Span}((f, g))$, the vector (F, G) can be uniquely decomposed over $V \oplus V^\perp$ as:

   $$(F, G) = (\tilde{F}, \tilde{G}) + (\breve{F}, \breve{G})$$

   where $(\tilde{F}, \tilde{G}) \in V^\perp$, and $(\breve{F}, \breve{G}) \in V$.

   - It is shown in [9, Lemma 3] that $\|(\tilde{F}, \tilde{G})\| = \beta$. By assumption, this implies $\|(\tilde{F}, \tilde{G})\| = O(n\|(f, g)\|)$.
   - We bound $\|(\breve{F}, \breve{G})\|$: after (F, G) is reduced using Reduce (Algorithm 1), the triangle inequality guarantees that $\|(\breve{F}, \breve{G})\| \leq n/2\|(f, g)\|$.

   It follows that:

   $$\|(F, G)\|^2 = \|(\tilde{F}, \tilde{G})\|^2 + \|(\breve{F}, \breve{G})\|^2 = O\left(n^2\|(f, g)\|^2\right) \tag{31}$$

   and therefore (F, G) can be stored in space $O(n(B + \ell))$. Of course, we also have to handle (F, G) when it is computed from $F', G', f^\times, g^\times$ and is therefore not yet reduced. We have $\|F\| \leq \sqrt{\frac{n}{2}}\|F'\|\|g\|$ and $\|G\| \leq \sqrt{\frac{n}{2}}\|G'\|\|f\|$. From (25) and (31), it follows that (F, G) can be stored in space $O(n(B+\ell))$ even before reduction.

Algorithm 4 needs to store ℓ successive values of $(\mathrm{N}^i(f), \mathrm{N}^i(g))$, each taking space $O(n(B + \ell))$, as well as one set of polynomials F', G', F, G at once, each taking space $O(n(B + \ell))$. The space complexity of Algorithm 4 is therefore $O(n(\ell(B + \ell)))$.

For Algorithm 5, the previous analysis remains valid, except that only a constant number of values $(\mathrm{N}^i(f), \mathrm{N}^i(g))$'s need to be stored simultaneously, as they can all be recomputed from (f, g) in space $O(n(B + \ell))$, according to Lemma 2. The space complexity of algorithm is therefore $O(n(B + \ell))$. □

About Lemma 3. A previous version of this work presented a version of Lemma 3 that did not require a condition on $\beta = \|(\tilde{F}, \tilde{G})\|$. However, the proof contained an erroneous upper bound on β.

This updated version corrects this by adding the condition $\beta = O(n\|(f, g)\|)$. This fixes the proof and also makes it much simpler. The obvious downside is that the proven statement is slightly less general. However, this constraint can be checked preemptively via Eq. (30), which give the possibility to re-sample (f, g) when the context allows it, as is the case for all applications we are aware of (see Section 1.2). It is also almost always satisfied in practice: a heuristic argument in [9] states that when coefficients of f, g are sampled according to a Gaussian, we can expect on average $\beta = O(1/\|(f, g)\|)$.

We are grateful to Thomas Espitau for pointing out the erroneous upper bound on $\|(\tilde{F}, \tilde{G})\|$.

Lemma 4 (Time complexity analysis). *With the conditions of Lemma 3, the time complexities of Algorithm 4 (TowerSolverR) and Algorithm 5 (TowerSolverI) are:*

- $\tilde{O}(nB)$ *for Algorithm 4 with Schönhage-Strassen;*
- $\tilde{O}(nB)$ *for Algorithm 5 with Schönhage-Strassen;*
- $O((nB)^{\log_2(3)}\ell)$ *for Algorithm 4 with Karatsuba;*
- $O((nB)^{\log_2(3)}\ell^2)$ *for Algorithm 5 with Karatsuba;*

We note that while the complexities given with Schönhage-Strassen are much better than with Karatsuba, they are misleading as the $\tilde{O}$ hides constant and logarithmic factors which are not negligible in practice. The complexities given with Karatsuba reflect much more accurately the running times that we observe for typical values of n and B.

