Debreceni Egyetem
Természettudományi és Technológiai Kar
Matematikai Intézet

Szakdolgozat

Titkosítási eljárások

készítette:

Molnár Alexandra

témavezető: Dr. Tengely Szabolcs

Debrecen, 2020

Tartalomjegyzék

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Tengely Szabolcsnak, kinek segítsége nélkül nem jöhetett volna létre ezen dolgozat.
Szeretném megköszönni családomnak, hogy tanulmányaim során mellettem voltak és támogattak, valamint páromnak, Gábor Sándornak, hogy mindig átsegített a nehézségeken, bíztatott, motivált.
Végül köszönöm tanáraimnak és csoporttársaimnak, hogy segítséget nyújtottak az egyetemi éveim alatt.

1. Bevezető

A nyilvános kulcsú kriptográfiai algoritmus radikális változásokat hozott a titkosítási módszerek tekintetében [6]. A hagyományos egykulcsos titkosítással szemben, mely során ugyanazt a kulcsot használjuk a kódolás és a dekódolás során is, a nyílvános kulcsú algoritmusok matematikai függvényeken alapulnak, és asszimetrikusak. Két kulcs használatával járnak, egy a titkosításhoz és egy tőle különböző a visszafejtéshez szükséges, továbbá az asszimetrikus titkosításra teljesül az a fontos tulajdonság, miszerint számítási szempontból megvalósíthatatlan a visszafejtő kulcs kiszámítása csak az algoritmus és a publikus kulcs ismeretében. Néhány ilyen algoritmus esetén a titkosító és visszafejtő kulcs fel is cserélhető, pl.: RSA.

A titkosítás lépései:

1. Minden rendszer generál egy kulcspárt.
2. Minden rendszer közzé teszi titkosítási kulcsát, ezt nevezzük publikus kulcsnak, illetve titokban tartja annak párját, amit privát kulcsnak hívunk.
3. Ha A üzenetet szeretne küldeni B-nek, akkor B publikus kulcsával titkosítja az üzenetét.
4. Amikor B megkapja az üznetet, a titkos kulcsa segítségével dekódolja az üzenetet. Más nem tudja visszafejteni az üzenetet, mert csak B ismeri a privát kulcsot.

Az algoritmus követelményei:

Bármely nyilvános kulcsú titkosítási algoritmusra teljesülnek a következő követelmények, melyeket Whitfield Diffie és Martin Hellman fogalmaztak meg:

- B-nek a kulcspár (publikus(KU) és privát(KR)) generálása könnyen számolható.
- A-nak a publikus kulcs (KU_b) tudatában könnyen kiszámítható a titkosított szöveg és így el tudja küldeni a megfelelő kriptoszöveget. $C = E_{KU_b}(M)$, E a titkosító függvény, M pedig az üzenet.

- B-nek szintén könnyen számítható a privát kulcsa (KR_b) segítségével a kriptoszövegből visszafejteni az eredeti szöveget.

$$M = D_{KR_b}(C) = D_{KR_b}[E_{KU_b}(M)],$$

  ahol D a dekódoló függvény.

- Egy külső személynek lényegében lehetetlen KU_b-ból KR_b-t kiszámolni.

- Csak KU_b és C tudatában lényegében lehetetlen visszafejteni M-et.

- A titkosítás és a visszafejtés bármilyen sorrendben elvégezhető:

$$M = E_{KU_b}[D_{KR_b}(M)] = D_{KR_b}[E_{KU_b}(M)].$$

A fenti "könnyen kiszámítható", "lényegében lehetetlen" fogalmak nincsenek matematikailag pontosan definiálva, mégis a gyakorlatban számos kriptorendszer biztonsága alapszik rajtuk.

Az NTRU az egyik leggyorsabb ma ismert, nyílvános kulcsú kriptográfiai algoritmus, melyet 1996-ban prezentált Jeffery Hoffstein, Jill Pipher és Joseph H. Silverman [3]. Biztonsága a legrövidebb vektor problémán alapszik. Ebben a dolgozatban az NTRU titkosítás és néhány változata kerül bemutatásra.
A szakdolgozat írása közben létrejött egy cikk is, melyet 2020 júniusában mutatott be Hayder Hashim a "20th Central European Conference on Cryptology" konferencián. Az előadás anyaga az alábbi linken elérhető: https://web.math.pmf.unizg.hr/cecc2020/files/hashim-molnar-tengely-talk.pdf.

2. NTRU

Először az alap NTRU felépítése és működése kerülne bemutatásra.

Legyen $R = \mathbb{Z}[x]/(x^n - 1)$ polinomgyűrű, melyben legfeljebb $(N-1)$-edfokú polinomok vannak. Az NTRU algoritmus műveleteit ebben a gyűrűben végezzük. Szükségünk van három egész paraméterre: N, p, q úgy, hogy $(p, q) = 1$. p-t kicsi, q-t pedig nagy modulusnak nevezzük. Legyenek továbbá L_m, L_f, L_g és $L_r \subseteq R$ olyan halmazok, melyekben kis együtthatós polinomok vannak.

