Fakultet elektrotehnike i računastva, Sveučilište u Zagrebu

Postkvantna kriptografija - FALCON
Tehnička dokumentacija

Verzija 2

Student: Slađan Tadić

Nastavnik: prof.dr.sc. Marin Golu

Sadržaj

1. UVOD

1.1 SVRHA

Cilj postkvantnih kriptičkih algoritama je pružanje sigurnosnih karakteristika pri suočavanju s kvantnim računalima. Kvantna računala su moguća, ali zbog trenutnih tehnoloških ograničenja još postoje brojni problem. Takvo kvantno računalo bi vrlo učinkovito razbilo uobičajene asimetrične enkripcije i algoritme digitalnog potpisa temeljene na teoriji brojeva.

1.2 FALCON

Shema potpisa temelji se na GPV(Gentry, Peikert i Vaikuntanathan) radnom okviru. Instanciramo okvir koristeći NTRU rešetke s pomoću takozvanog "brzog Fourierovog uzorkivača". Temeljna poteškoća je rješavanje problema kratkih cijelih brojeva (SIS) preko NTRU rešetki, za što ne postoji znani učinkoviti algoritam, čak ni uz pomoć kvantnih računala.

1.3 DOSEG

- Sigurnost: koristi se pravi Gaussov uzorkivač, koji garantira neznatno curenje informacija tajnog ključa, do praktično, beskonačnog broja potpisa
- Kompaktnost: zahvaljujući NTRU rešetkama, potpisi su znatno kraći nego bilo koja druga shema potpisa bazirana na rešetkama uz istu razinu sigurnosti, dok je javni ključ iste duljine
- Brzina: korištenjem brzog Fourierovog uzorkivača može se generirati tisuće ključeva po sekundi na prosječnom računalu, potvrda potpisa je pet do deset puta brža
- Skalabilnost: operacijska složenost je O(n log n) za stupanj n, što osigurava umjeren trošak
- Ekonomičnost: koristi manje od 30 kB RAM-a, Falcon je kompatibilan s malim komponentama

1.4 PERFORMANSE

Algoritam će doživjeti znatnu primjenu samo ako je razumno učinkovit, u našem svijetu, gdje nema kvantnih računala. Koristeći uobičajeno stolno računalo (Intel® Core® i5-8259U at 2.3 GHz, TurboBoost disabled), Falcon je postigao sljedeće performanse:

varijanta	keygen(ms)	keygen(RAM)	sign/s	verify/s	pub size	sig size
FALCON-512	8.64	14336	5948.1	27933.0	897	666
FALCON-1024	27.45	28672	2913.0	13650.0	1793	1280

Veličine osim *keygen* su izražene u bajtovima. Za usporedbu, FALCON-512 ekvivalentan je s RSA-2048 koji koristi 256 bajta.

2. ALGORITAM

2.1 PREGLED

Glavni element u FALCONu su polinomi stupnja n s cijelim koeficijentima. Stupanj n je, uobičajeno, potencija broja 2 (uglavnom, 512 ili 1024). Izračuni se rade po modulu monoma istog stupnja (označavamo s Φ, koji je uvijek oblika $\Phi = x^n + 1$).

Unutar algoritma, neki polinomi interpretiraju se kao vektori, a neki kao matrice koje određuju rešetku. Stoga, FALCON možemo uglavnom izražavati kroz operacije nad polinomima, čak i kad radimo s matricama koje određuju rešetku.

2.2 KLJUČEVI

2.2.1 PRIVATNI KLJUČ

Privatni ključ je osnova za rešetku dimenzije 2n, s parametrima:.

$$\left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$$

gdje su f, g, F i G kratki integralni polinomi po modulu Φ, također, polinom f je invertibilan.
Moraju biti zadovoljene jednadžbe:

- $h = \frac{g}{f} \ mod\ \Phi\ mod\ q$
- $f\ G\ -\ g\ F\ =\ q\ mod\ \Phi$ (*NTRU* jednadžba)

F i G se mogu izračunavati dinamički, ali je to skupo, stoga je bar očekivano spremanje F-a, uz f i g, a G izračunavamo učinkovito. Uz navedene polinome, iz privatnog ključa računamo (dinamično, ili ih spremamo uz f, g i F):

- FFT (Fast Fourier Transformation) reprezentaciju f, g, F i G-a , predstavljenih kao matricu:

$$\hat{\mathbf{B}} = \left[\begin{array}{c|c} \mathsf{FFT}(g) & -\mathsf{FFT}(f) \\ \hline \mathsf{FFT}(G) & -\mathsf{FFT}(F) \end{array}\right]$$

- FALCON stablo T – binarno stablo sa sljedećim pravilima:
  - stablo visine 0 ima samo jedan čvor čija je vrijednost realni σ > 0
  - stablo visine κ posjeduje sljedeća svojstva:
    - vrijednost korijena, T.value jest polinom $\ell \in Q[x]/(x^n + 1)$ gdje je $n = 2^\kappa$
    - njegova lijeva i desna djeca, su FALCON stable visine κ - 1

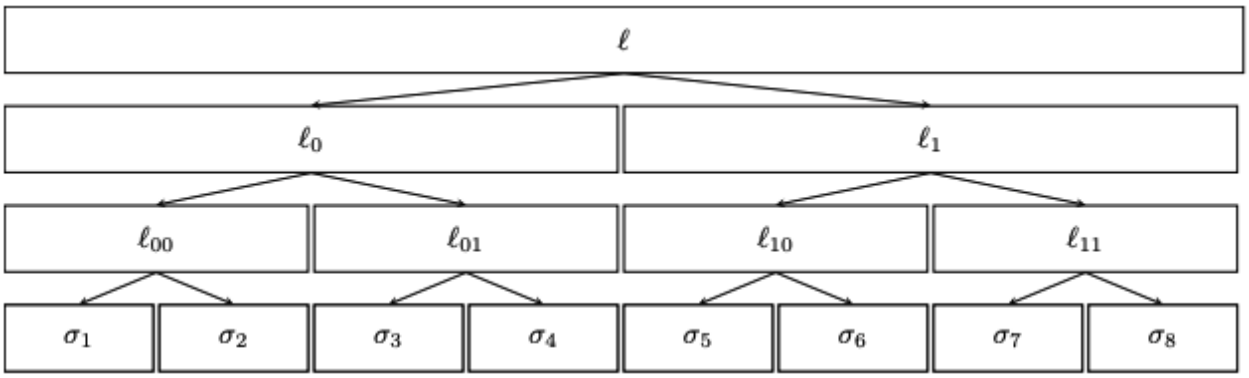


Figure 1. FALCON stablo visine 3

2.2.2 JAVNI KLJUČ

Javni ključ je osnova za rešetku dimenzije 2n, s parametrima:

$$\left[\begin{array}{c|c} -h & I_n \\ \hline qI_n & O_n \end{array}\right]$$

- I_n : matrica identiteta dimenzije n
- O_n : nul matrica
- -h : nxn matrica (polinom f mod Φ), cijeli broj u rasponu od 0 do n – 1
- qI_n : specifični, mali, prim broj (preporučeno 12289)

FALCON javni ključ *pk* korespondira s privatnim ključem *sk = (f, g, F, G)* je polinom h takav da vrijedi:
$h = gf^{-1} \ mod \ (\phi, q)$

2.3 BRZA FOURIEROVA TRANSFORMACIJA (FFT) I TEORIJA TRANSFORMACIJE BROJEVA (NNT)

2.3.1 BRZA FOURIEROVA TRANSFORMACIJA (FFT - FAST FOURIER TRANSFORMATION)

Postoje nekoliko načina implementacije FFT-a, koji mogu rezultirati malo drukčijim rezultatom. Koristi se u za operacije privatnog ključa, a svojstva implementacije u FALCON-u su:

- FFT se ne skalira konstantnim faktorom (npr. 1/ deg(ϕ))
- Nema ograničenja na redoslijed FFT-a (npr. rastući red u jediničnom krugu, smjer kazaljke na satu), ostaje na izbor onome tko ga implementira, ali mora biti konzistentan u cijelom algoritmu

2.3.2 TEORIJA TRANSFORMACIJE BROJEVA (NTT - NUMBER THEORETIC TRANSFORMATION)

NNT analogan je FFTu u polju $\mathbb{Z}_p$ gdje je p prim broj za koji vrijedi p = 1 mod 2n. I pod tim uvjetima ϕ ima točno n korijena. Konverzija iz, i u NTT reprezentacije može se učiniti učinkovito u O(nlogn) operacija. Služi za brzu implementaciju operacija javnog ključa.