Proof. For $i \in [\![0, \ell]\!]$, let $B_i = \log\max(\|\,\mathrm{N}^i(f)\|, \|\,\mathrm{N}^i(g)\|)$. Using (25) and the fact that $\|f\|_1 \leq \|f\|_2^2$, we have $B_j \leq 2^{j+1}B$. The two costliest steps in our algorithms are the descent (computing $\mathrm{N}^i(f), \mathrm{N}^i(g)$ for increasing i) and the lifting (computing $F^{[i]}, G^{[i]}$ for decreasing i).

- *Descent.* Computing $\mathrm{N}^{i+1}(f)$ from $\mathrm{N}^i(f)$ is essentially as costly as an NTT and an inverse NTT, which both take time $O(\frac{n}{2^i}\log(\frac{n}{2^i})\mathcal{M}(B_i))$.
  Thus, it can be done in time $D_i = O(\frac{n}{2^i} \cdot \log\frac{n}{2^i} \cdot (2^i B)^{\log_2(3)})$ with Karatsuba, or $\tilde{O}(nB)$ with Schönhage-Strassen (see Section 2.3). This step is repeated ℓ times (once for each depth) for Algorithm 4, and $O(\ell^2)$ times ($\ell - i$ times for the depth i) for Algorithm 5.
- *Lifting.* From (31), we know that $\mathrm{N}^i(f), \mathrm{N}^i(g), F^{[i+1]}, G^{[i+1]}$ have the log of their euclidean norm bounded by $B_i + \log\frac{n}{2^i}$, so computing F, G at the recursion depth i can be done in time $R_i = O(\frac{n}{2^i}\log(\frac{n}{2^i})\mathcal{M}(B_i + \log\frac{n}{2^i}))$. In both algorithms, this step is repeated once for each depth.

In Algorithm 5, the descent is the costliest part as computing $\mathrm{N}^{i+1}(f)$ from $\mathrm{N}^i(f)$ is done $O(\ell^2)$ times ($\ell - i$ times for the depth i). Its time complexity is therefore $\sum_{0\leq i<\ell}(\ell - i)D_i$, which ends the proof for Algorithm 5.

In Algorithm 4, the lifting is the costliest part as each individual step is slightly more expensive than for the descent. Its time complexity is then $\sum_{0\leq i<\ell} R_i$, which ends the proof for Algorithm 4. $\square$

General Case For *q*. The analysis above covered the situation where the right-hand side of the NTRU equation is $q = 1$. In the general case, we may target another value of q, usually a small integer. This is done by multiplying values by q at some point in the lifting phase. In the description of algorithms TowerSolverR and TowerSolverI, that multiplication was done right after the GCD, but it could be done later on. In any case, multiplying by q increases the size of polynomial coefficients by $\log q$ bits, and Babai's reduction will in practice absorb these bits. In the worst case, the $\log q$ bits subsist to the last step, implying a space overhead of at most $O(n \log q)$ bits. The same remark applies to ResultantSolver.

Failure probability. In a previous version of this work, we erroneously stated that Algorithms 4 and 5 find a solution for the NTRU equation (2) with inputs (f, g) if and only if such a solution exists. This is not necessarily true; Algorithms 2, 4 and 5 find a solution if and only if $\gcd(\mathrm{N}(f), \mathrm{N}(g)) \mid q$. However, (2) might admit such a solution even if $\gcd(\mathrm{N}(f), \mathrm{N}(g)) \nmid q$.

For example, in the ring $\mathbb{Z}[x]/(x^4 + 1)$, consider the ring elements:

$$(f, g, F, G) = (x - 2, 9x^3 + x^2, x, x^2 + 2x + 4).$$

One can check that $fG - gF = 1$, whereas $\mathrm{N}(f) = 17$ and $\mathrm{N}(g) = 17 \cdot 386$.

We are grateful to Eamonn Postlethwaite for pointing out and discussing this limitation with us.

Output quality. An important notion is the *quality* of the solutions (F, G), for example its Euclidean norm or its Gram-Schmidt norm (as defined in e.g. [14,9]). For any of these metrics, our algorithms will output solutions of exactly the same quality as existing algorithms.

Indeed, the set of solutions is of the form $\{(F_0+rf, G_0+rg) | r \in \mathbb{Z}[x]/(x^n+1)\}$, where (F_0, G_0) denotes an arbitrary solution pair. For any element in this set, Algorithm 1 will output the same solution, so the Euclidean norm of the output will be the same for Algorithms 2, 4 and 5. On the other hand, for a fixed input (f, g), all the solutions to the NTRU equation have the same Gram-Schmidt norm (see e.g. [9, Lemma 3]).