Publikus kulcs generálása:

Véletlenszerűen vegyünk két polinomot, $f \in L_f$-et és $g \in L_g$-t. f-et úgy kell megválasztanunk, hogy legyen inverze $\pmod p$ és $\pmod q$. Modulo p úgy dolgozunk, hogy a szokásos $[0, p-1]$ intervallum helyett a $\left[\frac{-p+1}{2}, \frac{p-1}{2}\right]$ intervallumot használjuk.

$$F_p * f \equiv 1 \pmod p,$$

$$F_q * f \equiv 1 \pmod q.$$

Ebből tudjuk megadni a publikus kulcsot, h-t:

$$h \equiv pF_q * g \pmod q.$$

2.1. Kódolás

Az üzenetet egy olyan m polinommal reprezentáljuk, hogy $m \in L_m$ és redukáljuk m együtthatóit modulo p szerint. Választunk még egy véletlenszerű polinomot, úgynevezett maszatoló változót, $r \in L_r$-t. Ennek a segítségével kapjuk meg a titkosított üzenetet a következő módon:

$$e \equiv r * h + m \pmod q.$$

Példa SageMath-ban:

```
sage: def cmap(t,p):
....:     if (ZZ(t)%p)>(p//2):
....:         return ((ZZ(t)%p)-p)
....:     else:
....:         return ZZ(t)%p
sage: Zx.<X>=ZZ[]
sage: N=7
sage: p=3
sage: q=41
sage: f=X^6+X^5-X^3-X^2-X-1
sage: g=X^6+X^4-X^3+1
sage: Pp.<b>=PolynomialRing(GF(p))
sage: Pq.<c>=PolynomialRing(GF(q))
sage: fp=Pp(f).inverse_mod(b^N-1)
sage: fq=Pq(f).inverse_mod(c^N-1)
sage: h=(p*fq*Pq(g))%(c^N-1)
sage: "publikus␣kulcs:", h
('publikus␣kulcs:', 25*c^6 + 6*c^5 + 40*c^4 + 17*c^3 +
   23*c^2 + 30*c + 20)
sage: r=X^6-X^5+X-1
sage: m=-X^5+X^3+X^2-X+1
sage: em=(Pq(r)*h)%(c^N-1)+Pq(m)%(c^N-1)
sage: "kodolt␣uzenet:", em
('kodolt␣uzenet:', 12*c^6 + 38*c^5 + 40*c^4 + 26*c^2 +
   36*c + 13)
```

2.2. Dekódolás

A kapott e titkosított üzenetet és az f publikus kulcsot összeszorozzuk, így kapjuk a-t.

$$\begin{aligned} a &\equiv f * e \pmod{q} \\ &\equiv f * (r * h + m) \pmod{q} \\ &\equiv f * h * r + f * m \pmod{q} \\ &\equiv pf * F_q * g * r + f * m \pmod{q} \\ &\equiv pg * r + f * m \pmod{q}. \end{aligned}$$

Végezetül az egyenletet modulo p vesszük:

$$a * Fp_{\equiv}(p * r * g + f * m) * F_p \equiv m \pmod{p}.$$

```
sage: A=(Pq(f)*em)%(c^N-1)                                          24
sage: A                                                             25
3*c^5 + 30*c^4 + 3*c^3 + 35*c^2 + 9*c                               26
sage: A1=[cmap(k,q) for k in A.list()]                              27
sage: Zx(A1)                                                        28
3*X^5 - 11*X^4 + 3*X^3 - 6*X^2 + 9*X                               29
sage: "dekodolt␣uzenet:", Zx([cmap(k,p) for k in Pp((Zx             30
    (A1)*Zx(fp))%(X^N-1)).list()])
('dekodolt␣uzenet:', -X^5 + X^3 + X^2 - X + 1)                      31
```

2.3. Törés LLL-algoritmussal

Ahogyan a bekezdésben már szó esett róla, az NTRU biztonsága a legrövidebb vektor problémán alapszik [1], ami a következő: egy tetszőleges Λ rácsban szeretnénk megtalálni a legrövidebb v vektort a helyvektorok között. Ebben az esetben $-v$ is a legrövidebb vektor lesz. A legfőbb elméleti eredményt ezzel kapcsolatban a Minkowski-tétel tartalmazza, amely a legrövidebb vektor felső határát adja meg.

Minkowski-tétel: Legyen Λ egy olyan rács, melynek rangja m. Ha λ a legrövidebb vektor normája, akkor

$$\lambda \leq \frac{2}{\sqrt{\pi}} \frac{m}{2}!^{\frac{1}{m}} \det\Lambda^{\frac{1}{m}}.$$

LLL-algoritmus:

Az LLL-redukciós algoritmus egy polinomiális idejű, rácsredukciós algoritmus, melyet Arjen Lenstra, Hendrik Lenstra és Lovász László találtak ki 1982-ben és utánuk is lett elnevezve [5]. Máig nem ismert hatékony, polinomiális idejű algoritmus, mely megoldaná a legrövidebb vektor problémát tetszőlegesen nagy dimenzióban. Az LLL-algoritmust viszont tudjuk alkalmazni a legrövidebb vektor közelítéséhez. Ez az algoritmus tulajdonképpen egy, a Gramm-Schmidt eljáráshoz hasonló módszerrel úgynevezett redukált (majdnem ortogonális) bázist szolgáltat.

Az $\mathbb{R}^n$ vektortértben egy $\{\mathbf{b_1}, \mathbf{b_2}, \cdots, \mathbf{b_m}\}$ bázist c-redukáltnak hívunk akkor és csak akkor, ha a $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \cdots, \mathbf{b_m^*}\}$ ortonormált bázisa teljesíti a következő egyenlőtlenséget $i = 1, \cdots, m-1$ esetén:

$$\|\mathbf{b_{i+1}^*}\|^2 \geq \frac{\|\mathbf{b_i^*}\|^2}{c}.$$

Jó redukciót jelent az, ha c értéke kicsi. Nem minden bázis 1-redukálható, de minden bázis $\frac{4}{3}$-redukálható.

