SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

NTRU-kryptering med Polynomringar och Gitter

av

Javier Cuadra Venegas

2022 - No K29

NTRU-kryptering med Polynomringar och Gitter

Javier Cuadra Venegas

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Jonas Bergström

2022

Abstrakt

Syftet med denna uppsats är att undersöka hur NTRU-kryptering fungerar. NTRU är ett modert krypto som man tror även är säkert mot kvantdatorer som försöker lösa krypteringen utan att använda en dekrypteringsnyckel. Arbetet kommer att gå igenom vad som behövs för att skapa krypteringen, till exempel polynomringar. Det kommer också att tas upp hur man dekrypterar med dekrypteringsnyckeln och ett par alternativ för att dekryptera ifall man saknar nyckeln.

Tack

Jag vill tacka min handledare Jonas Bergström som har alltid varit tillgänglig för att bolla idéer och gett flera goda råd med arbetet.

Innehållsförteckning

1 Bakgrund

1.1 Krypteringens historia

Det har alltid funnits behov för att skicka hemliga meddelanden. Till exempel användes Ceasarchiffret under romarriket för att skicka hemliga meddelanden mellan arméerna ifall fienden skulle få tag på romarnas budbärare. Chiffret fungerade genom att byta ut varje bokstav med en bokstav som var n steg bort i alfabetet. Idag används Ceasarchiffret för att lära barn om kryptering men under romarriket när majoriteten av människorna inte var läskunniga och såg bara blandade bokstäver var krypteringen effektiv.

När fienderna väl kom på hur chiffret fungerade och kunde tvinga fram ett svar genom att flytta fram bokstäverna n steg behövde romarna utveckla sina krypteringar genom att skriva med till exempel dubbla bokstäver. Ceasarchiffret är ett exempel på en symmetrisk kryptering som går ut på att det finns en nyckel som används för att kryptera meddelandet och för att dekryptera meddelandet. I ceasarchiffret används n som nyckel då bokstäverna flyttas n steg fram för att krypteras och sedan n steg bak för att dekrypteras.

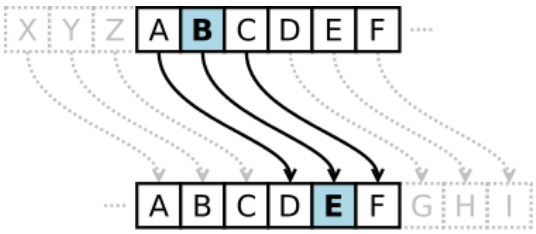


Figure 1: Ceasarchiffret.

Under 1900-talet tillverkades Enigma maskningen som var en av de första datordriva krypteringsmaskiner. Enigma maskinen använde sig av symmetrisk kryptering och krävde att mottagaren av meddelandet hade en egen Enigma maskin med samma inställningar som maskinen som krypterade meddelandet för att kunna dekryptera meddelandet. Då inställningarna på maskinen ändrades dagligen och innehöll tidskänsliga meddelanden var det svårt att dekryptera i tid. Det var i princip omöjligt att dekryptera meddelandena för hand. Istället skapades Bombe, en dator som använde en algoritm för att hitta Enigma maskinens inställningar och att sedan dekryptera meddelandena. Efter Enigma maskinen utfördes majoriteten av krypteringarna och dekrypteringarna med hjälp av datorer som hade lättare att skapa svårare nycklar och utföra mer avancerade algoritmer. Detta ledde tillslut till asymmetriska krypteringar som till skillnad från symmetriska krypteringar hade en publik nyckel för att kryptera och en hemlig nyckel för att dekryptera. Idéen är att personen som skickar meddelandet M krypterar det med hjälp av den publika nyckeln E,

$$E(M) = C.$$

Mottagaren av C, har en hemlig nyckel, D, som hen använder för att dekryptera

$$D(C) = M.$$

På så sätt kan en person skapa båda nycklarna, skicka ut den publika nyckeln och sedan kan endast skaparen dekryptera alla mottagna meddelanden till skillnad från symmetriska krypteringar där både D och E är samma nyckel. Majoriteten av asymmetriska krypteringar har längre beräkningar för att skapa och beräkna funktionerna. Detta har gjort datorer essentiella för kryptering då beräkning för hand kan ta flera livstider.

1.2 RSA

1977 skapades en av de mest välkända asymmetriska krypteringarna, RSA. Krypteringen fungerar genom att en person, Amanda, skapar tre heltal, (n, d, e) med hjälp av primtal. Hon börjar genom att välja två stora primtal, p & q för att räkna ut $pq = n$ och $(p-1)(q-1) = h$. Nästa steg är att hitta ett e som är relativt prim till h. Det finns massor av möjliga e och det bör gå fort att testa olika e med $SGD(e, h) = 1$. Fortsatt använder Amanda Euklides utökade algoritm för att hitta inversen till $e(\mod h)$ för att få fram d. Nu när Amanda har alla delar kan hon dela den publika nyckeln (n, e) så att Bob kan kryptera sitt meddelande M på följande sätt

$$M^e \pmod{n} = C.$$

När Amanda tar emot C kan hon dekryptera C med hjälp av sin privata nyckel (n, d) genom att ta

$$C^d \pmod{n} = M.$$

Exempel 1: RSA-kryptering kan gå till på följande sätt:

Amanda väljer primtalen $p = 11$, $q = 29$, och räknar ut: $n = 319$, $h = 280$, $e = 59$, $d = 19$.	
	Bob kan nu skicka meddelandet $M = 213$ genom att ta $C = 213^{59} \pmod{319} = 14$.
Amanda kan nu dekryptera C, $14^{19} \pmod{319} = 213$.	

Teorin för att knäcka Bobs meddelande utan att ha tillgång till den hemliga nyckeln är simpel. Eftersom den publika nyckeln är känd för alla, så kommer alla redan känna til n och e. För att få tag på d kan man faktorisera n till pq för att sedan räkna ut $(p-1)(q-1)$ och inversen till $e \pmod{(p-1)(q-1)}$. Under 70-talet när RSA skapades var det inte lika lätt att faktorisera n då det inte hade utvecklats någon effektiv metod. Nu för tiden finns det flera program som är mer effektiva med att faktorisera vilket har gjort att krypteringar med lägre värden av n inte är säkra. Detta innebär att för att RSA ska vara säkert behöver p och q vara väldigt stora vilket tar tid att hitta och därmed inte lika effektivt.

1.3 Kvantdatorer

Med ny teknologi introduceras nya sätt att kryptera och knäcka krypteringar. En av de senaste metoderna är användning av kvantdatorer som till skillnad från vanliga datorer är specialiserade för att lösa vissa komplexa problem. Detta är på grund av att istället för bits använder kvantdatorer qubits (eller quantum bits) som har egenskapen att lägga informationen som är i qubiten i superposition. När informationen är i superposition representerar informationen alla möjliga konfigurationer som den kan ha vilket sparar massor av dataminne. Qubits kan även kopplas med andra qubits, detta låter kvantdatorns kraft öka exponentiellt beroende på hur många qubits som är sammankopplade. Till exempel,

ifall en dator har 5 bits så kan den skicka $2^5 = 32$ olika kombinationer. Däremot kan 5 sammankopplade qubits i superposition skicka alla 32 kombinationer samtidigt. Med hjälp av qubits kan kvantdatorer hitta effektivare sätt att lösa komplexa problem samtidigt som det inte krävs lika mycket minne för att spara alla möjliga konfigurationer. Med hjälp av rätt algoritm och tillräckligt många qubits kan en kvantdator hitta nyckeln till några krypton som RSA mycket snabbare än en vanlig dator. Däremot finns det idag ingen algoritm som är tillräckligt bra för att snabbt lösa "kortaste/närmsta vektor-problemet" som är en av metoderna för att hitta nyckeln till NTRUkrypteringar. Nackdelen med kvantdatorer är qubits inte kan göra allt. Till skillnad från bits, är qubits skapade för att testa och lösa komplexa problem men inte för att tänka logiskt eller ge direkta svar. För krypteringar är kvantdatorer väldigt användbara så länge det finns en algoritm som kvantdatorn kan följa. I det fallet kommer kvantdatorn vara bättre på att hantera dataminne och uträkningar som skapas av algoritmerna vilket används för att skapa eller knäcka krypteringar.

