Sveučilište u Zagreb
Fakultet elektrotehnike i računarstva
Projekt R

Postkvantna kriptografija – odabrani algoritmi
Tehnička dokumentacija

Studenti: Krunoslav Tomičić, Velimir Kovačić,

Slađan Tadić, Filip Penzar

Nastavnik: doc.dr.sc. Marin Golub

1. UVOD

1.1 POSTKVANTNA KRIPTOGRAFIJA

Kvantna računala, za razliku od onih klasičnih, koriste q-bite umjesto normalnih bitova. Prednost q-bita je što se u istom trenutku ne mora nalaziti samo u stanju 0 ili 1, već može biti u oba stanja ili nekoj njihovoj kombinaciji. Ova svojstva omogućuju kvantnim računalima izvođenje određenih algoritama (npr. Shor's algorithm) koji razbijaju trenutne metode kriptografije. S obzirom na povećanje broja q-bita u kvantnim računalima, postoji opasnost od efikasnog izvođenja tih algoritama i rušenja trenutne kriptografske infrastrukture. Iz tog se razloga javila potreba za razvojem kriptografskih algoritama otpornih na navedene napade.

1.2 NIST

NIST (*National Institute of Standards and Technology*) je američka državna organizacija kojoj je cilj standardizacija u tehnologiji. 2017. raspisala je natječaj kojemu je cilj bio pronalazak i odabir kriptografskih algoritama otpornih na napade kvantnim računalima. Kroz 4 eliminacijska kruga, 2022. objavili su popis od 4 pobjednička algoritma. Crystals-Kyber, Crystals-Dilithium, Falcon i Sphincs+.

U nastavku dokumenta je pobliže opisan način rada svakog od navedenih algoritama.

Postkvantna kriptografija – CRYSTALS - Kyber
Tehnička dokumentacija
Verzija 1.0

Student: Krunoslav Tomičić

Nastavnik: doc.dr.sc.Marin Golub

Sadržaj

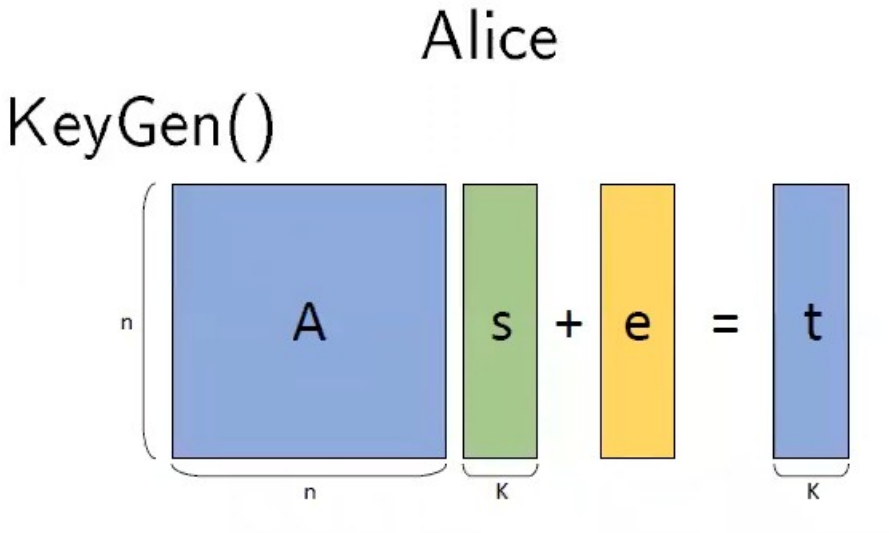
1. Uvod

1.1 CRYSTALS – Kyber

Kyber je kriptografski sustav razmjene ključeva i enkripcije koji se temelji na težini rješavanja problema učenja s greškama (LWE) preko modul rešetki. Osnovna jednadžba glasi **B = As + e** ***mod q****.* Finalist je NIST-ovog post-kvantnog kriptografskog natječaja.

1.2 Matematički opis enkripcije

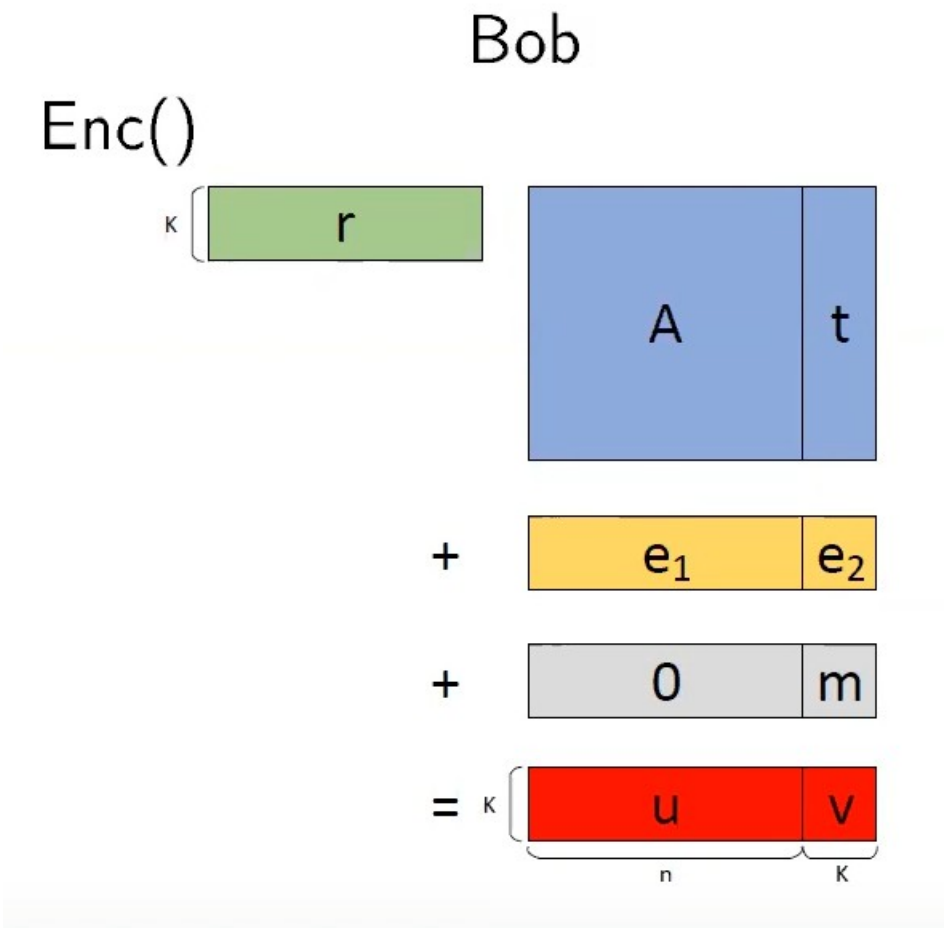
1.2.1 Generiranje ključa



Slika 1. Alice generira privatni i javni ključ

Alice generira matricu **A** dimenzija **n** x **n** nasumičnih koeficijenata, te dva vektora, **s** dimenzije **K** i vektor **e**, vrlo malih koeficijenata. Nakon toga računa vektor **t** također dimenzije **K**.Objavljuje javni ključ **(A,t)** te za sebe ostavlja privatni ključ **(s)**

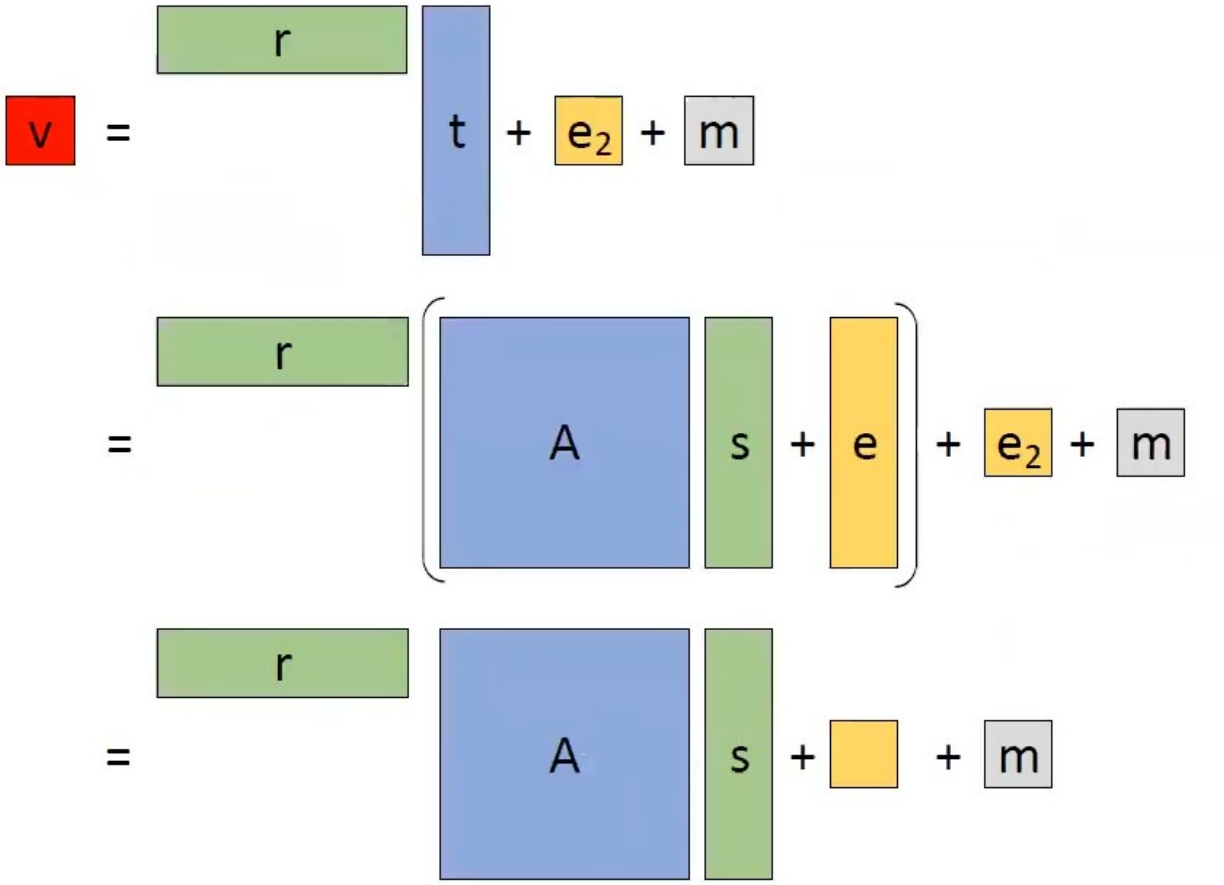
1.2.2 Enkripcija poruke



Slika 2. Bob enkriptira poruku

Bob generira nasumični vektor **r** duljine **K** te pomoću njega, javnog ključa dobivenog od Alice, malih error-a **e1** i **e2** te same poruke **m** dobiva šifru **(u, v)**.

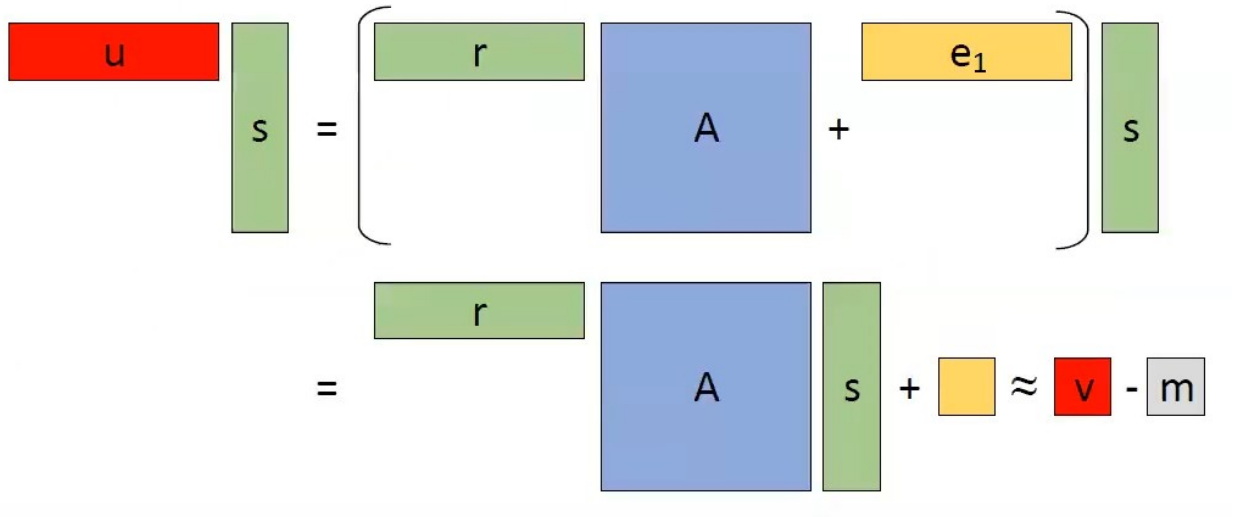
1.2.3 Računanje v dijela



Slika 3. Računanje međukoraka v

U ovom koraku **t** se rastavlja na svoje dijelove **A x s + e**. Množenjem svih komponenti s **r** dobivamo **rAs + re**. Imajući na umu da su koeficijenti greški **(e, e1, e2)** izrazito mali, oni ne pridonose mnogo cjelini. Onda je **r** x **e** vrlo mal i objedinjujemo ga s **e2** te i dalje imamo vrlo neznatan vektor. (Na slici žuti bez naziva). Ono što nam ostaje na kraju je **rAs + *greška* + m.**

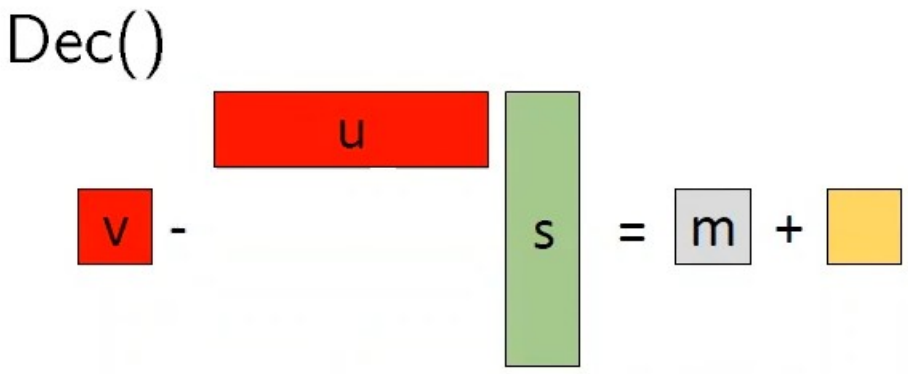
1.2.4 Računanje u x s dijela



Slika 4. Računanje međukoraka u x s

U se rastavlja na **r** x **A** + **e1** te se oba dijela vektorski množe sa **s**.
U prethodnom koraku smo se vidi da je **v** jednako **rAs + *greška* + m**.
No u ovom koraku dobivamo da je vektorski umnožak **u** x **s** jedak **rAs + e1s**.
Možemo prepoznati jednak dio obje jednadže - **rAs** i znajući da je **e1s** također vrlo malih koeficijenata govorimo kako je **u** x **s** aproksimativno jednako **v - m**!

1.2.5 Dekripcija poruke



Slika 5. Alice dekriptira poruku

Nakon što su prethodni koraci uspješno obavljeni sama dekripcija poruke postaje vrlo jednostavna. Oduzimanjem **u** x **s** od **v**. Uvrštavajući prethodno raspisane korake, poništavaju se izrazi **rAs** i ostaje samo **m + *greška*** gdje je ***greška*** dovoljno mala da ne utječe na originalnu poruku.

2. Algoritam

2.1 Parametri

Crystals-Kyber ostvaren je na nekoliko razina sigurnosti. Ovdje su navedeni odgovarajući setovi parametara. U astavku koristimo Kyber512 set parametara.

Set parametara	n	k	q	η1	η2	(d_u, d_v)	δ
Kyber512	256	2	3329	3	2	(10 , 4)	2^{-139}
Kyber768	256	3	3329	2	2	(10, 4)	2^{-164}
Kyber1024	256	4	3329	2	2	(11, 5)	2^{-174}

2.2 Funkcije enkapsulacije

2.2.1 Generiranje ključeva

KYBER.CCAKEM.KeyGen()

Izlaz: Javni ključ **pk** - B $(12 \cdot k \cdot n/8 + 32)$

Izlaz: Tajni ključ **sk** - B $(24 \cdot k \cdot n/8 + 96)$

```
z ← B^32
(pk, sk') := Kyber.CPAPKE.KeyGen()
sk := (sk' ‖ pk ‖ H(pk) ‖ z)
return (pk, sk)
```

Duljine uz definirane konstante (u bajtovima):
Javni ključ - **pk**: 800
Tajni ključ - **sk**: 1632

2.2.2 Enkapsulacija simetričnog ključa

Funkcija enkapsulacije ključa na ulazu uzima prethodno generirani javni ključ te pomoću njega vraća dva druga parametra. Šifru i zajedničku tajnu.Zajednička tajna je u ovom slučaju vektor duljine 32 bajta i predstavlja generirani simetrični ključkoji treba enkapsulirati u asimetričnom algoritmu. Šifra je enkapsulirani simetrični ključ.

KYBER.CCAKEM.Enc(pk)

Ulaz: Javni ključ **pk** - *B* $(12 \cdot k \cdot n/8 + 32)$

Izlaz: Šifra **c** - *B* $(d_u \cdot k \cdot n/8 + d_v \cdot n/8)$

Izlaz: Zajednička tajna **K** – B^*

```
m ← B^32
m ← H(m)
(K, r) := G(m ‖ H(pk))
c := Kyber.CPAPKE.Enc(pk, m, r)
K := KDF(K ‖ H(c))
return (c, K)
```

Duljine uz definirane konstante (u bajtovima):
Šifra - **c**: 768
Zajednička tajna - **K**: 32

2.2.3 Dekapsulacija simetričnog ključa

Funkcija dekapsulacije prima šifru te s pomoću tajnog ključa vraća dekapsuliranu zajedničku tajnu (ključ za simetričnu kriptografiju).

KYBER:CCAKEM:Dec(c, sk)
Ulaz: Šifra **c** - $B^{(d_u \cdot k \cdot n/8 + d_v \cdot n/8)}$
Ulaz: Tajni ključ **sk** - $B^{(24 \cdot k \cdot n/8 + 96)}$
Izlaz: Zajednička tajna **K** – B^*

```
pk := sk + 12 · k · n/8
h := sk + 24 · k · n/8 + 32 ∈ B^32
z := sk + 24 · k · n/8 + 64
m' := Kyber.CPAPKE.Dec(s, (u, v))
(K', r') := G(m' ‖ h)
c' := Kyber.CPAPKE.Enc(pk, m', r')
if c = c' then
return K := KDF(K' ‖ H(c))
else
return K := KDF(z ‖ H(c))
end if
return K
```

Duljine uz definirane konstante (u bajtovima):
Zajednička tajna - **K**: 32

2.3 Funkcije enkripcije

2.3.1 Generiranje ključeva

Funkcija generiranja ključeva vraća dva vektora različitih duljina.

KYBER.CPAPKE.KeyGen()

Izlaz: Tajni ključ **sk** - $B^{(12 \cdot k \cdot n/8)}$

Izlaz: Javni ključ **pk** - $B^{(12 \cdot k \cdot n/8 + 32)}$

```
d ← B^32
(ρ, σ) := G(d)
N := 0
for i from 0 to k - 1 do
  for j from 0 to k - 1 do
    Â[i][j] := Parse(XOF(ρ, j, i))
  end for
end for
for i from 0 to k - 1 do
  s[i] := CBD_η1(PRF(σ, N ))
  N := N + 1
end for
for i from 0 to k - 1 do
  e[i] := CBD_η1(PRF(σ, N ))
  N := N + 1
end for
ŝ := NTT(s)
ê := NTT(e)
t̂ := Â ∘ ŝ + ê
pk := (Encode_12(t̂ mod+q)‖ρ) . pk := As + e
sk := Encode_12(ŝ mod+q) . sk := s
return (pk , sk)
```

Duljine uz definirane konstante (u bajtovima):
Javni ključ - **pk**: 768
Tajni ključ - **sk**: 800

2.3.2 Enkripcija poruke

Za enkripciju poruke koristi se javni ključ I “salt”, što je vektor 32 nasumično odabrana bajta.
Bitno je naglasiti da je ulazna poruka također 32 bajtni vektor te ukoliko želimo enkriptirati poruku proizvoljne duljine,
moramo obaviti dopunu do duljine djeljive s 32 te šifrirati koristeći neki od poznatih modova operacije u blok kodovima.

KYBER.CPAPKE.Enc(pk ,m , r)

Ulaz: Javni ključ **pk** - *B* $(12 \cdot k \cdot n/8 + 32)$

Ulaz: Poruka **m** - *B* 32

Ulaz: Salt **r** - *B* 32

Izlaz: Šifra **c** - *B* $(d_u \cdot k \cdot n/8 + d_v \cdot n/8)$

```
N := 0
t̂ := Decode12(pk )
ρ := pk + 12 · k · n/8
for i from 0 to k - 1 do
  for j from 0 to k - 1 do
    Â^T [i][j] := Parse(XOF(ρ, i, j))
  end for
end for
for i from 0 to k - 1 do
  r[i] := CBD_η1(PRF(r, N ))
  N := N + 1
end for
for i from 0 to k - 1 do
  e1[i] := CBDη2 (PRF(r, N ))
  N := N + 1
end for
e2 := CBD_η1(PRF(r, N ))
r̂ := NTT(r)
u := NTT^-1(Â^T ◦ r̂) + e1
v := NTT^-1(t̂^T ◦ r̂) + e2 + Decompress_q(Decode_1(m), 1)
c1 := Encode_du(Compress_q(u, d_u))
c2 := Encode_dv(Compress_q(v, d_v))
return c = (c1 ‖ c2)
```

Duljine uz definirane konstante (u bajtovima):
Šifra - **c**: 768

2.3.3 Dekripcija poruke

Funkcija dekapsulacije prima šifru te s pomoću tajnog ključa vraća dekapsuliranu zajedničku tajnu (ključ za simetričnu kriptografiju).