Ha veszünk egy Λ rácsot és annak c-redukált $\{\mathbf{b_1}, \mathbf{b_2}, \cdots, \mathbf{b_m}\}$ bázisát úgy, hogy $c \leq \frac{4}{3}$, akkor a bázis közel orotogonális, abban az értelemben, hogy

$$\prod_{i=1}^{m} \|\mathbf{b_i}\| \leq c^{(m(m-1))/4} \det\Lambda.$$

Ha a c-redukált bázisunk $\{\mathbf{b_1}, \mathbf{b_2}, \cdots, \mathbf{b_m}\}$, m a rács rangja, λ pedig a legrövidebb vektor normája, akkor

$$\|\mathbf{b_1}\| \leq c^{(m-1)/4} \det\Lambda^{\frac{1}{m}},$$

$$\|\mathbf{b_1}\| \leq c^{(m-1)/2} \lambda.$$

Az LLL-algoritmus $c > \frac{4}{3}$-ra polinomiális időn belül megadja a rácsban a c-redukált bázist.

NTRU törése [8]:

A publikus kulcs kielégíti a

$$h \equiv g * pF_q \pmod{q}$$

ekvivalenciát, ezért írhatjuk, hogy $f * h \equiv g \pmod{q}$. Tekintsük a következőképpen definiált rácsot:

$$\Lambda = \{(F_1, F_2) \in R_q \times R_q : F_1 * h \equiv F_2 \pmod{q}\}.$$

Nyílvánvaló, hogy $(f, q) \in \Lambda$. Így az $f * h \equiv g \pmod{q}$ egyenlet helyett írható, hogy

$$f * h - u * q = g$$

néhány $u \in R_q$ esetén. Ez megegyezik azzal, hogy

$$\begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ h & q \end{pmatrix} \begin{pmatrix} f \\ -u \end{pmatrix}.$$

Vagy még hasznosabb alakban:

$$\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \\ g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ h_0 & h_1 & \cdots & h_{N-1} & q & 0 & \cdots & 0 \\ h_{N-1} & h_0 & \cdots & h_{N-2} & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_1 & h_2 & \cdots & h_0 & 0 & 0 & \cdots & q \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \\ -u_0 \\ -u_1 \\ \vdots \\ -u_{N-1} \end{pmatrix}.$$

Mivel az f és g polinomokban lévő együtthatók kicsik, így az (f, g) vektor rövid lesz a Λ rácsban. Alkalmazható az LLL-algoritmus.

A korábbi példa törése LLL-algoritmussal Sage-mathban:

```
sage: M=matrix(2*N)
....: for i in [0..N-1]: M[i,i]=1
....: for i in [N..2*N-1]: M[i,i]=q
....: for i in [0..N-1]:
....:     for j in [0..N-1]:
....:         M[i+N,j]=((Zx(GF(q)(1/p)*h)*X^i)%(X^N-1))
   [j]
sage: M
[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  1  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  1  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  1  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  1  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  1  0  0  0  0  0  0  0]
[34 10 35 33 27  2 22 41  0  0  0  0  0  0]
[22 34 10 35 33 27  2  0 41  0  0  0  0  0]
[ 2 22 34 10 35 33 27  0  0 41  0  0  0  0]
[27  2 22 34 10 35 33  0  0  0 41  0  0  0]
[33 27  2 22 34 10 35  0  0  0  0 41  0  0]
[35 33 27  2 22 34 10  0  0  0  0  0 41  0]
[10 35 33 27  2 22 34  0  0  0  0  0  0 41]
```

A kapott mátrixra tudjuk alkalmazni az LLL-algortitmust, hogy rövid vektorokat találjunk, és mivel f és g rövidek, így esélyes, hogy feltűnnek. A Sagemath beépített függvényként tartalmazza az LLL-t, így könnyen használható.

```
sage: f.coefficients(sparse=False)
```

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

```
sage: g.coefficients(sparse=False)
```

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

```
sage: M1=M.transpose().LLL()
sage: M1
```

$$\begin{pmatrix}
-1 & 1 & 1 & 0 & -1 & -1 & -1 & 1 & 1 & 0 & 1 & -1 & 0 & 0 \\
-1 & -1 & 0 & 1 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & -1 \\
0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 0 & 0 & 1 & 1 & 0 \\
-1 & 0 & 1 & -1 & -1 & 0 & -1 & 1 & 0 & 1 & 0 & 1 & -1 & 1 \\
0 & 1 & -1 & -1 & 0 & -1 & -1 & 0 & 1 & 0 & 1 & -1 & 1 & 1 \\
1 & 1 & 0 & -1 & -1 & -1 & -1 & 1 & 0 & 1 & -1 & 0 & 0 & 1 \\
-1 & 1 & 0 & 0 & 1 & 0 & -2 & 0 & 0 & 1 & 1 & -1 & 0 & 0 \\
-2 & -10 & 7 & -4 & 6 & -3 & 3 & 3 & -3 & 2 & -2 & -8 & 5 & 6 \\
-3 & 1 & 9 & -8 & 3 & -5 & 4 & -5 & -4 & 3 & -2 & 3 & 9 & -5 \\
3 & -2 & 1 & 9 & -7 & 3 & -7 & -5 & -5 & -3 & 4 & -3 & 3 & 9 \\
-6 & -6 & -2 & 3 & -4 & -6 & 1 & -5 & -10 & -1 & -5 & -8 & 0 & 8 \\
-3 & -3 & -6 & 4 & -9 & 6 & -10 & -5 & 3 & -8 & -6 & -3 & -1 & 0 \\
6 & -3 & 3 & -2 & -10 & 7 & -4 & -8 & 5 & 6 & 3 & -3 & 2 & -2 \\
10 & -7 & 4 & -6 & 3 & -3 & 2 & 3 & -2 & 2 & 8 & -5 & -6 & -3
\end{pmatrix}$$