I den här uppsatsen utforskas hur en modernare kryptering, NTRU, fungerar och vad som krävs för att knäcka den. Analysen består främst av NTRU-krypteringens skapare, Hoffstein, Pipher och Silvermans bok "An Introduction To Mathematical Cryptography". Men analysen kommer även bestå av artikeln "A Meet-In-The-Middle Attack on NTRU Private Key" av Howgrave-Graham, Silverman och Whyte.
Innan vi går igenom hur NTRU-krypteringen fungerar behöver vi gå igenom vad gitter och polynomringar är och hur dessa kan användas.

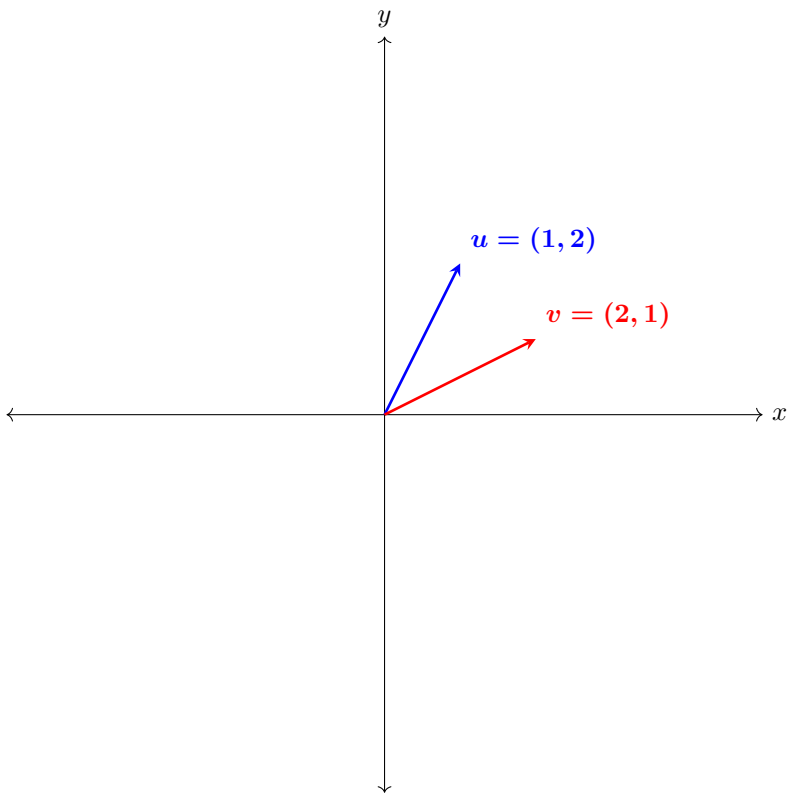
2 Gitter

Gitter är en mängd punkter som är sammankopplade i ett mönster. I just detta fall existerar gittret i vektorrummet där varje punkt i gittret representeras av vektorerna som sträcker sig i rummet.

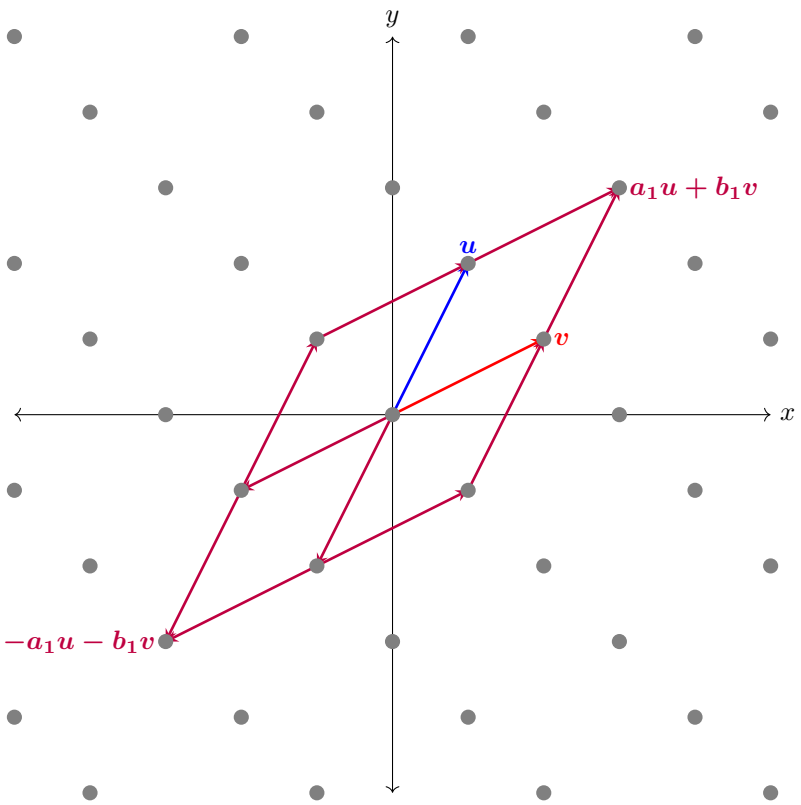
Definition 1: Låt n och d vara två positiva heltal. Låt $b_1 \ldots, b_d \in \mathbb{R}^n$ vara d linjärt oberoende vektorer. Gittret $\mathcal{G}$ genereras av $(b_1 \ldots, b_d)$ och är lika med mängden

$$\mathcal{G} = \sum_{i=1}^{d} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{d} x_i b_i | x_i \in \mathbb{Z} \right\}.$$

För att skapa ett gitter behövs basvektorer som består av en en vektor (eller nollvektor) för varje dimension som gittret består av. Under dessa exempel kommer basvektorerna besitta vektorer som är oberoende av varandra samt heltal då NTRU-krypteringen behöver detta för att den ska fungera. För att illustrera denna definition i $\mathbb{R}^2$ kan vi låta dessa två vektorer,



utgöra en bas för ett gitter. Med dessa basvektorer kan man generera ett gitter genom att addera samt subtrahera vektorerna med varandra där gitterns punkter w hamnar vid vektorernas spetsar. Anta att a_1 och $b_1 \in \mathbb{Z}$, detta ger oss alla möjliga gitterpunkter som nås av vektorerna i $\mathbb{R}^2$ med formeln $w = a_i u + b_j v$ vilket ger oss följande gitter



Detta ger oss även definitionen,
Definition 2: Ett heltalsgitter har endast vektorer med heltalskoordinater.
Det går även att skriva in basvektorerna i en matris för att till exempel hitta andra basvektorer som ger gitter.

Förslag 1: I varje $\mathcal{G}$ finns flera möjliga baser. Två godtyckliga baser är relaterade via en matris som har heltals koefficienter och en determinant som är 1 eller -1. Genom att använda matrismultiplikationen

$$UA = B \tag{1}$$

där U är matrisen med determinanten 1 eller -1 och A/B är två olika matriser som representerar två basvektorer i samma gitter.

Exempel 2: Ett gitter i $\mathbb{R}^3$ kommer att ha tre basvektorer som kan se ut på följande sätt

$$v_1 = (2, 3, 1), \qquad v_2 = (-2, 4, 3), \qquad v_3 = (1, 3, 0).$$

Dessa vektorer kan man lägga in i en matris vilket ger oss 3×3 -matrisen

$$A = \begin{pmatrix} 2 & 3 & 1 \\ -2 & 4 & 3 \\ 1 & 3 & 0 \end{pmatrix}$$

Vi kan skapa nya basvektorer genom att addera och subtrahera vektorerna från A med varandra, till exempel

$$w_1 = v_1 + v_2 - v_3, \qquad w_2 = v_2 - v_3, \qquad w_3 = v_1 + v_2 - 2v_3.$$

För att räkna ut matrisen med de nya basvektorerna kan man utföra matrismultiplikationen $UA = B$ där U är v - vektorernas sammansättning för basvektorerna w

$$U = \begin{pmatrix} 1 & 1 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -2 \end{pmatrix}$$

