Generating NTRU Trapdoors

Thomas Pornin, NCC Group

April 29th, 2020

Outline

- NTRU Lattices
- Tools: FFT & NTT, Babai's Nearest Plane, Resultants
- Classic NTRU Solver
- More Tools: Field Norm, Fast Resultants
- New Solver
- Implementation Issues

`https://eprint.iacr.org/2019/015`

`https://eprint.iacr.org/2019/893`

NTRU Lattices

Let $\phi \in \mathbb{Z}[x]$ a monic polynomial of degree n.

For any polynomial $f \in \mathbb{C}[x]/(\phi)$, we denote $C_\phi(f)$ the $n \times n$ matrix:

$$C_\phi(f) = \begin{bmatrix} f \bmod \phi \\ xf \bmod \phi \\ \dots \\ x^{n-1} f \bmod \phi \end{bmatrix}$$

For any $f, g \in \mathbb{C}[x]/(\phi)$:

$$\begin{aligned} C_\phi(f+g) &= C_\phi(f) + C_\phi(g) \\ C_\phi(fg) &= C_\phi(f) C_\phi(g) \end{aligned}$$

NTRU Lattices

C_ϕ is a ring isomorphism from $\mathbb{C}[x]/(\phi)$ onto its image.

We use polynomials to compute on matrices.

Let $f, g \in \mathbb{Z}[x]/(\phi)$ with small coefficients. Let q be a given integer. The NTRU equation is:

$$fG - gF = q \mod \phi$$

for two other polynomials $F, G \in \mathbb{Z}[x]/(\phi)$.

Most of the work in the trapdoor generation is *solving the NTRU equation*, i.e. finding a solution (F, G) with small coefficients.

NTRU Lattices

If $f, g, F, G, h \in \mathbb{Z}[x](\phi)$ such that:

$$\begin{aligned} fG - gF &= q \mod \phi \\ fh &= g \mod \phi \mod q \end{aligned}$$

then the two following matrices $2n \times 2n$:

$$B = \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right] \qquad P = \left[\begin{array}{c|c} h & I_n \\ \hline qI_n & O_n \end{array}\right]$$

denote two bases for the same lattice of dimension $2n$.

B is the trapdoor (private key) for P (public key). NTRUEncrypt only needs (f, g), but NTRUSign and Falcon also need (F, G).

NTRU Lattices

Falcon parameters

- $\phi = x^n + 1$ with $n = 512$ or 1024 (n is a power of two)
- $q = 12289$
- $|f_j|, |g_j| \leq 20, |F_j|, |G_j| \leq 120$

q is chosen such that there are $2n$-th primitive roots of 1 in $\mathbb{Z}_q$.

f must be invertible in $\mathbb{Z}_q[x]/(\phi)$.

ϕ is the $2n$-th *cyclotomic polynomial*. ϕ is irreducible over $\mathbb{Q}[x]$ and has n distinct roots in $\mathbb{C}$:

$$\gamma_j = e^{2i\pi((2j+1)/2n)}$$

NTRU Lattices

Toy example: Falcon-8

$$
\begin{aligned}
\phi &= x^8 + 1 \\
q &= 12289 \\
f &= -55 + 11x - 23x^2 - 23x^3 + 47x^4 + 16x^5 + 13x^6 + 61x^7 \\
g &= -25 - 24x + 30x^2 - 3x^3 + 36x^4 - 39x^5 + 6x^6 + 0x^7 \\
F &= 58 + 20x + 17x^2 - 64x^3 - 3x^4 - 9x^5 - 21x^6 - 84x^7 \\
G &= -41 - 34x - 33x^2 + 25x^3 - 41x^4 + 31x^5 - 18x^6 - 32x^7 \\
h &= -4839 - 6036x - 4459x^2 - 2665x^3 \\
&\quad -186x^4 - 4303x^5 + 3388x^6 - 3568x^7
\end{aligned}
$$

Public key is h. Private key is (f, g, F, G).

Fourier Transform

The (discrete) Fourier transform of $f \in \mathbb{C}[x]/(\phi)$ is $\hat{f} = (f(\gamma_j))$.

- The Fourier transform is a bijection.
- The Fourier transforms of $f + g$ and fg can be computed by simple term-wise additions and multiplications, respectively, of $\hat{f}$ and $\hat{g}$.
- If $f \in \mathbb{R}[x]/(\phi)$ then $f(\gamma_{n-1-j}) = \overline{f(\gamma_j)}$: we can store only $n/2$ complex values for $\hat{f}$.
- The *Fast Fourier Transform* (FFT) can compute f from $\hat{f}$, or vice versa, with $O(n \log n)$ operations.