5 Implementation Issues and Performances

Our new solving algorithm (TowerSolverI) is implemented as part of the key generation process of Falcon [22], a signature scheme submitted to the NIST call for post-quantum cryptographic schemes[21]. Falcon uses modulus $\phi = x^n + 1$ (with $n = 2^\ell$) or $\phi = x^n - x^{n/2} + 1$ (with $n = 3 \cdot 2^\ell$); these two sub-cases are called "binary" and "ternary", respectively. Our implementation supports the binary case for all degrees from 2 to 1024, and the ternary case for all degrees from 12 to 768; only the higher degrees (512, 768 and 1024) provide sufficient security, but the lower values are convenient to test the correctness of the key generation process.

In the context of Falcon, the target q value for the NTRU equation is fixed to $q = 12289$ (binary case) or $q = 18433$ (ternary case). The coefficients of the secret polynomials f and g are generated with a discrete Gaussian distribution of standard deviation $1.17\sqrt{q/(2n)}$ in the binary case, thus a size of a few bits at most; they are slightly larger in the ternary case, but in practice it can be assumed that they always fit over 8 bits each for normal key sizes.

We implemented TowerSolverI, and measured the costs of the various steps so as to estimate the computational overhead of TowerSolverI when compared to TowerSolverR. We also implemented the classic solver ResultantSolver, as a baseline to estimate the impact of our new techniques based on the field norm. Test system is a MacBook Pro laptop (Intel Core i7-6567U clocked at 3.30 GHz), running Linux in 64-bit mode. Implementations are in C and do not use platform integer types larger than 64 bits. Obtained performance is the following, for modulus $\phi = x^{1024} + 1$:

Algorithm	**CPU (ms)**	**RAM (kB)**
Classic algorithm: ResultantSolver	2000	3300
New algorithm (iterative): TowerSolverI	20	30
New algorithm (recursive): TowerSolverR	17	40

The following subsections describe various optimizations and other local techniques that together allow for these substantial performance gains. The source code can also be browsed on the Falcon Web site:

https://falcon-sign.info/impl/falcon-keygen.c.html

5.1 Value Sizes

The analyses presented in Section 4 allow computing absolute bounds on the size of intermediate values and resultants. However, these bounds are substantially larger than average cases.

An important point is that, *in the context of key pair generation*, it is acceptable for the solving algorithm to occasionally fail. Indeed, there are unavoidable failure conditions, when (for instance) the randomly generated f polynomial is not invertible in $\mathbb{Z}_q[x]/(\phi)$. If such a case arises, then it suffices to generate new random f and g. Similarly, we may arbitrarily reject (f, g) pairs for which the NTRU equation can be solved, but some internal implementation threshold is exceeded: such rejections imply a reduction of the space of possible keys, but have no significant impact on security as long as rejections are relatively infrequent. Even rejecting half of potential private keys only gives one bit of information to attackers.

Therefore, it is acceptable to *measure* the average maximum size of intermediate values, and use such sizes as the basis for memory allocation, with some margin. For instance, the theoretical maximum bound on the coefficients of f and g at maximum recursion depth (when they are constant polynomials, and equal to their resultants with $x + 1$) is about 12000 bits (for $n = 1024$); however, in practice, their average size was measured to be about 6308 bits, with a standard deviation of less than 25 bits. We can thus assume that they will almost always fit in 6500 bits, and may simply reject the very rare cases when that assumption does not hold.

This methodology allows the use of static memory allocation, that offers strong guarantees on memory usage and also helps with making the key generation process memory access pattern uncorrelated with the secret values.

5.2 RNS, CRT and NTT

A *Residue Number System* is a representation of an integer z by storing z mod r_j for a number of moduli r_j. Any integer in a range of length no more than the product of the r_j has a unique representation and can be unambiguously recomputed with the Chinese Remainder Theorem. Integers in RNS representation can be added and multiplied by simply computing the result modulo each r_j.

In our implementation, we use moduli r_j which are prime numbers slightly below, but close to, 2^{31}. We furthermore require that ϕ has n distinct roots modulo each r_j; in the binary case, this is achieved by ensuring that $r_j = 1 \bmod 2n$. We precomputed 521 such primes, ranging from 2135955457 to 2147473409.