Då U har determinanten -1 kan vi beräkna $U \times A$, vilket ger oss

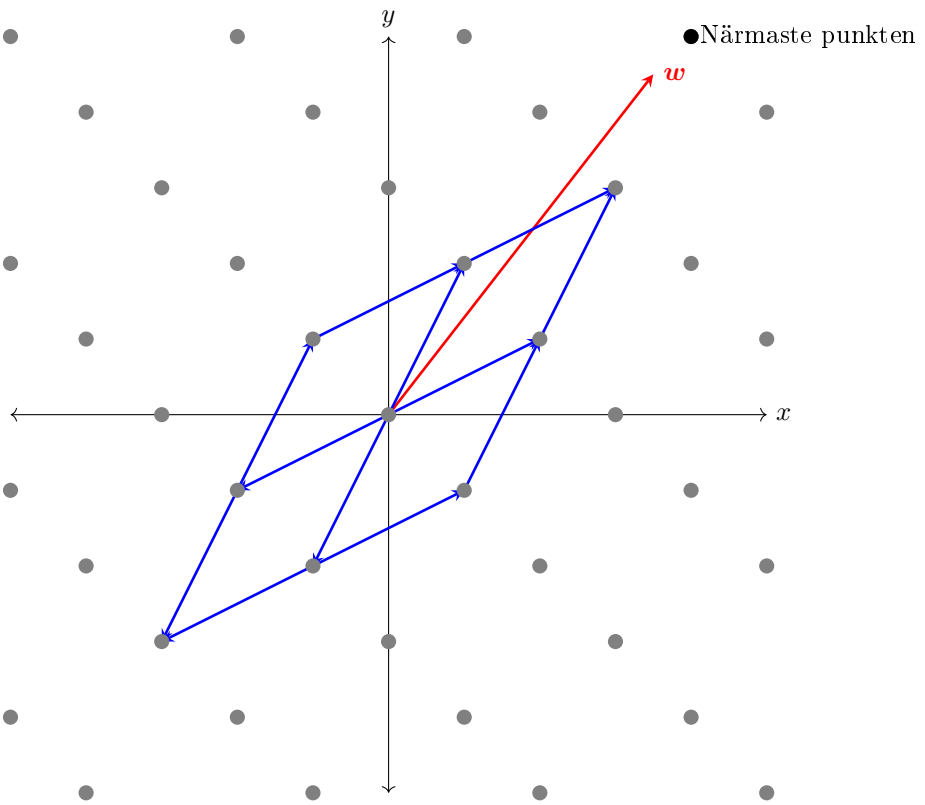
$$UA = B = \begin{pmatrix} -1 & 4 & 4 \\ -3 & 1 & 3 \\ -2 & 1 & 4 \end{pmatrix}$$

Så länge determinanten av U är ± 1 kommer B vara en ny bas för $\mathcal{G}$. U har även egenskapen att raderna av U^{-1} ger w - vektorernas sammansättning för basvektorerna v.

2.1 SVP & CVP

Det finns två problem som är kopplade till gittren. Den ena är det korta vektorproblemet (SVP) som handlar om att bland alla $v \in \mathcal{G}$ hitta den kortaste v som har egenskapen $||v|| > 1$ och minimerar den Euklidiska normen. När gittret har låg dimension är detta inte ett större problem men när gittret har hög dimension blir uträkningen svårare.

Det andra problemet är det närmsta vektorproblemet (CVP). Problemet går ut på att för en given vektor $w \in \mathbb{R}^n$ som inte är en del av $\mathcal{G}$, hitta den närmsta vektorn v som är en del av $\mathcal{G}$.
Exempel 3: Vektorn w är inte en del av $\mathcal{G}$:



Lösningen på CVP ger oss $||w - v||$ vilket även där minimerar den Euklidiska normen.

2.2 Babai's Algoritm

Ett förslag för att hitta en lösning till SVP och CVP är Babai's algoritm. Det speciella med den här algoritmen är att den inte alltid ger korrekt lösning. För att den ska ge korrekt lösning på problemen behöver basvektorerna vara relativt parvist ortogonala med varandra så att

$$v_i \cdot v_j \approx 0 \qquad \text{för alla } i \neq j.$$

Algoritm 1: *Låt $\mathcal{G} \in \mathbb{R}^n$ vara ett gitter med basen $v_1, \ldots, v_n$ och låt $w \in \mathbb{R}^n$ vara en godtycklig vektor. Om basvektorerna är tillräckligt ortogonala till varandra, då kan följande algoritm lösa CVP:*

$$\text{Skriv} \qquad w = t_1 v_1 + t_2 v_2 + \cdots + t_n v_n \text{ med } t_1, \ldots, t_n \in \mathbb{R}$$

$$\text{Sätt} \qquad a_i = [t_i] \text{ för } i = 1, 2, \ldots, n.$$

$$\text{Få tillbaka vektorn} \qquad v = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n.$$

Generellt, om basvektorerna är tillräckligt ortogonala med varandra kommer algoritmen lösa CVP. Men om basvektorerna är långt ifrån att vara ortogonala kommer vektorn som algoritmen ger tillbaka inte vara vektorn som är närmast w.

Exempel 4: Börja med att hitta vinkeln för våra vektorer $v_1 = (2, 7)$ och $v_2 = (8, -3)$ i basvektorn A för att se ifall vinkeln är tillräckligt ortogonal.

$$\begin{aligned}\cos x &= \frac{v_1 \cdot v_2}{|v_1||v_2|} \\ &= \frac{2 \cdot (8) + 7 \cdot (-3)}{\sqrt{2^2 + 7^2}\sqrt{8^2 + (-3)^2}} \\ &= \frac{-5}{\sqrt{53}\sqrt{73}} \\ &\approx -0.08\end{aligned}$$

Vi fortsätter genom att leta efter den närmsta $v \in \mathcal{G}$ till $w = (21, 12)$ på följande sätt

$$(21, 12) = a_1(2, 7) + a_2(8, -3)$$

Vilket ger oss ekvationerna och lösningarna

$$\begin{aligned}21 &= 2a_1 - 8a_2 \\ 12 &= 7a_1 - 3a_2 \\ a_1 &= \frac{159}{62} \\ a_2 &= \frac{123}{62}\end{aligned}$$

Då alla $v \in \mathcal{G}$ är heltal avrundar vi och får

$$\begin{aligned}a_1 &= [t_1] = 3 \\ a_2 &= [t_2] = 2\end{aligned}$$

Detta ger oss $3(2, 7) + 2(8, -3) = (22, 15)$ som är den närmsta v till vektorn w.
Däremot om vi istället hittar nya basvektorer som inte är i närheten av att vara ortogonala får vi ett annat svar. Vi testar med en ny bas genom att med hjälp av (1) beräkna

$$j_1 = 35v_1 + 27v_2 \qquad j_2 = 153v_1 + 118v_2,$$

som har determinanten -1. Fortsatt räkning ger

$$\begin{pmatrix} 35 & 27 \\ 153 & 118 \end{pmatrix} \begin{pmatrix} 2 & 7 \\ 8 & -3 \end{pmatrix} = \begin{pmatrix} 286 & 164 \\ 1250 & 717 \end{pmatrix},$$

vilket ger oss den nya basvektorn med vektorerna $j_1 = (286, 164)$ och $j_2 = (1250, 717)$. Snabb beräkning av $j_1 \cdot j_2$ visar att vektorerna inte är ortogonala med varandra. Vi kan nu använda oss av Babai's algoritm igen och får

$$(21, 12) = a_1(286, 164) + a_2(1250, 717).$$

Vilket ger oss de ungefärliga lösningarna

$$\begin{aligned}a_1 &= 1 \\ a_2 &= 0\end{aligned}$$

Som tidigare sätter vi in detta och får $(286, 164) + 0(1250, 717) = (286, 164)$ vilket vi vet sedan tidigare är långt ifrån den närmsta vektorn till w.
Med Babai's algoritm har vi med tillräckligt ortogonala basvektorer hittat en lösning till CVP medan med icke ortogonala basvektorer får vi fram fel vektor. Detta innebär att en person med rätt basvektorer kan lätt lösa CVP medan någon som fel basvektorer inte kan det. Ifall gittern $\mathcal{G}$ inte har en bas som är ortogonal går det att använda sig av LLL-algoritmen.