NTT

Similar to FFT but in $\mathbb{Z}_p[x]/(\phi)$ for a prime p such that ϕ splits over $\mathbb{Z}_p$.

For $\phi = x^n + 1$ and $n = 2^e$, we need $p = 1 \bmod 2n$. If w is a primitive $2n$-th root of unity in $\mathbb{Z}_p$, then the NTT of f is $(f(w^{2j+1}))$ for $0 \leq j < n$.

NTT and inverse NTT can be applied with $O(n \log n)$ operations (modulo p).

Given $f, g \in \mathbb{Z}[x]/(\phi)$, the NTT allows efficient computation of $f + g$ and fg modulo any prime p such that ϕ splits over $\mathbb{Z}_p$.

Babai's Nearest Plane

For $f \in \mathbb{C}[x]/(\phi)$, its *adjoint* is the unique polynomial $f^* \in \mathbb{C}[x]/(\phi)$ such that:

$$f^*(\gamma_j) = \overline{f(\gamma_j)}$$

for all roots γ_j of ϕ.

If $f \in \mathbb{R}[x]/(\phi)$ then $f^* \in \mathbb{R}[x]/(\phi)$.

If $\phi = x^n + 1$ (cyclotomic) then:

$$f^* = f_0 - \sum_{i=1}^{n-1} f_i x^{n-i}$$

Babai's Nearest Plane

Given a solution (F, G) to $fG - gF = q \bmod \phi$, then:

$$(G - kg)f - (F - kf)g = q \mod \phi$$

for any $k \in \mathbb{C}[x]/(\phi)$.

Thus, if $k \in \mathbb{Z}[x]/(\phi)$, then $(F - kf, G - kg)$ is also a solution.

Babai's Nearest Plane: Set:

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

This makes (F, G) smaller. Apply repeatedly if necessary.

Babai's Nearest Plane

We can use an approximation of k, e.g. using floating-point numbers.

- Approximate f and g with floating-point numbers: $f \approx 2^c \sum \tilde{f}_j x^j$ and $g \approx 2^c \sum \tilde{g}_j x^j$ (c such that $\max\{|\tilde{f}_j|, |\tilde{g}_j|\} \approx 1$)
- Approximate F and G similarly: $F \approx 2^d \sum \tilde{F}_j x^j$ and $G \approx 2^d \sum \tilde{G}_j x^j$ (d such that $\max\{|\tilde{F}_j|, |\tilde{G}_j|\} \approx 2^{25}$)
- Compute $\tilde{k}$:

$$\tilde{k} = \left\lfloor \frac{\tilde{F}\tilde{f}^* + \tilde{G}\tilde{g}^*}{\tilde{f}\tilde{f}^* + \tilde{g}\tilde{g}^*} \right\rceil$$

- Replace: $(F, G) \leftarrow (F - 2^{d-c}\tilde{k}f, G - 2^{d-c}\tilde{k}g)$
- Repeat while it works (each call reduces (F, G) by about 25 bits, until they have about the same size as (f, g)).

Resultants

For $a, b \in \mathbb{C}[x]$, of degree n and m respectively, the *resultant* of a and b is:

$$\mathrm{Res}(a, b) = a_n^m \prod_j b(\alpha_j) = (-1)^{mn} b_m^n \prod_k a(\beta_k)$$

where $(\alpha_j)_{1 \leq j \leq n}$ are the roots of a, and $(\beta_k)_{1 \leq k \leq m}$ are the roots of b.

- If $a, b \in \mathbb{Z}[x]$, then $\mathrm{Res}(a, b) \in \mathbb{Z}$.
- If $a, b \in \mathbb{Z}[x]$ are co-prime, then the extended Euclidean GCD yields coefficients $u, v \in \mathbb{Z}[x]$ such that:

$$au + bv = \mathrm{Res}(a, b)$$

Solving the NTRU Equation: Classical Method

Input: $f, g \in \mathbb{Z}[x]/(\phi)$. We want $F, G \in \mathbb{Z}[x]/(\phi)$ with small coefficients such that:

$$fG - gF = q \mod \phi$$

1. Using the extended Euclidean GCD algorithm (on polynomials), find $s, s', t, t' \in \mathbb{Z}[x]$ such that:

$$\begin{aligned} fs + \phi s' &= \mathrm{Res}(\phi, f) \\ gt + \phi t' &= \mathrm{Res}(\phi, g) \end{aligned}$$

