Postkvantna kriptografija – CRYSTALS - Dilithium

Tehnička dokumentacija

# Verzija 1

# Student: Velimir Kovačić

# Nastavnik: doc.dr.sc. Marin Golub

Sadržaj

[1. Uvod 1](#_Toc123681279)

[1.1 CRYSTALS – Dilithium 1](#_Toc123681280)

[1.2 Matematička pozadina 1](#_Toc123681281)

[1.2 Fiat-Shamir with Aborts 2](#_Toc123681282)

[1.3 Dilithium poboljšanja 3](#_Toc123681283)

[1.3.1 Spremanje matrice **A** 3](#_Toc123681284)

[1.3.2 Spremanje vektora polinoma **t** 3](#_Toc123681285)

[1.3.3 Determinizam 3](#_Toc123681286)

[1.3.4 Oblik polinoma NTT 4](#_Toc123681287)

[3. Algoritam 4](#_Toc123681288)

[3.1 Važne funkcije 4](#_Toc123681289)

[3.1.1 Hashing to a Ball 4](#_Toc123681290)

[3.2.2 Ekspanzija matrice A 5](#_Toc123681291)

[3.1.3 Generiranje vektora 5](#_Toc123681292)

[3.1.4 Hash otporan na kolizije 5](#_Toc123681293)

[3.1.5 Rastavljanje brojeva 5](#_Toc123681294)

[3.1.6 Hintovi 6](#_Toc123681295)

[3.2 Važni parametri 7](#_Toc123681296)

[3.3 Generiranje ključeva 8](#_Toc123681297)

[3.4 Potpisivanje 9](#_Toc123681298)

[3.5 Provjera potpisa 10](#_Toc123681299)

[4. Program za testiranje 11](#_Toc123681300)

[4.1 Uvod 11](#_Toc123681301)

[4.2 Demonstracija 11](#_Toc123681302)

[4.2.1 Početni prozor 11](#_Toc123681303)

[4.2.2 Prozor za digitalni potpis 12](#_Toc123681304)

[4.2.3 Prozor za enkripciju 14](#_Toc123681305)

[4.3 Implementacija 15](#_Toc123681306)

[5. Literatura 16](#_Toc123681307)

# 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 . Taj prsten sadrži sve polinome oblika , za koje vrijedi da im je stupanj manji od , a koeficijenti manji od što se osigurava time da se zbrajanje i množenje koeficijenata odvija pod .

Dakle:

Koristimo vrijednosti , .

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

Maksimalni koeficijent polinoma je:

Gdje je -ti koeficijent polinoma , a centrirani modulo , za :

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

predstavlja skup svih elemenata kao , ali bez onih koji imaju koeficijente jednake :

## 1.2 Fiat-Shamir with Aborts

GenerirajKljučeve():

**A** ←

(**s**1, **s**2) ← ×

**t** := **As**1 + **s**2

*pk* := (**A**, **t**)

*sk* := (**A**, **t**, **s**1, **s**2)

**vrati** (*pk*, *sk*)

Potpiši(*sk*, *M*):

**z** := ⊥

**dok je z** = ⊥:

**y** ←

**w** := **Ay**

**w**1 := HighBits(**w**, 2)

∈ := H(*M* ‖ **w**1)

**z** := **y** + **s**1

**ako je z** ≥ - **ili** LowBits(**Ay** − **s**2, 2) ≥ – :

**z** := ⊥

**vrati** *σ* = (**z**, )

ProvjeriPotpis(*pk*, *M*, *σ* = (**z**, )):