KYBER.CPAPKE.Dec(sk, c)

Ulaz: Tajni ključ **sk** - $B^{(12 \cdot k \cdot n/8)}$

Ulaz: Šifra **c** - $B^{(d_u \cdot k \cdot n/8 + d_v \cdot n/8)}$

Izlaz: Poruka **m** - B^{32}

$u := Decompress_q(Decode_{du}(c), d_u)$
$v := Decompress_q(Decode_{dv}(c + d_u \cdot k \cdot n/8), d_v)$
$\hat{s} := Decode_{12}(sk)$
$m := Encode_1(Compress_q(v - NTT^{-1}(\hat{s}^T \circ NTT(u)), 1))$
return m

Duljine uz definirane konstante (u bajtovima):
Poruka - **K**: 32

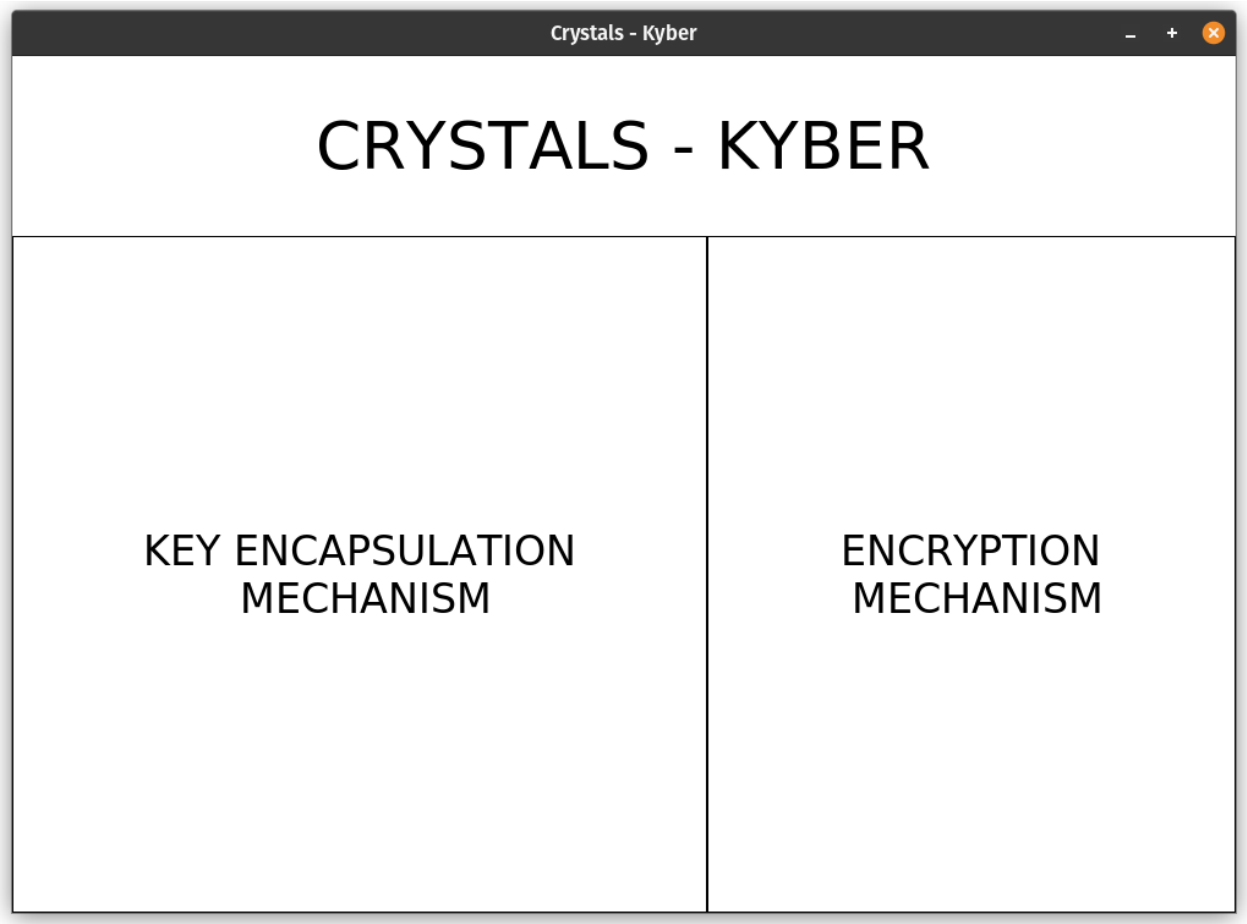
3. Demonstracijski program

3.1 Demonstracija

Sučelje za demonstraciju napravljeno je u programskome jeziku Python. Za ispravan rad potrebno je preuzeti datoteku **main.py** te ju pozicionirati u **/pyky.** Nakon toga pokretanjem naredbe **python3 main.py** iz **/pyky** direktorija pokreće se sučelje.

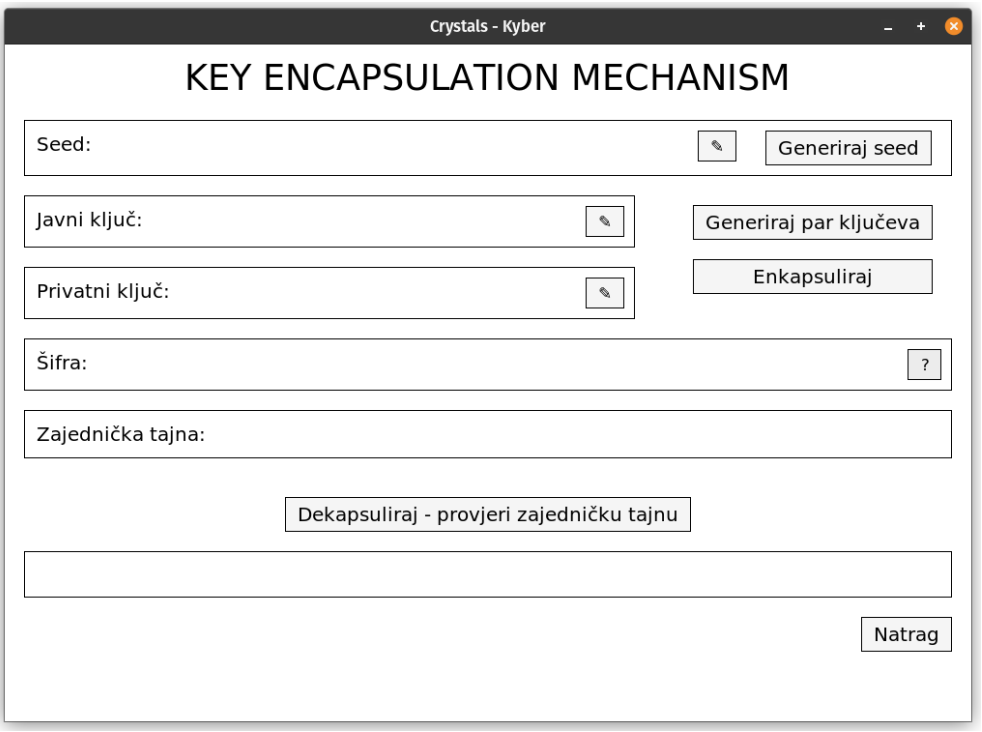
3.1.1 Početni zaslon

Pritiskom jednog od gumba “Key encapsulation mechanism” ili “Encryption mechanism” prelazimo na odgovarajuća sučelje

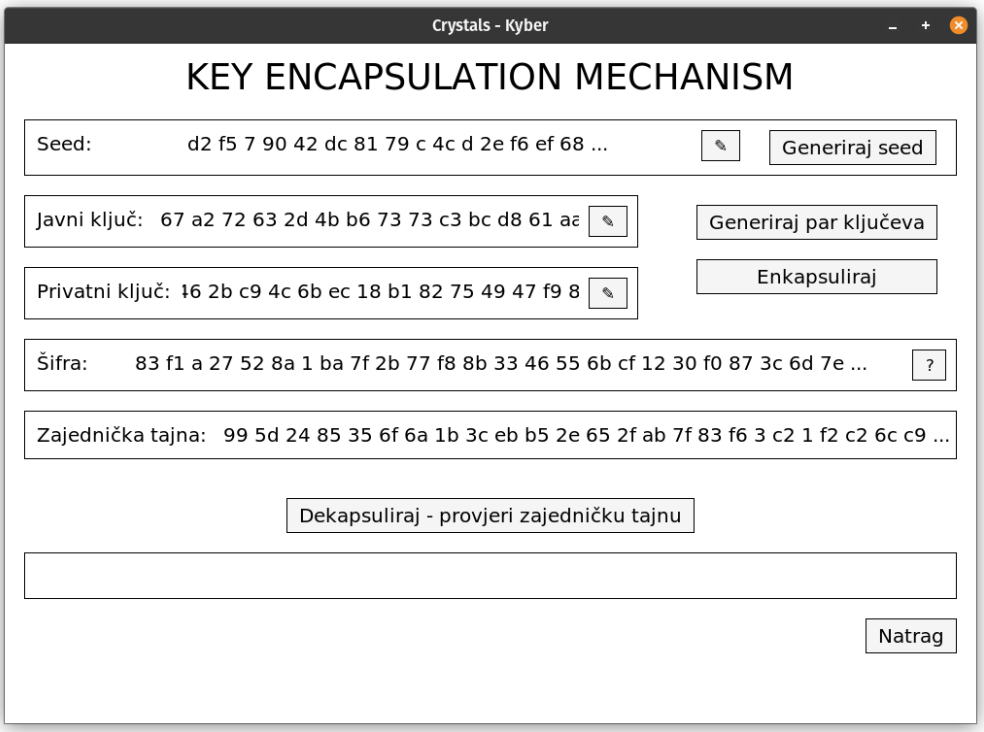


3.1.2 Sučelje “Key encapsulation”

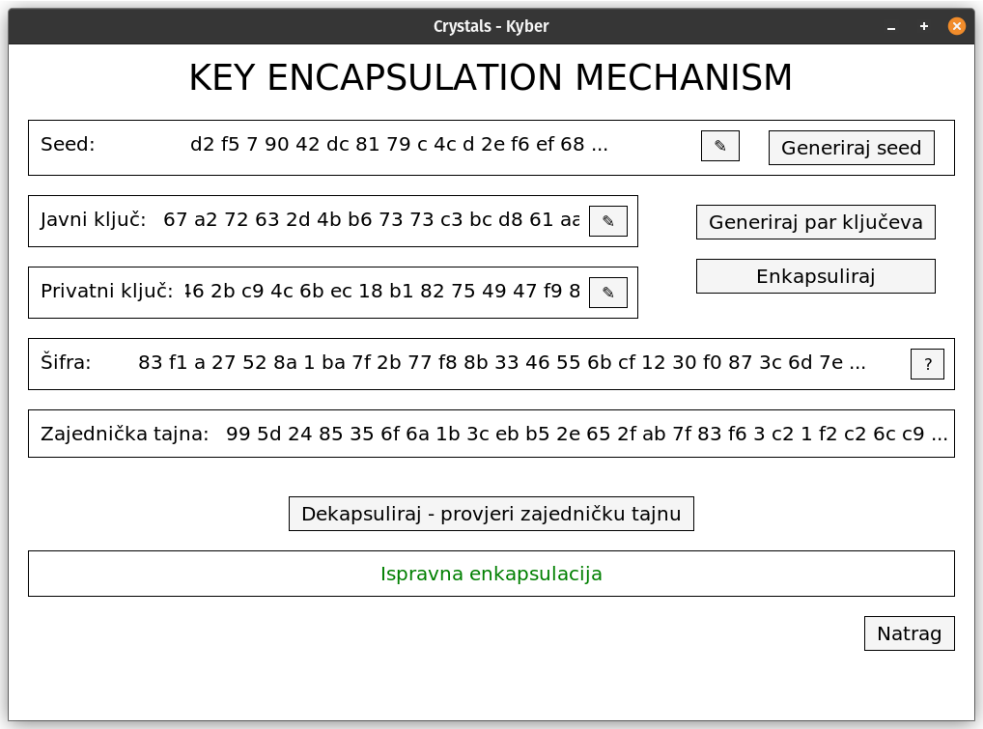
Pritiskom na

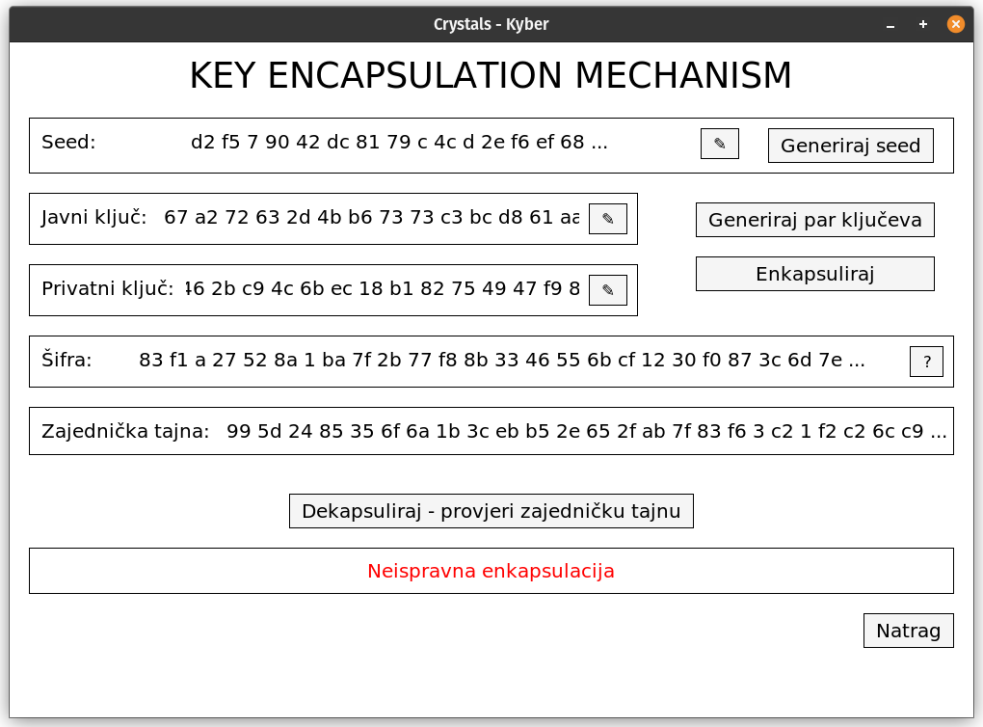


odgovarajuće gumbe možemo generirati seed i par ključeva (javni i privatni). Seed I ključeve možemo urediti klikom na olovku. Nakon generiranog seed-a I ključeva možemo enkapsulirati zajedničku tajnu (bajtovi za simetričnu enkripciju).

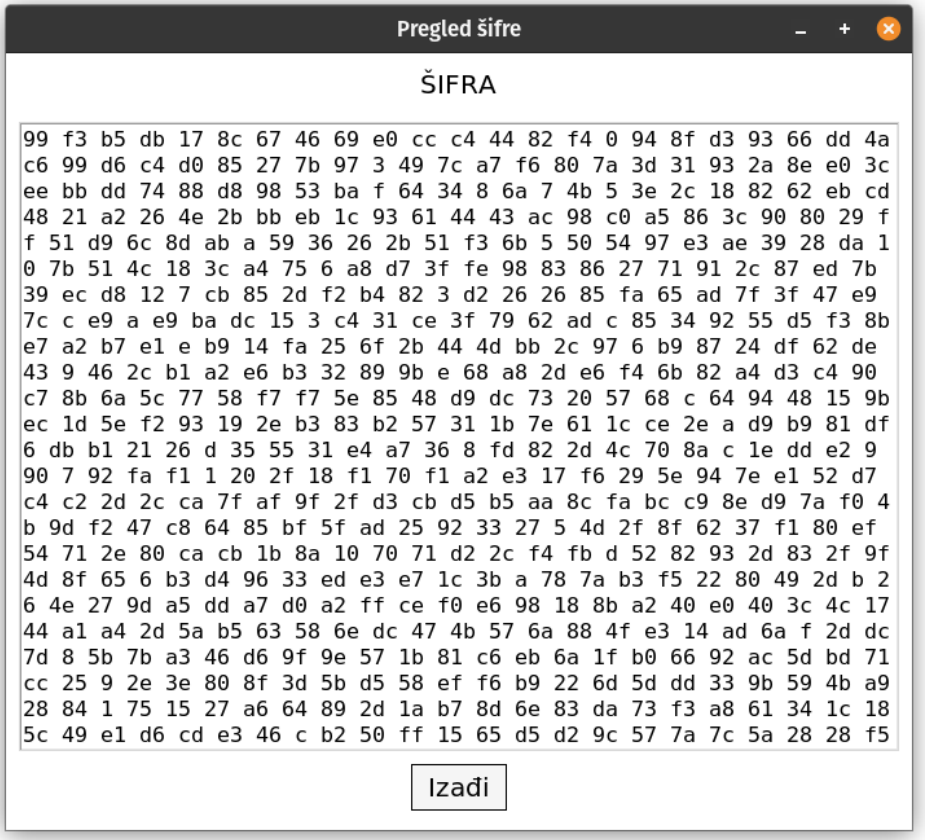


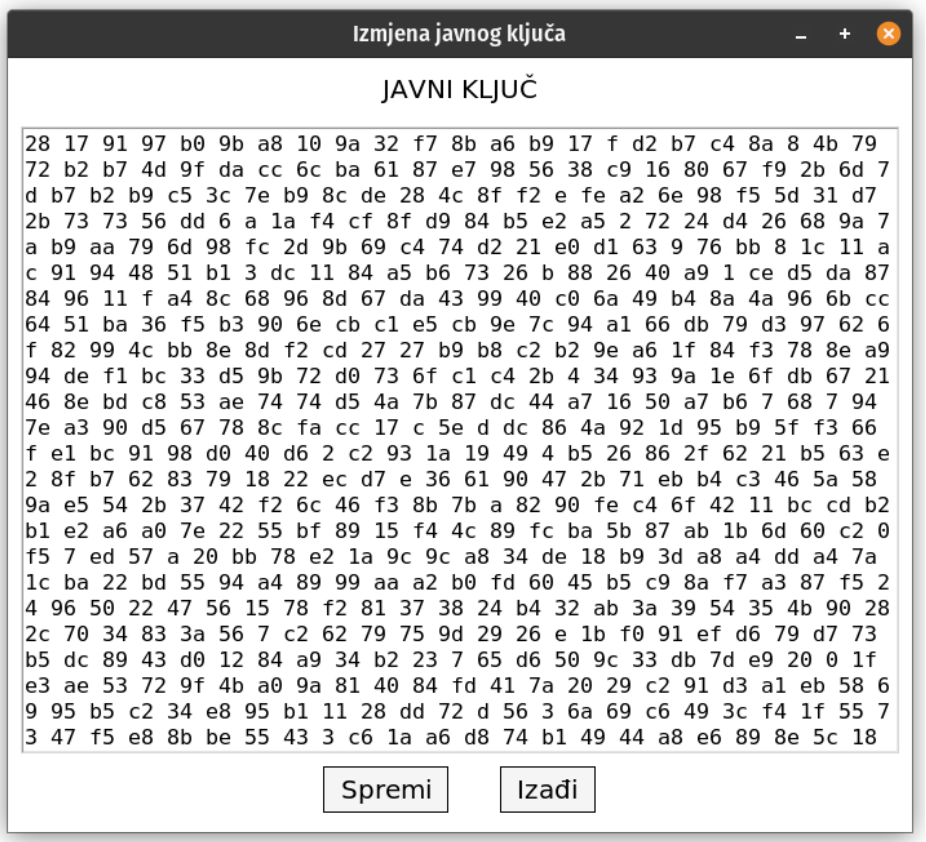
Ukoliko smo uspješno ili neuspješno dekapsulirali (nakon uređivanja kljueva) dobivamo odgovarajuće poruke:

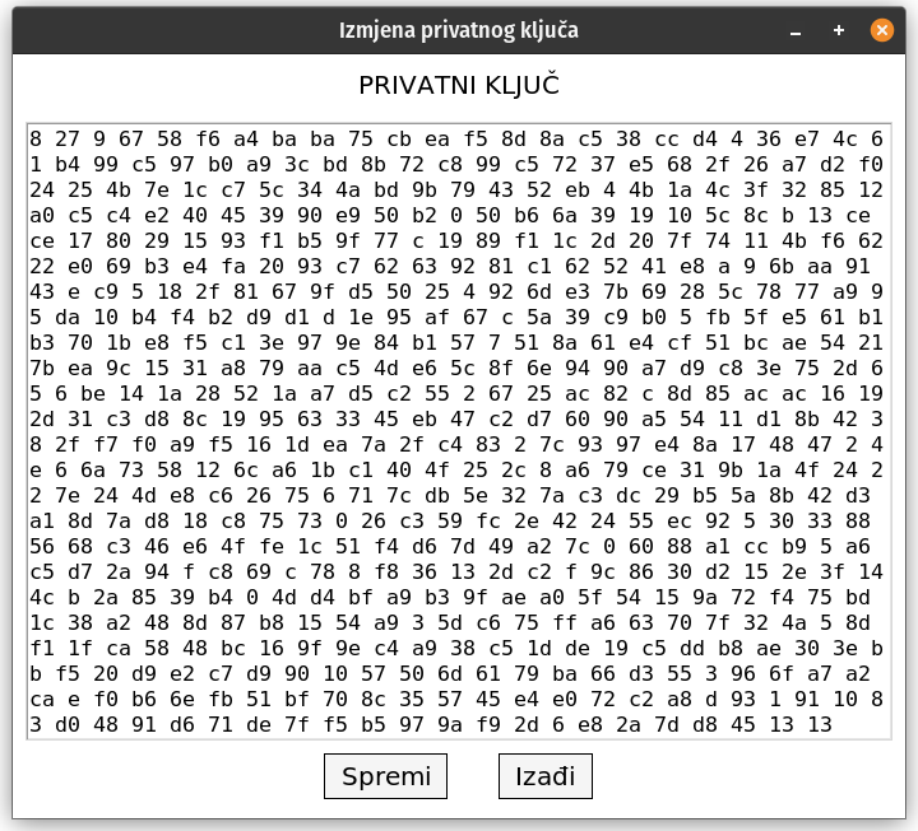




U nastavku su prozori za prikaz šifri te uređivanje javnog i privatnog ključa:

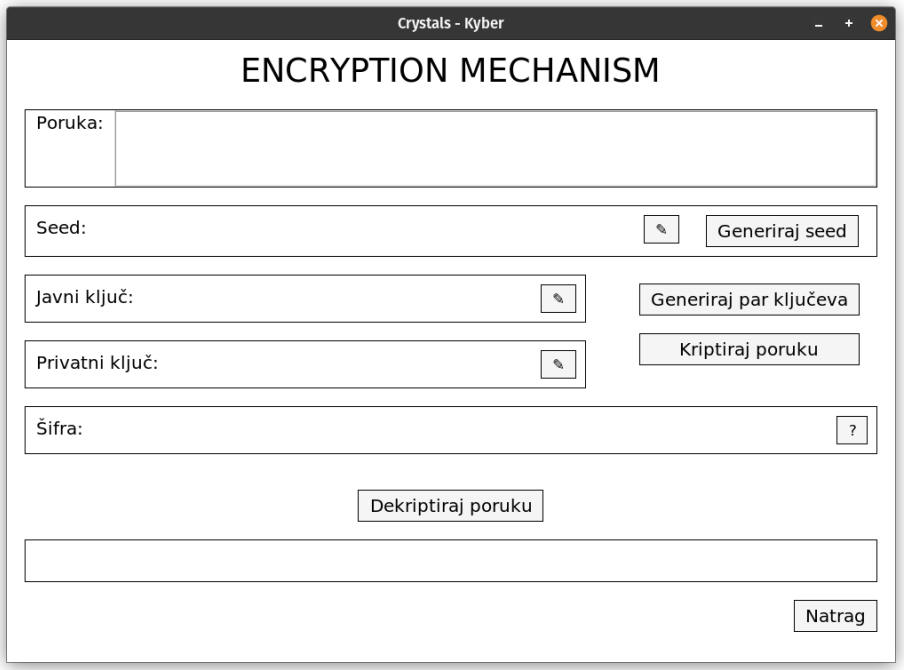


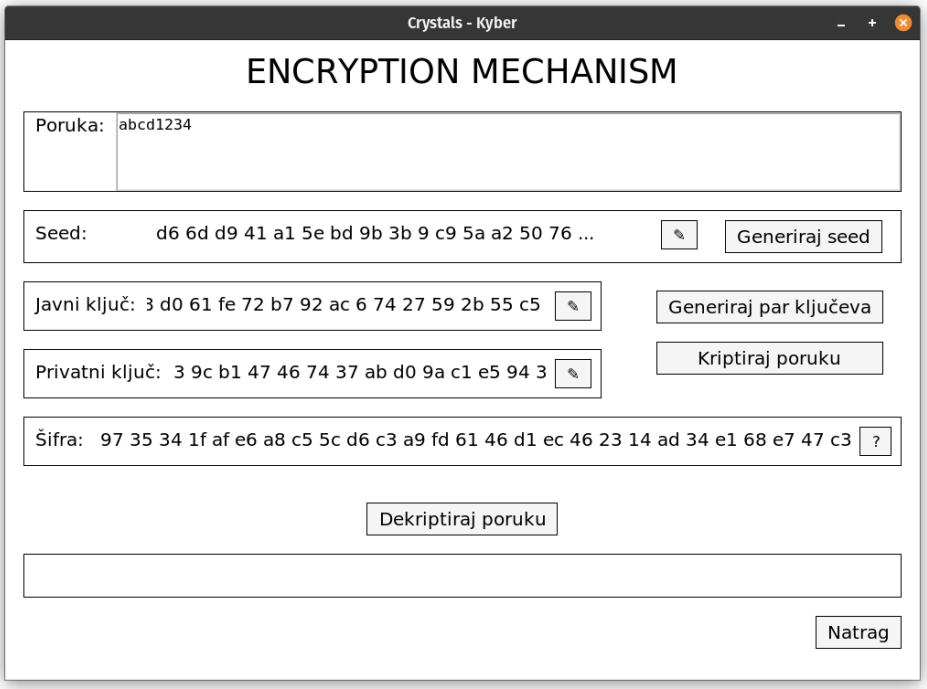


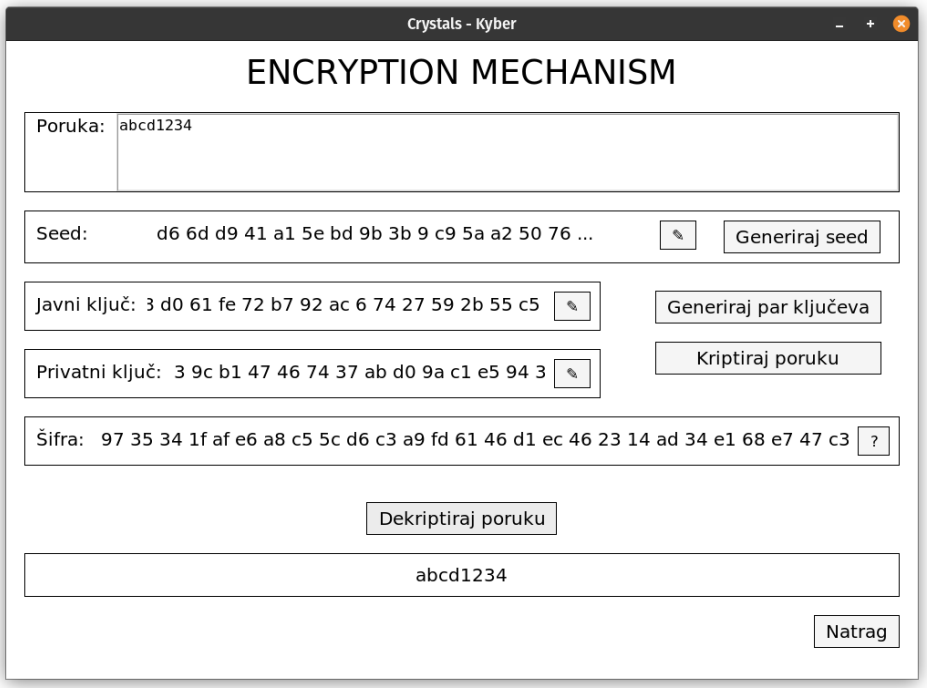


3.1.3 Sučelje „Key encryption”

Način rada programa za enkripciju poruke je vrlo sličan kao i prilikom enkapsulacije. Generiramo see, javni i privatni ključ, unosimo poruku te ju na kraju kriptiramo. Nakon toga ju dekrištiramo te provjeravamo uspješnost rada algoritma.




Crystals - Kyber
ENCRYPTION MECHANISM
Poruka:
abcd1234
Seed:
d6 6d d9 41 a1 5e bd 9b 3b 9 c9 5a a2 50 76 ...
Generiraj seed
Javni ključ:
3 d0 61 fe 72 b7 92 ac 6 74 27 59 2b 55 c5
Generiraj par ključeva
Privatni ključ:
3 9c b1 47 46 74 37 ab d0 9a c1 e5 94 3
Kriptiraj poruku
Šifra:
97 35 34 1f af e6 a8 c5 5c d6 c3 a9 fd 61 46 d1 ec 46 23 14 ad 34 e1 68 e7 47 c3
Dekriptiraj poruku
Natrag


Crystals - Kyber
ENCRYPTION MECHANISM
Poruka:
abcd1234
Seed:
d6 6d d9 41 a1 5e bd 9b 3b 9 c9 5a a2 50 76 ...
Generiraj seed
Javni ključ:
3 d0 61 fe 72 b7 92 ac 6 74 27 59 2b 55 c5
Generiraj par ključeva
Privatni ključ:
3 9c b1 47 46 74 37 ab d0 9a c1 e5 94 3
Kriptiraj poruku
Šifra:
97 35 34 1f af e6 a8 c5 5c d6 c3 a9 fd 61 46 d1 ec 46 23 14 ad 34 e1 68 e7 47 c3
Dekriptiraj poruku
abcd1234
Natrag

4. Literatura

- Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé
  CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation
- Hauke Malte Steffen, Lucie Johanna Kogelheide, Timo Bartkewitz
  In-depth Analysis of Side-Channel Countermeasures for CRYSTALS-Kyber Message Encoding on ARM Cortex-M4

Postkvantna kriptografija – CRYSTALS - Dilithium
Tehnička dokumentacija
Verzija 1

Student: Velimir Kovačić

Nastavnik: doc.dr.sc. Marin Golub

Sadržaj

1. Uvod

1.1 CRYSTALS – Dilithium

CRYSTALS – Dilithium postkvantni je algoritam digitalnog potpisa čija je težina utemeljena na težini pronalaženja najkraćih vektora u rešetkama (engl. *lattice*). Sama shema napravljena je po principu *Fiat-Shamir with Aborts*.

1.2 Matematička pozadina

Sve operacije s polinomima izvodit će se na prstenu polinoma $R_q = Z_q[X]/(X^n+1)$. Taj prsten sadrži sve polinome oblika $a_k X^k + a_{k-1} X^{k-1} + \ldots + a_1 X + a_0$, za koje vrijedi da im je stupanj manji od n, a koeficijenti manji od q što se osigurava time da se zbrajanje i množenje koeficijenata odvija pod $\bmod q$.

Dakle:

$$R_q = Z_q[X]/(X^n+1) = \left\{ a_k X^k + a_{k-1} X^{k-1} + \ldots + a_1 X + a_0 \middle| k < x \wedge a_i < q \right\}$$

Koristimo vrijednosti $q = 2^{23} - 2^{13} + 1 = 8380417$, $n = 256$.

R predstavlja isti prsten polinoma kao R_q, ali nema ograničenja na veličinu koeficijenata.

Maksimalni koeficijent polinoma je:

$$\|w\|_\infty = \max_i \left| w_i \bmod^{\pm} \alpha \right|$$

Gdje je w_i i-ti koeficijent polinoma w, a $\bmod^{\pm}$ centrirani modulo α, za $\alpha > 0$:

$$r' = r \bmod^{\pm} \alpha, r' \in \begin{cases} \left\langle \frac{-\alpha}{2}, \frac{\alpha}{2} \right], & \text{za parni } \alpha \\ \left[\frac{-\alpha-1}{2}, \frac{\alpha-1}{2} \right], & \text{za neparni } \alpha \end{cases}$$

S_η predstavlja skup svih elemenata iz prstena polinoma R kojima je maksimalni koeficijent polinoma manji od η:

$$S_\eta = \{ w \mid w \in R \wedge \|w\|_\infty < \eta \}$$

$\widetilde{S}_\eta$ predstavlja skup svih elemenata kao S_η, ali bez onih koji imaju koeficijente jednake η:

$$\widetilde{S}_\eta = \{ w \bmod^{\pm} 2\eta \mid w \in R \}$$

1.2 Fiat-Shamir with Aborts

GenerirajKlučeve():
    A ← $R_q^{k \times l}$
    (**s**$_1$, **s**$_2$) ← $S_\eta^l \times S_\eta^k$
    t := **A****s**$_1$ + **s**$_2$
    pk := (**A**, **t**)
    sk := (**A**, **t**, **s**$_1$, **s**$_2$)
    vrati (*pk*, *sk*)

Potpiši(*sk*, *M*):
    z := $\perp$
    dok je z = $\perp$:
        y ← $\widetilde{S}_{\gamma_1}^l$
        w := **Ay**
        w$_1$:= HighBits(**w**, $2\gamma_2$)
        $c \in B_\tau$:= H(*M* ‖ **w**$_1$)
        z := **y** + c**s**$_1$

        ako je $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **ili** $\|$LowBits(**Ay** − c**s**$_2$, $2\gamma_2$)$\|_\infty \geq \gamma_2 - \beta$:
            z := $\perp$
    vrati σ = (**z**, c)

ProvjeriPotpis(*pk*, *M*, σ = (**z**, c)):
    w'$_1$:= HighBits(**Az** − c**t**, $2\gamma_2$)
    ako je $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ **i** c == H(μ ‖ **w**'$_1$):
        vrati potpis je valjan
    inače:
        vrati potpis nije valjan

U funkciji GenerirajKlučeve(), koja ne prima nikakve argumente, prvo se stvori matrica **A** veličine $k \times l$ u čijoj je svakoj ćeliji polinom iz prstena polinoma R_q. Zatim se nasumično generiraju 2 vektora polinoma **s**$_1$ duljine l i **s**$_2$ duljine k, tako da su koeficijenti svih polinoma najviše η. Vektor polinoma **t** dobije se množenjem matrice **A** s vektorom polinoma **s**$_1$ i pribrajanjem vektora polinoma **s**$_2$. Javni ključ *pk* sastoji se od matrice **A** i vektora polinoma **t**, a privatni ključ *sk* se sastoji od matrice **A**, vektora polinoma **t**, vektora polinoma **s**$_1$ i vektora polinoma **s**$_2$. Funkcija vraća javni i privatni ključ.

Funkcija Potpiši(*sk*, *M*) kao ulazne argumente prima privatni ključ *sk* i poruku *M*. Vektor polinoma **z** postavi se na vrijednost $\perp$ što predstavlja vrijednost *none*. Petlja se provodi dok je vektor polinoma **z** jednak $\perp$. Vektor polinoma **y** duljine l nasumično se generira tako da su svi

koeficijenti manji od γ_1. **w** je vektor polinoma jednak umnošku matrice polinoma **A** i vektora polinoma **y**. $\mathbf{w}_1$ je vektor polinoma, koji se dobije uzimanjem samo visokih bitova svih koeficijenata svih polinoma vektora **w**. Koeficijenti se rastavljaju na više i niže bitove prema formuli $\mathbf{w} = \mathbf{w}_1 \cdot 2\gamma_2 + \mathbf{w}_0$, $\mathbf{w}_1$ predstavlja više, a $\mathbf{w}_0$ niže bitove, vrijedi $|\mathbf{w}_0| \leq \gamma_2$. c iz R_q je takav polinom da ima točno τ koeficijenata jednakih 1, dok su svi ostali jednaki 0. Dobije se primjenom hash funkcije nad konkatiniranom porukom M s vektorom polinoma $\mathbf{w}_1$. Vektor polinoma **z**, koji predstavlja potencijalni digitalni potpis, računa se kao zbroj vektora polinoma **y** i umnoška polinoma c s vektorom polinoma $\mathbf{s}_1$. Ako je najveći koefcijent polinoma vektora polinoma **z** veći ili jednak $\gamma_1 - \beta$ ili su niski bitovi $\mathbf{Ay} - c\mathbf{s}_2$ veći ili jednaki $\gamma_2 - \beta$, vektor polinoma **z** postavlja se na početnu vrijednost $\perp$ te se tako ponavlja petlja. Kada se pronađe vektor **z** i polinom c takvi da zadovolje uvjete vraća se potpis σ koji sadrži samo njih.

Za provjeru ispravnosti digitalnog potpisa koristi se funkcija ProvjeriPotpis(*pk*, *M*, $\sigma = (\mathbf{z}, c)$)
koja kao argumente prima javni ključ *pk*, poruku *M* i digitalni potpis σ. Vektor polinoma $\mathbf{w}'_1$ računa se kao visoki bitovi od $\mathbf{Az} - c\mathbf{t}$.

Ukoliko je potpis ispravan vrijedi: $\mathbf{w}'_1 = \text{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2) = \text{HighBits}(\mathbf{A}(\mathbf{y} + c\mathbf{s}_1) - c(\mathbf{As}_1 + \mathbf{s}_2), 2\gamma_2) = \text{HighBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2) = \text{HighBits}(\mathbf{Ay}) = \text{HighBits}(\mathbf{w}) = \mathbf{w}_1$.

Ako je $\mathbf{w}'_1$ jednak $\mathbf{w}_1$ iz funkcije Potpiši(*sk*, *M*), hash koji daje konkatiniran uz M biti će jednak polinomu c. Uz to se provjerava i svojstvo $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$, koje ima svaki vektor polinoma **z** koji se pojavi kao dio digitalnog potpisa na izlazu funkcije Potpiši(*sk*, *M*). Ako su ova dva uvjeta zadovoljena digitalni je potpis ispravan.

1.3 Dilithium poboljšanja

Algoritam CRYSTALS – Dilithium na osnovni princip *Fiat-Shamir with Aborts* uvodi određena poboljšanja i optimizacije.

1.3.1 Spremanje matrice **A**

Spremanje matrice **A** zauzimalo bi puno prostora jer se sastoji od $k \times l$ polinoma stupnja do 255. Umjesto eksplicitnog spremanja matrice **A**, sprema se *seed* ρ iz kojeg se hash funkcijom matrica generira kada god je to potrebno.

1.3.2 Spremanje vektora polinoma **t**

Problem s vektorom polinoma **t** je što zauzima mnogo memorije, a sprema se i u javnom i u privatnom ključu. Budući da rezultat $\mathbf{Az} - c\mathbf{t}$ ne ovisi

previše o nižim bitovima **t** možemo ih isključiti iz javnog ključa, to jest spremati samo više bitove **t**. Nastaje problem u tome što sada ne možemo uvijek točno izračunati **Az** – c**t** jer ne uključujemo niže bitove **t**-a iz kojih bi se mogli pojaviti prijenosi. To rješavamo stvaranjem vektora *hintova* **h** u kojem pohranjujemo informacije o prijenosima.

1.3.3 Determinizam

Dodaje se mogućnost determinističkog generiranja potpisa determinističkim generiranjem vektora polinoma **y** što će ujedno biti i zadana postavka.

1.3.4 Oblik polinoma NTT

Zahvaljujući Kineskom teoremu o ostacima možemo podijeliti prsten polinoma R_q na manje prstene polinoma koji u umnošku daju polinom jednak onom koji smo dijelili, istu stvar možemo i s polinomima unutar tog prstena. Pretvaranje polinoma u taj oblik naziva se NTT (eng. *Number Theoretic Transform*). Funkciju pretvaranja iz kanonskog u oblik NTT označavamo s NTT(), a funkciju za pretvorbu iz oblika NTT u kanonski oblik zovemo NTT^{-1}(). Polinomi u obliku NTT mogu se pomnožiti u vremenskoj složenosti $O(n)$, a pretvorba u oblik NTT odvija se u vremenskoj složenosti $O(\log n)$. Dakle, umjesto množenja polinoma u kanonskom obliku u vremenskoj složenosti $O(n^2)$, možemo ih pomnožiti u vremenskoj složenosti O($n \log n$).

$$\hat{a} = \mathrm{NTT}(a)$$

$$ab = NTT^{-1}(\hat{a}\,\hat{b})$$

$$ab = \mathrm{NTT}^{-1}\text{¿}$$

3. Algoritam

3.1 Važne funkcije

3.1.1 Hashing to a Ball

Funkcija SampleInBall(ρ) vraća jedan element iz skupa B_τ.

B_τ predstavlja skup svih elemenata iz R koji imaju točno τ koeficijenata koji su ili 1 ili -1, dok su svi ostali koeficijenti 0. Takvih elemenata u R ima $2^\tau \binom{256}{\tau}$.

Za funkciju SampleInBall(ρ) potreban je hash koja hashira na skup B_τ u 2 koraka:

1. *Second pre-image resistant* hash funkcija koja mapira niz bitova proizvoljne duljine na niz bitova duljine 256.
2. *Extendable-output* (XOF) hash funkcija koja koristi izlaz 1. koraka kao *seed*

SampleInBall(ρ):
Inicijaliziraj **c** = $c_0 c_1 \ldots c_{255} = 00\ldots0$
za i := 256 - τ do 255
j := broj iz skupa $\{0,1,\ldots,i\}$ odabran koristeći XOF funkciju
s := broj iz skupa $\{0,1\}$ odabran koristeći XOF funkciju
$c_i := c_j$
$c_j := (-1)^s$
vrati c

3.2.2 Ekspanzija matrice A

ExpandA(ρ)

Kako se u ključevima ne bi morala pohranjivati cijela vrijednost matrice **A**, pohranjuje se *seed* ρ i svaki put kada je potrebno generira se matrica $\hat{\mathrm{A}}$ u obliku NTT.

3.1.3 Generiranje vektora y

ExpandMask(ρ', κ)

Koristeći seed ρ i nonce κ, ova funkcija generira vektor duljine l koji se sastoji od polinoma iz S_{γ_1}.

3.1.4 Hash otporan na kolizije

CRH(δ)

Hash otporan na kolizije mapira ulaz δ u niz bitova duljine 384.

3.1.5 Rastavljanje brojeva

Power2Roundq(r, d)

Za binarni broj$r$ duljine n bitova, funkcija Power2Round$_q(r, d)$ vraća par u kojem je prvi član prvih (viših) $n-d$ bitova, a drugi član zadnjih (nižih) d bitova. $r=[(r-r_0)/2^d]*2^d+r_0$.

Power2Round$_q(r, d)$:
    $r := r \bmod^+ q$
    $r_0 := r \bmod^\pm 2^d$
    vrati $((r-r_0)/2^d, r_0)$

Decompose$_q(r, \alpha)$

Slično kao funkcija Power2Round$_q(r, \alpha)$, funkcija Decompose$_q(r, \alpha)$ rastavlja binarni broj na više i niže bitove (r_1 i r_0) i vraća ih. Koristeći α (parni djelitelj broja $q-1$) dijeli ih na način da $r=\alpha \cdot r_1+r_0$.

