# Implementação do Criptossistema Pós-Quântico NTRU conforme a Norma IEEE 1363.1

**Cassius de Oliveira Puodzius**[1] **, Paulo S. L. M. Barreto**[1*]

[1] Departamento de Engenharia de Computação e Sistemas Digitais,
Escola Politécnica, Universidade de São Paulo, Brazil.

cassius.puodzius@poli.usp.br,pbarreto@larc.usp.br

***Abstract.*** *This article addresses the* NTRU *cryptosystem, as a lattice based cryptosystem, specifically the* NTRU*'s scheme as the draft of standard [6] approaches. We discuss in particular some obscure aspects of the standard, aiming to clarify the points needed to obtain correct and secure implementations.*

***Resumo.*** *Descreve-se neste artigo o sistema criptográfico* NTRU *enquanto criptossistema baseado em reticulados, especificamente o esquema* NTRUEncrypt *acolhido no draft da norma [6]. Discutem-se em particular alguns aspectos obscuros da norma, visando a esclarecer os pontos necessários para obter implementações corretas e seguras.*

## 1. Introdução

Com exceção do sistema criptográfico *McEliece*, os sistemas criptográficos clássicos, tais como *RSA* ou criptografia baseada em curvas elípticas, estão fadados a ficarem obsoletos, pois são vulneráveis a ataques quânticos. Em meio a essa realidade tem-se feito novos sistemas criptográficos resistentes a ataques quânticos, alcunhando-os de "*Pós-Quânticos*".

Entre os sistemas criptográficos pós-quânticos está o *NTRU*. No entanto o *NTRU* possui peculiaridades únicas dentre os sistemas criptográficos pós-quânticos, pois o *NTRU* é mais eficiente do que os sistemas criptográficos mais populares atualmente, como destacado em [7]. A complexidade do *NTRU* é $O(n^2)$ e o custo sobre tamanho da chave é $O(n)$, enquanto o *RSA* tem complexidade $O(n^3)$ e custo $O(n^2)$. Além disso, o *NTRU* é um sistema criptográfico completo, com encriptação e assinatura digital e pode ser facilmente implementado. Portanto, apesar de pós-quântico, o *NTRU* é um sistema criptográfico adequado à realidade de hoje.

O *NTRU* baseia-se em problemas tidos por intratáveis, embora não se saiba se são NP-difíceis para o reticulado específico adotado nesse criptossistema. Um deles, usado em *NTRUEncrypt*, é o Problema do Vetor Mais Curto (*Shortest Vector Problem*, ou *SVP*), que consiste em encontrar o vetor mais curto do reticulado a partir de uma base qualquer.

### Contribuições do trabalho

Nossas contribuições neste artigo são apresentar e elucidar todos os pormenores necessários para implementar o *NTRUEncrypt* conforme definido no presente *draft* da norma [6] gerida pelo *IEEE*, além de apontar imprecisões e suprir omissões algorítmicas

*Orientador do trabalho. Bolsista de produtividade em pesquisa do CNPq, processo 303163/ 2009-7.

desse *draft*. Empenhamo-nos também em apresentar a estrutura do *NTRU* em reticulados, algo ainda infrequente na literatura. Não apresentamos o algoritmo de assinatura *NTRUSign* por falta de espaço e por não estar acolhido no *draft* da norma [6], mas necessário para um criptossistema completo.

O restante deste documento está organizado da seguinte maneira. Apresenta-se o sistema *NTRU* com ênfase nos reticulados na Seção 2. A implementação do *NTRUEncrypt*, onde veremos como tratá-lo de maneira simples em um anel de polinômios segundo o *draft* da norma [6] é exposta na Seção 3. Mostraremos nessa mesma Seção todos os cuidados que devemos tomar ao implementá-lo para que o código não fique susceptível a ataques e o compararemos ao RSA. Nossas conclusões são elencadas na Seção 4.

## 2. NTRU

O *NTRU* é um sistema criptográfico baseado em reticulado. Para definir-se os reticulados utilizados no *NTRU*, definimos primeiramente a matriz **T**

$$\mathbf{T} = \begin{pmatrix} \mathbf{0}_{1x(n-1)} & \mathbf{I}_{(n-1)x(n-1)} \\ 1 & \mathbf{0}_{(n-1)x1} \end{pmatrix} \tag{1}$$

Essa matriz representa a transformação linear que roda ciclicamente as coordenadas de um vetor de entrada.