Valóban, több sorban is megtalálható f-nek és g-nek valamilyen eltolt alakja. Ezek mindegyike jó vektor lehet a töréshez. Például, ha az első sort tekintjük, akkor f-ben az együtthatók listája [-1 1 1 0 -1 -1 -1].

```
sage: f=-X^6+X^5+X^4-X^2-X-1                                          57
sage: Zx.<X>=ZZ[]                                                     58
sage: N=7                                                             59
sage: p=3                                                             60
sage: q=41                                                            61
sage: Pp.<b>=PolynomialRing(GF(p))                                    62
sage: Pq.<c>=PolynomialRing(GF(q))                                    63
sage: fp=Pp(f).inverse_mod(b^N-1)                                     64
sage: fp                                                              65
```

$$b^6 + 2b^5 + b^4 + 2b^2 + 1$$

```
sage: em=12*c^6 + 38*c^5 + 40*c^4 + 26*c^2 + 36*c + 13                66
sage: A=(Pq(f)*em)%(c^N-1)                                            67
sage: A                                                               68
```

$$3c^4 + 30c^3 + 3c^2 + 35c + 9$$

```
sage: A1=[cmap(k,q) for k in A.list()]                                69
sage: Zx(A1)                                                          70
```

$$3X^4 - 11X^3 + 3X^2 - 6X + 9$$

```
sage: Zx([cmap(k,p) for k in Pp((Zx(A1)*Zx(fp))%(X^N-1)               71
   ).list()])
```

$$-X^5 + X^3 + X^2 - X + 1$$

Hasonlóan a harmadik és hatodik sor első hét tagjából képzett listák jó együtthatói lesznek az f polinomnak és sikeresen visszakaphatjuk segítségével az eredeti üzenetet. Ha viszont egy másik olyan sorból választunk, ami szintén csak a -1, 0, 1 számokat tartalmazza, akkor hamar elakad az algoritmus, mivel nem lesz f-nek inverze modulo p.

3. ITRU

Az NTRU ihlette ITRU [2] tirkosítási algoritmus nagyon hasonlít szerkezetében az előzőekben bemutatott NTRU-hoz, azzal a különbséggel, hogy műveletei a modulo n maradékosztály-gyűrűben helyezkednek el, amit $\mathbb{Z}/n\mathbb{Z}$-vel jelölünk. Ebben a gyűrűben az összeadás és a szorzás a hagyományos összeadás és szorzás, csak az eredményt modulo n vesszük.

Az ITRU paraméterei:
A paraméterek a $\mathbb{Z}/p\mathbb{Z}$ és $\mathbb{Z}/q\mathbb{Z}$ gyűrűkből vannak.

- p : kis modulus
- q : nagy modulus
- f : a privát kulcs generálásához szükséges privát szám
- g : a publikus kulcs generálásához szükséges random szám
- r : a kriptoszöveg generálásához szükséges random szám
- m : decimális reprezentása az üzenetnek
- K_{pr} : privát kulcspár, (f, F_p)
- K_{pb} : publikus kulcs, h
- a : közvetítő paraméter
- C : dekódolt üzenet

3.1. Paraméterek és kulcsok generálása

A kis modulust, p-t, 1000-nek választjuk és hozzá random választunk két egészet, g-t és r-t. f-et is random számként generáljuk úgy, hogy legyen multiplikatív inverze modulo p és modulo q, ahol $q > (p \cdot r \cdot g + f \cdot m)$ prím. Ezután meghatározzuk $f^{-1} \pmod{p}$-t, amit F_p-vel illetve $f^{-1} \pmod{q}$-t, amit F_q-val jelölünk. A kibővített euklideszi algoritmussal ez könnyen számolható.

A kapott inverzek teljesítik a következő feltételeket:

$$f \cdot F_p \equiv 1 \pmod{p},$$

$$f \cdot F_q \equiv 1 \pmod{q}.$$

A üzenet decimális alakját az ASCII-táblázat segítségével tudjuk megadni. A Sage-mathban ezek is rögzítve vannak. Egy karakter ASCII-kódját a "ord( )" paranccsal, a kódhoz tartozó karaktert pedig a "chr( )" paranccsal tudjuk előhívni.

A privát kulcspárt, a már korábban meghatározott f és F_p adja,

$$K_{pr} = (f, F_p).$$

A publikus kulcsot pedig a következőképpen számoljuk:

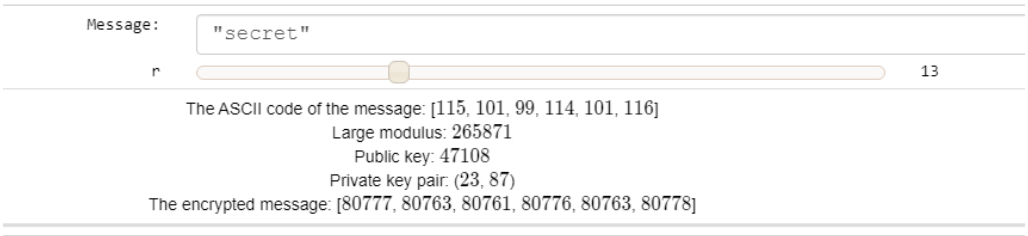
$$K_{pb} = h = (p \cdot F_q \cdot g) \pmod{q}.$$