2.4 CIJEPANJE I SPAJANJE

ϕ je *ciklotomski polinom*, stoga ga možemo zapisati kao $\phi(x) = \prod_{k \in Z \times m} (x - \zeta^k)$, m = 2n, a ζ je proizvoljan primitivni m-ti korijen od 1 (npr. $\zeta = \exp(2i\pi/m)$). Neka je ϕ' takav da vrijedi $\phi(x) = \phi'(x^2)$ (npr. $\phi(x) = x^n + 1$ i $\phi'(x) = x^{n/2}+1$).

2.4.1 FUNKCIJA splitfft

Funkcija rastava FFT(f), f se može jedinstveno rastaviti kao $f(x) = (x^2) + xf_1(x^2)$. Taj rastav možemo zapisati i kao:

$$f_0 = \sum_{0 \le 1 < \frac{n}{2}} a_{2i} x^i \text{ i } f_1 = \sum_{0 \le 1 < \frac{n}{2}} a_{2i+1} x^i$$

Rastavili smo f na parne i neparne koeficijente, stoga možemo jednostavno napisati $split(f) = (f_0, f_1)$.

Algoritam 1 *splitfft*(FFT(f))

Ulaz: $FFT(f) = (f(\zeta))_\zeta$
Izlaz: $FFT(f_0) = (f_0(\zeta))_\zeta \ i \ FFT(f_1) = (f_1(\zeta))_\zeta$
1. za ζ takav da $\phi(\zeta) = 0$ i $\text{Im}(\zeta) > 0$
2. $\zeta' \leftarrow \zeta^2$
3. $f_0(\zeta') \leftarrow \frac{1}{2}[f(\zeta) + f(-\zeta)]$
4. $f_1(\zeta') \leftarrow \frac{1}{2\zeta}[f(\zeta) - f(-\zeta)]$
5. vrati $FFT(f_0)$ i $FFT(f_1)$

2.4.2 FUNKCIJA mergefft – inverzna funkcija splitfft

Algoritam 2 *mergefft*(f_0, f_1)

Ulaz: $FFT(f_0) = (f_0(\zeta))_\zeta \ i \ FFT(f_1) = (f_1(\zeta))_\zeta$
Izlaz: $FFT(f) = (f(\zeta))_\zeta$
1. za ζ takav da $\phi(\zeta) = 0$
2. $\zeta' \leftarrow \zeta^2$

3. $f(\zeta) \leftarrow f_0(\zeta') + \zeta f_1(\zeta')$
4. vrati $FFT(f)$

2.5 HASHIRANJE

Prvi korak za potpisivanje poruke ili njezinu verifikaciju sastoji se od *hashiranja* poruke. U našem slučaju, poruka mora biti *hashirana* u polinomskom obliku. Za taj postupak koristit ćemo metodu XOF tj. odobrenu *hash* funkciju s proširenim izlazom, točnije SHAKE-256, za sve sigurnosne razine.

- SHAKE-256 -Init () označava inicijalizaciju SHAKE-256 konteksta *hashiranja*
- SHAKE-256 -Inject (ctx, str) označava ubacivanje podatka kontekst *hashiranja* ctx
- SHAKE-256 -Extract (ctx, b) označava ekstrakciju b pseudoslučajnih bitova iz hashing iz hashing konteksta ctx

Algoritam 3 HashToPoint(str, q, n)

Ulaz: tekst str, modul q <= 2^{16}, stupanj n
Izlaz: polinom c = $\sum_{i=0}^{n-1} c_i x^i$

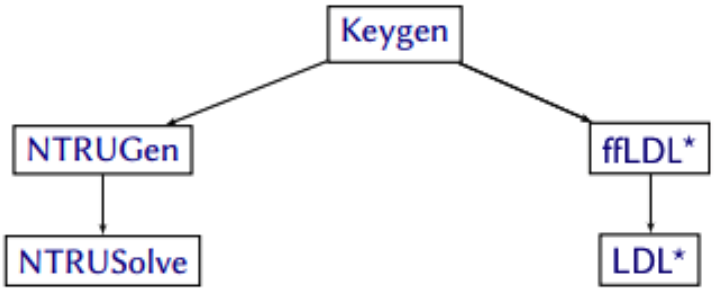
1. k <- [2^{16}/q]
2. ctx <- SHAKE-256-Init()
3. SHAKE-256-Inject(ctx, str)
4. SHAKE-256-Extract(ctx, n = 16), generira dijelove polinoma
5. Agregiraj generirane dijelove
6. Vrati polinom

2.6 GENERIRANJE PARA KLJUČEVA

Generiranje para ključeva možemo razdvojiti u dva koraka:

1. Rješavanje NTRU jednadžbe tj. generiranje f i g (lako), te F i G (teško)
2. Izračun *FALCON* stabla

Nakon čega slijedi normalizacija čvorova LDL stabla za pretvorbu u FALCON stablo. Pohranjeni rezultat umatamo u privatni ključ *sk* i odgovarajući javni ključ *pk* tj. $h = gf^{-1} mod\ q$.



Slika 1Dijagram toka za generiranje ključeva

Algoritam 4 *Keygen(ϕ, q)*

Ulaz: monom tj. $2^n + 1$ ➔ n , q = 12289
Izlaz: *pk, sk*

1. f, g, F, G ← NTRUGen(ϕ, q)
2. Stvori rešetku B

3. Izračunaj FFT(Fast Fourier Transformation) za rešetku B, tj. svaku njezinu komponentu
4. Izračunaj dekompoziciju LDL* stabla, $G \leftarrow \hat{B} \times \hat{B}^*$
   (* predstavlja tzv. Hermitian adjoint, u cijelom dokumentu, a × vektorski umnožak)
5. Izračun LDL* stabla T ← ffLDL* (G)
6. za svaki list od T (normalizacija stabla)
7. $list.vrijednost \leftarrow \sigma/\sqrt{list.vrijednost}$ (σ = standardna devijacija *trapdoor* uzorkivača)
8. Stvori privatni ključ, *sk* ← (Bˆ , T)
9. Izračunaj h = gf $^{-1}$ mod q.
10. $pk \leftarrow h$
11. vrati *sk*, *pk*

2.6.1 GENERIRANJE POLINOMA f, g, F i G

1. Generiramo nasumično polinome f i g. Zatim za njih provjeravamo jesu li prikladni za svrhu algoritma. To radimo na načine:
   a. Linija 7 osigurava da se h može izračunati iz f i g. To je istina samo onda NTT(f) ne sadrži koeficijente koji su 0.
   b. polinomi f, g, F, G moraju dozvoliti generiranje kratkih potpisa, To vrijedi za relaciju
   $$\gamma \leq 1.17\sqrt{q}$$
2. Izračunavamo F i G, tako da f, g, F i G prolaze provjeru. To radimo s algoritmom **NTRUSolve**