Seja **v** um vetor tal que $\mathbf{v} \in \mathbb{Z}^n$. Definimos a matriz circulante de **v** como

$$\mathbf{v} * \mathbf{T} = [\mathbf{v}, \mathbf{vT}, \ldots, \mathbf{vT}^{n-1}] \tag{2}$$

Os reticulados utilizados no *NTRU* obedecem a duas condições:

- São fechados sob a transformação linear que mapeia o vetor (**x**,**y**) em (**Tx**, **Ty**), onde dim(**x**) = dim(**y**) = $n$.
- Os reticulados são q-nários, i.e., sempre contêm $q\mathbb{Z}^{2n}$ como subreticulados. Portanto (**x**,**y**) pertencer ao reticulado depende unicamente de (**x**,**y**) $\pmod q$.

Esses reticulados são chamados de reticulados convolucionais modulares.

### 2.1. Chave Privada

A chave privada do *NTRU* é um vetor curto $(\mathbf{f},\mathbf{g}) \in \mathbb{Z}^{2n}$, assim, associa-se um reticulado à chave privada e ao parâmetro $q$. Os vetores **f**,**g** obedecem às retrições:

- A matriz $[\mathbf{f} * \mathbf{T}]$ deve ser inversível módulo $q$.
- $\exists \mathbf{v} \in \mathbb{Z}^n : \mathbf{f} - \mathbf{v} \in \{-p, 0, p\}^n$ e $\mathbf{g} \in \{-p, 0, p\}^n$, tais que **f** e **g** são aleatórios e $\mathbf{f} - \mathbf{v}$ e **g** possuam $d_f + 1$ coeficientes positivos e $d_f$ coeficientes negativos, os outros $n - 2d_f - 1$ coeficientes são nulos.

É importante que a questão dos coeficientes esteja clara, pois quando deseja-se implementar o *NTRU* é comum cometer-se o erro de gerar **f** e **g** com mesmo número de coeficientes positivos e negativos. Inclusive esse erro é cometido no draft [6], quando gera-se o vetor **g** e verifica-se se **g** possui inverso. De posse do corolário a seguir, vemos que se os coeficientes forem gerados segundo o draft [6], **g** não será inversível módulo $q$.

**Teorema 1.** $\sum_{j=0}^{d} a_j x^j \pmod{(x-1)} = \sum_{j=0}^{d} a_j$

*Demonstração.*

$$\begin{aligned}
& a_d x^d + a_{d-1}x^{d-1} + \ldots + a_0 = \\
= \quad & a_d x^d + (-a_d x^{d-1} + a_d x^{d-1}) + a_{d-1}x^{d-1} + \ldots + a_0 = \\
= \quad & a_d x^{d-1}(x-1) + (a_d + a_{d-1})x^{d-1} + \ldots + a_0 = \\
\vdots & \\
= \quad & a_d x^{d-1}(x-1) + \ldots + (a_d + \ldots + a_1)(x-1) + (a_d + \ldots + a_0) \\
\Rightarrow \quad & (a_d x^d + a_{d-1}x^{d-1} + ... + a_0) \pmod{(x-1)} = (a_d + a_{d-1} + ... + a_0) \pmod{(x-1)}
\end{aligned}$$

□

**Corolário 1.** $\sum_{j=0}^{d} a_j = 0 \Rightarrow \sum_{j=0}^{d} a_j x^j \pmod{(x^d-1)} = 0.$

*Demonstração.*

$$\sum_{j=0}^{d} a_j = 0 \Rightarrow \sum_{j=0}^{d} a_j x^j \pmod{(x-1)} = 0$$

$$(x-1)|(x^d-1) \Rightarrow \sum_{j=0}^{d} a_j x^j \pmod{(x^d-1)} = 0$$

□

Essas restrições para gerar **f**,**g** serão usadas como propriedades importantes na geração da chave pública, encriptação e decriptação. Sob essas restrições

$$\begin{aligned}
[\mathbf{f} * \mathbf{T}] &\equiv \mathbf{I} \pmod{p} \\
[\mathbf{g} * \mathbf{T}] &\equiv \mathbf{0} \pmod{p}
\end{aligned}$$