Definition 3: Låt $\mathcal{B} = \{v_1, v_2, \ldots, v_n\}$ vara basen för ett ortogonalt gitter $\mathcal{G}$ och låt $\mathcal{B}^* = \{v_1^*, v_2^*, \ldots, v_n^*\}$ vara den ortogonala basen som vart omvandlad med Gram-Schmidt metoden. Basen $\mathcal{B}$ är *LLL reducerad* ifall följande kriterier uppnås:

$$\textbf{Storleks kriteriet} \qquad |\mu_{i,j}| = \frac{v_i \cdot v_j^*}{||v_j^*||^2} \leq \frac{1}{2} \qquad \text{för alla } 1 \leq j < i \leq n,$$

$$\textbf{Lovász kriteriet} \qquad ||v_i^*||^2 \geq \Big(\frac{3}{4} - \mu_{i,i-1}^2\Big)||v_{i-1}^*||^2 \qquad \text{för alla } 1 < i \leq n.$$

Där $\mu_{i,j}$ är koefficienten från projektionen med Gram-Schmidt metoden

$$\mu_{i,j} = \frac{v_i \cdot v_j^*}{v_j^* \cdot v_j^*}.$$

LLL-algoritmen går ut på att upprepa Gram-Schimdt metoden tills båda kriterierna uppnås. Den nya basen som skapas kommer ha om möjligt, både kortare vektorer samt mer ortogonala än tidigare. Lägg märke till att den nya basen inte kommer vara helt ortogonal men detta är inte nödvändigt då Babai's algoritm endast kräver en tillräckligt ortogonal bas.

3 Polynomringar

Innan vi ger oss in på hur NTRU fungerar behöver vi förstå vad polynomringar är för något och hur man kan använda modulo för att förenkla dem.
Definition 4: Fixera ett positivt heltal N. För att förenkla polynomen till grad $N-1$ används ringen

$$R = \frac{\mathbb{Z}[x]}{x^N - 1}.$$

Liknande kan man använda modulo $q \in \mathbb{Z}$,

$$R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{x^N - 1}.$$

Detta innebär att då ett polynom kan skrivas som

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} + a_N x^N,$$

så kan ett element i polynomringen R skrivas som

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1},$$

med koefficienterna i $\mathbb{Z}$ respektive $\mathbb{Z}/q\mathbb{Z}$. Anledningen för att använda R eller R_q till uträkningar istället för vanliga polynom är så att man kan med hjälp av modulo få $x^N \equiv 1 \mod (x^N - 1)$ vilket underlättar beräkningarna. På detta sätt får man en ring med ändligt många element som kan användas i krypteringen. Detta kan även användas under polynomringmultiplikation där man istället använder $\star$ för att markera att multiplikation sker mellan polynom i R istället för vanliga polynom.

Förslag 2: Produkten av två polynom $a(x), b(x) \in R$ ges genom

$$a(x) \star b(x) = c(x) \text{ med } c_k = \sum_{i+j \equiv k \pmod N} a_i b_{k-i},$$

där $0 \leq i, j \leq N-1$ så att $i + j \equiv k \pmod N$ uppfylls. En skillnad när $a(x), b(x) \in R_q$ är att modulo q används på koefficienterna i c_k.

Bevis 1:

$$\begin{aligned}
a(x) \star b(x) &= (\sum_{i=0}^{N-1} a_i x^i) \star (\sum_{j=0}^{N-1} b_j x^j) \\
&= \sum_{k=0}^{2N-2} (\sum_{i+j=k} a_i b_j) x^k \\
&= \sum_{k=0}^{N-1} (\sum_{i+j=k} a_i b_j) x^k + \sum_{k=N}^{2N-2} (\sum_{i+j=k} a_i b_j) x^{k-N} \\
&= \sum_{k=0}^{N-1} (\sum_{i+j=k} a_i b_j) x^k + \sum_{k=0}^{N-2} (\sum_{i+j=k} a_i b_j) x^k \\
&= \sum_{k=0}^{N-1} (\sum_{i+j \equiv k \pmod N} a_i b_j) x^k.
\end{aligned}$$

□

Exempel 5: Vi börjar med att exemplifiera detta genom att ta två polynom $a(x), b(x) \in R$ där

$$\begin{aligned}
a(x) &= 4 - 3x + 5x^2 + 3x^3 + x^5 \\
b(x) &= 2 + 4x - x^3 + 2x^4 - 2x^5.
\end{aligned}$$

Vi sätter att $N = 6$ och våra polynom kommer då existera i $\mathbb{Z}[x]/(x^6 - 1)$. Uträkning ger oss

$$\begin{aligned}
a(x) \star b(x) &= 8 + 10x - 2x^2 + 22x^3 + 23x^4 - 17x^5 + 17x^6 - 4x^7 - 7x^8 + 2x^9 - 2x^{10} \\
&= 8 + 10x - 2x^2 + 22x^3 + 23x^4 - 17x^5 + 17 - 4x - 7x^2 + 2x^3 - 2x^4 \\
&= 25 + 6x - 9x^2 + 24x^3 + 21x^4 - 17x^5.
\end{aligned}$$

Om vi istället finner oss i R_{15} kan vi fortsätta med uträkningen då vi har $q = 15$ och kan därför utföra $\pmod{15}$ på koefficienterna vilket ger oss

$$\begin{aligned}
a(x) \star b(x) &= 25 + 6x - 9x^2 + 24x^3 + 21x^4 - 17x^5 \\
&= 10 + 6x + 6x^2 + 9x^3 + 6x^4 + 13x^5.
\end{aligned}$$

3.1 Centrering och multiplikativ invers av polynomringar

En av metoderna för att göra uträkningar enklare är att med hjälp av modulo q centrera siffrorna i polynomringen.
Definition 5: Låt $a(x) \in R_q$. Centreringen av $a(x)$ till R är det unika polynomet som uppfyller

$$a'(x) \pmod q = a(x)$$

vars koefficienter ligger i intervallet

$$-\frac{q}{2} < a'_i \leq \frac{q}{2}.$$

Exemplet ovan är alltså inte centrerat. För att $a(x) \star b(x)$ ska vara centrerade behöver koefficienterna befinna sig mellan $-\frac{15}{2} < a'_i \leq \frac{15}{2}$ då vi använder samma q, vilket i detta fall inkluderar heltalen $\{-7, -6, -5, \ldots, 5, 6, 7\}$. Centrering av $a(x) \star b(x)$ är alltså

$$a(x) \star b(x) = -5 + 6x + 6x^2 - 6x^3 + 6x^4 - 2x^5 \in R_{15}.$$

Notera att om

$$\begin{aligned}(\text{Centrering av } a(x)) &= a(x)\\ (\text{Centrering av } b(x)) &= b(x),\end{aligned}$$

så får vi att

$$\begin{aligned}(\text{Centrering av } a(x)) \star (\text{Centrering av } b(x)) &= 25 + 6x - 9x^2 + 24x^3 + 21x^4 - 17x^5\\ \text{Centrering av } (a(x) \star b(x)) &= -5 + 6x + 6x^2 - 6x^3 + 6x^4 - 2x^5\end{aligned}$$

ger olika resultat men med modulo q så är de lika.
Multiplikativ invers av polynom går endast att räkna ut ifall man multiplicerar ett polynom $f(x)$ med $1/f(x)$ vilket inte är ett polynom om inte graden för f är större än 0. Men med hjälp av polynomringar är det lättare att hitta ett $b(x)$ som ger oss $a(x) \star b(x) = 1$.
Exempel 6: Med $N = 2$, $q = 7$ och $a(x) = 1 + 2x$ behöver vi hitta ett $b(x)$ som ger oss

$$(1 + 2x) \star b(x) = 1.$$

Istället för att multiplicera $a(x)$ med $1/a(x)$ kan vi låta $b(x) = (2 + 3x)$ och kan nu beräkna

$$\begin{aligned}a(x) \star b(x) &= (1 + 2x)(2 + 3x)\\ &= 2 + 7x + 6x^2\\ &= 8 + 7x\\ &= 1.\end{aligned}$$

Lemma 1: Låt q vara ett primtal. Då kommer $a(x) \in R_q$ ha en multiplikativ invers precis då

$$\text{SGD}(a(x), x^N - 1) = 1 \text{ i } (\mathbb{Z}/q\mathbb{Z})[x]. \tag{2}$$

Ifall $a(x)$ och $(x^N - 1)$ har SGD=1 går det att utföra Euklides utökade algoritm för att hitta polynomen $u(x), v(x) \in (\mathbb{Z}/q\mathbb{Z})[x]$ som uppfyller

$$a(x)u(x) + (x^N - 1)v(x) = 1.$$

Vilket kommer ge oss $a(x)^{-1}$ som är $u(x)$.