2. Using the extended Euclidean GCD algorithm (on integers), find $\delta = \mathrm{GCD}(\mathrm{Res}(\phi, f), \mathrm{Res}(\phi, g))$, and $u, v \in \mathbb{Z}$ such that:

$$\mathrm{Res}(\phi, f)u + \mathrm{Res}(\phi, g)v = \delta$$

Solving the NTRU Equation: Classical Method

3. If ∂ does not divide q, then there is no solution. Otherwise, a solution to the NTRU equation is:

$$\begin{aligned} F &= -(vq/\partial)t \\ G &= (uq/\partial)s \end{aligned}$$

4. Apply Babai's Nearest Plane to make F, G small.

Solving the NTRU Equation: Classical Method

$$
\begin{aligned}
\phi &= x^8 + 1 \\
f &= -55 + 11x - 23x^2 - 23x^3 + 47x^4 + 16x^5 + 13x^6 + 61x^7 \\
\mathrm{Res}(\phi, f) &= 116876023987729 \\
s &= -1977840025967 - 760360482925x \\
&\quad -1187952761129x^2 + 2178875333716x^3 \\
&\quad +99053048645x^4 + 107066058579x^5 \\
&\quad -1300496523049x^6 - 1203258774093x^7 \\
fs &= \mathrm{Res}(\phi, f) \mod \phi
\end{aligned}
$$

Solving the NTRU Equation: Classical Method

$$
\begin{aligned}
g &= -25 - 24x + 30x^2 - 3x^3 + 36x^4 - 39x^5 + 6x^6 + 0x^7 \\
\mathrm{Res}(\phi, g) &= 799035204433 \\
t &= -4807592197 - 51641354937x \\
&\quad +19364169957x^2 + 16709964258x^3 \\
&\quad -52685146080x^4 + 9320244186x^5 \\
&\quad +33116290887x^6 - 32824810485x^7 \\
gt &= \mathrm{Res}(\phi, g) \mod \phi
\end{aligned}
$$

Solving the NTRU Equation: Classical Method

$\mathrm{Res}(\phi, f)$ and $\mathrm{Res}(\phi, g)$ are co-prime:

$$\begin{aligned} u &= -100370007727 \\ v &= 14681264812448 \\ 1 &= \mathrm{Res}(\phi, f)u + \mathrm{Res}(\phi, g)v \end{aligned}$$

Solving the NTRU Equation: Classical Method

$$\begin{aligned}
F &= -(vq/\delta)t \\
&= 867376473223614208793597984 \\
&\quad +931703324289756474248363126 4x \\
&\quad -349364604067006002025078070 4x^2 \\
&\quad -301477938890928095715976377 6x^3 \\
&\quad +950535201938662334006078976 0x^4 \\
&\quad -168154040533641689147943379 2x^5 \\
&\quad -597477706485539807857274966 4x^6 \\
&\quad +592218873524243167632445392 0x^7
\end{aligned}$$

Solving the NTRU Equation: Classical Method

$$
\begin{aligned}
G &= (uq/\delta)s \\
&= 24395608958700754945810936 01 \\
&\quad +9378643755587873644889662 75x \\
&\quad +146527679900414108163084 9287x^2 \\
&\quad -2687527298124415179709584748x^3 \\
&\quad -1221766881641064524972754 35x^4 \\
&\quad -1320603114281504647601366 37x^5 \\
&\quad +1604093567321845579875767047x^6 \\
&\quad +148415595515854173169273 2579x^7
\end{aligned}
$$

Solving the NTRU Equation: Classical Method

$$
\begin{aligned}
k &= \left\lfloor \frac{Ff^* + Gg^*}{ff^* + gg^*} \right\rceil \\
&= 4622123611531641739215813 5 \\
&\quad -6852692450030865339318221 3x \\
&\quad -3937994046682674957485721 2x^2 \\
&\quad +7481891514846849477203358 2x^3 \\
&\quad -2052140620263188803786872 0x^4 \\
&\quad -5479438415219601533778719 9x^5 \\
&\quad +1612289344818678659084635 4x^6 \\
&\quad +1459057490704990875841968 1x^7
\end{aligned}
$$