Esse reticulado convolucional modular que contém (**f**,**g**) é representado por $\Lambda_q((\mathbf{f} * \mathbf{T}, \mathbf{g} * \mathbf{T})^T)$. Repare que $(\mathbf{f},\mathbf{g}) \in \Lambda_q((\mathbf{f} * \mathbf{T}, \mathbf{g} * \mathbf{T})^T)$ é consequência direta das propriedades dos reticulados do *NTRU*. Isso já era de se esperar pois a chave privada será uma base de $\Lambda_q((\mathbf{f} * \mathbf{T}, \mathbf{g} * \mathbf{T})^T)$. Dizemos que a chave privada é uma "boa" base, uma vez que ela é composta por vetores curtos.

### 2.2. Chave Pública

A chave pública, em contraste com a chave privada, é uma base "ruim" para $\Lambda_q((\mathbf{f} * \mathbf{T}, \mathbf{g} * \mathbf{T})^T)$. A chave pública é gerada a partir da *Forma Normal de Hermite* (*HNF*) de $\Lambda_q((\mathbf{f} * \mathbf{T}, \mathbf{g} * \mathbf{T})^T)$, onde se obtém uma matriz triângular inferior, que é essencialmente única e pode ser calculada eficientemente usando uma variante integral do *Algoritmo de Eliminação de Gauss*. Devido às propriedades do reticulado convolucional modular e às restrições na escolha de **f**, a base pública HNF é da forma

$$\mathbf{H} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{h} * \mathbf{T} & q \cdot \mathbf{I} \end{pmatrix} : \mathbf{h} = [\mathbf{f} * \mathbf{T}]^{-1}\mathbf{g} \pmod{q} \tag{3}$$

que pode ser compactamente representada pelo vetor $\mathbf{h} \in q\mathbb{Z}^n$

### 2.3. Encriptação

Dada uma mensagem de entrada, obtém-se um vetor $\mathbf{m} \in \{-1, 0, 1\}^n$ que a representa. Além de $\mathbf{m}$, deve-se gerar aleatoriamente um vetor $\mathbf{r} \in \{-1, 0, 1\}^n$. Determinados os vetores $\mathbf{m}$ e $\mathbf{r}$, defini-se o vetor curto de erro concatenando $\mathbf{m}$ e $\mathbf{r}$, temos então que o vetor curto de erro é

$$(-\mathbf{r}, \mathbf{m}) \in \{-1, 0, 1\}^{2n}$$

Calculando $\mathbf{H}^{-1}(-\mathbf{r}, \mathbf{m}) \pmod q$, segue

$$\begin{pmatrix} q \cdot \mathbf{I} & \mathbf{0} \\ -\mathbf{h} * \mathbf{T} & \mathbf{I} \end{pmatrix} \begin{pmatrix} -\mathbf{r} \\ \mathbf{m} \end{pmatrix} \pmod q = \begin{pmatrix} \mathbf{0} \\ \mathbf{m} + [\mathbf{h} * \mathbf{T}]\mathbf{r} \pmod q \end{pmatrix} \tag{4}$$

Como as últimas $n$ coordenadas desse vetor são sempre nulas, omitimo-as e obtemos, portanto um vetor $n$-dimensional

$$\mathbf{e} = \mathbf{m} + [\mathbf{h} * \mathbf{T}]\mathbf{r} \pmod q \tag{5}$$

A mensagem encriptada é $\mathbf{e}$.

### 2.4. Decriptação

Para decriptarmos a mensagem, usamos a identidade

$$[\mathbf{f} * \mathbf{T}][\mathbf{h} * \mathbf{T}] = [([\mathbf{f} * \mathbf{T}]\mathbf{h}) * \mathbf{T}] \tag{6}$$

Válida para quaisquer vetores $\mathbf{f}$ e $\mathbf{h}$.