Lemma 2: Enligt lemma 1 kan vi hitta polynomen $u(x), v(x) \in (\mathbb{Z}/q\mathbb{Z})[x]$ som tillfredsställer

$$a(x)u(x) + (x^N - 1)v(x) = \text{ SGD}(a(x), x^N - 1).$$

Ifall SGD$= 1$ så kommer modulo $(x^N - 1)$ ge $a(x) \star u(x) = 1 \in R_q$. På andra sidan, ifall $a(x)$ är i R_q, då kan vi hitta polynomet $u(x)$ så att $a(x) \star u(x) = 1 \in R_q$. Med definitionen av R_q får vi

$$a(x)u(x) \equiv 1 \bmod (x^N - 1),$$

så med överensstämmelse av definitioner finns det ett polynom $v(x)$ som tillfredställer

$$a(x)u(x) - 1 = 1 = (x^N - 1)v(x) \text{ i } \mathbb{Z}/q\mathbb{Z})[x].$$

Exempel 7: Vi letar efter $a(x)^{-1}$ där $a(x) = (x^3 + x^2 + 2) \in (\mathbb{Z}/3\mathbb{Z})[x]/(x^4 - 1)$. Eftersom vi befinner oss i $q = 3$ kan vi använda oss av modulo 3 och kan därför skriva $x^4 - 1 = x^4 + 2$. Vi börjar räkna ut SGD

$$\begin{aligned}(x^4 + 2) &= (x + 2)(x^3 + x^2 + 2) + (x^2 + x + 1)\\ (x^3 + x^2 + 2) &= x(x^2 + x + 1) + (2x + 2)\\ (x^2 + x + 1) &= 2x(2x + 2) + 1.\end{aligned}$$

Då vi vet att SGD $= 1$ kan vi utföra Euklides utökade algoritm och får

$$\begin{aligned}1 &= (x^2 + x + 1) - (2x)(2x + 2)\\ &= (x^2 + x + 1) - (2x)((x^3 + x^2 + 2) - x(x^2 + x + 1))\\ &= (x^4 + 2) - (x + 2)(x^3 + x^2 + 2) - (2x)((x^3 + x^2 + 2) - x((x^4 + 2) - (x + 2)(x^3 + x^2 + 2)))\\ &= -(x^3 + x^2 + 2)(2x^3 + 4x^2 + 3x + 2) - (x^4 + 2)(2x^2 - 1)\\ &= (x^3 + x^2 + 2)(4x^3 + 8x^2 + 6x + 4) + (x^4 + 2)(4x^2 - 2)\\ &= (x^3 + x^2 + 2)(x^3 + 2x^2 + 1) + (x^4 + 2)(x^2 + 1).\end{aligned}$$

Vi har därmed hittat att $(x^3 + x^2 + 2)^{-1} = (x^3 + 2x^2 + 1)$ i R_3.

4 NTRU-kryptering

Innan vi förklarar hur NTRU-kryptering fungerar behöver vi gå igenom en sista sak.
Definition 6: För två positiva heltal d_1 och d_2, låter vi

$$\mathcal{T}(d_1, d_2) = \left\{ a(x) \in R : \begin{array}{c} a(x) \text{ ha } d_1 \text{ koefficienter som är lika med } 1, \\ a(x) \text{ ha } d_2 \text{ koefficienter som är lika med } -1, \\ a(x) \text{ ha resten av sina koefficienter lika med } 0 \end{array} \right\}.$$

Det här behövs för att kunna försäkra att krypteringen fungerar utan att felberäkningar sker i dekrypteringen.

Vi låter personen som ska skicka den offentliga nyckeln och ta emot det krypterade meddelandet heta Amanda och personen som skickar det krypterade meddelandet heta Bob. Amanda kommer att skicka ut de offentliga siffrorna (N, p, q, d) där $1 \leq N$, N är ett primtal, $SGD(N, q) = SGD(p, q) = 1$ och uträkningarna av polynomen kommer att ske i ringarna

$$R = \frac{\mathbb{Z}[x]}{x^N - 1} \qquad R_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{x^N - 1} \qquad R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{x^N - 1}.$$

Amanda börjar krypteringen med att skapa den offentliga nyckeln genom att först komma på två slumpmässiga polynom

$$f(x) \in \mathcal{T}(d+1, d) \qquad \text{och} \qquad g(x) \in \mathcal{T}(d, d).$$

Anledningen till att $f(x)$ befinner sig i $\mathcal{T}(d+1, d)$ istället för $\mathcal{T}(d, d)$ är för att polynom i $\mathcal{T}(d, d)$ inte har en invers vilket $f(x)$ behöver för nästa steg. Anledningen till att $\mathcal{T}(d, d)$ inte har en invers är för att polynomet har alltid jämnt antal och lika många positiva och negativa ettor vilket alltid kommer att ge minst resten $x-1$ när man utför Euklides algoritm enligt (2). Amandas nästa steg är att räkna ut

$$F_q(x) = f^{-1}(x) \in R_q \qquad \text{och} \qquad F_p(x) = f(x)^{-1} \in R_p.$$

Ifall inversen av $f(x)$ inte existerar i R_q eller R_p behöver Amanda hitta på ett nytt polynom. Sedan skapar Amanda den offentliga nyckeln som ett polynom vi döper till $h(x)$ på följande sätt

$$h(x) = F_q(x) \star g(x) \in R_q.$$

Nu när Bob har tillgång till den offentliga nyckeln kan han kryptera sitt hemliga meddelande $m(x)$. Enda kravet är att koefficienterna i $m(x) \in R_p$ är centrerade eller med andra ord, ligger mellan $-\frac{1}{2}p < m_i < \frac{1}{2}p$. Bob väljer sedan ut ett nytt slumpmässigt polynom $r(x) \in \mathcal{T}(d, d)$. Nu kan Bob kryptera sitt meddelande i ringen R_q på följande sätt

$$e(x) \equiv p \times h(x) \star r(x) + m(x) \pmod{q}$$

och skicka $e(x)$ till Amanda. Amanda börjar uträkningen genom att räkna ut $a(x) \equiv f(x) \star e(x) \pmod{q}$. Hela uträkningen, med hjälp av att Amanda vet att $h(x) = F_q(x) \star g(x)$ och att $f(x) \times F_q(x) = 1$ ger oss

$$\begin{aligned} a(x) &\equiv f(x) \star e(x) && \pmod{q} \\ &\equiv f(x) \star (p \times h(x) \star r(x) + m(x)) && \pmod{q} \\ &\equiv p \times f(x)F_q(x) \star g(x) \star r(x) + f(x) \star m(x) && \pmod{q} \\ &\equiv p \times g(x) \star r(x) + f(x) \star m(x) && \pmod{q}. \end{aligned} \tag{3}$$

Nu behöver Amanda endast räkna ut $b(x) \equiv F_p(x) \star a(x) \in R_p$. Till skillnad från tidigare räknar Amanda ut $b(x)$ i R_p vilket låter henne använda sig av modulo p. Uträkning ger Amanda

$$\begin{aligned} b(x) &\equiv F_p(x) \star a(x) && \pmod{p} \\ &\equiv F_p(x) \star (p \times g(x) \star r(x) + f(x) \star m(x)) && \pmod{p} \\ &\equiv F_p(x) \star f(x) \star m(x) && \pmod{p} \\ &\equiv m(x) && \pmod{p} \end{aligned} \tag{4}$$

vilket är Bobs meddelande. Om vi observerar polynomet $p \times g(x) \star r(x) + f(x) \star m(x)$ kan vi se att både $g(x) \star r(x) \in \mathcal{T}(d,d)$, $f(x) \star \in \mathcal{T}(d+1,d)$ och $m(x) \in R_p$ vilket är vart $f(x)$ existerar vilket leder till $m(x) \in \mathcal{T}(d+1,d)$. Vi kan hitta den största möjliga koefficienten i $g(x) \star r(x)$ och $f(x) \star m(x)$ på följande sätt:
Eftersom $g(x), r(x) \in \mathcal{T}(d,d)$ kommer den största möjliga koefficienten som kan ske vid multiplikationen, ske när $g(x)$ och $r(x)$ har matchande $1, -1$ och är därför samma polynom. När $g(x) = r(x)$ får vi $g(x) \star r(x) = g(x)^2$ som består av $2d$ positiva ettor med värden mellan x^0 och x^{N-1}. När ett sådant polynom kvadreras kommer den största möjliga koefficienten vara lika stor som antalet termer vilket i detta fall är $2d$.
I det andra fallet behöver man komma ihåg att $-\frac{1}{2}p < m(x) < \frac{1}{2}p$ vilket innebär att den största möjliga koefficienten för $f(x) \star m(x)$ är $(2d+1)\frac{1}{2}p$. Den största möjliga koefficienten för hela polynomet kommer alltså vara

$$p \times 2d + (2d+1)\frac{1}{2}p = \left(3d + \frac{1}{2}\right)p = (6d+1)p.$$

I uträkningen av krypteringen finns det en möjlighet att när Amanda går från steg (3) till (4) i uträkningen, kommer p i $a(x)$ vara större än q och kommer därför bli reducerad med q i sista raden på (3). Detta innebär att när Amanda räknar ut steg (4) kommer inte $F_p(x) \star (p \times g(x) \star r(x))$ förvinna när det reduceras med mod p. För att undvika detta bör man välja ett q som uppfyller $q > (6d+1)p$. Det är viktigt att notera att fastän man väljer ett q som inte uppfyller $q > (6d+1)p$ är det fortfarande möjligt att läsa meddelandet då det är väldigt lite chans att alla 1 och -1 matchar för den största möjliga koefficienten.