3.2. Kódolás

A kódoláshoz, az NTRU-hoz hasonlóan, szükségünk van egy random paraméterre, amit r-rel jelölünk és itt egy egész szám. A kódolt üzenetet e-vel jelöljük és az alábbi módon számoljuk:

$$e = ((r \cdot h) + m) \pmod{q}.$$

A kódoló rész interaktív megoldása SageMath-ban:



Ennek programkódja:

```
sage: @interact
sage: def itru(s=input_box('"secret"',label = "Message:"),
r=slider(8,25,1,3, label="r")):
....:   p=1000
....:   F=Set([k for k in range(2,50) if gcd(k,1000)==1])
....:   f=F.random_element()
....:   S=Set([8..25])
....:   g=S.random_element()
....:   m=[ord(k) for k in s]
....:   pretty_print('The ASCII code of the message:',m)
....:   q=next_prime(p*r*g+255*f)
....:   Fp=(1/f)%p
....:   Fq=(1/f)%q
....:   h=(p*Fq*g)%q
....:   pretty_print('Large modulus:', q)
....:   pretty_print('Public key:',h)
....:   pretty_print('Private key pair:', (f,Fp))
....:   e=[((r*h)+m[i])%q for i in [0..len(m)-1]]
....:   pretty_print('The encrypted message:',e)
```

3.3. Dekódolás

Hogy megkapjuk az eredeti üzenetet, először a kapott, kódolt üzenetből kiszámoljuk a-t:

$$a = (f \cdot e) \pmod{q}.$$

Végül a segítségével megkapjuk az dekódolt üzenetet, m-et:

$$C = (F_p \cdot a) \pmod{p},$$

$$C = m.$$

Részletes számolás:

Tudjuk, hogy

$$e = ((r \cdot h) + m) \pmod{q}.$$

Ezt behelyettesítve az

$$a = (f \cdot e) \pmod{q}$$

egyenletbe kapjuk, hogy

$$\begin{aligned} a &= (f \cdot (r \cdot h + m)) \pmod{q} \\ &= (f \cdot r \cdot h + f \cdot m) \pmod{q}. \end{aligned}$$

Figyelembe véve, hogy

$$h = (p \cdot F_q \cdot g) \pmod{q},$$

írhatjuk, hogy

$$a = (f \cdot r \cdot p \cdot F_q \cdot g + f \cdot m) \pmod{q}.$$

Mivel

$$f \cdot F_q \pmod{q} \equiv 1,$$

így

$$a = (r \cdot p \cdot g + f \cdot m) \pmod{q}.$$

q-ról tudjuk, hogy nagyobb, mint $(p \cdot r \cdot g + f \cdot m)$, ezért a egyenlő lesz $(r \cdot p \cdot g + f \cdot m)$-mel.
Az eredeti üzenetet úgy kapjuk, hogy a-t megszorozzuk F_p-vel a $\mathbb{Z}/p\mathbb{Z}$ gyűrűben.

$$C = (f \cdot F_p \cdot m) + (p \cdot F_p \cdot r \cdot g) \pmod{p}.$$

Felhasználva, hogy $f \cdot F_p \pmod{p} \equiv 1$ és $p \cdot F_p \cdot r \cdot g = 0 \pmod{p}$,

$$C = m.$$

Ezáltal visszakaptuk az eredeti üzenetünket.

A dekódoló rész interaktív megvalósítása SageMath-ban:

Ennek programkódja:

```
sage: @interact
sage: def decoditru(e=input_box('[24839, 24825, 24823, 24838,
24825, 24840]', label="Encrypted message:"), q=input_box('134887',
label="Large modulus"), h=input_box('70534', label="Public key"),
Fp=input_box('963', label="Fp"),
f=input_box('27', label="f")):
....:   p=1000
....:   a=[(f*e[i])%q for i in [0..len(e)-1]]
....:   C=[(Fp*a[l])%p for l in [0..len(a)-1]]
....:   D=[chr(k) for k in C]
....:   pretty_print('The original message:', ''.join(D))
```

3.4. Támadás gyakoriságanalízissel

A példán is jól látható, hogy az ITRU minden karakter ASCII kódján ugyanazokat a műveleteket, ugyanazon paraméterekkel hajtja végre, így megfeleltethető egy lináris kriptorendszernek. Az ilyen titkosítások nem módosítják a karakterek előfordulásának gyakoriságát, hanem csak azt, hogy a betűket milyen jellel jelöltük. Egy-egy betű előfordulásának gyakorisága minden nyelvben eltér, ugyanakkor szigorú szabályoknak is felel meg. [4]

Vegyünk például egy angol nyelvű szöveget:

A	B	C	D	E	F	G	H	I	J	K	L	M
7.3	0.9	3.0	4.4	13	2.8	1.6	3.5	7.4	0.2	0.3	3.5	2.5
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
7.8	7.4	2.7	0.3	7.7	6.3	9.3	2.7	1.3	1.6	0.5	1.9	0.1

A relatív gyakoriság különböző szövegtípusokban eltérő lehet, de ki lehet választani azokat a karaktereket, amelyek gyakorisága magas, illetve azokat, melyeknek alacsony.

Nagy gyakoriságú karakterek:	E, T, R, I, O, N, A
Kis gyakoriságú karakterek:	J, K, Q, X, Z

Gyakoriságvizsgálattal könnyedén feltörhetünk egy ITRU-val titkosított szöveget. Elegendő hozzá megállapítani a kriptoszövegben a leggyakoribb kódot és ha ebből kivonjuk az eredeti szövegben feltételezett leggyakoribb betű ASCII kódját, akkor megkapjuk az eltolás mértékét.