De posse dessa identidade, temos

$$[\mathbf{f} * \mathbf{T}]\mathbf{e} = [\mathbf{f} * \mathbf{T}]\mathbf{m} + [\mathbf{f} * \mathbf{T}][\mathbf{h} * \mathbf{T}]\mathbf{r} = [\mathbf{f} * \mathbf{T}]\mathbf{m} + [\mathbf{g} * \mathbf{T}]\mathbf{r} \pmod q \tag{7}$$

Os valores absolutos das coordenadas do vetor $[\mathbf{f} * \mathbf{T}]\mathbf{m} + [\mathbf{g} * \mathbf{T}]\mathbf{r}$ são limitados por $q/2$, assim, pode-se determinar os valores exatos das coordenadas sem redução módulo $q$. Podemos agora fazer a redução módulo $p$ desse vetor agora, obtendo de volta a mensagem $\mathbf{m}$.

$$[\mathbf{f} * \mathbf{T}]\mathbf{m} + [\mathbf{g} * \mathbf{T}]\mathbf{r} \pmod p = \mathbf{I} \cdot \mathbf{m} + \mathbf{0} \cdot \mathbf{r} = \mathbf{m} \tag{8}$$

## 3. Implementação

A implementação do *NTRU* recai sobre a representação de elementos no anel de polinômios truncado $(\mathbb{Z}/q\mathbb{Z})[X]/(X^N - 1)$, representado por $\mathscr{R}_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N - 1)$, além de funções para mudança entre tipos de dados, que nos permitam representar mensagens como elementos do anel de polinômios trucados, números inteiros como cadeia de bytes e suas operações inversas.

As operações de $\mathscr{R}_q$ utilizadas no NTRU são soma/subtração, multiplicação por escalar, inversão e convolução. Além disso, na operação de decriptação é necessária a operação de congruência centralizada.

A soma/subtração e multiplicação por escalar de elementos de $\mathscr{R}_q$ são análogas às soma/subtração e multiplicação do anel canônico de polinômios, com o fato de que após a operação deve-se obter a congruência de cada coeficiente módulo $q$ e corrigir o grau do polinômio resultante, se necessário.

A inversão polinomial pode ser obtida mediante o Algoritmo Estendido de Euclides. Os polinômios gerados no NTRU quase sempre serão inversíveis, uma prova formal disso pode ser vista em [8]. Já o pseudo-código da inversão polinomial apropriado para $\mathscr{R}_q$ pode ser obtido em [1].

A convolução de dois elementos de $\mathscr{R}_q$ é representada por

$$h = f \circledast g \ : \ h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i} \tag{9}$$

Essa operação é um mapeamento do produto de $\mathbb{Z}/q\mathbb{Z}[X]$ em $\mathbb{Z}/q\mathbb{Z}[X]/(X^N - 1)$, exatamente como acontece com as operações dos anéis modulares dos números inteiros. Assim, outra forma de se calcular a convolução seria calcular o produto de polinômios em $\mathbb{Z}/q\mathbb{Z}[X]$ e obter a congruência módulo $N$ dos expoentes. A complexidade da operação de convolução é $O(n^2)$, porém em geral os polinômios trabalhados no NTRU possuem coeficientes $0, +1$, ou $-1$, portanto essas operações são computadas com muita rapidez. Pode-se ainda diminuir cerca de 25% das operações no cálculo da convolução particionando o polinômios em dois, conforme [9]. J. H. Silverman propôs que para grandes valores de $N$, múltiplos de 2, poderia-se utilizar a Transformada Rápida de Fourier para computar a convolução em $O(n \log n)$, entretanto se $N$ não for primo o NTRU fica susceptível a um ataque devido a [2].

**Definição 1.** *Seja* $\mathfrak{B}(d) \subset \mathscr{R}$ *o conjunto dos polinômios tais que*

- *possuem* $d$ *coeficientes são iguais a* $1$
- *possuem* $d-1$ *coeficientes são iguais a* $-1$
- *o restante dos coeficientes são iguais a* $0$

**Definição 2.** *A redução centralizada módulo* $p$ *de um polinômio* $P$ *mapeia seus coeficientes em* $]\lfloor \frac{p}{2} \rfloor, \lfloor \frac{p}{2} \rfloor]$

### 3.1. Tipos de dados e conversões

A implementação do *NTRUEncrypt* exige converter tipos de dados entre polinômios, números inteiros e cadeias de bytes. Diferentemente do que sugere o *draft* [6], pode-se implementar estas funções eficientemente usando *bitwise operation*. Para *NTRUEncrypt* devemos ter

I2OSP(*Integer to Octet String Primitive*), OS2IP(*Octet String to Integer Primitive*): recebe-se um inteiro $n$ e devolve-se uma sequência de bytes onde os bits representam o inteiro na base binária, e vice-versa.