Decompose$_q(r, \alpha)$:
    $r := r \bmod^+ q$
    $r_0 := r \bmod^\pm \alpha$
    ako je $r - r_0 = q - 1$:
        $r_1 := 0$
        $r_0 := r_0 - 1$
    inače:
        $r_1 := (r - r_0)/\alpha$
    vrati (r_1, r_0)

HighBits$_q(r, \alpha)$ i LowBits$_q(r, \alpha)$

Pomoćne funkcije koje služe za dohvaćanja viših i nižih bitova koje vraća funkcija Decompose$_q(r, \alpha)$.

HighBits$_q(r, \alpha)$:
    $(r_1, r_0) := \text{Decompose}_q(r, \alpha)$
    vrati r_1

LowBits$_q(r, \alpha)$:
    $(r_1, r_0) := \text{Decompose}_q(r, \alpha)$
    vrati r_0

3.1.6 Hintovi

MakeHint$_q(z, r, \alpha)$

Cilj stvaranja hinta je da ne moramo spremati cijeli broj nego samo njegove visoke bitove, a *hint* će nam otkriti postoji li prijenos pri zbrajanju. Konkretno, ispituje postoji li prijenos s nižih bitova pri zbrajanu r sa z.

MakeHint$_q(z, r, \alpha)$:
    r_1 := HighBits$_q(r, \alpha)$
    v_1 := HighBits$_q(r + z, \alpha)$
    ako $r_1 == v_1$:
        vrati 0
    inače:
        vrati 1

UseHint$_q(h, r, \alpha)$

Funkcija UseHint$_q(h, r, \alpha)$ na temelju *hinta* h koji može biti 0 ili 1 ispravlja i vraća više bitove broja r.

UseHint$_q(h, r, \alpha)$:
    $m := (q-1)/\alpha$
    (r_1, r_0) := Decompose$_q(r, \alpha)$
    ako $h = 1$ i $r_0 > 0$:
        vrati $(r_1 + 1) \bmod^+ m$
    inače ako $h = 1$ and $r_0 \leq 0$:
        vrati $(r_1 - 1) \bmod^+ m$
    inače:
        vrati r_1

Leme vezane uz korištenje i stvaranje *hintova*

1. UseHint$_q$(MakeHint$_q(z, r, \alpha), r, \alpha)$ = HighBits$_q(r + z, \alpha)$.
    a. Stvara se *hint* na temelju zbroja z i r i koristi se na visokim bitovima r što je jednako visokim bitovima zbroja r i z.
2. **Ako** $\|s\|_\infty \leq \beta$ i $\|\text{LowBits}_q(r, \alpha)\|_\infty < \alpha/2 - \beta$ **vrijedi** HighBits$_q(r, \alpha)$ = HighBits$_q(r + s, \alpha)$
    a. Ova lema daje dovoljan uvjet za slučaj da se neće mijenjati visoki bitovi r kada mu se pribroji s.
3. **Neka je** (r_1, r_0) = Decompose$_q(r, \alpha)$, (w_1, w_0) = Decompose$_q(r + s, \alpha)$ i $\|s\|_\infty \leq \beta$ **vrijedi:**
   $\|s + r_0\|_\infty < \alpha/2 - \beta$ **ako i samo ako** $w_1 = r_1$ i $\|w_0\|_\infty < \alpha/2 - \beta$

3.2 Važni parametri

NIST Security Level	2	3	5
Parameters			
q [modulus]	8380417	8380417	8380417
d [dropped bits from $\mathbf{t}$]	13	13	13
τ [# of ± 1's in c]	39	49	60
challenge entropy $[\log \binom{256}{\tau} + \tau]$	192	225	257
γ_1 [$\mathbf{y}$ coefficient range]	2^{17}	2^{19}	2^{19}
γ_2 [low-order rounding range]	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
(k, ℓ) [dimensions of $\mathbf{A}$]	$(4, 4)$	$(6, 5)$	$(8, 7)$
η [secret key range]	2	4	2
β $[\tau \cdot \eta]$	78	196	120
ω [max. # of 1's in the hint $\mathbf{h}$]	80	55	75

U ovisnosti o NIST razini sigurnosti koriste se različite vrijednosti ključnih parametara.

3.3 Generiranje ključeva

GenerirajKljučeve():

$\zeta \leftarrow \{0, 1\}^{256}$
$(\rho, \varsigma, K) \in \{0, 1\}^{256\times3} := \mathrm{H}(\zeta)$
$(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k := \mathrm{H}(\varsigma)$

$\hat{\mathbf{A}} := \mathrm{ExpandA}(\rho)$
$\mathbf{t} := \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \cdot \mathrm{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$
$(\mathbf{t}_1, \mathbf{t}_0) := \mathrm{Power2Round}_q(\mathbf{t}, d)$
$tr \in \{0, 1\}^{384} := \mathrm{CRH}(\rho \parallel \mathbf{t}_1)$

$pk := (\rho, \mathbf{t}_1)$
$sk := (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
vrati (pk, sk)

Mala masna slova označavaju vektore, a velika masna slova označavaju matrice. Slova s kapicom označavaju oblik NTT. H je SHAKE-256 hash funkcija.

Funkcija GenerirajKljučeve() na ulazu ne prima ništa.

Prvo se generira ζ, nasumični 256-bitni broj. Na temelju hasha od ζ se generiraju brojevi ρ, ς i K. Hashiranjem ς generiraju se vektori: $\mathbf{s}_1$, duljine l i $\mathbf{s}_2$, duljine k. Koeficijenti ovih vektora su polinomi iz S_η. Iz broja ρ generira se i privremeno sprema $\hat{\mathbf{A}}$ (oblik NTT matrice $\mathbf{A}$). Svako polje matrice $\mathbf{A}$ polinom je iz prstena polinoma R_q, a sama matrica veličine je $k \times l$, dakle $\mathbf{A} \in R_q^{k\times l}$. Računa se vektor polinoma $\mathbf{t}$ koristeći pretvorbu NTT te se potom

dijeli na više bitove $\mathbf{t}_1$ i niže bitove $\mathbf{t}_0$ funkcijom Power2Round$_q$($\mathbf{t}$, d). Hash koji na ulazu prima konkatiniran broj ρ i $\mathbf{t}_1$ (više bitove vektora $\mathbf{t}$) na izlazu daje broj tr.

Javni ključ *pk* sastoji se od broja ρ i $\mathbf{t}_1$ (viših bitova vektora $\mathbf{t}$). Privatni ključ *sk* sastoji se od brojeva ρ, K i tr i vektora $\mathbf{s}_1$, $\mathbf{s}_2$ i $\mathbf{t}_0$ (viših bitova vektora $\mathbf{t}$).

Funkcija vraća javni i privatni ključ.

3.4 Potpisivanje

Potpiši(*sk*, *M*):
- $\hat{\mathbf{A}}$:= ExpandA(ρ)
- $\mu \in \{0,1\}^{384}$:= CRH($tr \| M$)
- κ := 0
- ($\mathbf{z}$, $\mathbf{h}$) := $\perp$
- $\rho' \in \{0,1\}^{384}$:= CRH($K \| \mu$) (ili $\rho' \leftarrow \{0,1\}^{384}$)

- $\hat{\mathbf{s}}_1$:= NTT($\mathbf{s}_1$)
- $\hat{\mathbf{s}}_2$:= NTT($\mathbf{s}_2$)
- $\hat{\mathbf{t}}_0$:= NTT($\mathbf{t}_0$)

- **dok je** ($\mathbf{z}$, $\mathbf{h}$) == $\perp$:
  - $\mathbf{y} \in \tilde{S}^{l}_{\gamma 1}$:= ExpandMask(ρ', κ)
  - $\mathbf{w}$:= NTT^{-1}($\hat{\mathbf{A}} \cdot$ NTT($\mathbf{y}$))
  - $\mathbf{w}_1$:= HighBits$_q$($\mathbf{w}$, $2\gamma_2$)
  - $\tilde{c} \in \{0,1\}^{256}$:= H($\mu \| \mathbf{w}_1$)
  - $\hat{c}$:= NTT(SampleInBall($\tilde{c}$))
  - $\mathbf{z}$:= $\mathbf{y}$ + NTT^{-1}($\hat{c} \cdot \hat{\mathbf{s}}_1$)
  - $\mathbf{r}_0$:= LowBits$_q$($\mathbf{w}$ − NTT^{-1}($\hat{c} \cdot \hat{\mathbf{s}}_2$), $2\gamma_2$)

  - **ako je** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **ili** $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$:
    - ($\mathbf{z}$, $\mathbf{h}$) := $\perp$
  - **inače:**
    - $\mathbf{h}$:= MakeHint$_q$(−NTT^{-1}($\hat{c} \cdot \hat{\mathbf{t}}_0$), $\mathbf{w}$ − NTT^{-1}($\hat{c} \cdot \hat{\mathbf{s}}_2$) + NTT^{-1}($\hat{c} \cdot \hat{\mathbf{t}}_0$), 2γ$_2$)
    - **ako je** $\|$NTT^{-1}($\hat{c} \cdot \hat{\mathbf{t}}_0$)$\|_\infty \geq \gamma_2$ **ili** (broj jedinica u $\mathbf{h}$) > ω:
      - ($\mathbf{z}$, $\mathbf{h}$) := $\perp$
  - $\kappa = \kappa + l$

Funkcija Potpiši(*sk*, *M*) na ulazu prima tajni ključ i poruku koja se potpisuje.

Iz broja ρ generira se i privremeno sprema $\hat{\mathbf{A}}$ (oblik NTT matrice $\mathbf{A}$). μ se generira kao 384-bitni broj iz hasha otpornog na kolizije (CRH) koji na ulazu prima konkatinirani broj tr iz tajnog ključa i poruku *M*. Broj κ postavlja se na 0, a par vektora ($\mathbf{z}$, $\mathbf{h}$) postavljaju se na vrijednost $\perp$ (*none*/ništa). U slučaju deterministčkog potpisivanja ρ'se generira kao

384-bitni broj iz hasha otpornog na kolizije (CRH) koji na ulazu prima konkatinirani broj K iz tajnog ključa i prethodno generirani broj μ. U slučaju nedeterminističkog potpisivanja ρ' se generira kao nasumični niz bitova.

Izračunaju se oblici NTT vektora polinoma $\mathbf{s}_1$, $\mathbf{s}_2$ i $\mathbf{t}_0$.

Provodi se petlja dok je par vektora (**z**, **h**) jednak $\perp$. Na temelju ρ' i κ generira se vektor polinoma duljine l, a svaki njegov polinom ima koeficijente manje od γ_1. Vektor polinoma **w** izračuna se kao umnožak matrice **A** i vektora **y** koristeći pretvorbu NTT. Vektor $\mathbf{w}_1$ dobije se primjenjujući HighBits_q funkciju uz $\alpha = 2\gamma_2$ na vektor **w**. $\tilde{c}$, 256-bitni broj dobije se hashiranjem kontkatiniranog broja μ s vektorom polinoma $\mathbf{w}_1$. Polinom c dobiva se funkcijom SampleInBall kojoj se kao ulazni argument prosljeđuje broj $\tilde{c}$. Vektor polinoma **z** dobiva se sumiranjem vektora polinoma **y** i umnoška polinoma c s vektorom polinoma $\mathbf{s}_1$. Množenje se provodi koristeći pretvorbe NTT. Vektor polinoma $\mathbf{r}_0$ jednak je razlici funkciji nižih bitova razlike $\mathbf{w} - c\mathbf{s}_2$ (množenje pretvorbama NTT) uz $\alpha = 2\gamma_2$.

Ako je najveći koeficijent polinoma u vektoru polinoma z veći ili jednak $\gamma_1 - \beta$ ili je najveći koeficijent polinoma u vektoru polinoma $\mathbf{r}_0$ veći ili jednak $\gamma_2 - \beta$, postavljamo par (**z**, **h**) na početnu vrijednost $\perp$.

Inače stvaramo vektor *hintova* **h** funkcijom MakeHint_q za zbroj $-c\mathbf{t}_0$ i $\mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0$. Ako je pak najveći koeficijent polinoma u vektoru polinoma umnoška $c\mathbf{t}_0$ veći ili jednak γ_2 ili je broj jedinica u vektoru *hintova* **h** veći od ω, postavljamo par (**z**, **h**) na početnu vrijednost $\perp$.

Na kraju svakog ponavljanja broju κ dodajemo broj l.

Kada se uspije stvoriti zadovoljavajući par (**z**, **h**), vraća se potpis σ koji se sastoji od vektora polinoma **z**, vektora **h** i broja $\tilde{c}$.

3.5 Provjera potpisa

ProvjeriPotpis(*pk*, *M*, σ = (**z**, **h**, $\tilde{c}$)):

Â := ExpandA(ρ)
$\mu \in \{0, 1\}^{384}$:= CRH(CRH(ρ ‖ $\mathbf{t}_1$) ‖ *M*)
c := SampleInBall($\tilde{c}$)
$\mathbf{w'}_1$:= UseHintq(**h**, NTT^{-1}(**Â** · NTT(**z**) – NTT(c) · NTT($\mathbf{t}_1 \cdot 2^d$)), 2γ_2)

ako je $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ **i** $\tilde{c}$ == H(μ ‖ $\mathbf{w'}_1$) **i** broj jedinica u **h** ≤ ω:
    vrati potpis je valjan
inače:
    vrati potpis nije valjan

Iz broja ρ generira se i privremeno sprema **Â**. Broj μ generira se kao i u digitalnom potpisu iz konkatiniranacije broja tr (koji se izračunava iz ρ i $\mathbf{t}_1$ kao u funkciji GenerirajKljučeve()) i poruke *M*. c se izračuna funkcijom SampleInBall iz $\tilde{c}$. Iskorištava se vektor hintova **h** iz potpisa na razliku **Az** – $c\mathbf{t}_1 \cdot 2^d$ funkcijom UseHint_q te se rezultat sprema kao $\mathbf{w'}_1$. Kao i do sada množenje se odvija uz pretvorbe NTT.

Za vektor polinoma **z** i vektor hintova **h** provjeravaju se uvjeti iz funkcije Potpiši(*sk*, *M*), a izračunati broj $\mathbf{w'}_1$ se provjerava preko hash funkcije iz čega se donosi odluka o ispravnosti potpisa.

4. Program za testiranje

4.1 Uvod

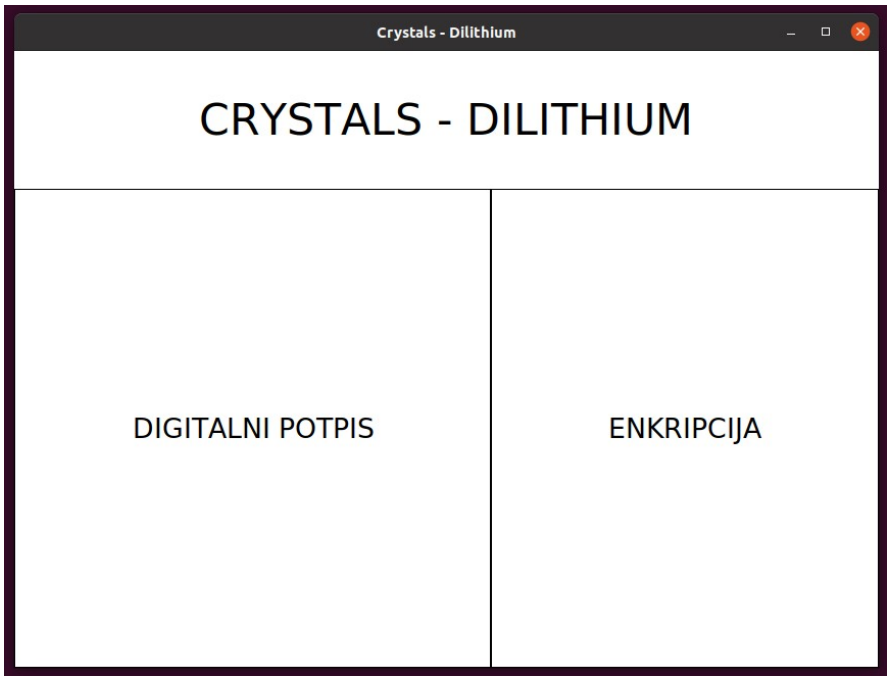
Program je namijenjen isprobavanju digitalnog potpisivanja i provjere digitalnog potpisa postkvantnim algoritmom CRYSTALS – Dilithium. Izvorni kod samog algoritma može se preuzeti na web-stranici https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022, korištena je preporučena referentna implementacija *dilithium3*. Osim digitalnog potpisa postoji dodatna funkcionalnost enkripcije.

4.2 Demonstracija

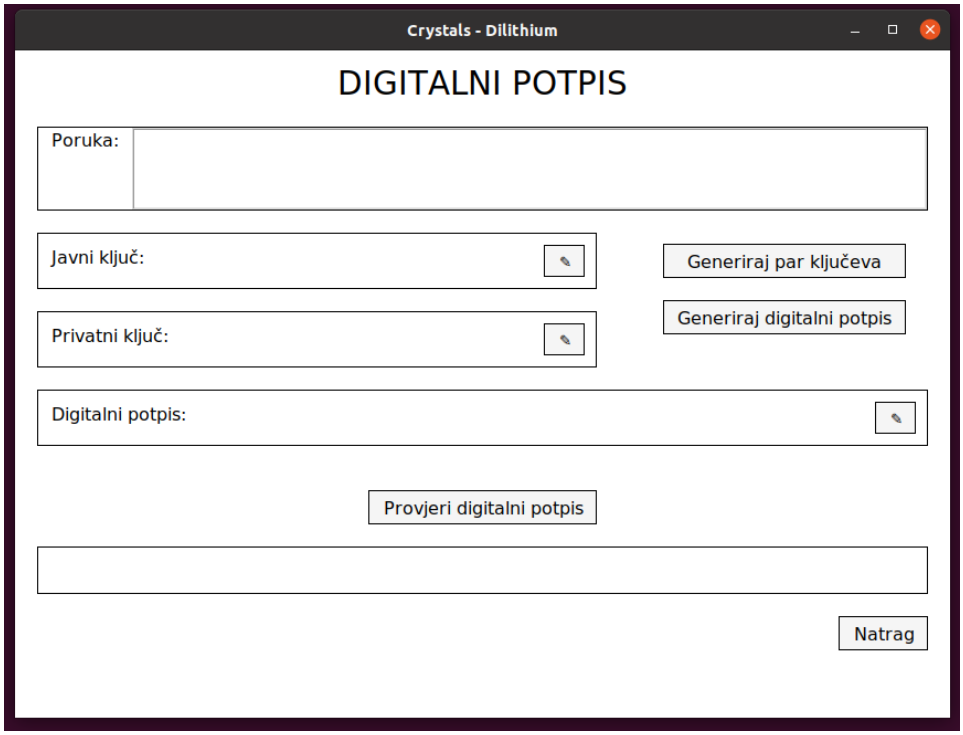
Program se može pokrenuti na Linuxu iz naredbenog retka pozicioniranjem unutar vršnog direktorija i pozivom naredbe „$ python main.py“.

4.2.1 Početni prozor

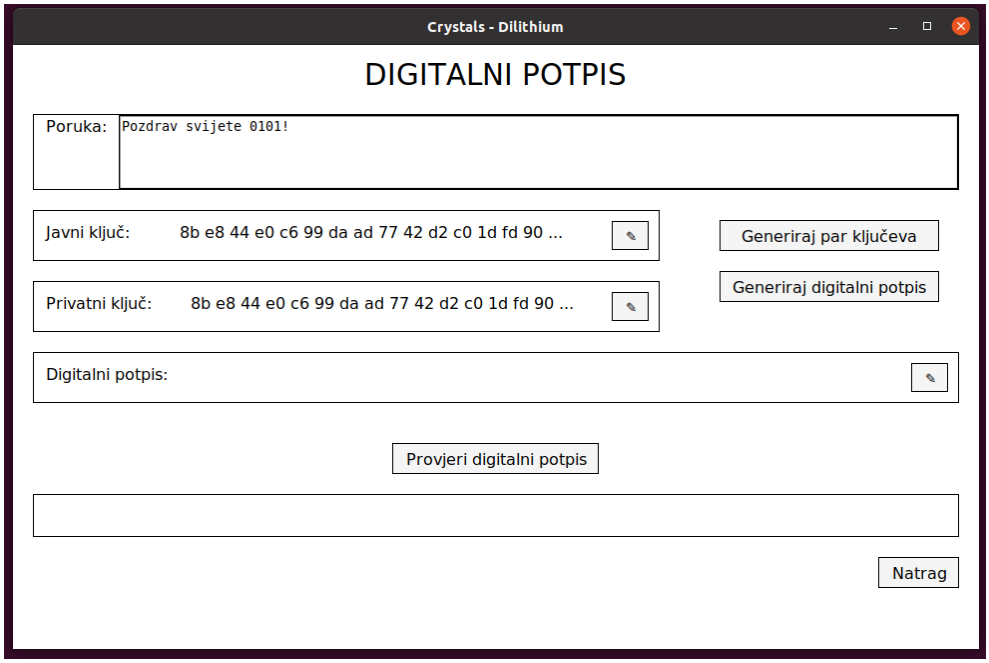
Na početnom prozoru možemo odabrati jednu od dvije opcije pritiskanjem na gumb: digitalni potpis ili enkripcija.