Exempel 8: Hela krypteringen och dekrypteringen kan till exempel ske på följande sätt. Amanda börjar med att lägga ut de offentliga siffrorna $(N, p, q, d) = (7, 5, 67, 2)$ som uppfyller alla krav och där

$$q = 67 > (6d+1)p = 65.$$

Amanda väljer härnäst sina två funktioner

$$f(x) = -x^5 + x^4 - x^3 + x + 1 \in \mathcal{T}(3,2), \qquad g(x) = x^6 - x^2 + x - 1 \in \mathcal{T}(2,2),$$

och inverserna till $f(x)$

$$F_p(x) = 3x^6 + 4x^5 + 3x^4 + x^3 + 3x^2 + 4x + 3 \in R_p$$
$$F_q(x) = 57x^6 + 62x^5 + 36x^4 + 26x^3 + 36x^2 + 62x + 57 \in R_q.$$

Den offentliga nyckeln räknar Amanda ut på följande sätt

$$h(x) = F_q(x) \star g(x) = 26x^6 + 5x^5 + 16x^4 + 51x^3 + 62x^2 + 41x \in R_q.$$

Nu väntar Amanda medan Bob skickar sitt meddelande genom skapa och räkna ut

$$m(x) = 2x^5 - 2x^4 - x^3 + x^2 + 1 \in R_p$$

$$r(x) = x^5 - x^4 - x^2 + x \in \mathcal{T}(2,2)$$

$$\begin{aligned} e(x) &\equiv p \times r(x) \star h(x) + m(x) \\ &\equiv 41x^6 + 24x^5 + 6x^4 + 66x^3 + 60x^2 + 45x + 27. \end{aligned}$$

När Amanda tar emot $e(x)$ från Bob kan hon räkna ut

$$\begin{aligned} a(x) &\equiv f(x) \star e(x) \\ &\equiv 14x^6 + 55x^5 + 3x^4 + 49x^3 + 15x^2 + 55x + 1 \\ &\equiv 14x^6 - 12x^5 + 3x^4 - 18x^3 + 15x^2 - 12x + 1 \in R_q. \end{aligned}$$

Till sist räknar Amanda ut sista steget i dekrypteringen och får

$$\begin{aligned} b(x) &\equiv F_p(x) \star a(x) \\ &\equiv 2x^5 + 3x^4 + 4x^3 + x^2 + 1 \\ &\equiv 2x^5 - 2x^4 - x^3 + x^2 + 1 \in R_p. \end{aligned}$$

Meddelandet som Bob skickar kommer att vara ett polynom. För att göra om polynomet till ord behöver Amanda och Bob komma överens sedan innan hur polynomet ska kodas genom att skapa ett lexikon där man kan översätta polynom till tecken som till exempel via ASCII. Då meddelandet $m(x)$ är centrerad i R_p finns det flera sätt att översätta polynomet till tecken. Ifall Bob behöver skicka ett längre meddelande kommer han behöva skicka flera separata till Amanda.

5 Dekryptering utan de hemliga polynomen

Ifall en tredje person, Cecilia, vill ta del av hemligheterna som Bob skickar behöver hon ta reda på Amandas hemliga polynom, $f(x)$ och $g(x)$. För att göra detta behöver man kolla på $h(x)$ som kan användas för att få

$$f(x) \star h(x) \equiv g(x) \mod q.$$

Då vi vet att $f(x)$ och $g(x)$ är teritiära polynom kan vi lättare hitta polynom som uppfyller ovanstående ekvation. Med den här metoden är det möjligt att hitta flera lösningar då $(x^k \star f(x), x^k \star g(x))$ också ger korrekt lösning för alla $0 \leq k < N$. Med dessa lösningar kommer man få lösningen $x^k \star m(x)$. Man säger att $x^k \star f(x)$ är en rotation av $f(x)$. Antal försök det kommer ta för Cecilia ifall hon ska tvinga fram nyckeln genom att testa alla möjliga teritiära polynom går att räkna ut genom att kolla på $\mathcal{T}(d_1, d_2)$. Vi låter koefficienterna i d_1 ha värdet 1 och låter resterande värden, $N - d_1$, av polynomet ha d_2 koefficienter med värdet -1. Vi kan därmed räkna ut antal permutationer Cecilia kan tvinga fram med

$$\binom{N}{d_1}\binom{N-d_1}{d_2} = \frac{N!}{d_1!(N-d_1)!} \times \frac{(N-d_1)!}{d_2!(N-d_1-d_2)!} = \frac{N!}{d_1!d_2!(N-d_1-d_2)!}.$$

För att d_1 och d_2 ska göra dekrypteringen så svår som möjligt, med flest antal möjliga permutationer, behöver d_1 och d_2 vara cirka $N/3$. Ifall Cecilia skulle försöka läsa Bobs meddelanden till Amanda från det förra kapitlet, skulle hon därför behöva testa

$$\binom{7}{3}\binom{7-3}{2} = \frac{7!}{3!(7-3)!} \times \frac{(7-3)!}{2!(7-3-2)!} = \frac{7!}{3!2!(7-3-2)!} = \frac{7!}{3!2!2!} = 210.$$

Fastän exemplet har väldigt små värden på (N, p, q, d) kommer det ta en hel del försök för Cecilia att tvinga fram svaret. I Cecilias beräkningar inkluderar hon inte rotationer då $x^k \star m(x)$ har sina egna permutationer som behöver beräknas. Rotationer används som bäst med hjälp av kvantdatorer som kan snabbt beräkna alla möjliga permutationer.

5.1 Dekryptering via CVP

Ett annat sätt att få fram hemliga nyckeln är genom att lösa CVP i gittret L_h^{NTRU} som är kopplat till

$$M_h^{NTRU} = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right)$$

som har bredd och längd $2N$. Matrisen består av 4 stycken, mindre mindre matriser med storleken $N \times N$ där:
Första matrisen är identitetsmatrisen,
Andra matrisen är cykliska permutationer av $h(x)$,
Tredje matrisen är nollmatrisen och
Fjärde matrisen är identitetsmatrisen multiplicerad med q.

Vi kan använda M_h^{NTRU} för att identifiera polynomen $f(x)$ och $g(x)$ i R som en $2N$-dimensionell vektor

$$(f, g) = (f_0, f_1, \ldots, f_{N-1}, g_0, g_1, \ldots, g_{N-1}) \in \mathbb{Z}^{2N}.$$

Lemma 3: Eftersom $f(x) \star h(x) \equiv g(x) \mod q$, låt $u(x) \in R$ vara polynomet som tillfredsställer

$$f(x) \star h(x) = g(x) + qu(x). \tag{5}$$

Då får man

$$(f, -u)M_h^{NTRU} = (f, g), \tag{6}$$

vilket innebär att (f, g) är i gittret L_h^{NTRU}.