RE2OSP(*Ring Element to Octet String Primitive*): No draft [6], em todos os níveis de segurança e otimizações, $q = 2056$, segue que os elementos de $\mathbb{Z}/q\mathbb{Z}[X]/(X^N - 1)$ são

representados por 11 bits. Portanto a cada 8 coeficientes obteremos 11 bytes. Se usarmos *bitwise operation* despenderemos cerca de 11 vezes menos memória do que se seguirmos o algoritmo apresentado em [6], além de reduzirmos o número de operações.

TP2OSP(*Ternary to Octet String Primitive*), OS2TPP(*Octet String to Ternary Primitive*): Estas funções convertem um polinômio ternário, i.e., com coeficientes $-1$, 0 e 1, em uma cadeia de bytes e vice-versa.

Em TP2OSP 16 coeficientes determinam 3 bytes. Os menores coeficientes determinam os bits mais significativos, assim, caso o grau do polinômio não tenha congruência 0, 6 ou 12 módulo 16, pode-se completá-lo com coeficientes nulos até que o seja. Se o grau do polinômio tiver congruência 0 módulo 16, a cadeia de bytes poderá ser determinada sem descartar nenhum bit, se tiver congruência 6 descarta-se o bit menos significativo e se tiver congruência 12 descartam-se os dois bits menos significativos, ambos do byte mais significativo. Em OS2TPP determinam-se 16 coeficientes a cada 3 bytes. Se o número de bytes não for um múltiplo de 3, ao final restará 1 ou 2 bytes. Se sobrar 1 byte, acrescenta-se o bit 0 na posição menos significativa da cadeia de bits, passando a ficar com 9 bits e, portanto, determinando-se 6 coeficientes. Se sobrarem 2 bytes, acrescenta-se o bit 0 nas duas posições menos significativas da cadeia de bits, passando a ficar com 18 bits e, portanto, determinando-se 12 coeficientes. Repare que os bits acrescentados são aqueles mesmos descartados em TP2OSP, preservando dessa forma a relação de inversibilidade entre OS2TPP e TP2OSP.

A conversão deve seguir a tabela contida em [6] em *Encryption Operation* e *Descryption operation*. Em OS2TPP utiliza-se a mesma tabela de TP2OSP, mas dessa vez com os bits como entrada.

OS2RE4P(*Octet String to Ring Element mod 4 Primitive*), RE42OSP(*Ring Elemente mod 4 to Octet String Primitive*): A única conversão de cadeia de bytes em elementos do anel tais que esse anel não é $\mathbb{Z}/q\mathbb{Z}[X]/(X^N - 1)$, é para $\mathbb{Z}/\mathbb{Z}_4[X]/(X^N - 1)$. Assim deve-se implementar funções que convertam uma cadeia de bytes em um elemento do anel $\mathbb{Z}/\mathbb{Z}_4[X]/(X^N - 1)$ e vice-versa. Essas operações são simples, porque os elementos de $\mathbb{Z}/\mathbb{Z}_4[X]/(X^N - 1)$ são representados por 2 bits e um byte (8 bits) é seu múltiplo, assim, cada byte é convertido em 4 coeficientes do polinômio $P \in \mathbb{Z}/\mathbb{Z}_4[X]/(X^N - 1)$ e vice-versa.

Note que não é necessário nenhuma função que converta cadeias de bytes em elementos do anel de polinômios truncados, apesar de haver no *draft* [6] o pseudo-código OS2REP que realiza essa operação.

### 3.2. Geração das chaves

Para gerar as chaves serão necessários os parâmetros $N$, $p$, $q$, $d_f$, $d_g$, bem como um gerador de polinômios aleatórios. Onde $N$ é a dimensão do anel polinômico utilizado no *NTRU*, $q$ é um inteiro que especifica o anel $\mathbb{Z}/q\mathbb{Z}$ no qual pertencem a chave pública e a mensagem encriptada, $p$ é um inteiro que especifica o anel $\mathbb{Z}/p\mathbb{Z}$ no qual pertence a chave privada, onde é realizada a redução modular centralizada, e por fim, $d_f$ e $d_g$ são inteiros que determinam a distribuição dos coeficientes de $f$, $g$. Escolhem-se $q$ e $p$ tais que $q \gg p$. Em geral $q$ é uma potência de 2 e $p$ é um primo, inclusive no draft [6], em todos os níveis de segurança e em todas as otimizações (tamanho e velocidade), $q = 2048$ e $p = 3$. Já $d_f$ define o limite do tamanho dos polinômios. É importante que $gcd(p, q) = 1$, pois se $p \mid q$, tem-se que $e \equiv m \pmod{p}$ onde $e$ é a mensagem encriptada e $m$ é a mensagem, o que

torna a encriptação extremamente insegura.