Tekintsük a következő angol nyelvű szöveget, melyből előre eltűntettük az összes szóközt, valamint annak egy titkosítását!

```
sage: Text="InordertofurtherenhancethesecurityofNTRU,"   72
sage: Text=Text+"researchhasbeenconductedonothervaria"   73
sage: Text=Text+"ntsofNTRU.Somevariantsproposetheuseo"   74
sage: Text=Text+"fpolynomialringswithcoefficientsinot"   75
sage: Text=Text+"herrings.In2002,Gaboritsuggestedtheu"   76
sage: Text=Text+"seoftheringofpolynomialsinsteadofthe"   77
sage: Text=Text+"ringofintegersandpresentedCTRU[5].Ko"   78
sage: Text=Text+"uzmenkosuggestedtheuseofGaussianInte"   79
sage: Text=Text+"gersandpresentedGTRUin2006[6].Otherv"   80
```

```
sage: Text=Text+"ariantsusealternativerings.In2005,Co"
sage: Text=Text+"glianese&GoisuggestedtheuseofmatriceJ"
sage: Text=Text+"sandpresentedMaTRU[7].In2011and2015J"
sage: Text=Text+"arvisandNevinssuggestedtheuseoftheri"
sage: Text=Text+"ngofEisensteinintegersandpresentedET"
sage: Text=Text+"RU[8,9].In2009Malekianetal.suggested"
sage: Text=Text+"theuseoftheringofQuaternionsandprese"
sage: Text=Text+"ntedQTRUin2015[10,11].OtherNTRUvaria"
sage: Text=Text+"ntsusevaryingcommutativestructures.I"
sage: Text=Text+"n2002,BankspresentedavariantofNTRUwh"
sage: Text=Text+"ichusesnoninvertiblepolynomials[12]."
sage: Text=Text+"In2003,RourkeandSunarpresentedavaria"
sage: Text=Text+"ntofNTRUwhichusesMontgomerymultiplic"
sage: Text=Text+"ation[13].In2007,Trumanpresentedtheu"
sage: Text=Text+"seofanoncommutativeNTRU[14].Furtherm"
sage: Text=Text+"ore,workby[15]presentsasimplifiedver"
sage: Text=Text+"sionofNTRUreferredtoasminiNTRU,which"
sage: Text=Text+"providesageneralizedparameterselecti"
sage: Text=Text+"oncriteriaandreducedparametersetswhi"
sage: Text=Text+"chfosterunderstandingoftheNTRUpublic"
sage: Text=Text+"keycryptosystem."
sage: p=1000
sage: F=Set([k for k in range(50) if gcd(k,1000)==1])
sage: f=F.random_element()
sage: S=Set([8..25])
sage: r=8
sage: g=S.random_element()
sage: m=[ord(k) for k in Text]
sage: q=next_prime(p*r*g+255*f)
sage: Fp=(1/f)%p
sage: Fq=(1/f)%q
sage: h=(p*Fq*g)%q
sage: e=[((r*h)+m[i])%q for i in [0..len(m)-1]]
sage: e[0:10]
```

[49834, 49871, 49872, 49875, 49861, 49862, 49875, 49877, 49872, 49863]

Az utolsó paranccsal az átláthatóság miatt csak az első néhány karakter kódját írja ki a program.

Ha feltételezzük, hogy az "e" betű a leggyakoribb karakter, akkor az alábbi módon eljuthatunk az eredeti szöveghez:
Készítünk egy olyan halmazt, melyben a különböző karakterek kódjai mellett feltüntetjük azoknak a titkosított szövegben való előfordulási számát is, majd ezt a halmazt rendezzük a gyakoriságot jelző tag szerinti csökkenő sorrendbe.

```
sage: se=Set(e)                                                    115
sage: Gyak=[(e.count(k),k) for k in se]                            116
sage: GyakRend=sorted(Gyak, key=lambda tup: -tup[0])               117
sage: GyakRend[0:6]                                                118
```

$$[(129, 49862), (83, 49871), (75, 49877), (74, 49876), (70, 49875), (63, 49866)]$$

A rendezett listánk első tagja mutatja meg, hogy melyik a leggyakoribb karakter a szövegben. Ebből kivonva az eredeti szövegben feltételezett leggyakoribb karakterhez rendelt ASCII kódot, esetünkben az 'e' betű kódja, ami a 101, kapjuk meg azt az értéket, mellyel minden karakter el lett tolva.

```
sage: Max=GyakRend[0]                                              119
sage: Max[1]                                                       120
```

$$49862$$

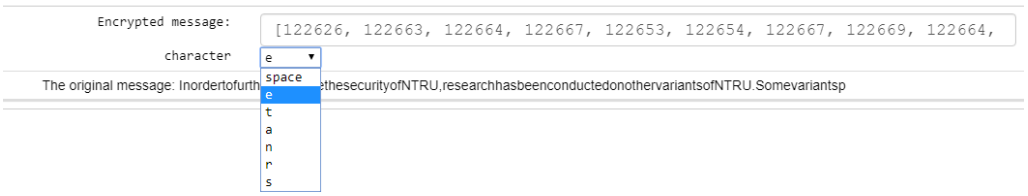
```
sage: eltolas=(Max[1])-(ord('e'))                                  121
sage: eltolas                                                      122
```

$$49761$$

Végül vesszük a titkosított szöveget és ha karakterenként az eltolás mértékét kivonjuk a kódokból, megkapjuk az eredeti sorrendben elhelyezkedő karakterek ASCII kódját. Ezekhez már csak hozzá kell rendelni a megfelelő karaktereket, és készen is van az eredeti szöveg.

```
sage: (''.join([chr(k-eltolas) for k in e]))[0:49]                 123
        InordertofurtherenhancethesecurityofNTRU,research
```

Interaktív megoldás SageMath-ban választható leggyakoribb karakterrel: Ebben a példában a "space"-t, mint karaktert, szintén belevettem a listába, hiszen szövegeink többségében az a leggyakrabban előforduló elem.