Computations modulo any r_j can be done with branchless code, which promotes efficiency. In the C language, addition is implemented thus:

```
static inline uint32_t
modp_add(uint32_t a, uint32_t b, uint32_t p)
{
        uint32_t d;

        d = a + b - p;
        d += p & -(d >> 31);
        return d;
}
```

This function computes the sum of `a` and `b` modulo `p`; the operation `a+b-p` is first computed modulo 2^{32}; if the result would have been negative, then the most significant bit will be set; we then *extend* that bit into a full-word mask in order to conditionally add the modulus again if necessary.

For multiplications, we use Montgomery multiplication:

```
static inline uint32_t
modp_montymul(uint32_t a, uint32_t b, uint32_t p, uint32_t p0i)
{
        uint64_t z, w;
        uint32_t d;

        z = (uint64_t)a * (uint64_t)b;
        w = ((z * p0i) & (uint64_t)0x7FFFFFFF) * p;
        d = (uint32_t)((z + w) >> 31) - p;
        d += p & -(d >> 31);
        return d;
}
```

Montgomery multiplication of a by b modulo p computes $ab/R \bmod p$, where R is a power of 2 greater than p (here, $R = 2^{31}$). The parameter `p0i` is a precomputed value equal to $-p^{-1} \bmod 2^{31}$. An integer a modulo p is said to be in "Montgomery representation" if it is kept as the value $aR \bmod p$; converting to and from Montgomery representation is done by computing a Montgomery multiplication with, respectively, $R^2 \bmod p$ or 1. The Montgomery multiplication of two integers which are in Montgomery representation, is equal to the Montgomery representation of the product of the two integers.

The Chinese Remainer Theorem (CRT), given $z_1 = z \bmod t_1$ and $z_2 = z \bmod t_2$, where t_1 and t_2 are prime to each other, allows recomputing z modulo $t_1 t_2$ with the following equation:

$$z = z_1 + t_1((t_1^{-1} \bmod t_2)(z_1 - z_2) \bmod t_2) \tag{32}$$

In our case, we use the CRT to convert an integer back from RNS representation, applying it on the moduli r_j one by one. At each step, we have the value z modulo t_1 and t_2, where:

$$\begin{aligned} t_1 &= \prod_{j<k} r_j \\ t_2 &= r_k \end{aligned} \tag{33}$$

The inverse of t_1 modulo t_2 is precomputed and stored along with the prime r_k itself. The CRT formula above can thus be applied with:

- a reduction of a big integer modulo r_k;
- a subtraction and a multiplication modulo r_k;
- a multiplication of a small integer (modulo r_k) with a large integer (t_1);
- an addition of two large integers.

This process can be done in place, if big integers are represented in basis 2^{31}, i.e. as sequences of 31-bit words; restricting words to 31 bits (instead of 32) also makes computations easier in standard C, where carry flags are not available. The aggregate products of r_j could be precomputed, but they can also be recomputed on the fly, for better space efficiency. If z fits over w words of 31 bits, and is represented in RNS modulo w small primes r_j, then the whole process of converting z back to a big integer in basis 2^{31} has cost $O(w^2)$ step, and is done mostly in place (we need an extra buffer of w words to rebuild the product of r_j, but that value may be shared if we have several integers z to convert).

It shall be noted that applying the CRT with r_j moduli one by one is not the most efficient method with regards to time complexity. For instance, we could assemble the r_j with a balanced binary tree, and use Karatsuba or Schönhage-Strassen for multiplications (each *modular* multiplication can be performed with two *integer* multiplications with Montgomery's method). However, such methods are more complex to implement, and require some extra space. In our implementation, the CRT reconstruction contributes only a small part to the total runtime cost, and can be performed mostly in-place.

In the course of the TowerSolverI algorithm, we often keep polynomials whose coefficients are both in RNS and NTT representations:

- The RNS representation means that a polynomial $f \in \mathbb{Z}[x]/(\phi)$ is replaced with w polynomials $f_j \in \mathbb{Z}_{r_j}[x]/(\phi)$.
- Each such polynomial f_j is furthermore in NTT representation (the moduli r_j where chosen so that ϕ splits over $\mathbb{Z}_{r_j}$, thereby allowing that representation).