Solving the NTRU Equation: Classical Method

$$\begin{aligned} F &\leftarrow F - kf \\ G &\leftarrow G - kg \end{aligned}$$

$$\begin{aligned} F &= 58 + 20x + 17x^2 - 64x^3 - 3x^4 - 9x^5 - 21x^6 - 84x^7 \\ G &= -41 - 34x - 33x^2 + 25x^3 - 41x^4 + 31x^5 - 18x^6 - 32x^7 \end{aligned}$$

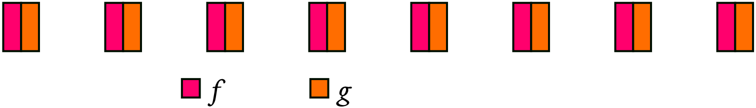
Solving the NTRU Equation: Classical Method

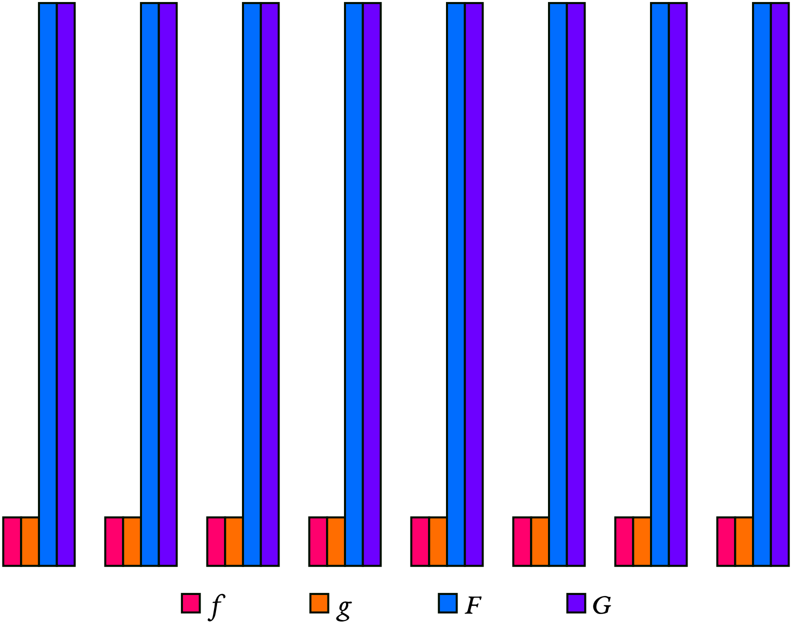
Main problem: intermediate F and G values have *many* coefficients and they are *large*.

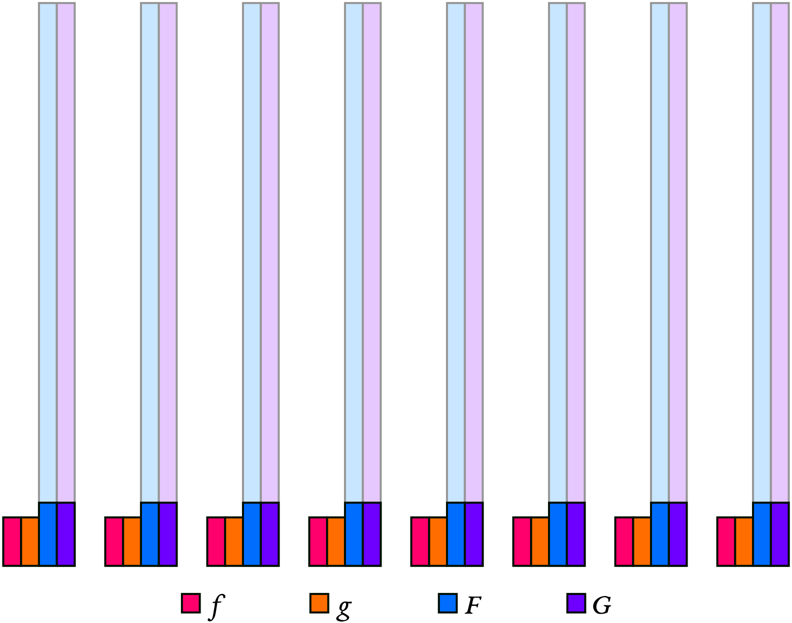
With Falcon-1024:

- $n = 1024$
- Size of each coefficient of s and t: ≈ 6300 bits
- Size of each coefficient of F and G: ≈ 13000 bits
- Total for (F, G): 3.3 megabytes

Small embedded systems typically have 64 kB of RAM (or less).


f
g


f
g
F
G



f
g
F
G

Degree Halving

Let $n = 2^e$.