Programkódja:

```
sage: @interact
sage: def BREAK(e=input_box('[826028, 826014, 826012, 826027,
826014, 826029]', label="Encrypted message:"),
character=['space', 'e','t', 'a', 'n', 'r', 's']):
....:   Gyak=[e.count(k) for k in e]
....:   m=max(Gyak)
....:   n=Gyak.index(m)
....:   A=[chr(k) for k in [0..255]]
....:   x=character
....:   if x=='space':
....:       eltolas=(e[n])-(A.index(' '))
....:   else:
....:       eltolas=(e[n])-(A.index(x))
....:   E=[e[i]-eltolas for i in [0..len(e)-1]]
....:   D=[chr(k) for k in E]
....:   pretty_print('The original message:', D)
```

4. MaTRU

A MaTRU kriptorendszer már hatékonyabb lineáris transzformációt alkalmaz, miközben jól összehasonlítható az NTRU-val [7]. Műveletei egy olyan $\mathbb{M}$ gyűrűben helyezkednek el, melyben $k \times k$ típusú mátrixok vannak, és azok elemei pedig a $R = \mathbb{Z}[x]/(x^n - 1)$ polinomgyűrű polinomjai. Az NTRU-tól eltérően, itt kétoldali mátrixszorzás van, emiatt két gyűrűelemre lesz szükségünk. Egy másik különbség a két kriptorendszer között a kommutativitás. A MaTRU gyűrűje nem kommutatív, ami azt jelenti, hogy a privát kulcs mátrixait és a véletlenszerű mátrixokat speciálisan úgy kell megkonstruálni a titkosítás során, hogy azok felcserélhetőek legyenek.

A mátrix paraméterei négy egész számból (n, k, p, q), illetve öt mátrixkészletből $(\mathcal{L}_F, \mathcal{L}_\Phi, \mathcal{L}_A, \mathcal{L}_W, \mathcal{L}_M) \subset \mathbb{M}$ állnak.

1. $\mathcal{L}_A$ tartalmazza a $C \in \mathbb{M}$ permutációmátrixokat oly módon, hogy $C^0, C^1, \ldots, C^{k-1}$ lineárisan függetlenek modulo q. A permutációmátrix egy olyan négyzetes mátrix, melynek minden sorában és minden oszlopában csak egy darab 1-es van, mindenhol máshol pedig nullák vannak.

$$\sum_{i=0}^{k-1} C^i = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}.$$

2. $\mathcal{L}'_F$ és $\mathcal{L}'_\Phi$ tartalmazzák az összes olyan $D \in \mathbb{M}$ mátrixot, melyek úgy vannak megkonstruálva, hogy $C \in \mathcal{L}_A$, $c_0, \ldots, c_{k-1} \in \mathbb{R}$ és $D = \sum_{i=0}^{k-1} c_i C^i$. Ezen kívül az $\mathcal{L}'_F$ mátrixainak meg kell felelniük annak a követelménynek, hogy van inverzük modulo p és modulo q.

3. Az üzenetek halmaza, $\mathcal{L}_m$, az összes olyan mátrixból áll, melyben a polinomok együtthatója modulo p van redukálva. Kifejezve:

$$\mathcal{L}_M = \left\{ M \in \mathbb{M} | \text{ az } M\text{-beli polinomok együtthatói} - \frac{p-1}{2}, \frac{p-1}{2} \text{ közöttiek} \right\}.$$

4.

$$\mathcal{L}(d) = \Big\{ M \in \mathbb{M} | i = \left[-\frac{p-1}{2} \right] \dots \left[\frac{p-1}{2} \right], i \neq 0, \text{minden } M\text{-beli polinomban van átlagosan } d \text{ együttható, ami egyenlő } i\text{-vel, a többi } 0 \Big\}.$$

Például, ha $p = 3$ és $n = 5$, akkor $\mathcal{L}(2)$ tartalmazza azokat a mátrixokat, melyekben minden polinomban van átlagosan 2 együttható, ami egyenlő 1-gyel, 2 együttható, ami egyenlő -1-gyel és 1 együttható, ami pedig nulla.

5. d_f és d_Φ segítségével $\mathcal{L}_F, \mathcal{L}_\Phi$ és $\mathcal{L}_W$ így definiálható:

$$\mathcal{L}_F = \mathcal{L}(d_f) \cap \mathcal{L}'_F,$$

$$\mathcal{L}_\Phi = \mathcal{L}(d_\Phi) \cap \mathcal{L}'_\Phi,$$

$$\mathcal{L}_W = \mathcal{L}(\lfloor n/p \rfloor).$$

Tipikusan $d_f \approx n/p$ és $d_\Phi \approx n/p$.

4.1. Kulcs generálása

A privát, illetve publikus kulcs megalkotásához először választunk két $k \times k$ típusú mátrixot, $A, B \in \mathcal{L}_A$. $F, G \in \mathcal{L}_f$ mátrixokat a következőképpen konstruáljuk meg:

$$F = \sum_{i=0}^{k-1} \alpha_i A^i, \qquad G = \sum_{i=0}^{k-1} \beta_i B^i.$$

Alkalmas random $\alpha_0, \alpha_1, ..., \alpha_{k-1} \in \mathbb{R}$ és $\beta_0, \beta_1, ..., \beta_{k-1} \in \mathbb{R}$ polinomokat használva F és G kielégíti azt, hogy $F, G \in \mathcal{L}(d_f)$.