As the algorithm goes deeper through the recursion, the degree of polynomials lowers, but the coefficients grow, thus requiring more moduli r_j. A common pattern is the following:

- Some polynomial inputs are provided modulo w small primes r_j and in NTT representation.
- The output is computed modulo these w small primes r_j, again in NTT representation. Moreover, the inverse NTT is applied on the inputs for each r_j.
- When all w small primes have been used, the CRT is applied to rebuild the full input coefficients.
- The rebuilt coefficients are then used to pursue the computation modulo w' more small primes r_j, each time computing the NTT.

5.3 Binary GCD

At the deepest recursion level, the polynomials f and g are plain integers (polynomials modulo $x + 1$ are constant), and the NTRU equation becomes a classic GCD computation with Bézout coefficients. Nominally, this algorithm uses repeated divisions, which are expensive and complex to implement. In order to both simplify and speed up that step, we use a binary GCD variant. The algorithm can be expressed as follows:

- Values a, b, u_0, u_1, v_0 and v_1 are initialized and maintained with the following invariants:

$$\begin{aligned} a &= fu_0 - gv_0 \\ b &= fu_1 - gv_1 \end{aligned} \tag{34}$$

  Initial values are:

$$\begin{aligned} a &= f \\ u_0 &= 1 \\ v_0 &= 0 \\ b &= g \\ u_1 &= g \\ v_1 &= f - 1 \end{aligned} \tag{35}$$

- At each step, a or b is reduced: if a and/or b is even, then it is divided by 2; otherwise, if both values are odd, then the smaller of the two is subtracted from the larger, and the result, now even, is divided by 2. Corresponding operations are applied on u_0, v_0, u_1 and v_1 to maintain the invariants. Note that computations on u_0 and u_1 are done modulo g, while computations on v_0 and v_1 are done modulo f.
- Algorithm stops when $a = b$, at which point the common value is the GCD of f and g.

This algorithm works only if both f and g are odd; otherwise, we cannot reliably compute divisions by 2 modulo f or g. Applying the principle explained in Section 5.1, we simply reject (f, g) pairs that would yield even resultants; this represents a reduction of the key space by a factor of 3, i.e. a loss of about 1.58 bits, which is considered negligible, as far as security is concerned. This rejection is easily done as a preliminary step, in which the resultants $\mathrm{Res}(\phi, f)$ and $\mathrm{Res}(\phi, g)$ are computed modulo 2: analysis of the TowerResultant algorithm in that specific case, when $\phi = x^n + 1$ and $n = 2^\ell$, shows that it suffices to add the coefficients of f modulo 2 (and similarly g).

The algorithm cost is quadratic in the size of the operands. The description above is bit-by-bit; in practice, we see that the decisions in the algorithm depend only on the few highest and lowest bits of each operand at each step. The implementation can thus be made considerably faster (experimentally, by a factor of about 12) by using the high and low bits to compute the action of 31 successive steps, and applying them on the values together with multiplications.

5.4 Babai's Reduction

When reducing candidate (F, G) relatively to (f, g), we must compute a reduction factor k:

$$k = \left\lfloor \frac{Ff^\star + Gg^\star}{ff^\star + gg^\star} \right\rceil \tag{36}$$

The polynomial division can be implemented efficiently in FFT representation with floating-point values. This implies, however, a loss of precision: thus, the resulting k will be only approximate, and the reduction will need to be applied repeatedly until F and G have reached an adequate size or cannot be reduced any further.

In our implementation, we extract the high bits of f, g, F and G and compute k with the FFT and a scaling factor, such that the resulting coefficients for k are equal to small integers (that fit on 30 bits each) multiplied by 2^s for some integer s. We shall then subtract kf and kg from F and G, respectively.

The computation of kf and kg is the most expensive part of the reduction. We have the choice between two methods:

1. Use a plain quadratic algorithm: if the degree is d, we thus need d^2 multiplications of a big integer (a coefficient of f or g) by a small integer (a coefficient of k).