Intuition: computations over polynomials $a, b \in \mathbb{C}[x]/(x^{n/2}+1)$ are equivalent to computations over polynomials $a(x^2), b(x^2) \in \mathbb{C}[x]/(x^n+1)$.

$$\begin{aligned} a(x^2) + b(x^2) &= (a+b)(x^2) \\ a(x^2)b(x^2) &= (ab)(x^2) \end{aligned}$$

This works for any ϕ such that all non-zero coefficients have even indices.

With $\phi = x^n + 1$ and $n = 2^e$ this can be done repeatedly.

Degree Halving

Let $f \in \mathbb{C}[x]/(x^n + 1)$.

We can write separately the even-indexed and odd-indexed coefficients of f as:

$$f = f_e(x^2) + x f_o(x^2)$$

with $f_e, f_o \in \mathbb{C}[x]/(x^{n/2} + 1)$.

Let $f' = f_e(x^2) - x f_o(x^2)$. We then have:

$$\begin{aligned} ff' &= (f_e(x^2) + x f_o(x^2))(f_e(x^2) - x f_o(x^2)) \\ &= (f_e(x^2))^2 - x^2 (f_o(x^2))^2 \\ &= (f_e^2 - x f_o^2)(x^2) \end{aligned}$$

Degree Halving

We write $N(f)(x^2) = ff'$.

This is the *field norm* for $\mathbb{Q}[x]/(x^n + 1)$ as a field extension of degree 2 of $\mathbb{Q}[x]/(x^{n/2} + 1)$.

Fact:

$$\mathrm{Res}(x^n + 1, f) = \mathrm{Res}(x^{n/2} + 1, N(f))$$

Therefore:

$$\begin{aligned} \mathrm{Res}(x^n + 1, f) &= \mathrm{Res}(x^{n/2} + 1, N(f)) \\ &= \mathrm{Res}(x^{n/4} + 1, N(N(f))) \\ &= \mathrm{Res}(x^{n/8} + 1, N(N(N(f)))) \\ &= \ldots \end{aligned}$$

Fast Resultant

Let p prime such that $p = 1 \bmod 2n$. Let $w \in \mathbb{Z}_p$ such that $w^n + 1 = 0$.

Then $(w^2)^{n/2} + 1 = 0$: w^2 is a root of $x^{n/2} + 1$. Moreover:

$$\begin{aligned} N(f)(w^2) &= f(w)f'(w) \\ &= f(w)(f_e(w^2) - wf_o(w^2)) \\ &= f(w)(f_e((-w)^2) + (-w)f_o((-w)^2)) \\ &= f(w)f(-w) \end{aligned}$$

But $-w$ is also a root of $x^n + 1$. Therefore:

The NTT representation of $N(f)$ (in $\mathbb{Z}_p[x]/(x^{n/2} + 1)$) is obtained by pair-wise multiplying the elements of the NTT representation of $N(f)$ (in $\mathbb{Z}_p[x]/(x^n + 1)$).

Fast Resultant

Let $\phi = x^n + 1$ with $n = 2^e$. Let $f \in \mathbb{Z}[x]/(\phi)$. To compute $\mathrm{Res}(\phi, f)$:

1. For many small primes p such that $p = 1 \bmod 2n$:
   (a) Set $f_p \in \mathbb{Z}_p[x]/(\phi)$ such that $f_p = f \bmod p$.
   (b) Convert f_p to NTT representation (in $\mathbb{Z}_p$).
   (c) Multiply together (modulo p) all elements of the NTT representation of f_p into $r_p \in [0 \dots p-1]$.
2. For each p, $\mathrm{Res}(\phi, f) = r_p$. Use the CRT to rebuild $\mathrm{Res}(\phi, f)$.

This improves the speed of the first steps of the classic NTRU solver, but does *not* reduce memory usage.

Recursive Solver

The field norm is a *product*: $N(f)(x^2) = ff'$

Let $f_n, g_n \in \mathbb{Z}[x]/(x^n + 1)$. Let:

$$\begin{aligned} f_{n/2} &= N(f_n) \\ g_{n/2} &= N(g_n) \end{aligned}$$

We have:

$$\begin{aligned} f_{n/2}(x^2) &= f_n f_n' \\ g_{n/2}(x^2) &= g_n g_n' \end{aligned}$$

Consequence: $f_1 = \mathrm{Res}(x^n + 1, f)$ and $g_1 = \mathrm{Res}(x^n + 1, g)$.