As chaves são determinadas gerando-se aleatoriamente polinômios $f \in \mathfrak{B}(d_f)$ e $g \in \mathfrak{B}(d_g)$. Entretanto $f$ deve ter a restrição de possuir inverso módulo $p$ e módulo $q$ e $g$ possuir inverso módulo $q$, pois a prova de segurança [5] necessita que a chave pública $h$ seja inversível módulo $q$ e para tanto $g$ também tem que ser. Para aumentar a eficiência é usual sortear um polinômio $F$ com $d_f$ coeficientes iguais a 1 e $d_f$ coeficientes iguais a $-1$, depois calcular $f = pF + 1$ e verificar se $f$ possui inverso módulo $q$, note que dessa forma a soma dos coeficientes de $f$ é igual a 1 e que $f_p = f^{-1} = 1 \pmod{p}$. Se $f$ tiver inverso módulo $p$ e módulo $q$, a chave pública $h$ será determinada por

$$h = f_q \circledast g \pmod{q} \tag{10}$$

A chave pública é $h$ e a chave privada $f$. Deve-se guardar $(f_p, f)$, mas como o *draft* [6] utiliza-se da otimização descrita acima, então $f_p \equiv 1$, o que nos exime de armazenar $f_p$.

É importante ressaltar que $p$ é um primo pequeno, entretanto o custo computacional para determinar-se $f_p$ é o mesmo custo para determinar-se $f_q$, levando em conta que a operação de maior custo no *NTRU* é o cálculo de polinômios inversos, gerar $f = pF + 1$ otimiza a geração da chave pública. Essa otimização fornece o nível de segurança desejado, como abordado em [4].

### 3.3. Encriptação

Para a encriptação são necessários os parâmetros $N$, $p$, $q$, $h$ (*chave pública*) e $m$, que é a cadeia de bytes da *mensagem* a ser encriptada. No draft [6] a encriptação e decriptação está de acordo com [3], onde utiliza-se uma função geradora e uma função hash para solucionar uma insegurança de envelope digital presente na formulação original do *NTRU*. Para tanto formam-se as cadeias de bytes

$$\begin{aligned} M &\leftarrow b \,||\, octL \,||\, m \,||\, p0 \\ sData &\leftarrow ODI \,||\, m \,||\, b \,||\, \text{hTrunc} \end{aligned}$$

Onde $b$ é uma cadeia de bytes aleatórios de tamanho pré-determinado, $octL$ o byte que representa o tamanho da mensagem $m$, que pode ser obtido a partir de I2OSP e $p0$ é a cadeia de bytes nulos que completam a mesagem $M$ em um tamanho pré-determinado, $ODI$ é uma cadeia de bytes pré-determinada e hTrunc é formada pelo primeiros $pkLen$ bytes da cadeia de bytes obtida pela conversão da chave pública $h$ em uma cadeia de bytes, mediante RE2OSP. Pode-se obter $pkLen$ em [6].

Uma vez determinado $M$, obtemos o polinômio $M_{ter}$ com OS2TPP. De posse de $sData$, obtém-se o polinômio $r$ mediante a função hash e entrada $sData$. Calcula-se então

$$R = h \circledast r \pmod{q} \tag{11}$$

Converte-se $R$ na cadeia de bytes $oR4$ mediante RE42OSP. Então gera-se o polinômio ofuscador $mask$ com a função geradora e $oR4$ de entrada e calcula-se

$$\begin{aligned} m' &= M_{ter} + mask \pmod{p} \\ e &= m' + R \pmod{q} \end{aligned}$$

Onde $e$ é a mensagem encriptada.

### 3.4. Decriptação

Os parâmetros para a decriptação são os mesmos utilizados na encriptação acrescidos por $e$, a mensagem encriptada, $f$ e $h$, as chaves privada e pública, respectivamente e $f_p$.