Bevis 2: När $(f, -u)$ multipliceras med första och tredje matrisen i M_h^{NTRU} får man tydligt tillbaka f då f multipliceras med identitetsmatrisen och $-u$ multipliceras med nollmatrisen. Multiplikationen med första kolumnen i andra och fjärde kvadranten kommer att ge

$$h_0 f_0 + h_{N-1} f_1 + \cdots + h_1 f_{N-1} - q u_0,$$

vilket är en av möjligheterna som kan ges av $f(x) \star h(x) - qu(x)$. Enligt (5) kommer detta vara en möjlig kandidat för g(x) så den andra och fjärde raden kommer därför att ge vektorn g. Enligt lemma 3 kommer (6) ge oss (f, g) genom att använda sig av en specifik linjär kombination av raderna i M_h^{NTRU} vilket innebär att $(f, g) \in L_h^{NTRU}$. □

Genom att skriva M_h^{NTRU} som en 2×2 matris kan man skriva beviset på följande sätt

$$(f, -u) \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} = (f, f \star h - qu) = (f, g).$$

För att visa att (f, g) troligen är den kortaste vektorn i L_h^{NTRU} kan man jämföra den förväntade kortaste vektorn med den Gaussiska Heuristiken.
Definition 7: Låt $\mathcal{G}$ vara ett gitter av dimensionen n. Den Gaussiska förväntade vektorns längd är

$$\sigma(\mathcal{G}) = \sqrt{\frac{n}{2\pi e}} (\det \mathcal{G})^{\frac{1}{n}}.$$

Den Gaussiska Heuristiken säger att den kortaste vektorn kommer tillfredställa

$$||v_{\text{kortaste}}|| \approx \sigma(\mathcal{G}).$$

Förslag 3: Låt (N, p, q, d) vara parametrarna för NTRU-krypteringen. Anta att $p = 5$ för enkelhetens skull samt låt $d \approx N/3$ och $q \approx 6pd \approx 2pN$ vara så stora som möjligt för att att krypteringen ska säkert fungera.
Eftersom både f och g har $d + 1$ koordinater som är lika med 1 och d koordinater som är lika med -1 får vi fram det ungefärliga värdet på avståndet till origo

$$||(f, g)|| \approx \sqrt{4d} \approx \sqrt{4\frac{N}{3}} \approx 1.55\sqrt{N}.$$

Samtidigt, eftersom L_h^{NTRU} har längden $2N$ och $det(L_h^{NTRU}) = q^N$ så får vi enligt den Gaussiska Heuristiken

$$\sigma(L_h^{NTRU}) \approx \sqrt{\frac{2N}{2\pi e}} (\det L)^{1/2N} \approx \sqrt{N\frac{q}{\pi e}} \approx 1.082N.$$

Vi kan även se att

$$\frac{||(f, g)||}{\sigma(L)} \approx \frac{1.43}{\sqrt{N}}.$$

Man kan nu se att om N är tillräckligt stort så är det stor sannolikhet att de kortaste vektorerna i L_h^{NTRU} är (f, g). För att hitta SVP i L_h^{NTRU} används LLL-algoritmen som klarar av det genom att använda sig av 2^N bits. När $250 < N < 1000$ är säkerhetsnivån av krypteringen ungefär lika säker som RSA. Nackdelarna med att använda sig av LLL-algoritmen är att svårighetsnivån ökar exponentiellt när N ökar i värde.

5.2 Attack på NTRU-nyckeln: Möta upp i mitten

5.2.1 Metod

En annan metod för att knäcka NTRU-krypteringen är "Möta upp i mitten"-metoden som har använts för att knäck andra krypteringar. Metoden går brett beskrivet ut på att ta två nycklar från olika delar av krypteringen för att sedan delvis lösa båda krypteringarna för att sedan kunna hitta nyckeln som fungerar för hela krypteringen. Den här algoritmen fokuserar på en äldre version av NTRU som använder binära polynom istället för ternära polynom, men för enkelhetens skull diskuteras denna metod som vanligt då det enda som påverkas är tiden för att utföra algoritmen. Algoritmen går ut på att dela upp $f(x)$ i två delar, f_1 och f_2. Båda kommer att ha längden $N/2$ och $d/2$ stycken 1 blandade med -1. Fastän hälften av alla ettor inte hamnar i f_1 eller f_2 så vet vi att minst en rotation av f har hälften av sina ettor i f_1 och f_2. För att visa uppdelningen av $f(x)$ används "||" och används genom

$$\begin{aligned} f \times h &= g \pmod{\text{q}} \\ \Rightarrow (f_1||f_2) \times h &= g \pmod{q} \\ \Rightarrow f_1 \times h &= g - f_2 \times h \pmod{q} \\ \Rightarrow (f_1 \times h)_i &= \{0,1\} - (f_2 \times h)_i \pmod{q} \forall i, \end{aligned}$$

där a_i betecknar det i:te numret av a och $\{0,1\}$ är innehållet i den unika "lådan" som f_1 och f_2 integrerar med och även en möjlig kandidat för g. Algoritmen har två steg:

Steg 1: Första steget går ut på att sortera alla möjliga f_1 till sin låda. Lådorna som används är tidigare skapade av datorn som kör algoritmen. Dessa lådor har storleken N och innehåller endast de binära talen 0 och 1 och har alla möjliga binära tal upp till 2^k där k är ett heltal som är tillräckligt stort för att alla f_1 ska ha plats men inte större än N. Det förväntas att $\binom{N/2}{d/2}$ lådor kommer att användas men det är inte garanterat att endast så många lådor kommer fyllas, alltså bör k vara större än $\binom{N/2}{d/2}$, men ett högre värde på k innebär även mer datorminne som används. Sorteringen av f_1 går till genom att räkna ut $f_1 \times h$, sedan kommer koeffienterna i f_1 få värdet 1 ifall koefficienten är $\geq q/2$ eller värdet 0 ifall den är $< q/2$. Till exempel, om $(f_1 \times h) = 12x^7 + 4x^6 + 9x^5 - 7x^4 + 3x^3 + x^2 - 14x + 9$ och $q/2 = 8$ kommer koefficienten för x^7 få värdet 1, koefficienten för x^6 få värdet 0 och koefficienten för x^4 få värdet 0. Sedan kommer f_1 matchas med lådan $[1,0,1,0,0,0,1,1]$ för att sparas där. Precis som att det är möjligt att vissa av lådorna inte kommer användas, kommer några av lådorna få flera f_1 i sig.

Steg 2: Andra steget börjar med att förbereda alla möjliga binära kombinationer av f_2 och räkna ut $f_2 \times h$. Matchingen med lådorna kommer ske liknande som för f_1 förutom att vi beräknar $-(f_2 \times h)$ mod q för att hitta rätt låda. Inte nog med att f_2 matchas med lådan som koefficienterna anger, ifall man adderar 1 till koefficienten och den får ett annat värde kommer även f_2 att matchas med den nya lådan. Till exempel, ifall $f_2 = 4x^3 + x^2 - 2x$, $N = 4$ och $q = 5$ kommer f_2 testas med alla f_1 i lådorna $[1,0,0,0], [1,0,1,0], [0,0,1,0]$ och $[0,0,0,0]$. När f_2 matchas med en låda kommer f_1 och f_2 sättas ihop till f igen så att ekvationen $(f_1 \times h)_i = \{0,1\} - (f_2 \times h)_i \pmod{q} \forall i$ kan lösas. Ifall lösningen är ternär med endast $0, 1$ eller -1 som värden, är den ternära funktionen g och f är då en fungerande nyckel ifall allt har gått rätt till. Ifall lösningen inte är ternär sätts f_2 ihop med nästa (ifall det finns) f_1 för att testa ekvationen.

Exempel 9: För att göra det lätt kommer exemplet visa hur algoritmen skulle hitta f i exempel 8. Algoritmen kommer ha de publika siffrorna $(N, p, q, d) = (7, 5, 67, 2)$ och den publika nyckeln $h = 26x^6 + 5x^5 + 16x^4 + 51x^3 + 62x^2 + 41x$. Första steget är att skapa lådorna som möjligen kommer att användas. Eftersom $N = 7$ är det rimligt att anta att $f_1 \times h$ kommer behöva en låda av storleken 7, därför skapas alla möjliga lådor med 7 bits

$$[0,0,0,0,0,0,0], \ldots [1,0,1,0,1,0,1], \ldots [1,1,1,1,1,1,1].$$

Nästa steg är att skapa f_1. Då $N = 7$ kommer f_1 ha 4 koefficienter med värderna 0, 1 och -1. Algoritmen radar upp alla möjliga versioner av f_1 med tanke på att f_2 har tre värden som kompleterar f_1. Ett möjligt värde på f_1 är

$$f_{i1} = x^6 - x^5 + x^4 - x^3.$$

För att göra exemplet tydligt använder vi även

$$f_{j1} \times h = 26x^6 + 12x^5 + 46x^4 + 57x^3 + 51x^2 + 6$$

och värdet vi vet stämmer via exempel 8,

$$f_{k1} = 0 - x^5 + x^4 - x^3.$$

Sista delen av steg ett är att räkna ut $(f_1 \times h)$, göra om till binärt tal och spara i rätt låda. Med $q = 33.5$ får vi

$$\begin{aligned}
f_{i1} \times h =& 37x^6 + 5x^5 + 5x^4 + 37x^3 + 14x^2 + 22x + 14 \Rightarrow [1,0,0,1,0,0,0],\\
f_{j1} \times h =& 26x^6 + 12x^5 + 46x^4 + 57x^3 + 51x^2 + 6 \Rightarrow [0,0,1,1,1,0,0],\\
f_{k1} \times h =& 37x^6 + 46x^5 + 21x^3 + 30x^2 + 27x + 40 \Rightarrow [1,1,0,0,0,0,1].
\end{aligned}$$