Algoritam 5 *NTRUGen(ϕ, q)*

Ulaz: monom stupnja n , q = 12289
Izlaz: polinomi f, g, F, G

1. $\sigma\{f,g\} \leftarrow 1.17\sqrt{q/2n}$ (odabiremo standardnu devijaciju σ tako da zadovoljava relaciju $||(f,g)|| = 1.17\sqrt{q}$)
2. za *i* od 1 do n - 1
3. $f_i \leftarrow D_{Z,\sigma\{f,g\},0}$ (diskretan Gaussov uzorkivač)
4. $g_i \leftarrow D_{Z,\sigma\{f,g\},0}$
5. $f \leftarrow \sum_i f_i x^i$ ($f \in \mathbb{Z}[x]/(\phi)$)
6. $g \leftarrow \sum_i g_i x^i$ ($g \in \mathbb{Z}[x]/(\phi)$)
7. ako NTT(f) sadrži 0 kao koeficijent
8. ponovi postupak
9. $\gamma \leftarrow \max\{\| (g,-f) \|, \| (\frac{qf^*}{ff^*+gg^*}, \frac{qg^*}{ff^*+gg^*}) \|\}$
10. ako je $\gamma > 1.17\sqrt{q}$ onda (provjeri je li Gram-Schmidt norma od B kratka)
11. ponovi postupak
12. $F, G \leftarrow NTRUSolve_{n,q}(f,g)$ (F, G takav da vrijedi $fG - gf = q \bmod \phi$)
13. ako su $(F,G) = \perp$ onda
14. ponovi postupak

Algoritam 6 $NTRUSolve_{n,q}(f,g)$

Ulaz: *f, g* (monom x^n+1 gdje je *n* potencija broja 2)
Izlaz: polinomi F, G koji zadovoljavaju jednadžbu $f\ G\ -\ g\ F\ =\ q\ mod\ \Phi$
1. ako je *n* = 1 onda
2. izračunaj *u*, *v* tako da $uf\ -\ vg\ =\ gcd(f,g)$ (gcd = najveći zajednički djelitelj)
3. ako $\gcd(f,g) \neq 1\ onda$
4. prekini i vrati ⊥
5. $(F,G) \leftarrow (vq, uq)$
6. vrati F, G
7. inače
8. $f' \leftarrow N(f)$, gdje je *N(f) = f(x)*f(-x)* $(f', g', F', G'\ \in \mathbb{Z}[x]/(x^{\frac{n}{2}}+1))$
9. $g' \leftarrow N(g)$
10. $F', G'\ \leftarrow NTRUSolve_{n/2,q}(f', g')$ (rekurzivni poziv)
11. $F\ \leftarrow\ F'(x^2)g(-x)$ $F, G \in \mathbb{Z}[x]/(x^n+1)$
12. $G\ \leftarrow\ G'(x^2)f(-x)$
13. Reduce(f, g, F, G) (reduciraj (F, G) vodeći računa o (f, g))
14. vrati (F, G)

Reduciranje rješenja za vektore ***u, v*** reduciramo ***u*** uzimajući u obzir ***v*** radeći transformaciju $\boldsymbol{u\ \leftarrow\ u\ -\ \lfloor\frac{\langle \mathbf{u},\mathbf{v}\rangle}{\langle \mathbf{v},\mathbf{v}\rangle}\rceil\ \mathbf{v}}$.

Za naš slučaj ***u*** zamjenjujemo s (***F, G***), a ***v*** zamjenjujemo s (***f, g***).

Algoritam 7 $Reduce(f,g,F,G)$

Ulaz: *polinomi f, g, F, G*
Izlaz: (***F, G***) reduciran s obzirom na (***f, g***)
1. radi
2. $k \leftarrow \lfloor\frac{Ff^*+Gg^*}{ff^*+gg^*}\rceil$ $\left\lfloor\frac{Ff^*+Gg^*}{ff^*+gg^*}\right\rceil \in \mathbb{Q}, k\ \in \mathbb{Z}$
3. $F \leftarrow F - kf$
4. $G \leftarrow G - kg$
5. dok je $k \neq 0$

2.6.2 IZRAČUN FALCON STABLA

Tajni ključ sk je oblika $sk = (\widehat{\boldsymbol{B}}, T)$. Postupak za izračun LDL stabla sastoji se od:

1. Izračunamo LDL dekompoziciju **G**: napišemo $\boldsymbol{G = L\ \times D \times\ L^*}$, gdje je **L** donja trokutasta matrica sa "1" na dijagonalama i **D** dijagonalnom matricom. **L** spremamo u *T.value* koja je vrijednost korijena od T. **L** je oblika $\mathbf{L} = \left[\begin{array}{c|c} 1 & 0 \\ \hline L_{10} & 1 \end{array}\right]$, stoga samo trebamo spremiti $\boldsymbol{L_{10}} \in \mathbb{Q}[x]/(\phi)$.

2. Onda koristimo *splitting* operator za rastav svake dijagonalne matrice **D** u manje matrice. Za svaki dijagonalni element rastavljen u novu matricu $\mathbf{G_i}$, rekurzivno izračunavamo LDL stablo kao u 1. koraku i spremamo rezultat u lijevo ili desno dijete korijena T (tj. *T.leftchild* i *T.rightchild)*

Algoritam 8 $LDL^*(\boldsymbol{G})$

Ulaz: *matrica* $\boldsymbol{G} = (G_{ij}) \in FFT(\mathbb{Q}[x]/(\phi))^{2\times 2}$
Izlaz: LDL* dekompozicija $\boldsymbol{G = LDL^*}$
Svi polinomi su u FFT formatu
1. $D_{00}\ \leftarrow\ G_{00}$
2 .$L_{10}\ \leftarrow\ G_{10}/G_{00}$
3. $D_{11}\ \leftarrow\ G_{11} - L_{10} \odot L_{10}^* \odot G_{00}$

4. $\mathbf{L} \leftarrow \left[\begin{array}{c|c} 1 & 0 \\ \hline L_{10} & 1 \end{array}\right], \mathbf{D} \leftarrow \left[\begin{array}{c|c} D_{00} & 0 \\ \hline 0 & D_{11} \end{array}\right]$
5. $vrati\ (\mathbf{L},\ \boldsymbol{D})$

Algoritam 9 $ffLDL^*(\boldsymbol{G})$

Ulaz: $Gram\ matrica\ \mathrm{G} \in FFT(\mathbb{Q}[x]/(x^n+1))^{2\times 2}$
Izlaz: binarno stablo T
Svi polinomi su u FFT formatu
1. $(\mathbf{L},\ \boldsymbol{D}) \leftarrow \boldsymbol{LDL^*}(\mathbf{G})$
2. $T.value \leftarrow \boldsymbol{L}_{10}$
3. $ako\ (n = 2)$
4. $T.leftchild \leftarrow D_{00}$
5. $T.rightchild \leftarrow D_{11}$
6. vrati T
7. inače
8. $d_{00}, d_{01} \leftarrow spliftfft(D_{00})$
9. $d_{10}, d_{11} \leftarrow spliftfft(D_{11})$
10. $\mathbf{G}_0 \leftarrow \left[\begin{array}{c|c} d_{00} & d_{01} \\ \hline d_{01}^* & d_{00} \end{array}\right], \mathbf{G}_1 \leftarrow \left[\begin{array}{c|c} d_{10} & d_{11} \\ \hline d_{11}^* & d_{10} \end{array}\right]$
11. $T.leftchild \leftarrow ffLDL^*(\boldsymbol{G}_0)$
12. $T.rightchild \leftarrow ffLDL^*(\boldsymbol{G}_1)$
13. vrati T

2.7 GENERIRANJE POTPISA

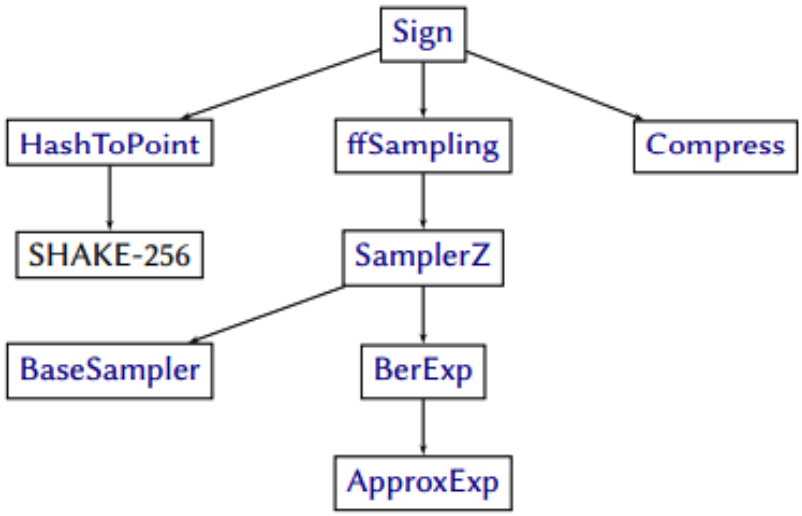


Figure 2 Slijedni dijagram generiranja potpisa

Iz poruke *m* generiramo *hash* vrijednost *c* i *salt r*, koristimo znanje o tajnom ključu *f*, *g*, *F*, *G* za izračunavanje dvije kratke vrijednosti s_1, s_2 tako da vrijedi $s_1 + s_2 h = c \bmod q$