Primeiramente calcula-se o polinômio candidato a mensagem $m'_c$

$$\begin{aligned} a &= f \circledast e \pmod{q} \\ m'_c &= f_p \circledast a \pmod{p} \end{aligned}$$

Onde $m'_c$ deve ter redução modular centralizada, é importante insistir nesta condição da redução modular ser centralizada, pois isto é crucial para que a decriptação funcione como esperado, porém na maior parte da literatura não é dada a ênfase devida, inclusive o *draft* [6] da norma que contempla o *NTRUEncrypt* pretere essa parte das instruções técnicas para a obtenção de $m'_c$. Então calcula-se o polinômio $R_c$ candidato a $h \circledast r \pmod{q}$

$$R_c = e - m'_c \pmod{q} \tag{12}$$

De posse de $R_c$, determina-se o polinômio ofucador *mask*, fazendo a conversão de $R_c$ para $coR4$ mediante RE42OSP. Então determina-se o polinômio o candidato à cadeia de bytes $cM$, da seguinte forma

$$cM_{ter} = m'_c - mask \pmod{q} \tag{13}$$

convertendo o polinômio $cM_{ter}$ para $cM$ mediante TP2OSP.

Se a decriptação for bem sucedida $cM$ será a cadeia de bytes

$$b \parallel octL \parallel m \parallel p0 \tag{14}$$

Para verificar, determina-se hTrunc a partir da chave pública $h$ e a cadeia de bytes

$$sData \leftarrow ODI \parallel cm \parallel cb \parallel \text{hTrunc} \tag{15}$$

Onde $cb$ é a cadeia de bytes formada pelos primeiros bytes de $cM$. O tamanho de $cb$ é o mesmo de $b$ da função de encriptação, tamanho esse que fora pré-determinado. O próximo byte de $cM$ após àqueles de $cb$ contém o número de bytes do candidato à mensagem $cm$.

De posse de $sData$, obtém-se o polinômio $r_c$ mediante a função hash e entrada $sData$. Calcula-se então

$$R'_c = h \circledast r_c \pmod{q} \tag{16}$$

Se $R'_c = R_c$ a decriptação terá sido, de fato, bem sucedida.

Escolhendo-se apropriadamente os parâmetros, como em [6], há uma probabilidade da decriptação falhar. Nesse caso, ainda pode-se tentar recuperar a mensagem escolhendo os coeficientes de $a \equiv f \circledast e$ em um intervalo $]\lfloor \frac{-q}{2} + x \rfloor, \lfloor \frac{q}{2} + x \rfloor]$, para algum $x$ tal que $x \in \mathbb{Z}$. Se ainda assim a mensagem não for recuperada, tem-se então falha de intervalo e a mensagem não poderá ser recuperada facilmente.

### 3.4.1. Por que o NTRU funciona?

Seja $r = \mathcal{H}(m)$ o polinômio gerado pela função hash, que tem indiretamente como parâmetro a mensagem $m$ e $\mathcal{G}(r \circledast h)$ o polinômio ofuscador gerado pela função geradora de polinômios.

Temos

$$\begin{aligned} a &\equiv f \circledast e \pmod{q} \\ &\equiv f \circledast \mathcal{H}(m) \circledast h + f \circledast [M_{ter} + \mathcal{G}(r \circledast h)]_p \pmod{q} \\ &\equiv f \circledast f_p \circledast pr \circledast g + f \circledast [M_{ter} + \mathcal{G}(r \circledast h)]_p \pmod{q} \\ &\equiv pr \circledast g + f \circledast [M_{ter} + \mathcal{G}(r \circledast h)]_p \pmod{q} \end{aligned}$$

Quase sempre o polinômio $a$ os coeficientes poderá ser reduzido a $]\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor]$, assim, fazendo $f_p \circledast a$ e fazendo a redução centralizada dos coeficientes de $f_p \circledast a$, obtemos o polinômio ternário $m'_c$. Podemos então determinar $r \circledast h$ fazendo $r \circledast h = e - m'_c$

Como

$$\begin{aligned} & f_p \circledast a &&= pf_p \circledast r \circledast g + [M_{ter} + \mathcal{G}(r \circledast h)]_p \\ \Rightarrow\ & f_p \circledast a \pmod{p} &&= [M_{ter} + \mathcal{G}(r \circledast h)]_p \end{aligned}$$