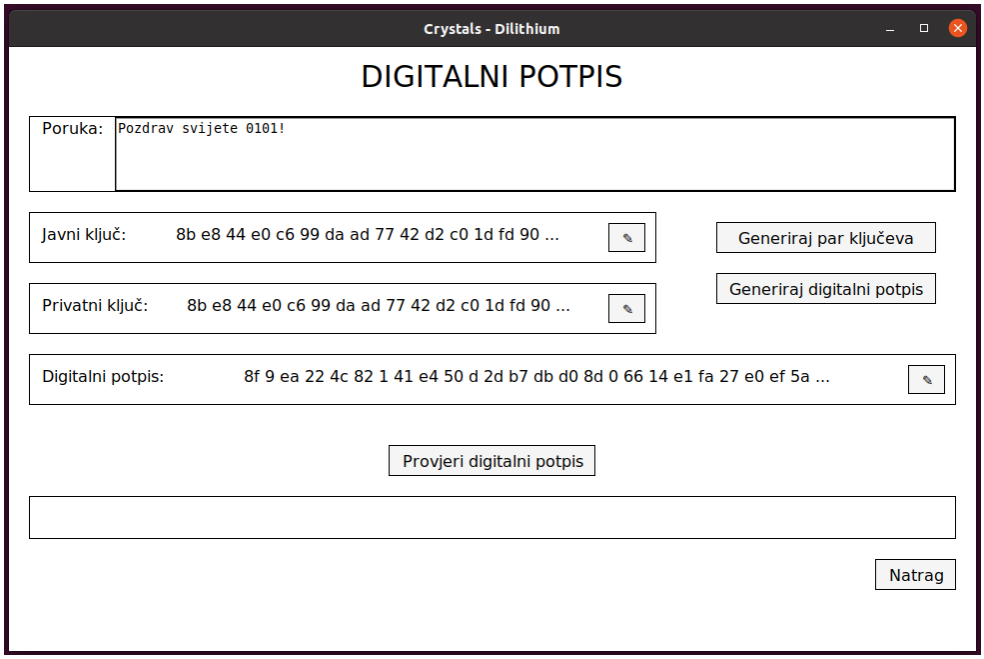
4.2.2 Prozor za digitalni potpis



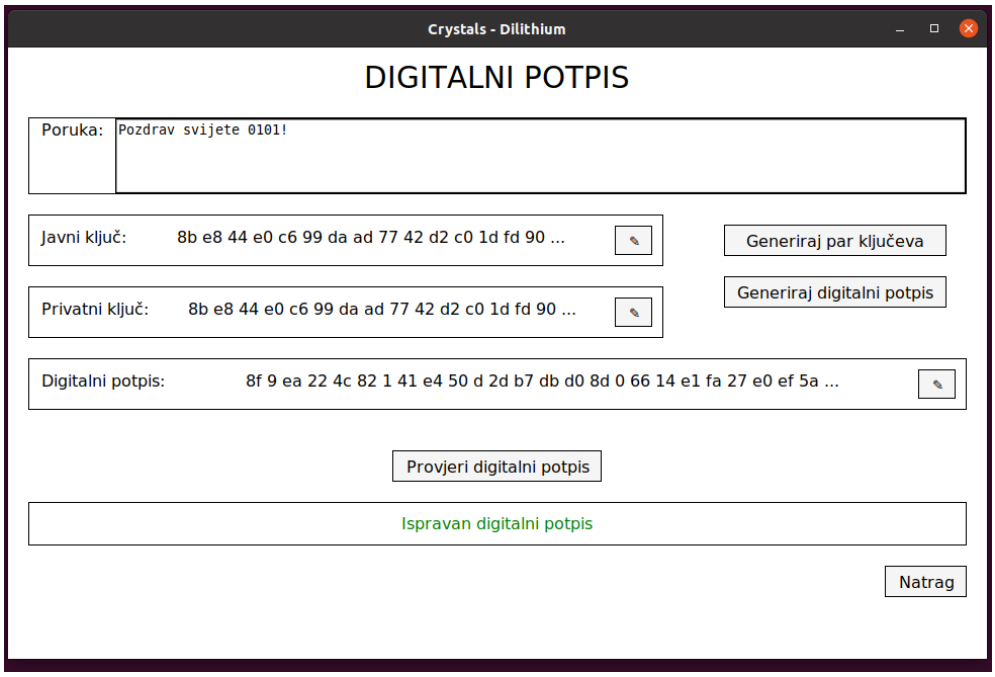
Na prozoru za digitalni potpis možemo unijeti proizvoljnu poruku u za to predviđenom okviru za unos, zatim možemo generirati par ključeva (javni i privatni) čijih će se prvih nekoliko bajtova prikazati na ekranu.



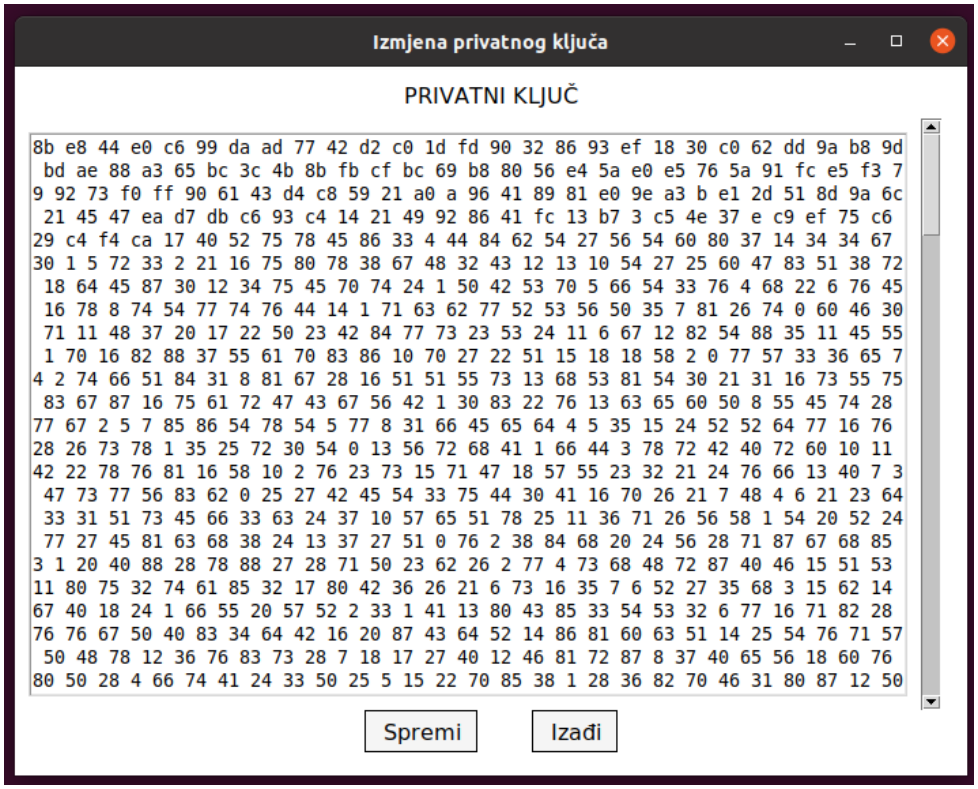
Nakon što smo generirali ključeve možemo generirati digitalni potpis na temelju privatnog ključa i poruke pritiskom na gumb „Generiraj digitalni potpis“.



Pritiskom na gumb „Provjeri digitalni potpis“ provjeriti će se digitalni potpis na temelju javnog ključa, poruke i samog digitalnog potpisa ispisat će se poruka „Ispravan digitalni potpis“ ili „Neispravan digitalni potpis“ ovisno o ispravnosti te trojke.

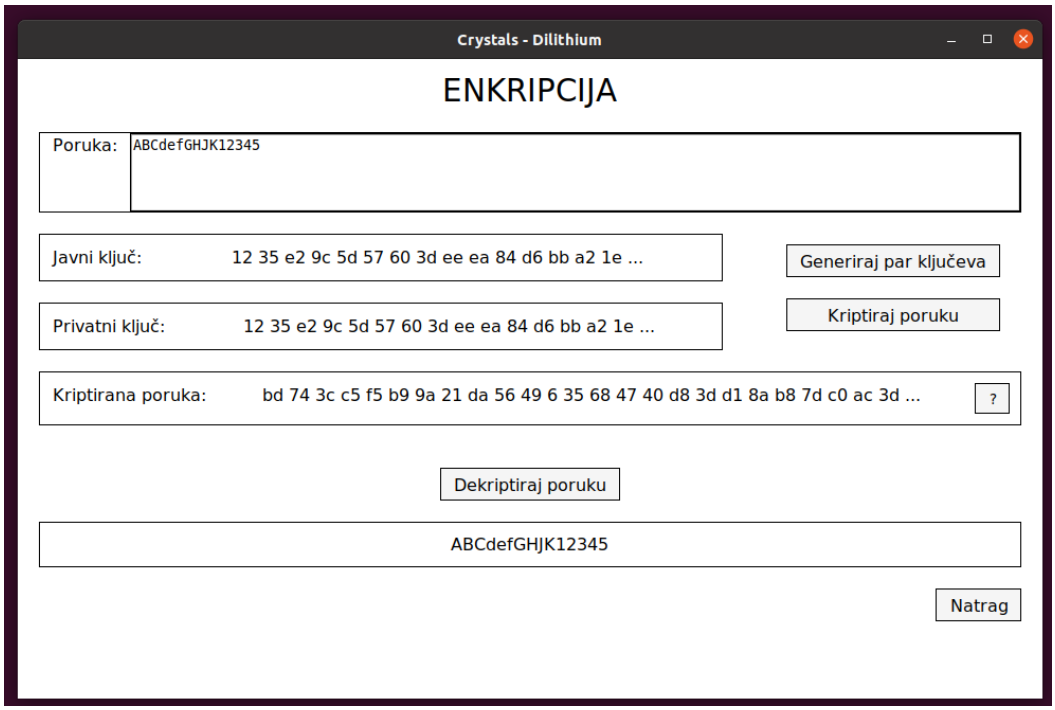


Pritiskom na gumbe s ikonom olovke pored javnog ključa, privatnog ključa i digitalnog potpisa otvorit će se prozor za pregled i izmjenu bajtnog zapisa u heksadekadskom formatu. Može se izmijeniti i spremiti (samo ako se broj bajtova i format ne narušava) pritiskom na gumb „Spremi“ ili odustati od izmjena i izaći pritiskom na gumb „Izađi“.



4.2.3 Prozor za enkripciju

Ponašanje je vrlo slično kao na prozoru za digitalni potpis, može se upisati poruka, generirati par ključeva, ali umjesto potpisa, kriptira se poruka, a umjesto provjere potpisa je dekripcija poruke. Pritiskom na gumb sa znakom upitnika otvara se prozor za pregled bajtnog zapisa kriptirane poruke u heksadekadskom formatu.



4.3 Implementacija

Kao prvi korak potrebno je generirati „dijeljene biblioteke“ *libpqcrystals_aes256ctr_ref.so*, *libpqcrystals_fips202_ref.so* i *libpqcrystals_dilithium3_ref.so* pozivom naredbe „$ make shared“ iz izvornog koda CRYSTALS-Dilithiuma. Zatim je potrebno generirati dodatnu „dijeljenu biblioteku“ randombytes.so naredbom „$ cc -fPIC -shared -o randombytes.so randombytes.c“.

```
dll_name = "randombytes.so"
dllabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
CDLL(dllabspath, mode = RTLD_GLOBAL)

dll_name = "libpqcrystals_aes256ctr_ref.so"
dllabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
CDLL(dllabspath, mode = RTLD_GLOBAL)

dll_name = "libpqcrystals_fips202_ref.so"
dllabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
CDLL(dllabspath, mode = RTLD_GLOBAL)

dll_name = "libpqcrystals_dilithium3_ref.so"
dllabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
dilithium = CDLL(dllabspath, mode = RTLD_GLOBAL)
```

„Dijeljene biblioteke“ su unesene kroz python datoteku „dilithium.py“ u obliku DLL-ova.

```python
def generate_keypair():
    public_key = (c_ubyte * PUBLIC_BYTES)()
    private_key = (c_ubyte * PRIVATE_BYTES)()

    dilithium.pqcrystals_dilithium3_ref_keypair(public_key, private_key)

    return public_key, private_key


def generate_signature(message, private_key):
    signature = (c_ubyte * BYTES)()
    signature_len = c_ulong(0)

    dilithium.pqcrystals_dilithium3_ref_signature(signature, byref(signature_len), message, c_ulong(len(message)), private_key)

    return signature, signature_len


def verify_signature(signature, siglen, message, public_key):
    return 0 == dilithium.pqcrystals_dilithium3_ref_verify(signature, siglen, message, c_ubyte(len(message)), public_key)
```

Napravljene su funkcije koje tipove podataka bliske pythonu pretvaraju u one bliske jeziku C i pozivaju funkcije „dijeljenih biblioteka“.

U datoteci „main.py“ nalazi se kod potreban za grafičko sučelje programa i barata se s python tipovima podataka koji se prosljeđuju funkcijama iz „dilithium.py“ kako bi se dobili željeni rezultati.

5. Literatura

- Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler and Damien Stehlé. (2020) *CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation*. 3rd round (https://pq-crystals.org/dilithium/data/dilithium-submission-nist-round3.zip)
- Amber Sprenkels. (2020) *The Kyber/Dilithium NTT* (https://electricdusk.com/ntt.html)

Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

Postkvantna kriptografija - FALCON
Tehnička dokumentacija

Verzija 2

Student: Slađan Tadić

Nastavnik: prof.dr.sc. Marin Golub

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

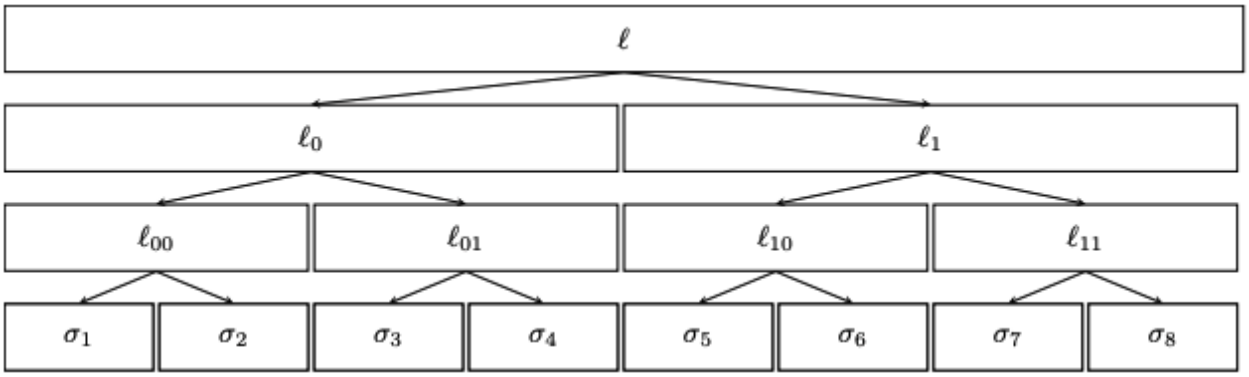


Figure 1. FALCON stablo visine 3

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:

$h = gf^{-1} \bmod (\phi, q)$

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 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 $f(x) = (x^2) + x f_1(x^2)$. 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 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 = $\sum_{i=0}^{n-1} c_i x^i$

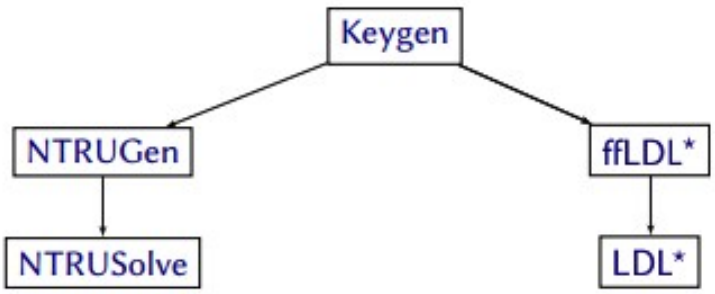
1. k <- [2^{16}/q]
2. ctx <- SHAKE-256-Init()
3. SHAKE-256-Inject(ctx, str)
4. SHAKE-256-Extract(ctx, n = 16), generira dijelove polinoma
5. Agregiraj generirane dijelove
6. Vrati polinom

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, $G \leftarrow \hat{B} \times \hat{B}^{\square}$
   (* 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. $f \leftarrow \sum_i f_i x^i$ ($f \in Z[x]/(\phi)$)
6. $g \leftarrow \sum_i g_i x^i$ ($g \in Z[x]/(\phi)$)
7. ako NTT(f) sadrži 0 kao koeficijent
8. ponovi postupak
9. $\gamma \leftarrow max\{\|(g,-f)\|, \left\|\left(\frac{qf^{\square}}{ff^{\square}+gg^{\square}}, \frac{qg^{\square}}{ff^{\square}+gg^{\square}}\right)\right\|\}$
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 $f G - g F = q \bmod \Phi$
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 $\perp$
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)* $\left(f', g', F', G' \in Z[x]/\left(x^{\frac{n}{2}} + 1\right)\right)$
9. $g' \leftarrow N(g)$
10. $F', G' \leftarrow NTRUSolve_{n/2,q}(f', g')$ (rekurzivni poziv)
11. $F \leftarrow F'(x^2)\, g(-x)$ $F, G \in Z[x]/(x^n + 1)$
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 $u \leftarrow u - \lfloor \frac{\langle u, v \rangle}{\langle v, v \rangle} \rceil v$.

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. $k \leftarrow \lfloor \frac{F f^{*} + G g^{*}}{f f^{*} + g g^{*}} \rceil$
$\lfloor \frac{F f^{*} + G g^{*}}{f f^{*} + g g^{*}} \rceil \in Q, k \in Z$
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 $sk=(\hat{B},T)$. Postupak za izračun LDL stabla sastoji se od:

1. Izračunamo LDL dekompoziciju **G**: napišemo $G=L\times D\times L^{\square}$, 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 $L_{10}\in Q[x]/(\phi)$.

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 $LDL^{\square}(G)$

Ulaz: *matrica* $G=(G_{ij})\in FFT(Q[x]/(\phi))^{2\times 2}$

Izlaz: LDL* dekompozicija $G=LD\,L^{\square}$

Svi polinomi su u FFT formatu

1. $D_{00}\leftarrow G_{00}$
2. $L_{10}\leftarrow G_{10}/G_{00}$
3. $D_{11}\leftarrow G_{11}-L_{10}\odot L_{10}^{\square}\odot G_{00}$
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. $vrati(L,D)$

Algoritam 9 $ff\ LDL^{\square}(G)$

Ulaz: $Gram\,matrica\,G\in FFT(Q[x]/(x^n+1))^{2\times 2}$

Izlaz: binarno stablo T

Svi polinomi su u FFT formatu

1. $(L,D)\leftarrow LD\,L^{\square}(G)$
2. $T.value\leftarrow L_{10}$
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. $T.leftchild\leftarrow ff\ LDL^{\square}(G_0)$
12. $T.rightchild\leftarrow ff\ LDL^{\square}(G_1)$
13. vrati T

2.7 GENERIRANJE POTPISA

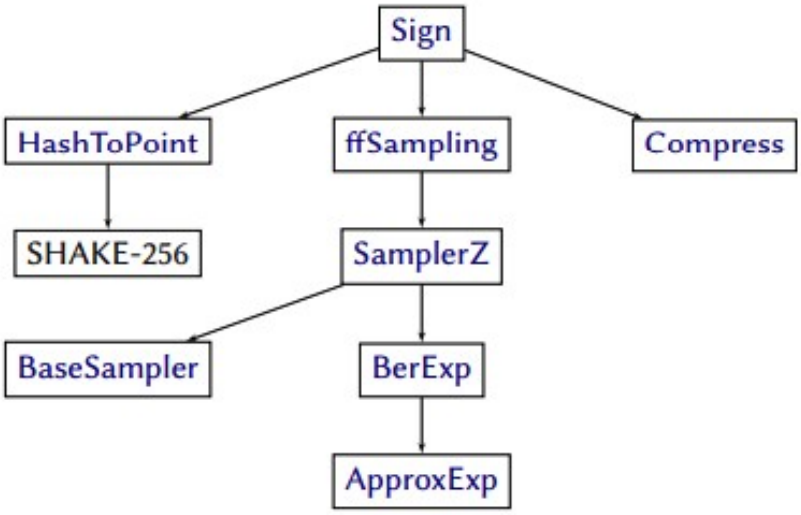


Figure 2 Slijedni dijagram generiranja potpisa

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. $c \leftarrow HashToPoint(r \vee m, q, n)$
3. $t \leftarrow \left(\frac{-1}{q} FFT(c) \odot FFT(F), \frac{1}{q} FFT(c) \odot FFT(f) \right)$ ($\odot$ - množenje FFT domena, t = (FFT(c), FFT(0)) $\cdot \hat{B}^{-1}$
4. radi
5. radi
6. $z \leftarrow ffSampling_n(t, T)$
7. $s = (t - z) \cdot \hat{B}$ (***s*** prati Gaussovu razdiobu)
8. dok $\|s^2\| > \lfloor \beta^2 \rfloor$ (s je u FFT prikazu)
9. $(s1, s2) \leftarrow invFFT(s)$ $s_1 + s_2 h = c \bmod (\phi, q)$
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 $ffSampling_n(t, T)$

Ulaz: $t = (t_0, t_1) \in FFT(Q[x]/(x^n+1))^2$, *FALCON stablo* T

Izlaz: $z = (z_0, z_1) \in FFT(Z[x]/(x^n+1))^2$

Svi polinomi su u FFT formatu.

---

1. ako je n = 1 onda
2. $\sigma' \leftarrow T.value$ $\quad \sigma' \in [\sigma_{min}, \sigma_{max}]$
3. $z_0 \leftarrow SamplerZ(t_0, \sigma')$ $\quad za\ n=1, t_i = invFFT(t_i) \in Q\ i\ z_i = invFFT(z_i) \in Z$
4. $z_1 \leftarrow SamplerZ(t_1, \sigma')$
5. vrati $z=(z_0, z_1)$
6. $(l, T_0, T_1) \leftarrow (T.value, T.leftchild, T.rightchild)$
7. $t_1 \leftarrow splitfft(t_1)$
8. $z_1 \leftarrow ffSampling_{n/2}(t_1, T_1)$
9. $z_1 \leftarrow mergefft(z_1)$
10. $t' \leftarrow t_0 + (t_1 - z_1) \odot l$
11. $t_0 \leftarrow splitfft(t_0')$
12. $z_0 \leftarrow ffSampling_{n/2}(t_0, T_0)$
13. $z_0 \leftarrow mergefft(z_0)$
14. *vrati* $z=(z_0, z_1)$

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, …, $2^k - 1$}.