Algoritam 10 $Sign(m, sk, \lfloor\beta^2\rfloor)$

Ulaz: poruka m, tajni ključ *sk*, granica$\lfloor\beta^2\rfloor$
Izlaz: potpis poruke m
1. $r \leftarrow \{0,1\}^{320}$ uniformno raspoređeno

2. $c \leftarrow HashToPoint(r||m,q,n)$
3. $t \leftarrow \left(-\frac{1}{q}FFT(c) \odot \text{FFT(F)}, \frac{1}{q}FFT(c) \odot \text{FFT(f)}\right)$ ($\odot$ - množenje FFT domena, t = (FFT(c), FFT(0))$\cdot \widehat{\boldsymbol{B}}^{-1}$)
4. radi
5. radi
6. $\boldsymbol{z} \leftarrow ffSampling_n(\boldsymbol{t}, T)$
7. $\boldsymbol{s} = (\boldsymbol{t} - \boldsymbol{z}) \cdot \widehat{\boldsymbol{B}}$ (**s** prati Gaussovu razdiobu)
8. dok $||s^2|| > \lfloor\beta^2\rfloor$ (s je u FFT prikazu)
9. $(\boldsymbol{s1}, \boldsymbol{s2}) \leftarrow \boldsymbol{invFFT}(\boldsymbol{s})$ $s_1 + s_2 h = c \bmod (\phi, q)$
10. $s \leftarrow Compress(s_2, 8 \cdot sbyteled - 328)$ (ukloni 1 bajt iz zaglavlja i 40 bajtova za r)
11. $dok\ (s = \perp)$
12. $vrati\ sig * (r, s)$

2.7.1 FAST FOURIER SAMPLING

Algoritam 11 $ffSampling_n(\boldsymbol{t}, T)$

Ulaz: $\boldsymbol{t} = (t_0, t_1) \in FFT(\mathbb{Q}[x]/(x^n+1))^2, FALCON\ stablo\ T$
Izlaz: $\boldsymbol{z} = (z_0, z_1) \in FFT(\mathbb{Z}[x]/(x^n+1))^2$
Svi polinomi su u FFT formatu.
1. ako je n = 1 onda
2. $\sigma' \leftarrow T.value$ $\sigma' \in [\sigma_{min}, \sigma_{max}]$
3. $z_0 \leftarrow SamplerZ(t_0, \sigma')$ $za\ n = 1, t_i = invFFT(t_i) \in \mathbb{Q}\ i\ z_i = invFFT(z_i) \in \mathbb{Z}$
4. $z_1 \leftarrow SamplerZ(t_1, \sigma')$
5. vrati $\boldsymbol{z} = (z_0, z_1)$
6. $(\ell, T_0, T_1) \leftarrow (T.value, T.leftchild, T.rightchild)$
7. $\boldsymbol{t}_1 \leftarrow splitfft(\boldsymbol{t}_1)$
8. $\boldsymbol{z}_1 \leftarrow ffSampling_{n/2}(\boldsymbol{t}_1, T_1)$
9. $z_1 \leftarrow mergefft(\boldsymbol{z_1})$
10. $t' \leftarrow t_0 + (t_1 - z_1) \odot \ell$
11. $t_0 \leftarrow splitfft(t'_0)$
12. $z_0 \leftarrow ffSampling_{n/2}(\boldsymbol{t_0}, T_0)$
13. $z_0 \leftarrow mergefft(z_0)$
14. $vrati\ \boldsymbol{z} = (z_0, z_1)$

Za algoritam *BaseSampler() koristimo vrijednost RCDT iz tablice 1.*
Stupci predstavljaju vrijednosti:

- *distribucija vjerojatnosti pdt[i]*
- *kumulativna distribucija vjerojatnosti* $cdt[i] = \sum_{j<i} pdt[j]$
- *obrnuta kumulativna distribucija* $RCDT[i] = \sum_{j>i} pdt[j] = 2^{72} - cdt[i]$

i	pdt[i]	cdt[i]	RCDT[i]
0	1 697 680 241 746 640 300 030	1 697 680 241 746 640 300 030	3 024 686 241 123 004 913 666
1	1 459 943 456 642 912 959 616	3 157 623 698 389 553 259 646	1 564 742 784 480 091 954 050
2	928 488 355 018 011 056 515	4 086 112 053 407 564 316 161	636 254 429 462 080 897 535
3	436 693 944 817 054 414 619	4 522 805 998 224 618 730 780	199 560 484 645 026 482 916
4	151 893 140 790 369 201 013	4 674 699 139 014 987 931 793	47 667 343 854 657 281 903
5	39 071 441 848 292 237 840	4 713 770 580 863 280 169 633	8 595 902 006 365 044 063
6	7 432 604 049 020 375 675	4 721 203 184 912 300 545 308	1 163 297 957 344 668 388
7	1 045 641 569 992 574 730	4 722 248 826 482 293 120 038	117 656 387 352 093 658
8	108 788 995 549 429 682	4 722 357 615 477 842 549 720	8 867 391 802 663 976
9	8 370 422 445 201 343	4 722 365 985 900 287 751 063	496 969 357 462 633
10	476 288 472 308 334	4 722 366 462 188 760 059 397	20 680 885 154 299
11	20 042 553 305 308	4 722 366 482 231 313 364 705	638 331 848 991
12	623 729 532 807	4 722 366 482 855 042 897 512	14 602 316 184
13	14 354 889 437	4 722 366 482 869 397 786 949	247 426 747
14	244 322 621	4 722 366 482 869 642 109 570	3 104 126
15	3 075 302	4 722 366 482 869 645 184 872	28 824
16	28 626	4 722 366 482 869 645 213 498	198
17	197	4 722 366 482 869 645 213 695	1
18	1	4 722 366 482 869 645 213 696	0

Table 1 Tablica distribucija, skalirana za faktor 2^{72}

Vrijedi relacija $\chi(i) = 2^{-72} \cdot pdt[i]$.

Za bilo koju logičku propoziciju P vrijedi, [[P]] = $\begin{cases} 1\ ako\ je\ P\ istina \\ 0\ ako\ je\ P\ laž \end{cases}$

$UniformBits(k)$ uzorkuje *z* uniformno u rasponu $\{0, 1, \ldots, 2^k - 1\}$.

Koristimo notaciju (>>) i (&) kako bi prikazali logički pomak u desno i operator AND.

Algoritam 12 BaseSampler()

Ulaz: -

Izlaz: cijeli broj $z_0 \in \{0, \ldots, 18\}$ takav da $z \sim \chi$

Svi polinomi su u FFT formatu

1. $u \leftarrow UniformBits(72)$
2. $z_0 \leftarrow 0$
3. za i = 0, … , 17
4. $z_0 \leftarrow z_0 + [[u < RCDT[i]]]$ (ne pdt ili cdt)
5. $vrati\ z_0$

Neka je C lista 64bitnih brojeva (u *hex* zapisu):

C =[0x0000004741183A3, 0x00000036548CFC06, 0x0000024FDCBF140A, 0x0000171D939DE045,
0x0000D00CF58F6F84, 0x000680681CF796E3, 0x002D82D8305B0FEA, 0x011111110E066FD0,
0x0555555555070F00, 0x155555555581FF00, 0x400000000002B400, 0x7FFFFFFFFFF4800,
0x8000000000000000].