De posse de $r \circledast h$ determina-se $\mathcal{G}(r \circledast h)$, seguindo

$$M_{ter} = f_p \circledast a \pmod{p} - \mathcal{G}(r \circledast h) \tag{17}$$

## 3.5. Comparação com o RSA

Testamos nossa implementação feita em C++, ainda sem otimizações além das adotadas no *draft* [6], e a comparamos com o RSA em um AMD Turion 64X2 2.4 GHz. Os dados do RSA-15360 foram omitidos devido ao enorme tempo necessário para gerar os parâmetros apropriados. Os dados experimentais obtidos foram medidos em ms e estão na tabela 1.

| level | Geração das chaves | | Encriptação | | Decriptação | |
|---|---|---|---|---|---|---|
| | RSA | NTRU | RSA | NTRU | RSA | NTRU |
| 112 | 1971 | 15.753 | 1.548 | 1.250 | 110.34 | 4.067 |
| 128 | 4998 | 19.399 | 3.467 | 1.448 | 349.91 | 5.132 |
| 192 | 628183 | 55.110 | 22.320 | 2.615 | 5094.10 | 10.486 |
| 256 | - | 117.024 | - | 5.649 | - | 25.447 |

**Tabela 1. Dados de execução para parâmetros típicos**

# 4. Conclusão

Abordamos o sistema criptográfico *NTRU*, que pertencente à família criptográfica baseada em reticulados, uma das mais promissoras famílias pós-quânticas. O sistema

NTRU é completo no sentido de oferecer encriptação e assinatura digital, ambas muito eficientes e com ordem de crescimento linear no custo de armazenamento da chave em função do nível de segurança.

Apresentamos a estrutura dos reticulados utilizados no *NTRU*, aproveitando essa abordagem para apresentar o *NTRUEncrypt*, e tratamos das questões pertinentes para que se possa implementar o *NTRUEncrypt* conforme o draft da norma [6] gerida pelo IEEE concernentes a criptografias assimétricas baseadas em reticulados. Descrevemos concisamente os algoritmos essenciais e apontamos imprecisões e pontos em aberto no texto do *draft* atual, em particular a necessidade do número exato de elementos não nulos em vetores curtos aleatórios conforme o Corolário 1.

Conforme vimos, mesmo sendo pós-quântico, eficiente, como mostra a tabela 1, e com boas propriedades de segurança, o *NTRUEncrypt*, conforme [6], possui uma implementação muito simples, onde o reticulado convolucional modular do *NTRU* é representado por anéis de polinômios truncados e os vetores do reticulado são polinômios.

## Referências

[1] P. S. L. M. Barreto. The NTRU Cryptosystem. http://www.larc.usp.br/~pbarreto/PQC-A.pdf.

[2] C. Gentry. Key Recovery and Message Attacks on NTRU-Composite. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 182–194, London, UK, 2001. Springer-Verlag.

[3] J. Hoffstein, D. Lieman, J. Pipher, and J. H. Silverman. NTRU: A Public Key Cryptosystem. http://grouper.ieee.org/groups/1363/lattPK/submissions/ntru.pdf.

[4] J. Hoffstein and J. Silverman. Optimizations for NTRU. In *In Publickey Cryptography and Computational Number Theory. DeGruyter*, pages 11–15, 2000.

[5] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte. NAEP: Provable security in the presence of decryption failures. http://www.securityinnovation.com/cryptolab/pdf/NAEP.pdf.

[6] IEEE P1363 Working Group. *IEEE 1363-1: Standard Specifications for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices (Draft)*, 2009. http://grouper.ieee.org/groups/1363/lattPK/index.html.

[7] R. Kouzmenkoe. Generalizations of the NTRU Cryptosystem. Master's thesis, École Polytechnique Fédérale de Lausanne, 2005.

[8] J. H. Silverman. Invertibility in Truncated Polynomial Ring, October 1998. http://www.securityinnovation.com/cryptolab/pdf/NTRUTech009.pdf.

[9] J. H. Silverman. High-Speed Multiplication of (Truncated) Polynomials, January 1999. http://www.securityinnovation.com/cryptolab/pdf/NTRUTech010.pdf.