Recursive Solver

Suppose that we found $F_{n/2}, G_{n/2} \in \mathbb{Z}[x]/(x^{n/2}+1)$ such that:

$$f_{n/2}G_{n/2} - g_{n/2}F_{n/2} = q$$

Then set:

$$\begin{aligned} F_n &= g'_n F_{n/2}(x^2) \\ G_n &= f'_n G_{n/2}(x^2) \end{aligned}$$

We then have:

$$\begin{aligned} f_n G_n - g_n F_n &= f_n f'_n G_{n/2}(x^2) - g_n g'_n F_{n/2}(x^2) \\ &= f_{n/2}(x^2) G_{n/2}(x^2) - g_{n/2}(x^2) F_{n/2}(x^2) \\ &= (f_{n/2}G_{n/2} - g_{n/2}F_{n/2})(x^2) \\ &= q \end{aligned}$$

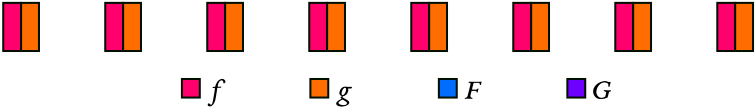
Recursive Solver

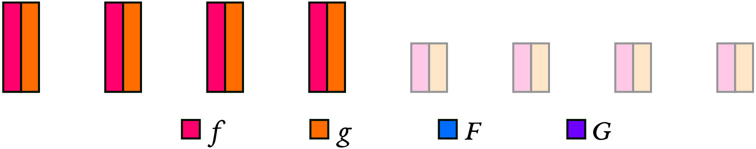
Input: $f_n, g_n \in \mathbb{Z}[x]/(x^n + 1)$

1. If $n = 1$, then:
   (a) Use extended GCD on *integers*: $f_1 u + g_1 v = \delta$.
   (b) If δ does not divide q, then Fail.
   (c) Set $F_1 = -(vq/\delta)$ and $G_1 = (uq/\delta)$.

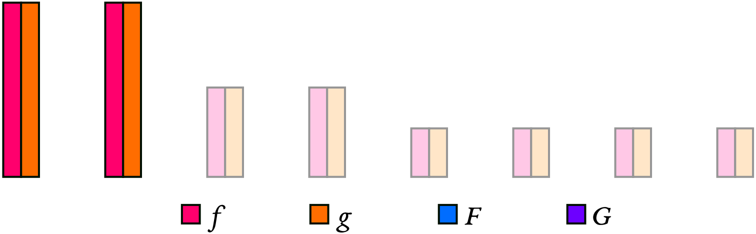
   else:

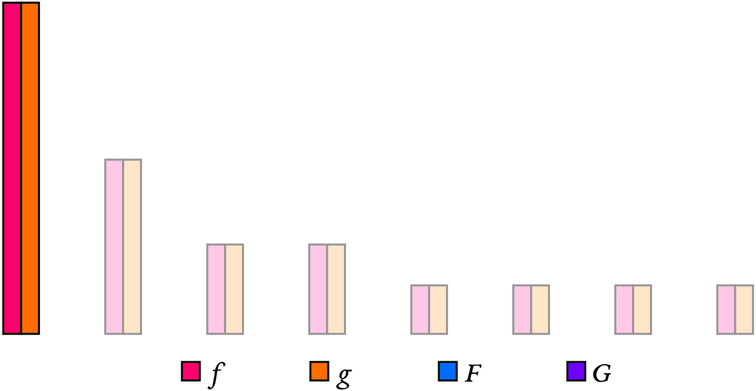
   (a) Set $f_{n/2} = N(f_n)$ and $g_{n/2} = N(g_n)$.
   (b) Call the solver recursively to get $F_{n/2}$ and $G_{n/2}$.
   (c) Set $F_n = g_n' F_{n/2}(x^2)$ and $G_n = f_n' G_{n/2}(x^2)$.
2. Apply Babai's Nearest Plane reduction on (F_n, G_n) to make the coefficients about the same size as those of (f_n, g_n).
3. Return the solution (F_n, G_n).