Neka je polinom $f \in \mathbb{R}$ takav da vrijedi: $f(x) = 2^{-63} \cdot \sum_0^{12} C[i] \cdot x^{12-i}$

$f(-x)$ je dobra aproksimacija za *exp*(-x) u granicama [0, ln(2)]. Stoga koristimo algoritam *ApproxExp* kako bi približno izračunali vrijednost $2^{63} \cdot ccs \cdot \exp(-x)$ za x u određenom rasponu. Međuvrijednosti varijabli *y, z* u ApproxExp su uvijek u rasponu $\{0, \ldots, 2^{63} - 1\}$, sa jednom iznimkom: ako je x = 0, onda na kraju *for* petlje vrijedi $y = 2^{63}$. Iz tog razloga, pogodno je prikazivati *x, y* koristeći, npr. C tip *uint64_t*.

Algoritam 13 ApproxExp(x, ccs)

Ulaz: decimalna vrijednost $x \in [0, \ln(2)]$ i $css \in [0, 1]$
Izlaz: integralna aproksimacija $2^{63} \cdot ccs \cdot \exp(-x)$
1. C = [0x00000004741183A3,0x00000036548CFC06,0x0000024FDCBF140A,0x0000171D939DE045, 0x0000D00CF58F6F84, 0x000680681CF796E3, 0x002D82D8305B0FEA, 0x011111110E066FD0, 0x0555555555070F00, 0x155555555581FF00, 0x400000000002B400, 0x7FFFFFFFFFF4800, 0x8000000000000000]
2. $y \leftarrow C[0]$ (y i z ostaju u rasponu {0, … , 2^{63} - 1} cijeli algoritam)
3. $z \leftarrow [2^{63} \cdot x]$
4. za i = 1, … , 12
5. $y \leftarrow C[u] - (z \cdot y) \gg 63$ $(z \cdot y)\ stane\ u\ 126\ bita, ali\ nama\ je\ potrebno\ samo\ gornjih\ 63$
6. $z \leftarrow [2^{63} \cdot ccs]$
7. $y \leftarrow (z \cdot y) \gg 63$
8. vrati y

Za dane ulaze $x, ccs \geq 0$, BerExp (Algoritam 14) vraća jedan bit 1 sa vjerojatnošću $\approx ccs \cdot \exp(-x)$

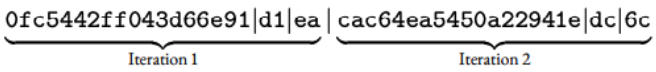
Algoritam 14 BerExp(x, ccs)

Ulaz: decimalne vrijednosti $x, ccs \geq 0$
Izlaz: jedan bit, jednak 1 s vjerojatnošću $\approx ccs \cdot \exp(-x)$
1. $s \leftarrow [x/\ln(2)]$ (izračunava jedinstvenu dekompoziciju $x = 2^s \cdot r, gdje\ je\ (r, s) \in [0, \ln(2)) \times \mathbb{Z}^+$
2. $r \leftarrow x - s \cdot \ln(2)$
3. $s \leftarrow \min(s, 63)$
4. z $\leftarrow (2 \cdot \mathrm{ApproxExp(r, ccs)} - 1) \gg s$ $(z \approx 2^{64-s} \cdot ccs \cdot \exp(-r) = 2^{64} \cdot ccs \cdot \exp(-x))$
5. $i \leftarrow 64$
6. $radi$
7. $i \leftarrow i - 8$
8. $w \leftarrow UniformBits(8) - ((z \gg i) \& 0xFF)$
9. $dok\ je((w = 0)$ i $(i > 0))$
10. vrati $[[w < 0]]$

Algoritam 15 SamplerZ(μ, σ')

Ulaz: decimalne vrijednosti $\mu, \sigma' \in \mathcal{R}$ takvi da $\sigma' \in [\sigma_{min}, \sigma_{max}]$
Izlaz: cijeli broj $z \in \mathbb{Z}$ uzorkovan iz distribucije vrlo blizu $D_{\mathbb{Z},\mu,\sigma'}$
1. $r \leftarrow \mu - \lfloor \mu \rfloor$ (r mora biti u intervalu [0, 1>)
2. $ccs \leftarrow \sigma_{min}/\sigma'$ (*ccs* omogućava algoritmu vrijeme pokretanja neovisno o σ')
3. $dok(1)$
4. $z_0 \leftarrow BaseSampler()$
5. $b \leftarrow UniformBits(8)\ \&\ 0x1$
6. $z \leftarrow b + (2 \cdot b - 1)z_0$
7. $x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{max}^2}$
8. $ako\ je\ BerExp(x\ ccs) = 1\ onda$
9. vrati $z + \lfloor \mu \rfloor$

Testovi s poznatim odgovorima ili *Known Answer Tests* (KAT). Kako bi pravilno implementirali Sampler Z i njegove podrutine, koristimo Tablicu 2. Svaki redak daje četvorku takvu da pri zamjeni internih poziva za *UniformBits*() čitajući bitove iz *randombytes* (koji se ponaša kao nasumični *bytestring)*: SamplerZ(μ, σ′) → z. Za čitljivost, tablica razdvaja *randombytes* za svaku iteraciju *SamplerZ-a*. Na primjer, redak 1, SamplerZ iterira dva puta prije terminiranja:



Pri svakoj iteraciji, prvih 9 nasumičnih bajtova koristi *BaseSampler*, idući koristi redak 5, dok zadnji koristi *BerExp*. Imajte na umu da *BerExp* ima vjerojatnost $1/2^8$ za korištenje jednog nasumičnog bajta; ovo je rijetko, ali se događa. To možemo vidjeti u retku 9 tablice 2, koji sadrži jedan takav primjer gdje koristi 2 nasumična bajta.

Table 2 Testni vektor za SamplerZ(σ_{min} = 1.277 833 697 za FALCON-512)

	disperzija μ	standardna devijacija σ'	*nasumični bitovi*	izlaz z
1	-91.90471153063714	1.7037990414754918	0fc5442ff043d66e91d1ea cac64ea5450a22941edc6c	-92
2	-8.322564895434937	1.7037990414754918	f4da0f8d8444d1a77265c2 ef6f98bbbb4bee7db8d9b3	-8
3	-19.096516109216804	1.7035823083824078	db47f6d7fb9b19f25c36d6 b9334d477a8bc0be68145d	-20
4	-11.335543982423326	1.7035823083824078	ae41b4f5209665c74d00dc c1a8168a7bb516b3190cb4 2c1ded26cd52aed770eca7 dd334e0547bcc3c163ce0b	-12
5	7.9386734193997555	1.6984647769450156	31054166c1012780c603ae 9b833cec73f2f41ca5807c c89c92158834632f9b1555	8
6	-28.990850086867255	1.6984647769450156	737e9d68a50a06dbbc6477	-30
7	-9.071257914091655	1.6980782114808988	a98ddd14bf0bf22061d632	-10
8	-43.88754568839566	1.6980782114808988	3cbf6818a68f7ab9991514	-41
9	-58.17435547946095	1.7010983419195522	6f8633f5bfa5d26848668e 3d5ddd46958e97630410587c	-61
10	-43.58664906684732	1.7010983419195522	272bc6c25f5c5ee53f83c4 3a361fbc7cc91dc783e20a	-46
11	-34.70565203313315	1.7009387219711465	45443c59574c2c3b07e2e1 d9071e6d133dbe32754b0a	-34
12	-44.36009577368896	1.7009387219711465	6ac116ed60c258e2cbaeab 728c4823e6da36e18d08da 5d0cc104e21cc7fd1f5ca8 d9dbb675266c928448059e	-44
13	-21.783037079346236	1.6958406126012802	68163bc1e2cbf3e18e7426	-23
14	-39.68827784633828	1.6958406126012802	d6a1b51d76222a705a0259	-40
15	-18.488607061056847	1.6955259305261838	f0523bfaa8a394bf4ea5c1 0f842366fde286d6a30803	-22
16	-48.39610939101591	1.6955259305261838	87bd87e63374cee62127fc 6931104aab64f136a0485b	-50