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 $z \; \chi$

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 =[0x00000004741183A3, 0x00000036548CFC06, 0x0000024FDCBF140A, 0x0000171D939DE045,
0x0000D00CF58F6F84, 0x000680681CF796E3, 0x002D82D8305B0FEA, 0x011111110E066FD0,
0x0555555555070F00, 0x155555555581FF00, 0x400000000002B400, 0x7FFFFFFFFFF4800,
0x8000000000000000].

Neka je polinom $f \in R$ 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, … , 2^{63} - 1}, s 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 $\approx ccs \cdot \exp(-x)$

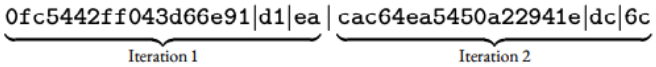
Algoritam 14 BerExp(x, ccs)

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

Algoritam 15 $SamplerZ(\mu, \sigma')$

Ulaz: decimalne vrijednosti $\mu, \sigma' \in R$ takvi da $\sigma' \in [\sigma_{min}, \sigma_{max}]$
Izlaz: cijeli broj $z \in Z$ uzorkovan iz distribucije vrlo blizu $D_{Z,\mu,\sigma'}$
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. $\quad z_0 \leftarrow BaseSampler()$
5. $\quad b \leftarrow UniformBits(8) \wedge 0x1$
6. $\quad z \leftarrow b + (2 \cdot b - 1) z_0$
7. $\quad x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{max}^2}$
8. $\quad ako\ je\ BerExp(x\ ccs) = 1\ onda$
9. $\quad\quad$ 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 ulančani u *string* (*r*||*m*) koji je) koji je *hashiran* u polinom $c \in Z[x]/(\phi)$ prema funkciji *HashToPoint* (Algoritam 3)
2. *s* se dekodira (dekompresira) u polinom $s_2 \in Z[x]/(\phi)$, koristeći funkciju *Decompress* (Algoritam 18)
3. Izračunavamo vrijednost $s_1 = c - s_2 h \bmod q$
4. Ako vrijedi $\vee (s_1, s_2) \vee^2 \leq \lfloor \beta^2 \rfloor$, 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 $pk = h \in Z[x]/(\phi)$ granica $\lfloor \beta^2 \rfloor$

Izlaz: *prihvati* ili *odbaci*

1. $c \leftarrow HashToPoint(r \vee m, q, n)$
2. $s_2 \leftarrow Decompress(s, 8 \cdot sbytelen - 328)$
3. $ako(s_2 = \perp)$
4. odbaci (nevaljano kodiranje)
5. $s_1 = c - s_2 h \bmod q$ (s_1 treba biti normaliziran između $\lceil -\frac{q}{2} \rceil i \lfloor \frac{q}{2} \rfloor$)
6. *ako* ($\vee (s_1, s_2) \vee^2 \leq \lfloor \beta^2 \rfloor$) (!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 $s \in Z[x]/(\phi)$ 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 ***s*** je ulančani string $s \leftarrow (str_0 \vee str_1 \vee \ldots \vee str_{n-1})$
3. **str** se nadopunjuje **nulama** do fiksne duljine **slen**

Algoritam 17 $Compress(s, len)$

Ulaz: polinom $s = \sum s_i x^i \in Z[x]$ 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. $str \leftarrow$ (kodiraj predznak od s_i)
4. $str \leftarrow$ (kodiraj binarno više bitove $s_i \vee$)
5. $k \leftarrow |s_i| \gg 7$ (kodiraj unarno niže bitove $s_i \vee$)
6. $str \leftarrow (str \vee 0^k 1)$
7. $ako(|str| > slen)$
8. $str \leftarrow \perp$ (prekini ako je *str* predugačak)
9. *inače*
10. $str \leftarrow (str \vee 0^{slen - |str|})$ (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 $n=2^l, l \in [0,10]$

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 prošireni 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* $str = ¿$, *string bitlength* **slen**

Izlaz: polinom $s = \sum s_i x^i \in Z[x]$, ili $\perp$

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

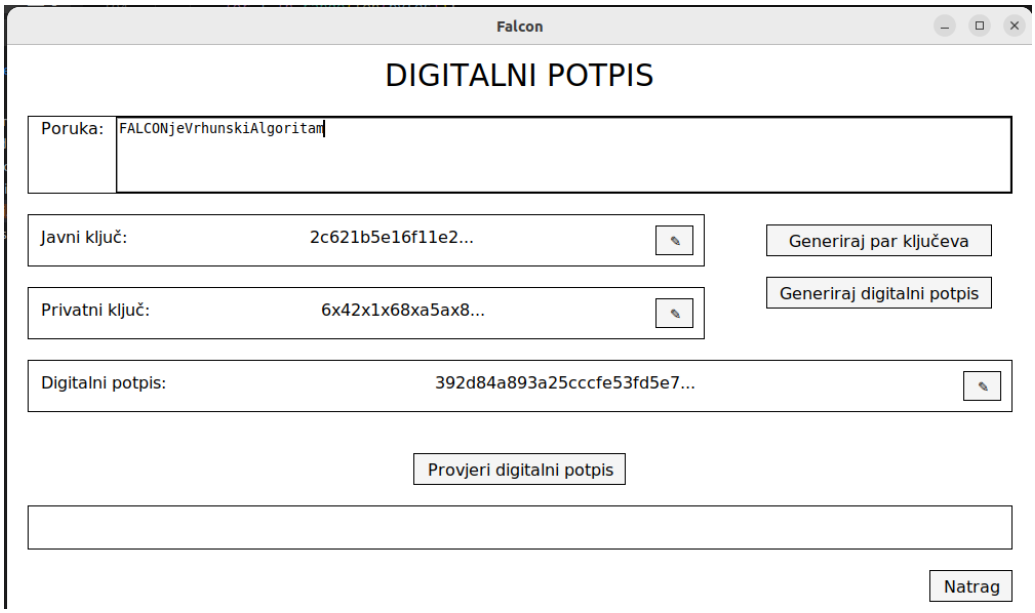
Kako bi mogli koristiti postojeću implementaciju moramo koristiti vanjsku knjižnicu *Crypto.Hash*, tj. iz nje *SHAKE256*. Upute za korištenje, kao i za instalaciju možete pogledati ovdje. Također, trebat će vam datoteka *main.py* koju sam napisao i koju možete dohvatiti ovdje, i koju trebate pohraniti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

3.3 DEMONSTRACIJA

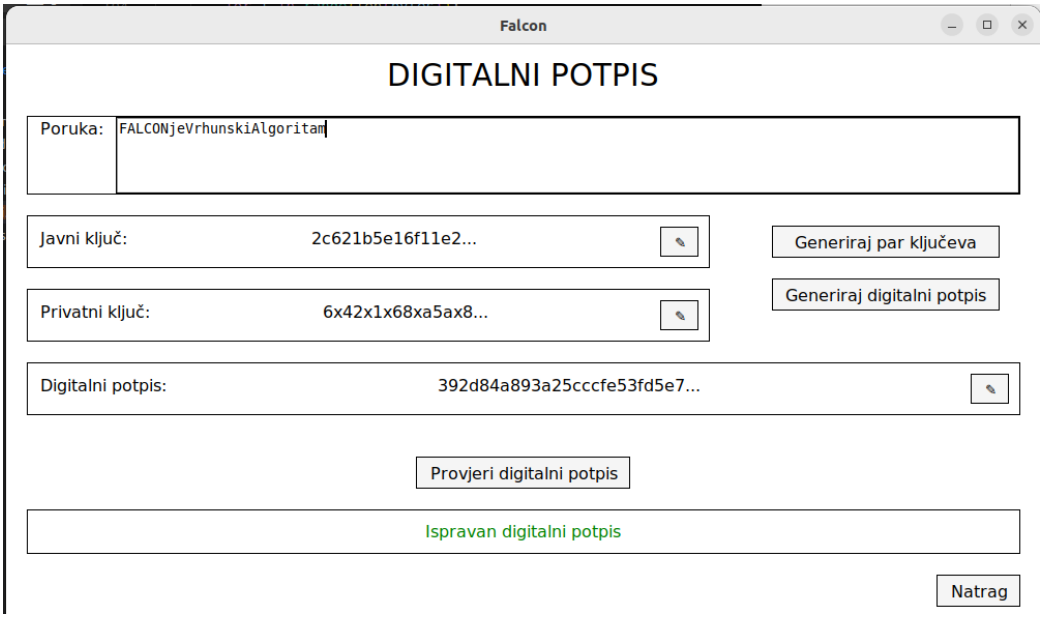
Program pokrećete iz direktorija u kojem se nalaze *main.py* i *falcon.py* koristeći naredbu:
python3 main.py

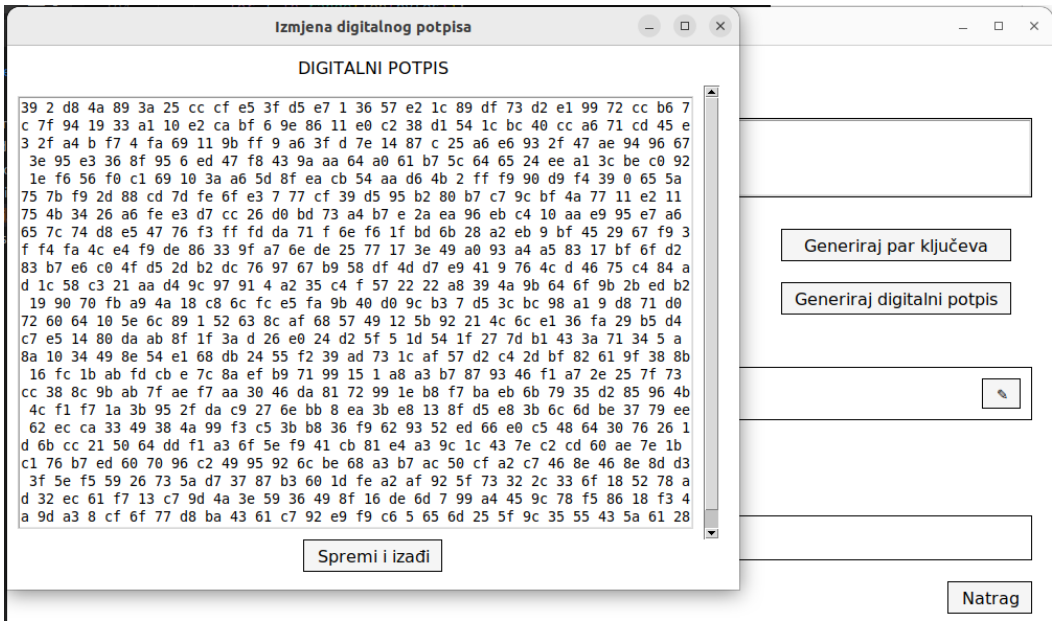
Prilikom pokretanja programa otvara se prozor:

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.

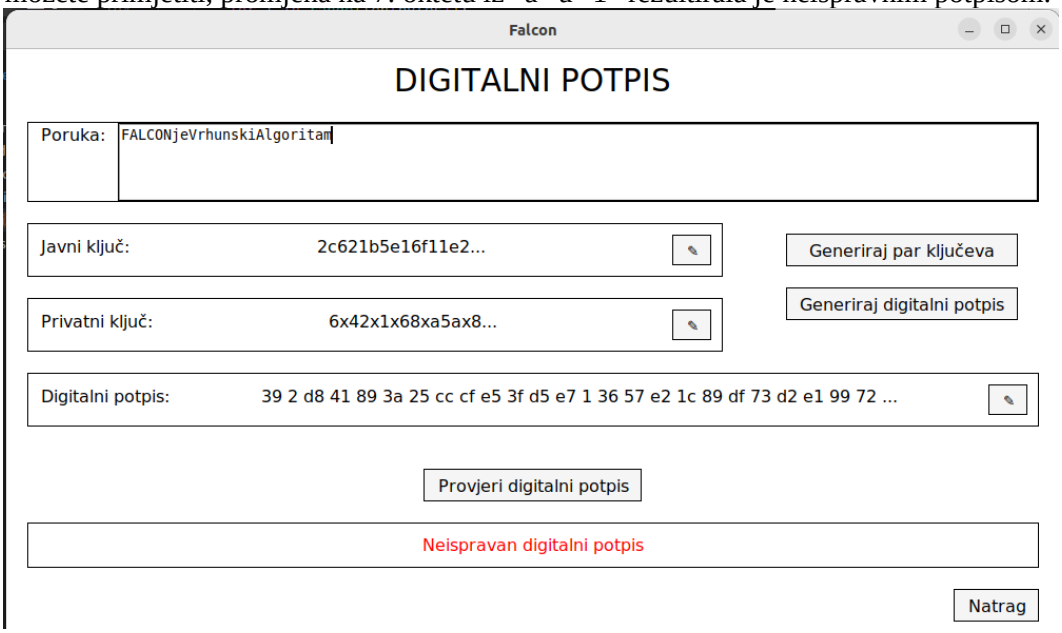


Poruka o ispravnosti digitalnog potpisa prikazuje se u donjem prozoru. Pritiskom na gumb olovke u polju "Digitalni potpis" moguće je ručno promijeniti vrijednosti navedenog polja. Ako se neka od tih vrijednosti promjeni ručno, provjera digitalnog potpisa bit će potencijalno neispravna.

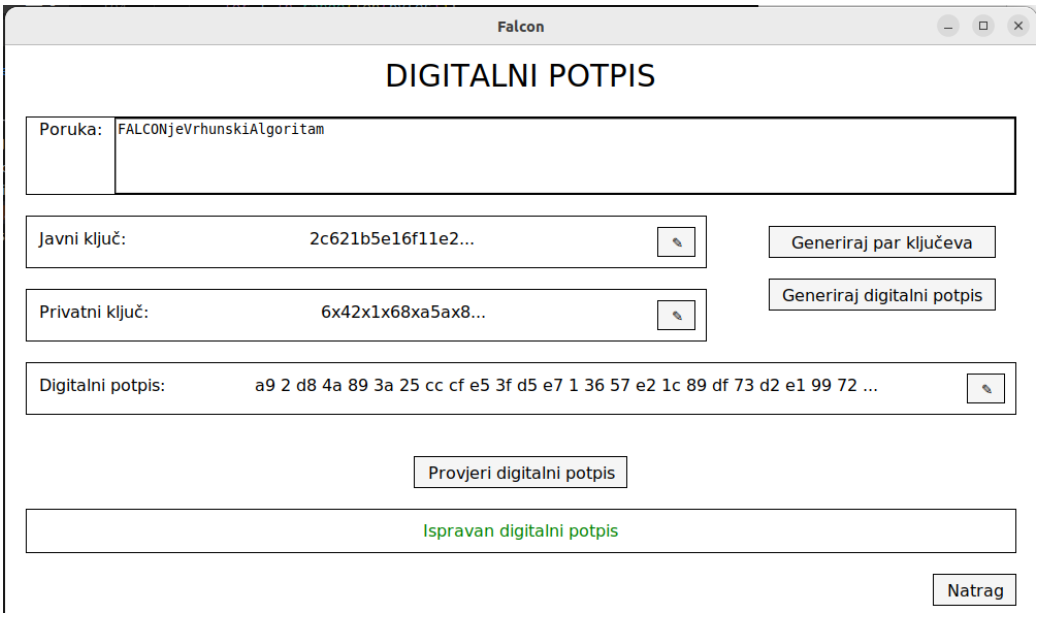




Kao što možete primjetiti, promjena na 7. oktetu iz “a” u “1” rezultirala je neispravnim potpisom.



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

U dokumentaciji i implementaciji vidljivo, da uz vrlo male promjene vrijednosti digitalnog potpisa je i dalje valjana. Unatoč toj činjenici, algoritam zadovoljava sve sigurnosne standard, a takvi slučajevi ne predstavljaju sigurnosni problem za njegovu uspješnu implementaciju.

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 s 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 s 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

Postkvantna kriptografija – SPHINCS+
Tehnička dokumentacija
Verzija 4

Student: Filip Penzar

Nastavnik: doc.dr.sc. Marin Golub

SADRŽAJ

1. UVOD

1.1 SPHINCS+

SPHINCS+ je shema digitalnog potpisa bez stanja koja se temelji na raspršivanju. Bazira se na ranijem algoritmu SPHINCS koji je prezentiran na EUROCRYPT-u 2015. U nastavku dokumenta je pobliže objašnjen i demonstriran rad algoritma.

2. ALGORITAM

2.1 POTPIS WOTS+

2.1.1 OTS I WOTS+

One time signature (OTS) je tip sheme digitalnog potpisa u kojem se isti privatni ključ može sigurno koristiti samo jedanput. Jednom kada se potpiše neka poruka s parom ključeva, ti isti ključevi efektivno postaju javni i više ih ne možemo koristiti za daljnje potpisivanje. *Winternitz one time signature* (WOTS+) vrsta je sheme jednokratnog digitalnog potpisa temeljenog na raspršvanju koji se koristi u sklopu SPHINCS+ algoritma.

2.1.2 GENERIRANJE KLJUČEVA

Definiramo Winternitzov parametar *w* kao element iz skupa {4, 16, 256}. Njega koristimo u funkciji pretvorbe poruke zapisane u bitovima u niz brojeva u rasponu [0, w – 1] na sljedeći način:

- poruka za kodiranje je duljine m bitova
- poruku rastavimo na l segmenata duljine $\log_2(w)$ bitova ($l = m / \log_2(w)$)
- svaki segment duljine $\log_2(w)$ bitova zasebno interpretiramo kao jedan broj u navedenom rasponu

Primjer:

w = 4
$M = 37_{(10)} = 100101_{(2)}$ (*big endian*)
m = len(M) = 6

$l = m / \log_2(w) = 6 / \log_2(4) = 3$
M = 10 | 01 | 01
→ 2 | 1 | 1

Ukoliko imamo poruku duljine *m* bitova, za nju generiramo l tajnih ključeva, gdje je $l = m / log_2(w)$. Na temelju l tajnih ključeva generiramo njihove javne ključeve tako da svaki od l tajnih ključeva uzastopce *hashiramo* nekim od algoritama raspršivanja (w – 1) puta. To nam daje l javnih ključeva.

Primjer:

skn => privatni ključ indeksa n
pkn => javni ključ indeksa n
H(skn) => primjena algoritma raspršivanja na privatni ključ indeksa n

---

sk0 → H(sk0) → H(H(sk0)) → H(H(H(sk0))) → … → H^(w-1)(sk0) = pk0
sk1 → H(sk1) → H(H(sk1)) → H(H(H(sk1))) → … → H^(w-1)(sk1) = pk1
sk2 → H(sk2) → H(H(sk2)) → H(H(H(sk2))) → … → H^(w-1)(sk2) = pk2
. . . . . … .
. . . . . … .
. . . . . … .
sk(l-1) → H(sk(l-1)) → H(H(sk(l-1))) → H(H(H(sk(l-1)))) → … → H^(w-1)(sk(l-1)) = pk(l-1)

Cjelokupni privatni ključ dobivamo spajanjem svih skn ključeva SK = (sk0 || sk1 || sk2 || … || sk(l-1)), a po istom principu dobivamo i javni ključ PK = (pk0 || pk1 || pk2 || … || pk(l-1)) gdje je || operator nadovezivanja.

2.1.3 POTPISIVANJE

Ukoliko imamo poruku *message,* njezin sažetak (*message digest*) *M* duljine *m* bitova dobiven korištenjem nekog algoritma raspršivanja (npr. Harak, SHA256), primjenom gore navedene funkcije pretvorbe dobitit ćemo l segmenata poruke. Koristeći algoritam opisan na stranici 1 dobivamo l segmenata privatnog i l segmenata javnog ključa. Primjenom funkcije pretvorbe pomoću Winternitzovog parametra dobivamo i l segmenata sažetka poruke M. Potpis generiramo tako da svaki od l segmenata sažetka poruke duljine $\log_2(w)$ bitova zasebno interpretiramo kao jedan broj u rasponu [0, w – 1] . Taj broj uz indeks segmenta određuje koliko ćemo puta nad privatnim ključem istog indeksa primjeniti funkciju raspršivanja (funkcija MORA biti ista kao i ona upotrebljena za generiranje javnog ključa i mora biti javno poznata). Tako dobivene vrijednosti *d*n nadovezujemo te one zajedno čine potpis poruke D = (d0 || d1 || d2 || … || d(l-1)).

Primjer:

poruka → H(poruka) = M = 1001110110
m = len(M) = 10
w = 4
$l = m / \log_2(w) = 10 / \log_2(4) = 5$
---
sk0 → H(sk0) → H(H(sk0)) → H(H(H(sk0))) = pk0
sk1 → H(sk1) → H(H(sk1)) → H(H(H(sk1))) = pk1
sk2 → H(sk2) → H(H(sk2)) → H(H(H(sk2))) = pk2
sk3 → H(sk3) → H(H(sk3)) → H(H(H(sk3))) = pk3
sk4 → H(sk4) → H(H(sk4)) → H(H(H(sk4))) = pk4

M = (10 | 01 | 11 | 01 | 00) → (2 | 1 | 3 | 1 | 0)

Segmenti poruke M i njihov indeks određuje koliko puta primjenjujemo funkciju adresiranja na odgovarajući privatni ključ:

M[0] = 2
sk0 → H(sk0) → H(H(sk0)) = d0

M[1] = 1
sk1 → H(sk1) = d1

M[2] = 3
sk2 → H(sk2) → H(H(sk2)) → H(H(H(sk2))) = d2

M[3] = 1
sk3 → H(sk3) = d3

M[4] = 0
sk4 = d4

Digitalni potpis D = (d0 || d1 || d2 || d3 || d4).