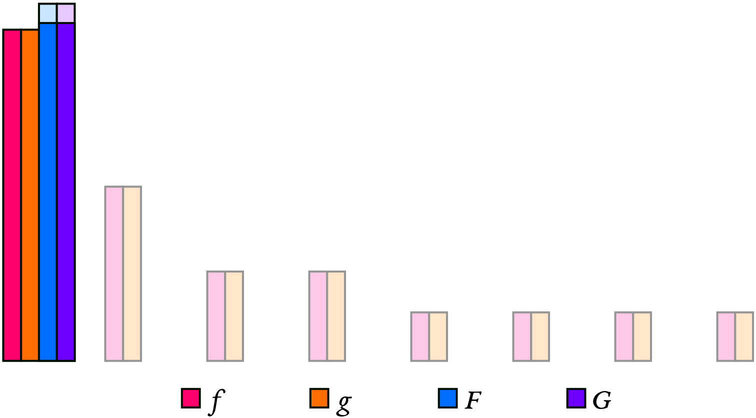

f
g
F
G



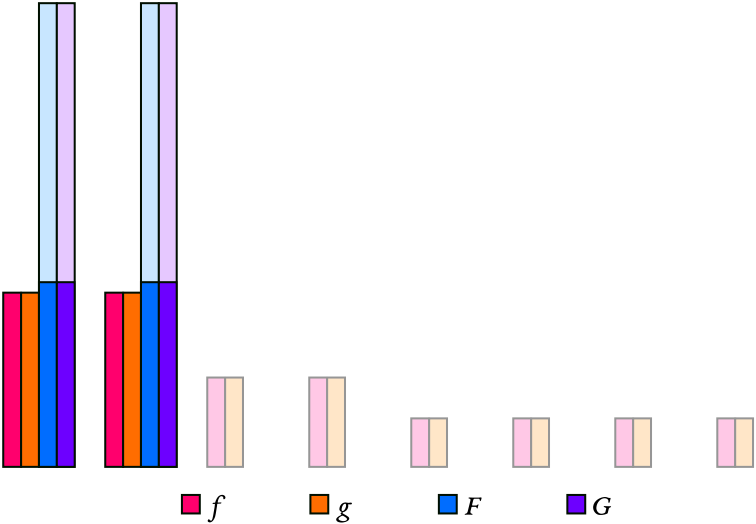
f
g
F
G


f
g
F
G

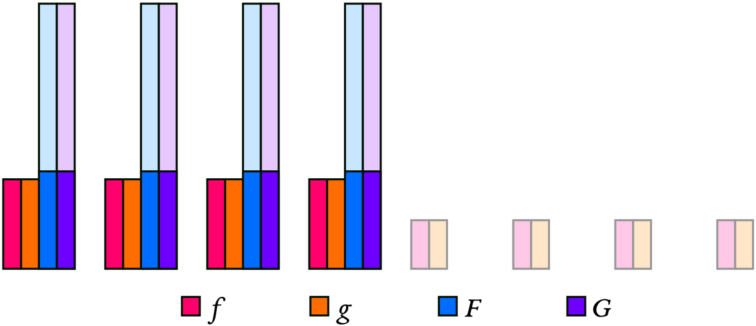

f
g
F
G

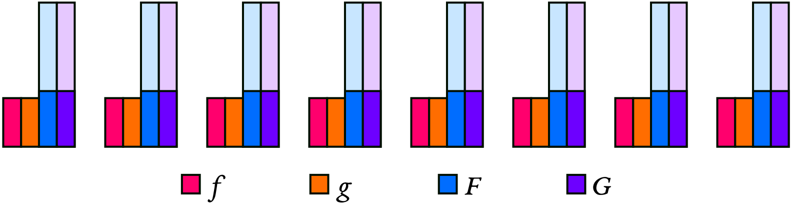


f
g
F
G



f
g
F
G


f
g
F
G


f
g
F
G

Recursive Solver

At each recursive call, polynomial coefficients are twice bigger, but degree is halved: *no uncontrolled memory expansion*.

Total size for $n = 1024$: 28.7 kB

- New solver is 115 times smaller!
- It is also about 100 times faster (20 milliseconds instead of about 2 seconds).

Implemented on ARM Cortex M4: about 170 million cycles for Falcon-512, 510 million cycles for Falcon-1024.

Implementation

Principle: key pair generation is allowed to fail.

- It is not a problem if a small(-ish) proportion of potential private keys are rejected.
- Whether a solution to the NTRU equation is correct or not is inexpensive to verify (small coefficients, can use FFT or NTT).
- For each value, we can *measure* the average size in bits and standard deviation, and allocate a one-size-fits-all buffer. E.g. size of $\mathrm{Res}(\phi, f)$ (deepest recursion):

$$\log|\mathrm{Res}(\phi, f)| \approx 6307.52 \pm 24.48$$

$\Rightarrow$ allocate a 6455-bit buffer, which will almost always be sufficient.