2.8 PROVJERA POTPISA

Proces provjere potpisa je puno jednostavniji od generiranja para ključeva i generiranje potpisa. Za javni ključ $pk = h$, poruku m, potpis $sig = (r, s)$ i granicu β^2, prilikom provjere koristimo pk kako bi provjerili valjanost potpisa sig za poruku m prema sljedećim pravilima:

1. Vrijednost r (tzv. "*salt*") i poruka m su konkatenirani u *string* $(r||m)$ koji je) koji je *hashiran* u polinom $c \in \mathbb{Z}[x]/(\phi)$ prema funkciji *HashToPoint* (Algoritam 3)
2. s se dekodira (dekompresira) u polinom $s_2 \in \mathbb{Z}[x]/(\phi)$, koristeći funkciju *Decompress* (Algoritam 18)
3. Izračunavamo vrijednost $s_1 = c - s_2 h \ mod \ q$
4. Ako vrijedi $||(s_1, s_2)||^2 \leq \lfloor \beta^2 \rfloor$, onda je vrijednost potpisa valjan, inače se odbacuje.

Algoritam 16 $Verify(m, sig,\ pk, \lfloor\beta^2\rfloor)$

Ulaz: poruka m, potpis *sig*=(*r*, *s*), javni ključ $pk = h \in \mathbb{Z}[x]/(\phi)$ granica$\lfloor\beta^2\rfloor$
Izlaz: *prihvati* ili *odbaci*

1. $c \leftarrow HashToPoint(r||m, q, n)$
2. $s_2 \leftarrow Decompress(s, 8 \cdot sbytelen - 328)$
3. $ako\ (s_2 = \perp)$
4. odbaci (nevaljano kodiranje)
5. $s_1 = c - s_2 h\ mod\ q$ (s_1 treba biti normaliziran između $\left\lceil -\frac{q}{2} \right\rceil\ i\ \left\lfloor \frac{q}{2} \right\rfloor$)
6. *ako* ($||(s_1, s_2)||^2 \leq \lfloor\beta^2\rfloor$) (!razlog zašto je pri nekim promjenama potpis ispravan, a pri nekima ne)
7. *prihvati*
8. *inače*
9. *odbaci* (ako je potpis predug)

2.9 FORMATI ZA KODIRANJE

2.9.1 SAŽIMANJE GAUSSIANA

U FALCON-u, potpis se sastoji od polinoma $s \in \mathbb{Z}[x]/(\phi)$ čiji su koeficijenti distribuirani oko 0 prema Gaussovoj diskretnoj distribuciji standardne devijacije $\sigma \approx 1.55\sqrt{q} \ll q$. Naivno kodiranje *s* zahtijevalo bi otprilike $\lceil \log_2 q \rceil \cdot \deg(\phi)$ bitova, što je daleko od optimalnog za komunikacijsku složenost.

Opis procedure sažimanja jest:

1. Za svaki koeficijent s_i,sažeta poruka str_i definira se kao:
   a) prvi *bit* od $\boldsymbol{str}_i$ jest predznak $\boldsymbol{s}_i$
   b) idućih 7 bitova $\boldsymbol{str_i}$ su 7 najmanje značajnih bitova $|\boldsymbol{s_i}|$
   c) zadnji bitovi $\mathbf{str}_i$ su kodirani od najznačajnijih bitova $|\boldsymbol{s_i}|$ koristeći unarno (Huffmanovo) kodiranje. Ako vrijedi $\lfloor |s_i|/2^7 \rfloor = k$, onda vrijedi $\mathbf{0^k 1}$
2. Sažeta poruka $\boldsymbol{s}$ je konkatenirani string $\boldsymbol{s} \leftarrow (\boldsymbol{str_0}||\boldsymbol{str}_1\ ||\ ...\ ||\ \mathbf{str}_{n-1})$
3. **str** se nadopunjuje **nulama** do fiksne duljine **slen**

Algoritam 17 $Compress(s, len)$

Ulaz: polinom $\boldsymbol{s} = \sum s_i x^i \in \mathbb{Z}[x]$ stupnja < n, *string bitlength* **slen**
Izlaz: kompresiran prikaz ***str*** polinoma ***s*** duljine ***slen*** bitova, ili $\perp$

1. $str \leftarrow \{\}$
2. $za\ i\ od\ 0\ do\ n-1$
3. $str \leftarrow (str||b), gdje\ je\ b = 1\ ako\ s_i < 0, inače\ b = 0$ (kodiraj predznak od s_i)
4. $str \leftarrow (str||b_6 b_5 \dots b_0), gdje\ b_j = (|s_i| \gg j)\ \&\ 0x1$ (kodiraj binarno više bitove $|s_i|$)
5. $k \leftarrow |s_i| \gg 7$ (kodiraj unarno niže bitove $|s_i|$)
6. $str \leftarrow (str||0^k 1)$
7. $ako\ (|str| > slen)$
8. $str \leftarrow \perp$ (prekini ako je *str* predugačak)
9. $inače$
10. $str \leftarrow (str||0^{slen-|str|})$ (nadopuni nulama)
11. vrati ***str***

2.9.2 POTPISI

FALCON potpis sastoji se od dva *stringa r* i *s*. *Salt r* mora biti poznat prije početka *hashiranja* poruke, dok vrijednost *s* može biti verificirana tek nakon što je cijela poruka procesirana. Ipak, ovdje definiramo kodiranje koje uključuje *r* i *s*.

Prvi okret je zaglavlje sljedećeg formata (bitovi s lijeva značajniji): **0 c c 1 n n n n**

- bitovi ***cc*** su 01 ili 10 za određivanje metode kodiranja za **s**. Kodiranje 01 koristi Algoritam 17, dok se za 01 koristi alternativno nekompresirano kodiranje u kojem se svaki koeficijent za **s** kodira fiksnim brojem bitova.
- bitovi **nnnn** kodiraju vrijednost ℓ takav da je FALCON stupanj $n = 2^{\ell}, \ell \in [0, 10]$

Nakon okteta zaglavlja slijedi *nonce string r* (40 okteta), a zatim kodiranje samog **s**-a.

Potpisi se normalno *proširuju* s nulama do propisane duljine (sbytelen). Verifikatori mogu podržavati neproširene potpise, koji nemaju fiksnu veličinu, ali su (u prosjeku) nešto kraći nego proširени potpisi. *Djelomično* proširivanje nije valjano: ako potpis ima oktete proširenja, onda svi moraju biti jednaki nuli, i ukupna duljima mora biti jednaka **sbytelen.**

Pri korištenju nekompresiranog formata kodiranja (*cc* je 10 u zaglavlju), svi elementi od **s** kodiraju se s točno 12 bitova (*signed big-endian, koristi se dvojni komplement za negativne brojeve; valjani raspon je -2047 do +2047).* Ovaj format daje veće potpise i služi za neuobičajene situacije u kojima vrijednosti potpisa i potpisana poruka su tajni: nekompresiran potpis može biti dekodiran i kodiran u konstantnom vremenu bez curenja podataka.