2.1.4 VERIFIKACIJA

Podaci koji su poznati verifikatoru su: w, D, PK i poruka. Na temelju poruke, verifikator računa sažetak poruke koristeći isti algoritam raspršivanja kao i potpisatelj. Time generira sažetak poruke M duljine m bitova. Segmenti poruke M i njihovi indeksi određuju koliko je puta primjenjena funkcija adresiranja na odgovarajući privatni ključ. Znajući to, za svaki indeks možemo odrediti koliko puta trebamo odgovarajući segment digitalnog potpisa *hashirati* kako bi dobili odgovarajući segment javnog ključa. Uzimajući primjer iz prošlog odjeljka:

Primjer:

Digitalni potpis D = (d0 || d1 || d2 || d3 || d4)
poruka → H(poruka) = M = 1001110110
m = len(M) = 10
w = 4
(w – 1) = 3
$l = m / \log_2(w) = 10 / \log_2(4) = 5$
PK = (pk0 || pk1 || pk2 || pk3 || pk4)

M = (10 | 01 | 11 | 01 | 00) → (2 | 1 | 3 | 1 | 0)

M[0] = 2
3 – 2 = 1 // segment d0 potrebno je hashirati još jednom kako bi dobili segment javnog ključa
---
d0 → H(d0) = pk0’

M[1] = 1
3 – 1 = 2 // segment d1 potrebno je hashirati još dvaput kako bi dobili segment javnog ključa
---
d1 → H(d1) → H(H(d1)) = pk1’

M[2] = 3
3 – 3 = 0 // segment d2 jednak je segmentu javnog ključa
---
d2 = pk2’

M[3] = 1
3 – 1 = 2 // segment d3 potrebno je hashirati još dvaput kako bi dobili segment javnog ključa
---
d3 → H(d3) → H(H(d3)) = pk3’

M[4] = 0
3 – 0 = 3 // segment d4 potrebno je hashirati još triput kako bi dobili segment javnog ključa
---
d4 = sk4 →H(d4) → H(H(d4)) → H(H(H(d4))) = pk4’

Dobivene javne ključeve nadovežemo i dobijemo PK’ = (pk0’ || pk1’ || pk2’ || pk3’ || pk4’).
Ukoliko se PK’ poklapa s PK, digitalni potpis je ispravan, inače je neispravan.

Ova se metoda za jedan par ključeva može upotrijebiti samo jedanput (OTS) jer ukoliko znamo *dn* za vrijednost poruke M[n] = z, znamo i vrijednost *dn* za sve vrijednosti poruke M[n] = z’ gdje vrijedi z’ >= z.

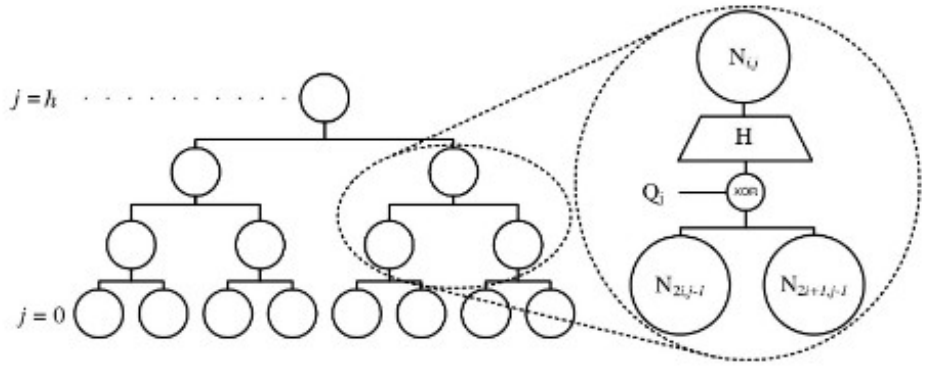
2.2 SHEMA XMSS

2.2.1 UVOD

Zbog nepraktičnosti korištenja jednog para ključeva za potpisivanje samo jedne poruke, Merkle je 1970. razvio shemu digitalnog potpisa temeljenih na *OTS* pomoću koje se isti par ključeva može koristiti za veći broj poruka, iako konačan.

2.2.2 GENERIRANJE KLJUČEVA

XMSS (*Extended Merkle Signature Scheme*) novija je implementacija originalne Merklove sheme. Shema se sastoji od stabla u kojemu su listovi parovi WOTS+ privatnih i javnih ključeva, a najgornji korijen stabla je potpisateljev javni ključ. Ostali čvorovi (uključujući i sam korijen stabla) su izlazi *hashrianja* njihove djece unutar stabla.
Na temelju privatnog *seed*-a (koji je zapravo privatni ključ) generiraju se svi ostali privatni ključevi u listovima stabla. Iz tih privatnih ključeva u listovim generiraju se javni ključevi za listove po WOTS+ algoritmu opisanom na stranici 1. Zatim se iz parova javnih ključeva listova koristeći odabrani algoritam raspršivanja, određuju čvorovi stabla. Postupak *hashiranja* djece kako bi se dobio roditelj nastavlja se sve do vrha stabla (korijena odnosno javnog ključa). Tako dobiveni korijen javno je dostupan i on je generirani javni ključ.

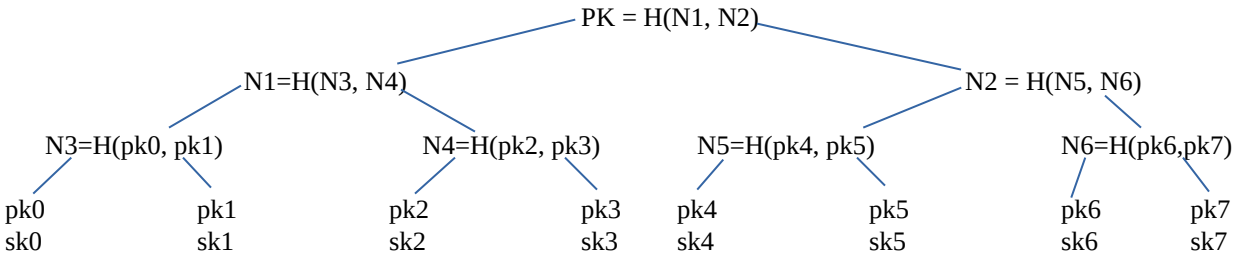


Slika 2.1: Struktura XMSS sheme

2.2.3 POTPISIVANJE

Kako bi se potpisala poruka potrebno je odabrati jedan od listova, ali se isti list smije koristiti samo jedanput. Iz odabranog lista pomoću algoritma potpisivanja poruke sa stranice 2 WOTS+ algoritmom dobiva se WOTS+ digitalni potpis poruke. WOTS+ digitalni potpis poruke, korišteni javni ključ lista, javni ključ susjednog lista, vrijednost čvora susjednog roditelja, vrijednost čvora susjednog roditelja roditelja … do korijena čini krajnji digitalni potpis. Ovime se stvara autentifikacijski put XMSS stabla.

Primjer:

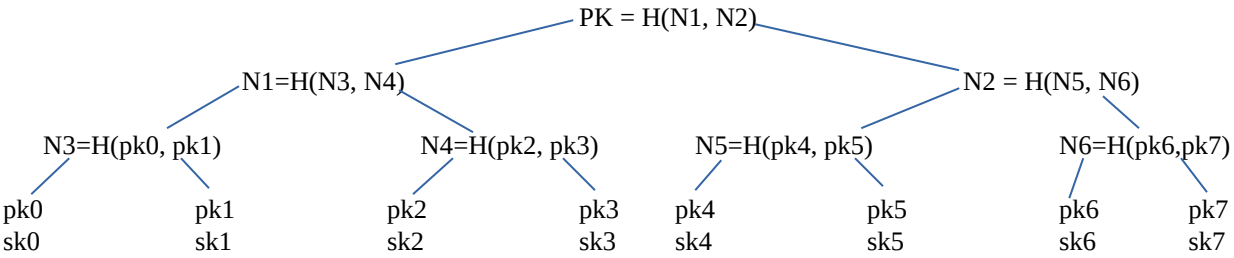


Koristeći pk0 i sk0 generiramo WOTS+ digitalni potpis *d*. Konačan digitalni potpis *D* definiramo D = (d || pk0 || pk1 || N4 || N2).

2.2.4 VERIFIKACIJA

Iz dobivenog digitalnog potpisa D, javnog ključa potpisatelja te korištenog algoritma raspršivanja, digitalni potpis se verificira tako da se iz njega pročita digitalni potpis WOTS+ algoritma i verificira po algoritmu opisanom na stranici 3. Ako se tako izračunati javni ključ pk' ne slaže s javnim ključem dobivenim u potpisu D, potpis nije valjan. Dalje se iz dobivenog javnog ključa i javnog ključa susjednog lista *hashiranjem* računa vrijednost njihovog roditelja. Dobivena vrijednost se uz vrijednost čvora susjednog roditelja ponovo *hashira* kako bi se dobila vrijednost njihovog roditelja i tako dalje, sve do korijena stabla. Dobivena vrijednost korijena PK' uspoređuje se s poznatim javnim ključem PK. Ako su PK' i PK isti, potpis je valjan, ako su različiti, potpis nije valjan.

Primjer:



D = (d || pk0 || pk1 || N4 || N2)

Iz d se računa pk0' i uspoređuje s pk0. Ako nisu isti potpis nije valjan.
Iz pk0 i pk1 računa se N3'=H(pk0, pk1).
Iz N3' i N4 računa se N1'=H(N3',N4).
Iz N1' i N2 računa se PK'=H(N1',N2).
Ako su PK' I PK isti, digitalni potpis je valjan, inače nije valjan.

2.3 STRUKTURA HT

2.3.1 UVOD

HT (*The Hypertree*) vrsta je certifikacijskog stabla XMSS stabala. Ovakva arhitektura omogućuje brže potpisivanje i verifikaciju većeg broja poruka. HT je stablo od više slojeva XMSS stabala. Stabla na vrhu i u sredini služe za potpisivanje javnih ključeva (korijena) XMSS stabala njihove djece. Stabla na najnižem sloju služe potpisivanju poruke, odnosno potpisivanju FORS javnih ključeva u slučaju SPHINCS+.

Slika 2.2: Struktura HT

T<i> označava javni ključ XMSS stabla indeksa *i*. Vidimo da se stablo indeksa *i* koristi za potpisivanje stabala s indeksima (2**i*) i (2**i* + 1). T1 javni je ključ HT. Kao privatni ključ HT koristimo *seed* koji nam služi kako bi deterministički mogli generirati sve privatne ključeve svih XMSS stabala unutar HT strukture. Na ovaj način ne moramo pohraniti cijelu strukturu HT, nego možemo izračunati samo potrebna XMSS stabla za potpisivanje

određene poruke.

2.3.2 POTPISIVANJE

Za sažetak poruke, i dovoljno veliku HT strukturu (broj listova je dovoljno velik, tako da je vjerojatnost slučajnog odabira već iskorištenog lista zanemariva) slučajnim odabirom izabire se list XMSS-a najnižeg sloja HT. Sažetak se potpisuje XMSS strukturom te se dobiva javni ključ XMSS najnižeg sloja HT. Dobiveni javni ključ tog XMSS stabla *hashira* se i potpisuje XMSS stablom roditelja. Navedeni postupak nastavlja se sve do najgornjeg XMSS stabla. Dobiveni korijen ujedno je i javni ključ potpisatelja. Ukupan potpis sastoji se od svih autentifikacijskih puteva svih korištenih XMSS, od najnižeg sloja do najvišeg.

2.3.3 VERIFIKACIJA

Za dobiveni potpis računaju se javni ključevi korištenih XMSS od dna prema vrhu. Kreće se sa izračunatim sažetkom poruke, za njega i pripadni autentifikacijski put računa se javni ključ prvog korištenog XMSS. Za tako dobiveni javni ključ ponovo se izračuna sažetak, te se računa javni ključ sljedećeg XMSS pomoću pripadnog autentifikacijskog puta dobivenog u potpisu. Postupak se nastavlja sve do posljednjeg XMSS. Ako je izračunati javni ključ korijena jednak javnom ključu potpisatelja, potpis je valjan, a u suprotnom je nevaljan.

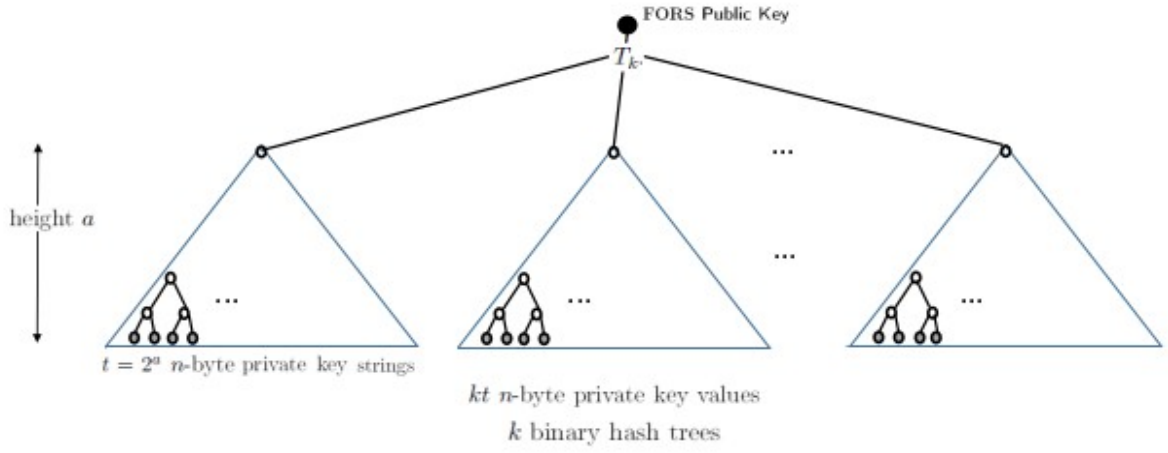
2.4 STRUKTURA FORS

2.4.1 UVOD

HT se ne koristi unutar SPHINCS+ kako bi se potpisala sama poruka, nego kako bi se potpisali javni ključevi FORS instanci. FORS (*Forest Of Random Subsets*) je shema digitalnog potpisa, FTS (*few-times signature*), u kojoj se isti privatni ključ može koristi više od jednom.

2.4.2 STRUKTURA

FORS instanca sastoji se od *k* binarnih hash stabala, svako stablo od *t* listova, te *hasha* T koji se dobije *hashiranjem* svih korijena od *k* binarnih stabala. Svaki list predstavlja jedan privatni ključ. Vrijednosti privatnih ključeva generiraju se iz *seeda*, koji je ujedno i privatni ključ same FORS instance.



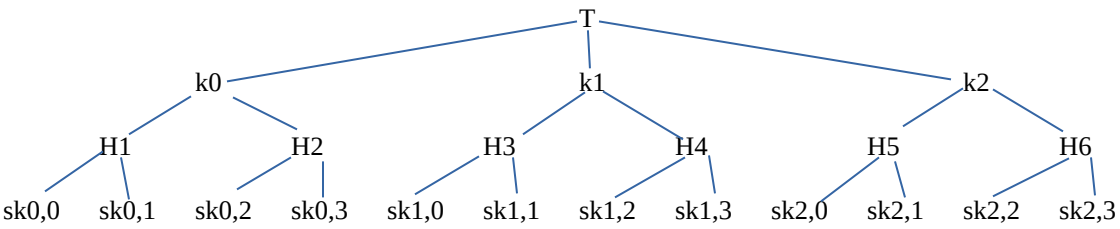
Slika 2.3: Struktura FORS

2.4.3 POTPISIVANJE

Sažetak poruke, M, podijeli se na *k* segmenata, svaki duljine $\log_2(t)$ bitova. Svaki segment interpretira se kao zasebna dekadska vrijednost, koja za odgovorajuće binarno hash stablo kodira list, odnosno privatni ključ tog stabla. Potpis čine svi autentifikacijski putevi svih binarnih hash stabala.

Primjer:

FORS instanca:



k = 3
t = 4
$M = 19_{(10)} = 010011_{(2)}$

M = (01 |00 | 11) = (1 | 0 | 3)
M[0] = 1
k0[M[0]] = k0[1] = sk0,1

M[1] = 0
k1[M[1]] = k1[0] = sk1,0

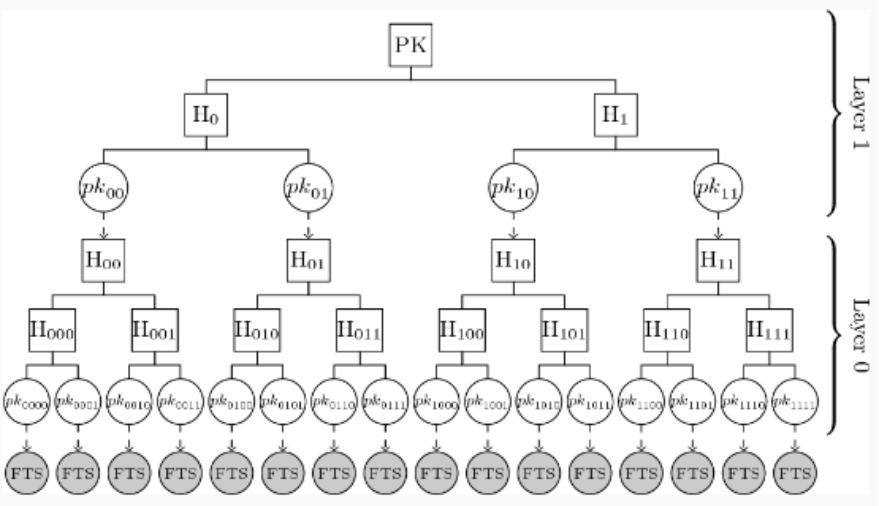
M[2] = 3
k2[M[2]] = k2[3] = sk2,3
Potpis D = (sk0,1 || sk0,0 || H2 || sk1,0 || sk1,1 || H4 || sk2,3 || sk2,2 || H5)

2.4.4 VERIFIKACIJA

FORS se ne koristi za verifikaciju potpisa, već samo za generiranje potpisa iz kojega se zatim može dobiti javni ključ T' korištene FORS instance. T' se dobiva na isti način kao i kod XMSS, uzastopnim *hashiranjem* dobivenih autentifikacijskih puteva sve do korijena.

2.5 STRUKTURA SPHINCS+

SPHINCS+ je shema digitalnog potpisa koja koristi sve gore opisane strukture. Sastoji se od HT strukture kojoj listovi služe potpisivanju javnih ključeva FORS instanci, koje potpisuju samu poruku.

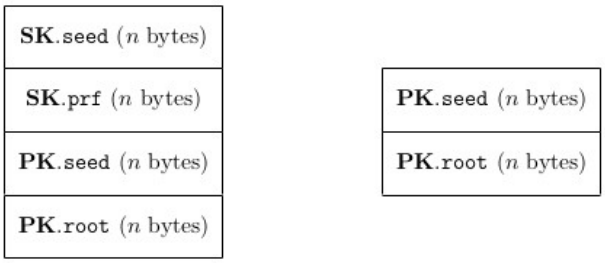


Slika 2.4: Struktura SPHINCS+

2.5.1 KLJUČEVI

Privatni ključ sastoji se od *seed*-a (SK.seed) koji se koristi za generiranje svih ostalih privatnih ključeva svih korištenih XMSS i FORS instanci. Privatni ključ sadrži i dio koji služi za deterministično generiranje vrijednosti randomizacije slučajnog sažetka poruke (SK.prf). Privatnom ključu se pridodaje javni ključ.

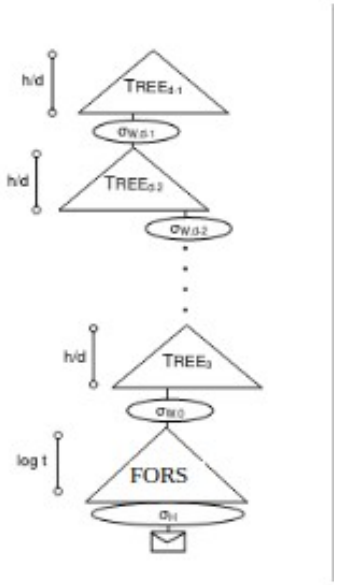
Javni se ključ također sastoji od dva dijela; javnog ključa HT strukture (PK.root) i *seed*-a (PK.seed).



Slika 2.5: Privatni i javni ključ SPHINCS+

2.5.2 POTPISIVANJE

Na poruku se prvo dodaje pseudo-slučajna vrijednost *R*, te se nad tako dobivenom porukom generira sažetak poruke *M*. Na temelju izračunatog sažetka poruke *M*, izabire se indeks FORS instance. Izabrana FORS instanca koristi se za potpisivanje sažetka *M*. Dalje se dobiveni korijen FORS instance potpisuje HT strukturom. Ukupan potpis sadrži pseudo-slučajnu vrijednost *R*, autentifikacijski put FORS instance, kao i autentifikacijski put HT strukture.



Slika 2.6: Autentifikacijski put SPHINCS+ potpisa

2.5.3 VERIFIKACIJA

Iz dobivenog potpisa uzima se pseudo-slučajna vrijednost *R*, na temelju *R* i poruke računa se njen sažetak. Iz sažetka i dobivenog autentifikacijskog puta računa se javni ključ FORS instance, a iz izračunatog ključa i

autentifikacijksog puta HT strukture računa se javni ključ PK'. Ukoliko je PK' jednak PK, digitalan potpis je valjan, u suprotnome je nevaljan.

3. PROGRAM

3.1 UVOD

Sa službene stranice natječaja (https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022) može se preuzeti zip datoteka koja sadrži izvorni kod algoritma i službena dokumentacija. U sklopu projekta, u programskom jeziku Python, napravljeno je sučelje za demonstraciju rada SPHINCS+.