2. Use the RNS representation and the NTT to compute the multiplication of k by f.

In general terms, throughout the TowerSolverI algorithm, we use polynomials of degree d with coefficients of size w words, such that dw remains roughly equal to n (coefficients double in size when the degree is halved). Babai's reduction will require $O(w)$ iterations (at that point, the size of F and G is about three times the size of f and g). The plain quadratic algorithm involves d^2 multiplications of a big integer (size w words) by a small one, thus $O(d^2w)$ operations per step, and a total of $O((dw)^2) = O(n^2)$. The use of NTT, however, implies the following elements:

- f and g must be converted to RNS and NTT. This is done once for the whole reduction. Conversion to RNS is $O(w^2d)$; the NTT has cost $O(wd\log d)$.
- For each iteration, k must be converted to NTT modulo each of the small primes ($O(wd\log d)$), multiplied with f and g ($O(wd)$), and converted back to big integers for the subtraction ($O(w^2d)$ for the CRT of d values of size w words each).

Thus, the RNS+NTT method has cost $O(w^3d + w^2d\log d) = O(n(w^2 + w\log d))$. At low recursion depth, where w is small and d is large, this method is thus faster than the plain quadratic algorithm; however, at high depth, w becomes large, the CRT cost dominates, and the plain quadratic algorithm becomes faster. Therefore, there is threshold at which implementation strategies should be switched.

In our implementation, we found that the threshold was at depth 4: when the polynomial degree is $n/16$ or more, the NTT method is faster. This threshold heavily depends on implementation details and the involved hardware, and thus should be measured.

5.5 Asymptotic And Real Performance

Asymptotic analysis would call for using big integer arithmetics, and efficient algorithms, e.g. Karatsuba or Schönhage-Strassen for integer multiplications. But such analysis is a valid approximation of real implementation performance only when inputs are "large enough". Our experience, when implementing the algorithms in the case of Falcon, is that practical degrees such as $n = 1024$ are below that threshold. This is why our code uses for instance RNS and a simple quadratic CRT process; our measures indicate that the dominant cost remains Babai's reduction.

With our use of quadratic algorithms for RNS and CRT, the expected asymptotic time complexity (for values of q and coefficients of f and g small enough to be considered "elementary") of TowerSolverI is $O(n^2\log n)$, while ResultantSolver would use $O(n^3)$. For $n = 1024$, this implies a factor of $n/\log n \approx 100$, which matches measured time.

Similarly, TowerSolverR is theoretically faster than TowerSolverI, since it stores intermediate values instead of recomputing them; but the execution time overhead of TowerSolverI is, in practice, less than 15%. We prioritized space efficiency and used TowerSolverI.

6 Conclusion and Open Problems

We presented the use of the field norm to optimize some computations on polynomial rings, in particular resultants and solving the NTRU equation. A practical consequence of the latter is that the post-quantum signature algorithm Falcon is fully usable on small microcontrollers or even smartcards, since 32 kB of RAM are enough to run our algorithm even for a long-term security NTRU lattice (degree $n = 1024$): all operations related to signatures (signature production, verification, and key pair generation) can fit on such constrained hardware.

We list below some open questions.

Non-cyclotomic polynomials. In our description, we covered the case of cyclotomic polynomials as moduli. The method can be extended to other moduli; in fact, for every modulus $\phi = \phi'(x^d)$ for some $d > 1$, application of the "field norm" can divide the degree by d for purposes of computing resultants and solving the NTRU equation. This holds even if ϕ is not irreducible over $\mathbb{Q}[x]$, i.e. if $\mathbb{Q}[x]/(\phi)$ is not, in fact, a field. The description of the general case remains a problem to explore; however, the use of reducible moduli in NTRU lattices is usually not recommended.

Floating-point arithmetic. Efficient implementation still relies, for Babai's reduction, on FFT and floating-point numbers. Fixed-point representation is probably usable, but the required range and precision must still be investigated. Whether the reduction may be performed efficiently without the FFT is an open problem.

Large integers. While our gains, in terms of memory, are significant, we still need to handle large integers. From an implementation complexity point of view, it would be interesting to get rid of large integers, for example by performing all operations in RNS, without negatively impacting the running time and memory requirements of our algorithms.

Other applications to cryptographic constructions. We think it is worthwhile to investigate whether our techniques can improve the efficiency of other cryptographic algorithms. In addition, just like we provided a constructive application of the field norm (as opposed to [1]), a constructive application of the trace (as opposed to [6]) would be, in our opinion, very interesting. Finally, [18] showed that an algebraic perspective is not necessary in the case of [1]; this raises the question of whether it is in our case.