Algoritam 18 $Decompress(str, len)$

Ulaz: *bitstring* $\boldsymbol{str} = (str[i]_{i=0,\dots,slen-1}$, *string bitlength* **slen**
Izlaz: polinom $\boldsymbol{s} = \sum s_i x^i \in \mathbb{Z}[x]$, ili $\perp$

1. $ako\ str\ \neq slen$ (osiguraj fiksnu duljinu)
2. $\quad vrati\ \perp$
3. $od\ i = 1\ do\ (n-1)$
4. $\quad s_1' \leftarrow \sum_{j=0}^{6} 2^{6-j} \cdot str[1+j]$ (oporavljamo najniže bitove $|s_i|$)
5. $\quad k \leftarrow 0$ (oporavljamo najviše bitove $|s_i|$)
6. $\quad dok\ je\ str[8+k] = 0$
7. $\quad\quad k \leftarrow k+1$
8. $\quad s_i \leftarrow (-1)^{str[0]} \cdot (s_1' + 2^7 k)$ (preračunavamo $|s_i|$)
9. $\quad$ ako $(s_i = 0)\ i\ (str[0] = 1)$ (nametnite jedinstveno kodiranje ako vrijedi)
10. $\quad\quad vrati\ \perp$
11. $\quad str \leftarrow str[9+k \dots \ell - 1]$ (uklanjamo bitove *str*-a koji kodiraju s_i)
12. $ako\ (str\ \neq 0^{|str|})$ (osiguraj da su bitovi na kraju nule)
13. $\quad vrati\ \perp$
14. $vrati\ s = \sum_{i=0}^{n-1} s_i x^i$

3. PROGRAM

3.1 UVOD

Ovdje možete preuzeti dokumentaciju algoritma, izvorni kod, implementaciju u C jeziku i Pythonu (koju sam koristio za lakšu implementaciju, radi grafičkog sučelja). Oprez, pratite istaknute upute za rad s *python* verzijom, koju možete pronaći u datoteci *README.md*.

3.2 IMPLEMENTACIJA

Kako bi mogli koristiti postojeću implementaciju moramo koristiti vanjsku knjižnicu *Crypto.Hash*, tj. iz nje *SHAKE256.* Upute za korištenje, kao i za instalaciju možete pogledati ovdje. Također, trebat će vam datoteka *main.py* koju sam napisao i koju možete dohvatiti ovdje, i koju trebate pohraniti u isti direktorij kao i prethodno pohranjenu implementaciju. Po potrebi, morat ćete instalirati NumPy, a upute možete pronaći ovdje.

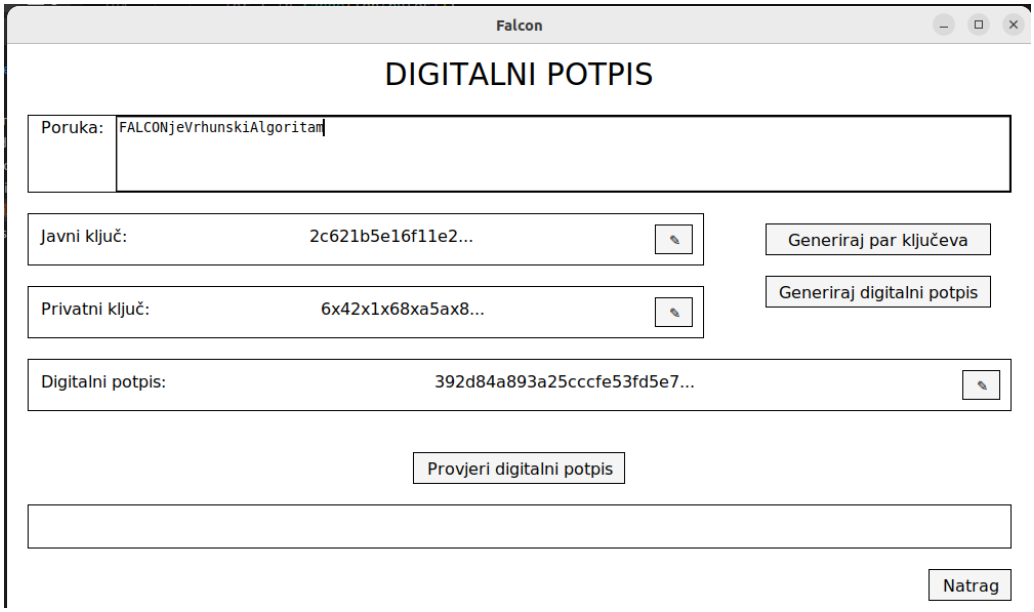
3.3 DEMONSTRACIJA

Program pokrećete iz direktorija u kojem se nalaze *main.py* i *falcon.py* koristeći naredbu:

python3 main.py

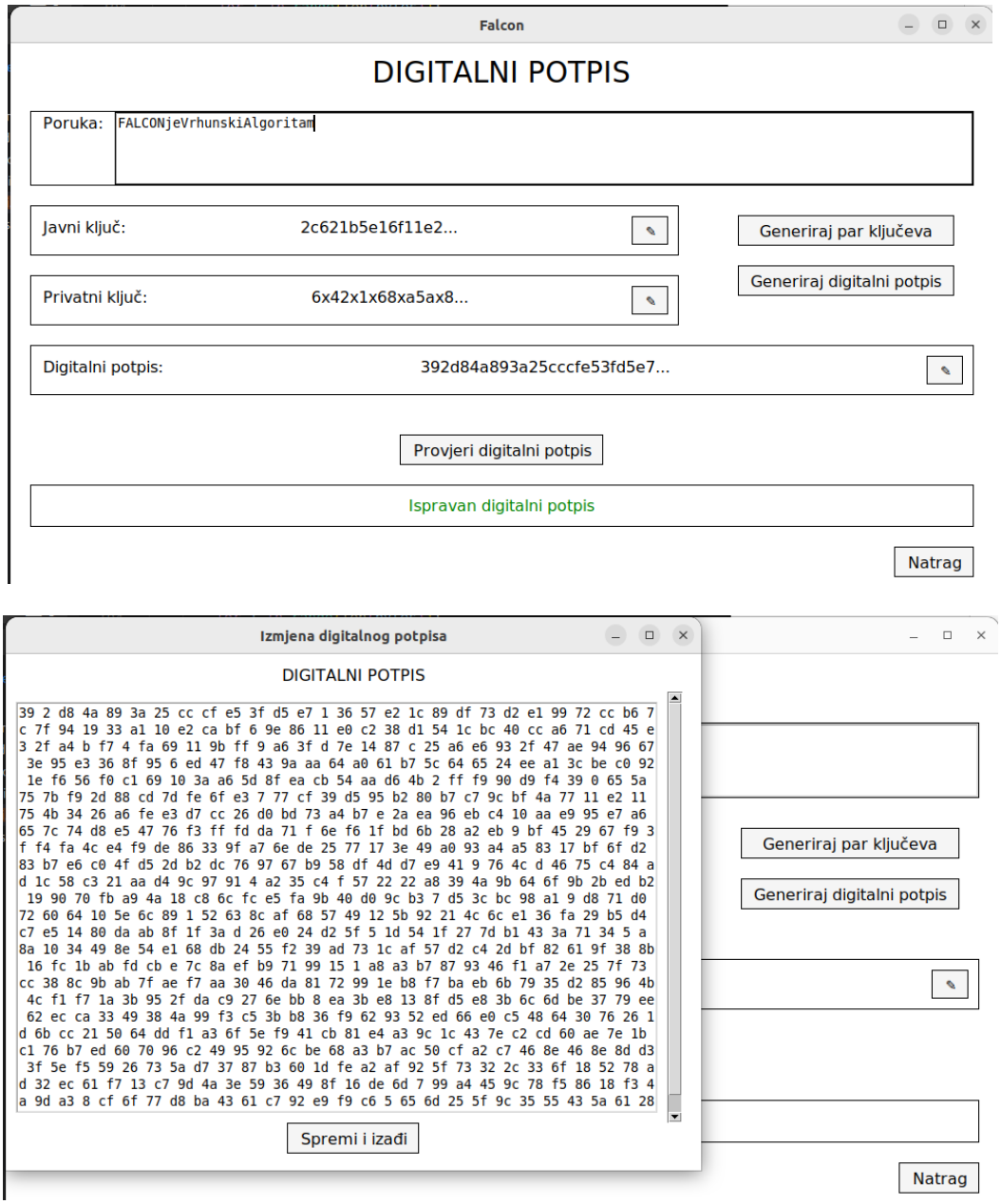
Prilikom pokretanja programa otvara se prozor:

Pritiskom na dio prozora u kojemu piše "DIGITALNI POTPIS" otvara se sljedeći prozor -> u polje "Poruka" upisuje se željena poruka za koju će se generirati digitalni potpis na osnovu generiranih ključeva.