3.2 IMPLEMENTACIJA

Preuzetu zip datoteku potrebno je raspakirati, zatim se pozicionirati u '`/SPHINCS-Round3/NIST-PQ-Submission-SPHINCS-20201001/Reference_Implementation/crypto_sign/sphincs-haraka-128f-robust`', te izvesti naredbu:

```
gcc -fPIC -shared -o lib_sphincs.so address.c fors.c haraka.c
hash_haraka.c PQCgenKAT_sign.c rng.c sign.c thash_haraka_robust.c utils.c
wots.c -lssl -lcrypto
```

Ovime se stvara dll datoteka naziva `lib_sphincs.so`. U datoteci api.h, opisane su funkcije koje se mogu koristiti. Njih se može pozvati iz pythona, a upute su opisane ovdje.

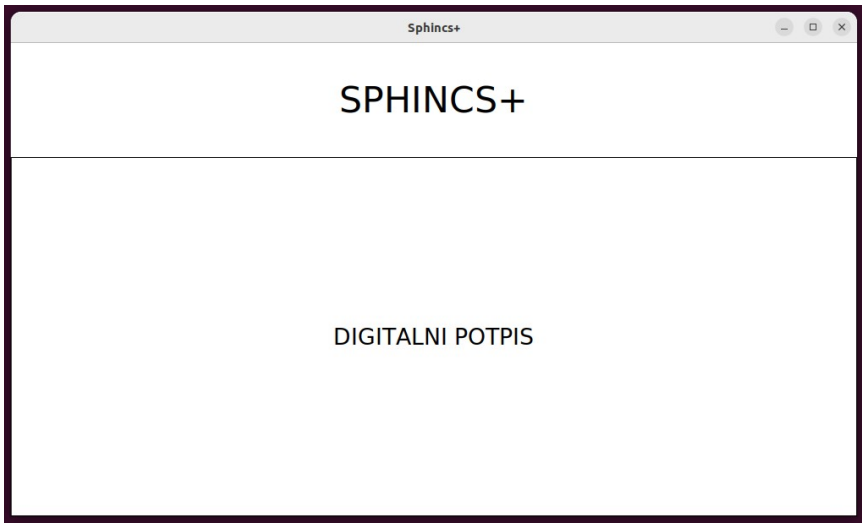
3.3 DEMONSTRACIJA

Python program pokreće se iz direktorija u kojemu se nalaze datoteke `main.py` i `sphincs.py` sljedećom naredbom:

```
$ python main.py
```

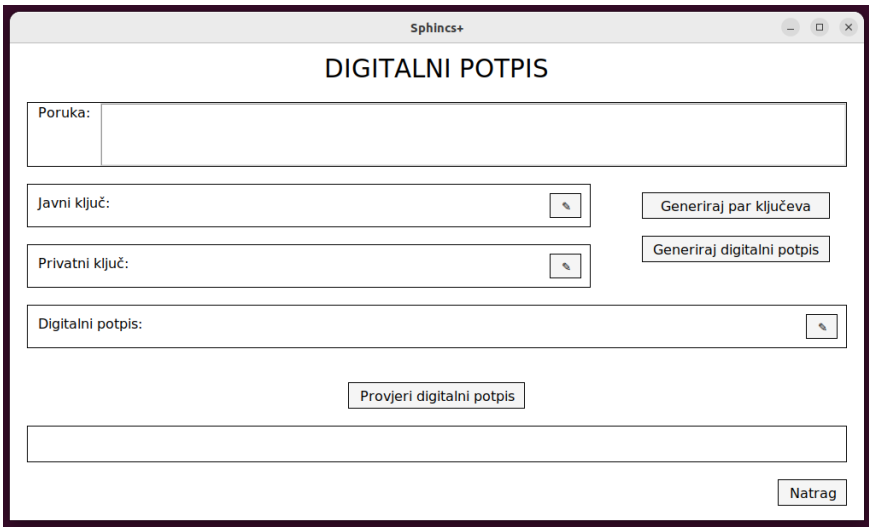
Slika 3.1: Pokretanje python programa

Otvara se prozor:



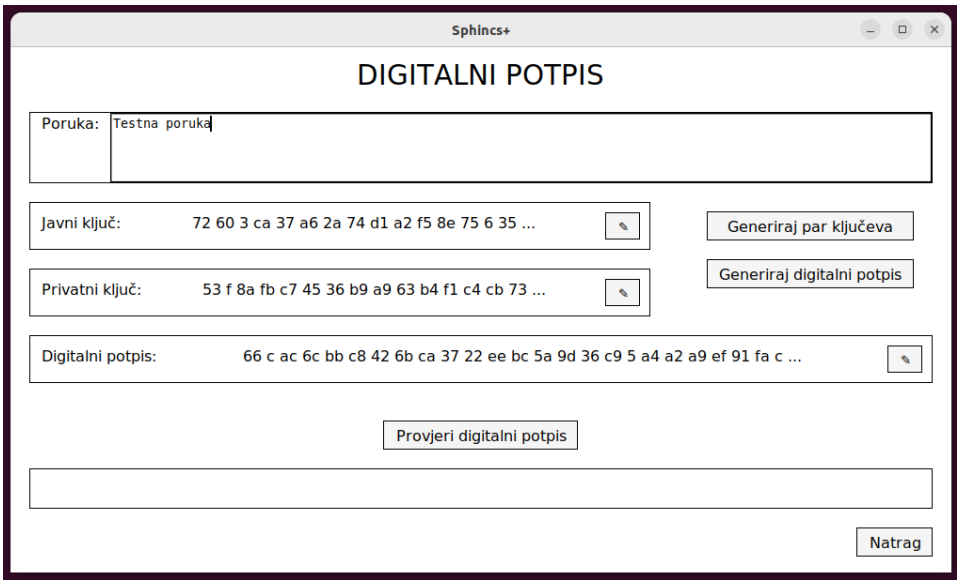
Slika 3.2: Početni zaslon

Pritiskom na dio prozora u kojemu piše "DIGITALNI POTPIS" otvara se sljedeći prozor:



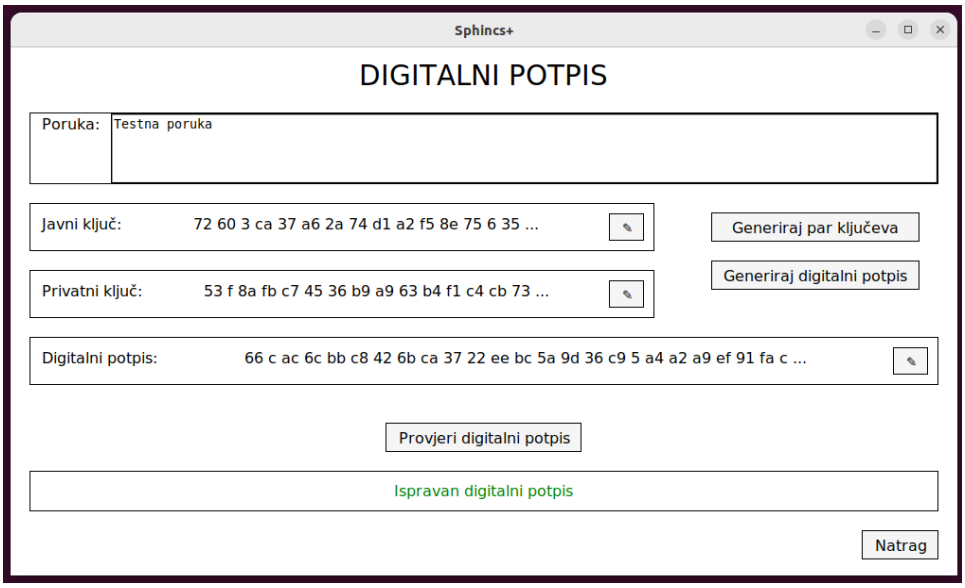
Slika 3.3: Digitalni potpis

U polje "Poruka" upisuje se željena poruka za koju će se generirati digitalni potpis.
Pritisak na gumb "Generiraj par ključeva" generira privatni i javni ključ. Pritisak na gumb "Generiraj digitalni potpis" generira digitalni potpis za upisanu poruku te par generiranih ključeva.



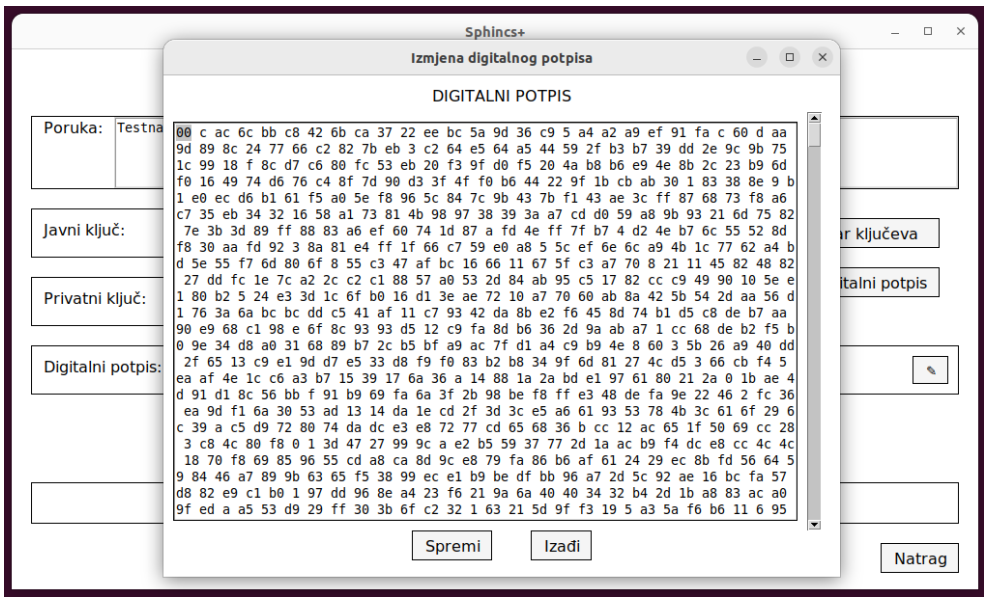
Slika 3.4: Unos poruke i generiranje ključeva

Pritiskom na gumb "Provjeri digitalni potpis" provjerava se je li digitalni potpis u polju "Digitalni potpis" valjan za par ključeva i upisanu poruku.

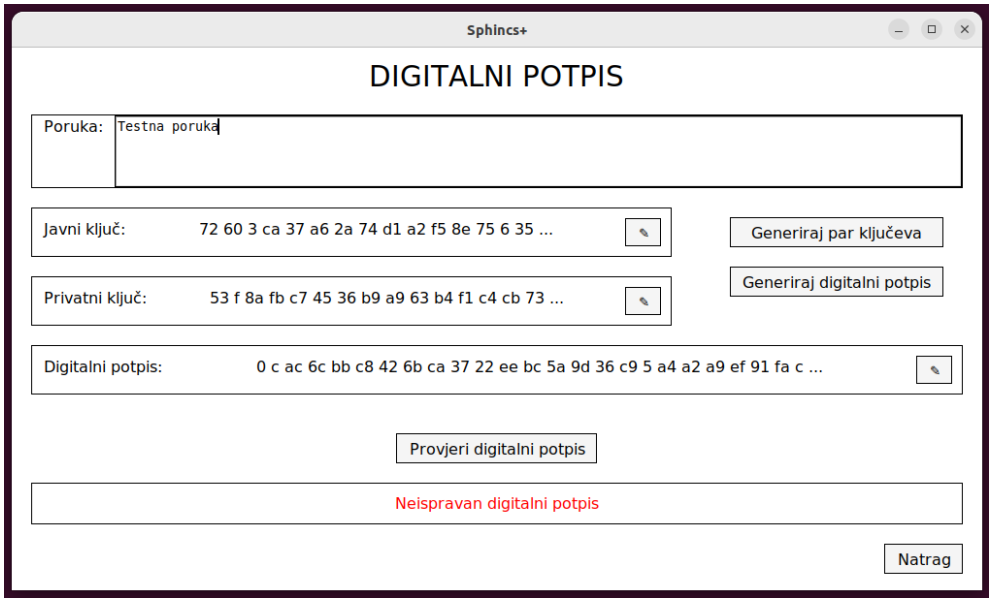


Slika 3.5: Provjera ispravnog digitalnog potpisa

Poruka o ispravnosti digitalnog potpisa prikazuje se u doljnjem prozoru. Pritiskom na gumb olovčice u poljima "Javni ključ", "Privatni ključ" i "Digitalni potpis" moguće je ručno promijeniti vrijednosti navedenih polja. Ukoliko se neka od tih vrijednosti promijeni ručno, provjera digitalnog potpisa biti će neispravna.

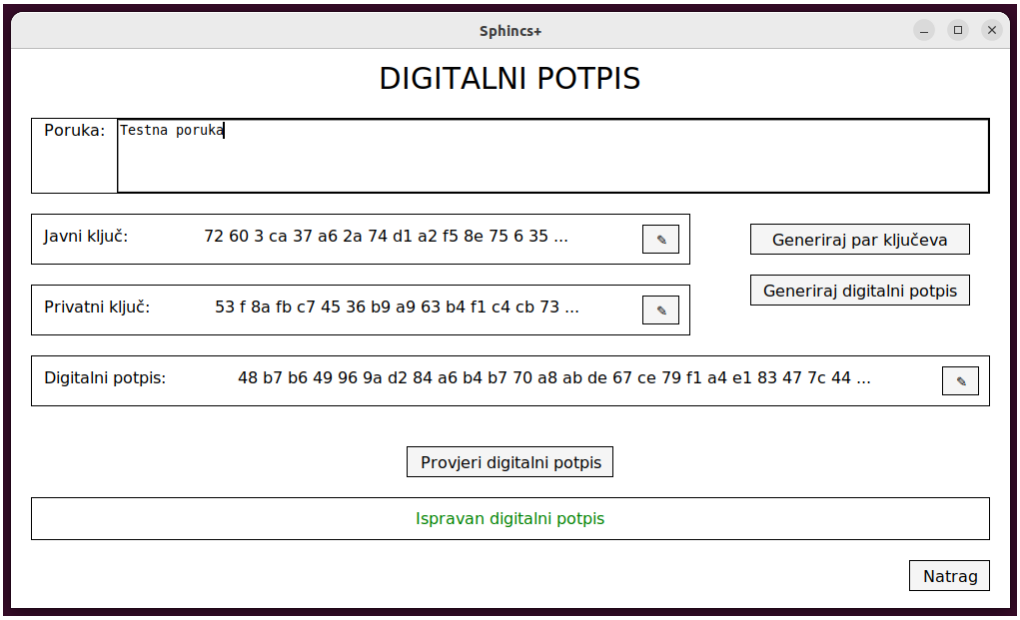


Slika 3.6: Ručna promjena digitalnog potpisa



Slika 3.7: Provjera neispravnog digitalnog potpisa

Zanimljivost SPHICS+ algoritma je što se za istu ulaznu poruku i isti par ključeva generira drugačiji valjani digitalni potpis. Navedeno ponašanje moguće je provjeriti ponovnim pritiskom na gumb "Generiraj digitalni potpis" te zatim pritiskom na "Provjeri digitalni potpis". Ovo je ponašanje objašnjeno na stranici 8.



Slika 3.8: Ponovno generiranje digitalnog potpisa

3.4 FUNKCIJE

U preuzetoj zip datoteci SPHINCS+ s NIST-ove stranice natječaja, nalazi se datoteka `api.h`, u kojoj su

deklarirane funkcije potrebne za korištenje algoritma.

3.4.1 POMOĆNE FUNKCIJE

```
unsigned long long crypto_sign_secretkeybytes(void);
```

→ Vraća duljinu privatnog ključa u oktetima.

```
unsigned long long crypto_sign_publickeybytes(void);
```

→ Vraća duljinu javnog ključa u oktetima.

```
unsigned long long crypto_sign_bytes(void);
```

→ Vraća duljinu potpisa u oktetima.

```
unsigned long long crypto_sign_seedbytes(void);
```

→ Vraća duljinu *seed*-a potrebnog za generiranje ključeva u oktetima

3.4.2 GENERIRANJE KLJUČEVA

```
int crypto_sign_seed_keypair(unsigned char *pk, unsigned char *sk, const
                             unsigned char *seed);
```

→ *pk = pokazivač na javni ključ, *sk = pokazivač na privatni ključ, *seed = pokazivač na seed
→ Funckija generira privatni i javni ključ koristeći dani seed.

```
int crypto_sign_keypair( unsigned char *pk, unsigned char *sk);
```

→ *pk = pokazivač na javni ključ, *sk = pokazivač na privatni ključ
→ Funkcija generira privatni i javni ključ.

3.4.3 GENERIRANJE POTPISA

```
int crypto_sign_signature(uint8_t *sig, size_t *siglen, const uint8_t
                          *m, size_t mlen, const uint8_t *sk);
```

→ *sig = pokazivač na potpis, *siglen = pokazivač na duljinu potpisa, *m = pokazivač na poruku, mlen = duljina poruke u oktetima, *sk = pokazivač na privatni ključ
→ Funkcija na temelju poruke m, te privatnog ključa sk generira digitalni potpis i sprema ga u sig.

```
int crypto_sign(unsigned char *sm, unsigned long long *smlen,
                const unsigned char *m, unsigned long long mlen,
                const unsigned char *sk);
```

→ *sm = pokazivač na potpis + poruku, *smlen = pokazivač na duljinu potpisa + poruke, *m = pokazivač na poruku, mlen = duljina poruke u oktetima, *sk = pokazivač na privatni ključ
→ Funkcija na temelju poruke m, te privatnog ključa sk generira digitalni potpis te ga zajedno s porukom spremu u sm

3.4.4 PROVJERA POTPISA

```
int crypto_sign_verify(const uint8_t *sig, size_t siglen, const uint8_t
                       *m, size_t mlen, const uint8_t *pk);
```

→ *sig = pokazivač na potpis, siglen = duljina potpisa u oktetima, *m = pokazivač na poruku, mlen = duljina poruke u oktetima, *pk = pokazivač na javni ključeva
→ Ukoliko je potpis valjan funkcija vraća 0, inače vrijednost različitu od 0

```
int crypto_sign_open(unsigned char *m, unsigned long long *mlen, const
                     unsigned char *sm, unsigned long long smlen,
                     const unsigned char *pk);
```

→ *m = pokazivač na poruku, *mlen = pokazivač na duljinu poruke u oktetima, *sm = pokazivač na potpis + pouku, smlen = duljina poruke + potpisa u oktetima, *pk = pokazivač na javni ključ
→ Ukoliko je potpis valjan funkcija vraća 0, inače vrijednost različitu od 0

3.5 TEST

Kako bi se provjerila ispravnost rada algoritma potrebno je provjeriti izlaz za unaprijed određene ulazne vektore. U zip datoteci nalazi se `Makefile` koji generira strojni kod upravo za takvu provjeru. Potrebno je odabrati inačicu algoritma raspršivanja koja se korisit u SPHINCS+ algoritmu (u primjeru je to haraka), pozicionirati se u taj direktorij te pokrenuti prevođenje programa `PQCgenKAT_sign.c` naredbom `make`.

```
SPHINCS-Round3/NIST-PQ-Submission-SPHINCS-20201001/Reference_Implementation/crypto_sign/sphincs-haraka-128f-robust$ make
```

Slika 3.9: Prevođenje testnog programa

Zatim pokrenuti prevedeni program naredbom `./PQCgenKAT_sign`.

```
/sphincs-haraka-128f-robust$ ./PQCgenKAT_sign
```

Slika 3.10: Pokretanje testnog programa

Ovime se u istom direktoriju generiraju dvije datoteke: `PQCsign_KAT.req` i `PQCsign_KAT.rsp`.
Njih je potrebno usporediti s istoimenim datotekama u direktoriju `/KAT/sphincs-haraka-128f-robust`.
Ako su navedene datoteke iste, algoritam radi ispravno.

4. ZAKLJUČAK

SPHINCS+ je vjerojatno najkonzervativnija post-kvantna shema digitalnog potpisa, ali je također neefikasna što se tiče veličine potpisa i brzine.

4.1 NEDOSTATCI

Jedan on glavnih nedostataka je brzina generiranja potpisa, jer se za svaki potpis moraju generirati pripadni privatni ključevi, pripadna FORS instanca te pripadna XMSS stabla iznova.

Drugi nedostatak je veličina digitalnog potpisa, jer se unutar svakog digitalnog potpisa moraju nalaziti i pripadni autentifikacijski putevi kako bi se iz njih mogao izračunati javni ključ za usporedbu.

4.2 PREDNOSTI

Za razliku od ostalih post-kvantnih algoritama, cijela sigurnost SPHINCS+ digitalnog potpisa temelji se isključivo na sigurnosti odabranih *hash* funkcija. Budući da i sigurnost svih ostalih algoritama ovisi o *hash* funkcijama, ovo smanjuje mjesto za pogrešku i potencijalne napade, te čini algoritam nezavisnijim.

Mala duljina ključeva je također jedna od prednosti, pogotovo duljina javnog ključa koji se često prenosi za vrijeme komunikacije.

5. LITERATURA

Radovi:

(1) Autori: Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Bas Westerbaan
Ime rada: SPHINCS+

(2) Autori: Andreas Hülsing
Ime rada: W-OTS+– Shorter Signatures for Hash-Based Signature Schemes

(3) Autori: Johannes Buchmann, Erik Dahmen, and Andreas Hülsing
Ime rada: XMSS – A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions

(4) Autori: Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn
Ime rada: SPHINCS: practical stateless hash-based signatures

(5) Autori: A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, A. Mohaisen
Ime rada: XMSS: eXtended Merkle Signature Scheme

Internet stranice:

(1) NIST-National Institute of Standards and Technology
Stranica: https://www.nist.gov/

(2) SPHINCS+, **Autor**: Peter Schwabe
Stranica: https://sphincs.org/

(3) Hash-Based Signatures Part IV: XMSS and SPHINCS, **Autor**: David Wong
Stranica: https://cryptoservices.github.io/quantum/2015/12/08/XMSS-and-SPHINCS.html

(4) Tweakable Hash Functions in Stateless Hash-Based Signature Schemes**, Autor**: Lena Heimberger
Stranica: https://www.iaik.tugraz.at/wp-content/uploads/teaching/mfc/THash.pdf