På samma sätt kommer alla versioner av f_2 skapas förutom att det kommer vara max 3 koefficienter. Vi använder oss av $f_{j2} = x^2 - x + 1$ och $f_{k2} = x + 1$ som vi vet kommer ge rätt. Vi vet att $q/2 = 33.5$ vilket ger

$$\begin{aligned}
-(f_{j2} \times h) =& -(37x^6 + 40x^5 + 27x^4 + 30x^3 + 21x^2 + 46) \pmod{67}\\
=& 30x^6 + 27x^5 + 40x^4 + 37x^3 + 46x^2 + 21 \Rightarrow [0,0,1,1,1,0,0],\\
-(f_{k2} \times h) =& -(31x^6 + 21x^5 + 46x^3 + 36x^2 + 41x + 26) \pmod{67}\\
=& 36x^6 + 46x^5 + 21x^3 + 31x^2 + 26x + 41 \Rightarrow [1,1,0,0,0,0,1].
\end{aligned}$$

För exemplets skull använder vi även följande f_2,

$$f_{i2} \times h = 66x^6 + 58x^5 + 6x^4 + 37x^3 + 15x^2 + 3x + 6 \Rightarrow [1,1,0,1,1,0,0],$$

som inte går att få med just detta h och så pass litet N. En efter en kommer f_2 matchas med lådorna. Först kan f_{j2} testas med alla f_1 i lådan [0,0,1,1,1,0,0] och sedan kan f_{i2} testas

med alla f_1 i lådorna

$$[1,1,0,1,1,0,0],[1,1,0,1,1,0,1],[1,1,1,1,1,0,0],[1,1,1,1,1,0,1],$$
$$[0,1,0,1,1,0,0],[0,1,0,1,1,0,1],[0,1,1,1,1,0,0],[0,1,1,1,1,0,1],$$

och f_{k2} testas med alla f_1 i lådan [1,1,0,0,0,0,1]. För att lösa ekvationen räcker det med att addera $(f_2 \times h)$ på båda sidorna

$$(f_{j1} \times h) = \{0,1\} - (f_{j2} \times h) \bmod q$$
$$(56x^6 + 47x^5 + 46x^4 + 57x^3 + 51x + 6) = \{0,1\} - (37x^6 + 40x^5 + 27x^4 + 30x^3 + 21x^2 + 46) \bmod q$$
$$63x^6 + 52x^5 + 6x^4 + 20x^3 + 5x^2 + 52 = \{0,1\} \bmod q,$$

vilket inte är en fungerande nyckel då polynomet inte vart ternärt. Testar vi med polynomen vi vet är nycklar får vi

$$(f_{k1} \times h) = \{0,1\} - (f_{k2} \times h) \bmod q$$
$$(37x^6 + 46x^5 + 21x^3 + 30x^2 + 27x + 40) = \{0,1\} - (31x^6 + 21x^5 + 46x^3 + 36x^2 + 41x + 26) \bmod q$$
$$x^6 + 66x^2 + x + 66 = \{0,1\} \bmod q$$
$$x^6 - x^2 + x - 1 = \{0,1\} \bmod q.$$

Då svaret är ternärt är $(f_{k1}||f_{k2}) = -x^5 + x^4 - x^3 + x + 1$ en nyckel till krypteringen.

5.2.2 Tid och datorminne

För att räkna ut den totala tiden för att använda sig av "Möta upp i mitten"-metoden låter vi τ_c vara tiden för att räkna ut $f_1 \times h \pmod q$. Vi låter även τ_l vara tiden för att leta upp låda i och integrera med lådan. Alltså kommer den första delen av uträkningen, då f_1 placeras i lådor, ta max

$$\tau_1 = \binom{N/2}{d/2}(\tau_c + \tau_l).$$

Den andra delen av uträkningen tar lite längre tid. Först behöver $f_2 \times h$ beräknas vilket tar τ_c. Sedan adderas τ_l för varje förväntade låda som f_2 kommer testas mot vilket är $2k/q$ antal. Sist behöver man addera τ_c för alla förväntade träffar mellan f_1 och f_2 vilket är $\binom{N/2}{d/2}/2^k$. Den totala tiden blir

$$\tau_2 = \binom{N/2}{d/2}\left(\tau_c + \frac{2k}{q}\tau_l + \frac{\binom{N/2}{d/2}}{2^k}\tau_c\right).$$

Då k är kopplat till antal lådor, går det att öka mängden lådor för att minska tiden det tar att utföra beräkningen men detta kommer resultera i ökad använding av dataminne. Samtidigt, för att öka effektiviteten, är det viktigt att inte använda för mycket minne då det tar tid att både skapa och integrera med lådorna. Minnet som behövs kommer vara antalet lådor samt minnet för att förvara f_1 i lådorna vilket inte kommer öka exponentiellt.
Det går att optimera beräkningen och det använda minnet ytterligare då den ovanstående beräkningen för f_1 och f_2 letar endast efter nyckeln som har hälften av ettorna i f_1 och resten i f_2. Detta inkluderar inte alla möjliga rotationer när det finns approximativt $\sqrt{N}$ rotationer

som fungerar som dekrypteringsnyckel. Optimeringen går ut på att ändra algoritmen så att istället för att hantera f_1 och f_2 var för sig, hanteras de samtidigt. Till skillnad från tidigare förvaras varje f_1 i var sin låda och samtidigt förvaras f_2 i approximativt $2N/q$ lådor. Om det finns r rotationer som fungerar, förväntas det att hitta en rotation som fungerar efter att algoritmen har räknat igenom $1/\sqrt{r}$ av alla f_1 och f_2 som är i samma låda och har en del av en möjlig rotation. Den förväntade tiden blir då summan av τ_c för att räkna ut f_2, τ_l för att förvara f_1, f_2 och τ_c för att beräkna kombinationerna av f_1 och f_2 på följande sätt

$$\tau_2 \approx \frac{\binom{N/2}{d/2}}{\sqrt{r}} \left(\tau_c + \left(1 + \frac{2N}{q} \right) \right) \tau_l + \frac{\tau_c}{2^k} \sum_i (\text{Träffar})_i.$$

Om man nu väljer ett k som är tillräckligt stort kommer antalet felberäkningar (tillfällen då f_1 och f_2 inte är en nyckel) att minska så pass mycket att att den andra termen i princip kan ignoreras. Unika beräkningar av τ_c och τ_l tar inte lång tid. Av multiplikationen vi har kvar är det alltså

$$\frac{\binom{N/2}{d/2}}{\sqrt{r}}$$

som har det större värdet och kommer bestämma tiden för att algoritmen ska hitta nyckeln. Värdet av r är svår att förutspå men r kommer aldrig att vara större än N vilket ger oss den längsta möjliga tiden

$$\frac{\binom{N/2}{d/2}}{\sqrt{N}}.$$

6 Referenser

Hoffstein. J, Pipher. J, Silverman. J (2014) - An Introduction to Mathematical Cryptography, Second Edition. Kapitel 3-3.2, 7-7.13 och 8.11.

Howgrave-Gaham. N, Silverman. J, Whyte. W. NTRU Cryptosystems Technical Report. Report #004, Version 2: A Meet-In-The-Middle Attack on an NTRU Private Key. Kapitel 1. https://ntru.org/f/tr/tr004v2.pdf

Quantum computing hype vs reality, IBM https://www.ibm.com/topics/quantum-computing. 2022-07-13

Bild - Figure 1 tagen från: Kryptologins historia - https://fr.wikipedia.org/wiki/Fichier:Caesar3.svg 2022-06-16