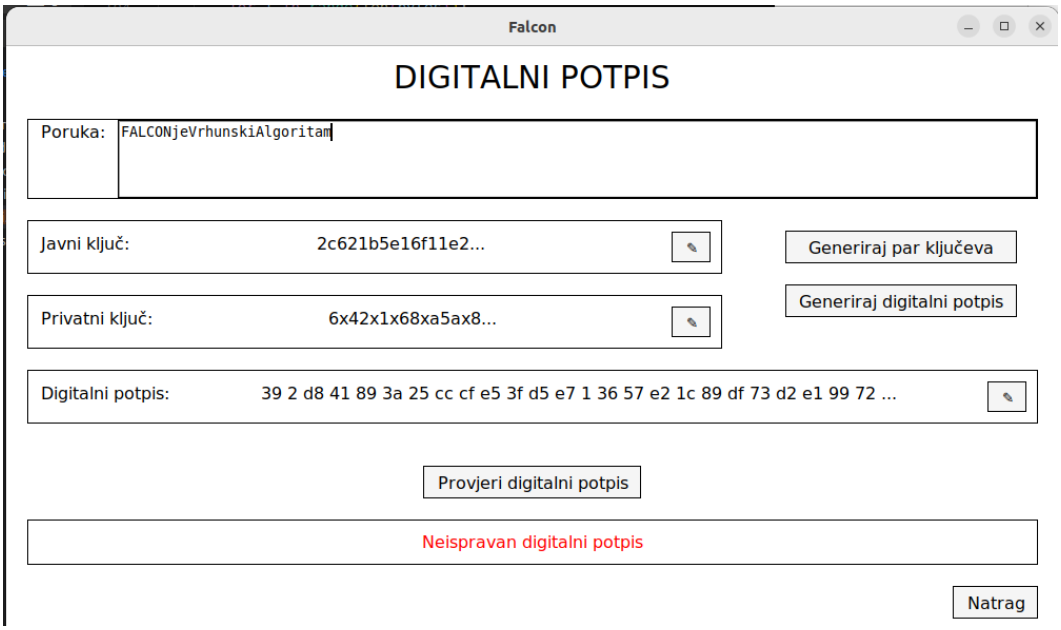


Poruka o ispravnosti digitalnog potpisa prikazuje se u doljnjem prozoru. Pritiskom na gumb olovčice u

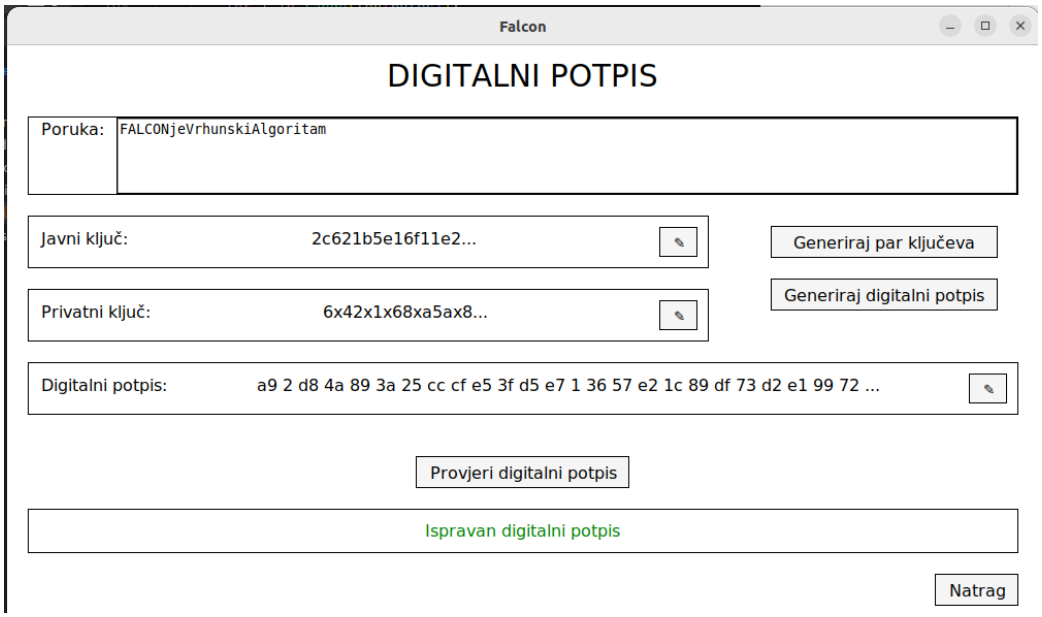
polju "Digitalni potpis" moguće je ručno promijeniti vrijednosti navedenog polja. Ukoliko se neka od tih vrijednosti promjeni ručno, provjera digitalnog potpisa biti će potencijalno neispravna.



Kao što možete primjetiti, promjena na 7. oktetu iz “a” u “1” rezultirala je neispravnim potpisom.



Dok je promjena na prvom oktetu iz "3" u "a" rezultirala ispravnim potpisom. To je očekivano ponašanje algoritma, objašnjenje je u algoritmu 16 pod korakom 6 (obratite pažnju na ispis konzole).



3.4 FUNKCIJE

3.4.1 POMOĆNE FUNKCIJE

- ntrugen(n) – izračunava polinome f, g, F i G te provjerava njihovu ispravnost
- repr(self) - prikaz objekta u čitljivom format (*self* predstavlja instancu klase)
- hash_to_point(self, message, salt) – *hashira* poruku i dodaje *salt*
- urandom – pomoćna funkcija za pseudoslučajne vrijednosti

3.4.2 GENERIRANJE KLJUČEVA

- **SecretKey.__init__(self, n, polys=none) – funkcija generira privatni ključ**
  - *n* je stupanj polinoma, preko njega izračunat je *phi*
  - *q* je prim broj, preporučeno 12289
  - polys su polinomi f, g, F i G dobiveni pozivom funkcije ntru_gen(n) unutar same funkcije

- **PublicKey.__init__(self, sk) – funkcija generira javni ključ**
  - *sk* je privatni ključ

3.4.3 GENERIRANJE POTPISA

- **SecretKey.sign(self, message, randombytes=urandom) – funkcija vraća generirani potpis**
  - *message* – poruka, mora biti *byte string* ili *byte polje*
  - procedura potpisivanja ponavlja se dok potpis nije dovoljno kratak

3.4.4 PROVJERA POTPISA

- **SecretKey.verify(self, message, signature) – provjera potpisa**
  - *message* – poruka
  - *signature* – potpis
  - ako je potpis ispravan vraća **true**, inače **false**

3.5 TEST

Kako bi provjerili ispravnost rada algoritma potrebno je usporediti izlaz iz algoritma s unaprijed određenim ulaznim vektorima (KAT). U tu svrhu koristite ćemo *makefile*, kojeg pokrećemo naredbom ***make test***.

4. ZAKLJUČAK

4.1 PREDNOSTI

Glavna prednost FALCON algoritma je njegova kompaktnosti, i izrađen je poglavito vodeći računa o tom kriteriju. Provjera ispravnosti potpisa je izrazito brza, ali čak i algoritam za potpisivanje može izvršiti 1000 potpisivanja na umjereno jakom računalu. Algoritam je modularan pa je moguće NTRU rešetke po potrebi zamijeniti drugim tipom rešetke ako je potrebno, isto tako možemo za uzorkovanje koristiti nešto drugo umjesto *Fast Fourier Sampling-a.* Potpisivanje sa oporavkom poruke i oporavkom ključa, te jednostavna provjera ispravnosti potpisa.

4.2 NEDOSTACI

Delikatna implementacija, potrebno netrivijalno razumijevanje matematičkih alata što je njegova najveća mana.

Također, koristi *floating-point* aritmetiku sa 53 bitnom preciznošću. što ne predstavlja poteškoću za softversku implementaciju, ali može biti veliki nedostatak pri ugrađivanju na ograničene uređaje.

5. LITERATURA

Autori: Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang
Ime rada: FALCON