Applications to cryptanalysis. A final line of research would be to use our techniques to improve the attacks based on the field norm [1], or even on the field trace [6].

References

1. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 153–178. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_6
2. Babai, L.: On lovasz' lattice reduction and the nearest lattice point problem. In: Proceedings on STACS 85 2Nd Annual Symposium on Theoretical Aspects of Computer Science. Springer-Verlag New York, Inc., New York, NY, USA (1985), http://dl.acm.org/citation.cfm?id=18858.18860
3. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU Prime. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
4. Campbell, P., Groves, M.: Practical post-quantum hierarchical identity-based encryption. 16th IMA International Conference on Cryptography and Coding (2017), http://www.qub.ac.uk/sites/CSIT/FileStore/Filetoupload,785752,en.pdf
5. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_27
6. Cheon, J.H., Jeong, J., Lee, C.: An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low level encoding of zero. Cryptology ePrint Archive, Report 2016/139 (2016), https://eprint.iacr.org/2016/139
7. Chuengsatiansup, C., Prest, T., Stehlé, D., Wallet, A., Xagawa, K.: ModFalcon: Compact signatures based on module-NTRU lattices. In: Sun, H.M., Shieh, S.P., Gu, G., Ateniese, G. (eds.) ASIACCS 20. pp. 853–866. ACM Press (Oct 2020). https://doi.org/10.1145/3320269.3384758
8. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965)
9. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_2
10. Espitau, T., Fouque, P.A., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of falcon. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 222–253. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_9
11. Fouque, P.A., Kirchner, P., Pornin, T., Yu, Y.: Bat: Small and fast kem over ntru lattices. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(2), 240265 (Feb 2022). https://doi.org/10.46586/tches.v2022.i2.240-265, https://tches.iacr.org/index.php/TCHES/article/view/9487
12. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra (3. ed.). Cambridge University Press (2013)
13. Gentleman, W.M., Sande, G.: Fast Fourier transforms: for fun and profit. In: Proceedings of the November 7-10, 1966, fall joint computer conference. pp. 563–578. ACM (1966)

14. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
15. Harvey, D., van der Hoeven, J.: Integer multiplication in time $O(n \log n)$. Annals of Mathematics **193**(2), 563 – 617 (2021). https://doi.org/10.4007/annals.2021.193.2.4, https://doi.org/10.4007/annals.2021.193.2.4
16. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J.H., Whyte, W.: NTRUSIGN: Digital signatures using the NTRU lattice. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 122–140. Springer, Heidelberg (Apr 2003). https://doi.org/10.1007/3-540-36563-X_9
17. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Buhler, J. (ed.) Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer (1998). https://doi.org/10.1007/BFb0054868, https://doi.org/10.1007/BFb0054868
18. Kirchner, P., Fouque, P.A.: Revisiting lattice attacks on overstretched NTRU parameters. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 3–26. Springer, Heidelberg (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7_1
19. Micciancio, D., Warinschi, B.: A linear space algorithm for computing the herite normal form. In: Kaltofen, E., Villard, G. (eds.) Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation, ISSAC 2001, ORCCA & University of Western Ontario, London, Ontario, Canada, July 22-25, 2001. pp. 231–236. ACM (2001). https://doi.org/10.1145/384101.384133, http://doi.acm.org/10.1145/384101.384133
20. NIST: Security requirements for cryptographic modules (2001), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf
21. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
22. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
23. Schönhage, A., Strassen, V.: Schnelle multiplikation großer zahlen. Computing **7**(3-4), 281–292 (1971). https://doi.org/10.1007/BF02242355, https://doi.org/10.1007/BF02242355
24. Smart, N.P., Albrecht, M.R., Lindell, Y., Orsini, E., Osheter, V., Paterson, K., Peer, G.: LIMA. Tech. rep., National Institute of Standards and Technology (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
25. Stehlé, D., Steinfeld, R.: Making NTRUEncrypt and NTRUSign as secure as standard worst-case problems over ideal lattices. Cryptology ePrint Archive, Report 2013/004 (2013), https://eprint.iacr.org/2013/004
26. Zhao, R.K., McCarthy, S., Steinfeld, R., Sakzad, A., O'Neill, M.: Quantum-safe HIBE: does it cost a latte? Cryptology ePrint Archive, Report 2021/222 (2021), https://eprint.iacr.org/2021/222