Polynomial Representation

Three competing representations:

- Base-2^{31}: each coefficient f_j is represented as an array of 31-bit words:

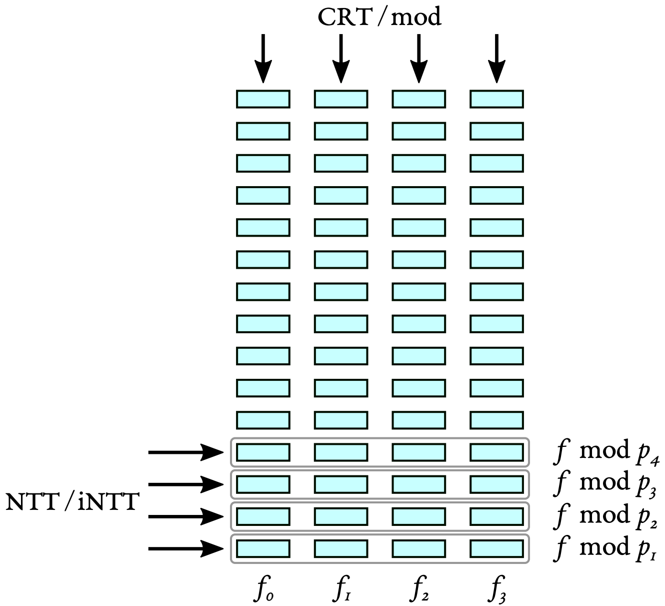
$$f_j = \sum_k f_{j,k} 2^{31k}$$

  (Two's complement for negative values)

- Residue Number System: each coefficient f_j is represented as an array of 31-bit words modulo small primes p_k:

$$\breve{f}_{j,k} = f_j \bmod p_k$$

- RNS + NTT: like RNS, but each *polynomial* $f \bmod p_k$ is in NTT representation.


CRT / mod
NTT / iNTT
f mod p_4
f mod p_3
f mod p_2
f mod p_1
f_0
f_1
f_2
f_3

Polynomial Representation

- Field norms: use RNS + NTT.
- Polynomial multiplications: use RNS. Also use NTT if degree is high (threshold: $n = 16$ or 32).
- RNS to RNS with more moduli: go to base-2^{31} with CRT, then reduce modulo all target primes p_k.
- CRT can be done mostly in-place.
  Requires for each p_k: $\left(\prod_{j<k} p_j\right)^{-1} \bmod p_k$ (precomputed).
- NTT requires a table of primitive root powers: this is regenerated dynamically.

Polynomial Representation

Integers modulo p_k:

- All p_k are chosen close to 2^{31} (but lower).
- Montgomery representation: $z \in \mathbb{Z}_{p_k}$ is represented by $2^{31}z \bmod p_k$.
- Efficient constant-time additions, subtractions and multiplications.
- Inversion modulo p_k: uses Fermat's Little Theorem (raise to power $p_k - 2$ in $O(\log p_k)$ multiplications).

Floating-Point

Babai's Nearest Plane uses non-integers:

- IEEE-754 "binary64" floating-point values can be used (in C, use type `double`).
- On recent x86, operations are constant-time (no subnormals, infinites or NaNs), except some shortcuts for divisions by a power of two (very rare, never observed over thousands of key pair generations).
- If using constant-time emulation of floating-point operations with pure integer code (much slower, but constant-time on all relevant architectures), then key generation time is multiplied by about 2.3.
- Fixed-point (with integer multiplication) probably possible, but requires extra care not to leak information on size of f, g.

Extra Tweaks

- (f_n, g_n) can be discarded at each recursive call: they have to be recomputed afterwards (CPU overhead: +15%) but this saves some kilobytes of RAM and avoids recursive calls (better for shallow stacks).

- When q is prime, and $\mathrm{GCD}(\mathrm{Res}(\phi, f), \mathrm{Res}(\phi, g)) \neq 1, q$, the current f, g are discarded.

  - We can quickly compute $\mathrm{Res}(\phi, f) \bmod 2$: it is the sum modulo 2 of the coefficients.

  - If both $\mathrm{Res}(\phi, f)$ and $\mathrm{Res}(\phi, g)$ are even, we can discard f, g immediately.