$\mathcal{L}_f$ definiálásánál már említettük, hogy F-nek és G-nek rendelkeznie kell modulo p és modulo q szerinti inverzekkel. Ez megfelelő paraméterek választása esetén így lesz. Az inverzek legyenek F_p, F_q és G_p, G_q, ahol I a $k \times k$-as egységmátrix és

$$F_p F \equiv I \pmod{p}, \qquad F_q F \equiv I \pmod{q},$$
$$G_p G \equiv I \pmod{p}, \qquad G_q G \equiv I \pmod{q}.$$

Ezáltal meg is van a privát kulcsunk, ami az (F, G) pár.

A publikus kulcs három mátrixból fog állni, (H, A, B), melyekből kettőt már ismerünk. H generálásához random választunk egy $W \in \mathcal{L}_w$ mátrixot, és:

$$H \equiv F_q W G_q \pmod{q}.$$

4.2. Kódolás

Az elküldendő üzenet titkosításához a publikus kulcs elemei közül A-t illetve B-t felhasználva elkészítjük $\Phi, \Psi \in \mathcal{L}_\Phi$ mátrixokat:

$$\Phi = \sum_{i=0}^{k-1} \phi_i A^i, \qquad \Psi = \sum_{i=0}^{k-1} \psi_i B^i.$$

Alkalmas $\phi_0, \phi_1, ..., \phi_{k-1} \in \mathbb{R}$ és $\psi_0, \psi_1, ..., \psi_{k-1} \in \mathbb{R}$ polinomokra $\Phi, \Psi \in \mathcal{L}(d_\Phi)$ is teljesülni fog.

Ezt követően a titkosítandó üzenetet a megfelelő alakban véve, $M \in \mathcal{L}_M$, létrehozzuk a titkosított szöveget:

$$E \equiv p(\Phi H \Psi) + M \pmod{q}.$$

Ebben az alakban küldi el a feladó az üzenetét.

4.3. Dekódolás

Az üzenet megfejtéséhez elsőként kiszámoljuk X-et:

$$X \equiv FEG \pmod{q}.$$

A kapott mátrixban az együtthatókat a $-q/2$ és $q/2$ tartományba helyezzük, az NTRU-hoz hasonlóan. Ezután egészként kezelve az együtthatókat, dekódoljuk az üzenetet az alábbi számolással:

$$\begin{aligned} D &\equiv F_p X G_p \pmod{p}, \\ D &= M. \end{aligned}$$

Részletes számolás:

$$\begin{aligned} X &\equiv FEG \pmod{q} \\ &\equiv F\left(p\left(\Phi H \Psi\right) + M\right) G \pmod{q} \\ &\equiv p\left(F \Phi F_q W G_q \Psi G\right) + FMG \pmod{q}. \end{aligned}$$

Habár a mátrixszorzás általában nem kommutatív, F és Φ azok lesznek:

$$\begin{aligned}
F\Phi &\equiv \left(\sum_{i=0}^{k-1} \alpha_i A^i\right)\left(\sum_{i=0}^{k-1} \phi_i A^i\right) \pmod q \\
&\equiv \sum_{i=0}^{k-1} \sum_{i\equiv j+l \pmod k} \alpha_j A^j \phi_l A^l \pmod q \\
&\equiv \sum_{i=0}^{k-1} \sum_{i\equiv j+l \pmod k} \phi_l A^{j+l} \alpha_j \pmod q \\
&\equiv \sum_{i=0}^{k-1} \sum_{i\equiv j+l \pmod k} \phi_l A^l \alpha_j A^j \pmod q \\
&\equiv \left(\sum_{i=0}^{k-1} \phi_i A^i\right)\left(\sum_{i=0}^{k-1} \alpha_i A^i\right) \equiv \Phi F \pmod q.
\end{aligned}$$

$G\Psi \equiv \Psi G \pmod q$ hasonlóan belátható. Tehát írható, hogy

$$X \equiv p(F\Phi F_q W G_q \Psi G) + FMG \equiv p(\Phi W \Psi) + FMG \pmod q.$$

Elegendően nagy q választása esetén tekinthetjük X polinomjainak az együtthatóit modulo q redukció nélkül, azaz mintha $\mathbb{Z}$-beliek lennének. Tudjuk venni az együtthatókat modulo p, elhagyva a $FMG \pmod p$ részt. Az eredeti üzenetet megkapjuk egy F_p-vel való balszorzással és egy G_p-vel való jobbszorzással.

$$F_p X G_p \equiv M \pmod p.$$

Irodalomjegyzék

[1] Helfer Etienne. LLL lattice basis reduction algorithm. pages 1–17, 2010.

[2] J. N. Gaithuru, M. Salleh, and I. Mohamad. ITRU: NTRU-Based Cryptosystem Using Ring of Integers. *International Journal of Innovative Computing*, 7(1):33–38, 2017.

[3] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[4] David R. Kohel. Cryptography. pages 19–21, 2008.

[5] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982. https://doi.org/10.1007/BF01457454.

[6] Sourav Mukhopadhyay. Public Key Cryptography, 2018. http://www.facweb.iitkgp.ac.in/ sourav/PublicKeyCrypto.pdf.

[7] Jeong Eun SONG, Tae Youn HAN, and Mun-Kyu LEE. Analysis and Improvement of MaTRU Public Key Cryptosystem. pages 982–988, 2015.

[8] Tengely Szabolcs. Lecture Notes on Cryptography. Online published, 2019. http://shrek.unideb.hu/ tengely/crypto/section-8.htmlsubsection-29.