**w**'1 := HighBits(**Az** − **t**, 2)

**ako je z**< – **i**  == H( **w**'1):

**vrati** potpis je valjan

**inače:**

**vrati** potpis nije valjan

U funkciji GenerirajKljučeve(), koja ne prima nikakve argumente, prvo se stvori matrica **A** veličine u čijoj je svakoj ćeliji polinom iz prstena polinoma . Zatim se nasumično generiraju 2 vektora polinoma **s**1 duljine i **s**2 duljine , tako da su koeficijenti svih polinoma najviše . 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 ⊥ što predstavlja vrijednost *none*. Petlja se provodi dok je vektor polinoma **z** jednak ⊥. Vektor polinoma **y** duljine nasumično se generira tako da su svi koeficijenti manji od . **w** je vektor polinoma jednak umnošku matrice polinoma **A** i vektora polinoma **y**. **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 **w** = **w**1 · 2 + **w**0, **w**1 predstavlja više, a **w**0 niže bitove, vrijedi |**w**0| ≤ . iz *Rq* 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 **w**1. Vektor polinoma **z**, koji predstavlja potencijalni digitalni potpis, računa se kao zbroj vektora polinoma **y** i umnoška polinoma c s vektorom polinoma **s**1. Ako je najveći koefcijent polinoma vektora polinoma **z** veći ili jednak – ili su niski bitovi **Ay** − c**s**2 veći ili jednaki – , vektor polinoma **z** postavlja se na početnu vrijednost ⊥ te se tako ponavlja petlja. Kada se pronađe vektor **z** i polinom takvi da zadovolje uvjete vraća se potpis *σ* koji sadrži samo njih.

Za provjeru ispravnosti digitalnog potpisa koristi se funkcija ProvjeriPotpis(*pk*, *M*, *σ* = (**z**, c))

koja kao argumente prima javni ključ *pk*, poruku *M* i digitalni potpis *σ*. Vektor polinoma **w**'1 računa se kao visoki bitovi od **Az** – **t**.

Ukoliko je potpis ispravan vrijedi: **w**'1 = HighBits(**Az** − **t**, 2) = HighBits(**A**(**y** + **s**1)– (**As**1 + **s**2), 2) = HighBits(**Ay** – **s**2, 2) = HighBits(**Ay**) = HighBits(**w**) = **w**1.

Ako je **w**'1 jednak **w**1 iz funkcije Potpiši(*sk*, *M*), hash koji daje konkatiniran uz M biti će jednak polinomu . Uz to se provjerava i svojstvo **z**< – , 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 polinoma stupnja do 255. Umjesto eksplicitnog spremanja matrice **A**, sprema se *seed* 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 **Az** – **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** – **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 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 , a pretvorba u oblik NTT odvija se u vremenskoj složenosti . Dakle, umjesto množenja polinoma u kanonskom obliku u vremenskoj složenosti , možemo ih pomnožiti u vremenskoj složenosti O().

# 3. Algoritam

## 3.1 Važne funkcije

### 3.1.1 Hashing to a Ball

Funkcija SampleInBall() vraća jedan element iz skupa .

predstavlja skup svih elemenata iz koji imaju točno τ koeficijenata koji su ili 1 ili -1, dok su svi ostali koeficijenti 0. Takvih elemenata u ima .

Za funkciju SampleInBall() potreban je hash koja hashira na skup 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():

Inicijaliziraj **c** = . . . =

**za**  := 256 - do 255

:= broj iz skupa odabran koristeći XOF funkciju

:= broj iz skupa odabran koristeći XOF funkciju

:=

:=

**vrati c**

### 3.2.2 Ekspanzija matrice A

ExpandA()

Kako se u ključevima ne bi morala pohranjivati cijela vrijednost matrice **A**, pohranjuje se *seed* i svaki put kada je potrebno generira se matrica u obliku NTT.

### 3.1.3 Generiranje vektora

ExpandMask(, )

Koristeći seed i nonce , ova funkcija generira vektor duljine koji se sastoji od polinoma iz .

### 3.1.4 Hash otporan na kolizije

CRH()

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

### 3.1.5 Rastavljanje brojeva

#### Power2Roundq(, )

Za binarni broj duljine bitova, funkcija Power2Round*q*(, ) vraća par u kojem je prvi član prvih (viših) bitova, a drugi član zadnjih (nižih) bitova. .

Power2Round*q*(, ):

:= mod+

:= mod±

**vrati**

#### Decompose*q*(, )

Slično kao funkcija Power2Roundq(, ), funkcija Decomposeq(, ) rastavlja binarni broj na više i niže bitove ( i ) i vraća ih. Koristeći (parni djelitelj broja ) dijeli ih na način da .

Decomposeq(, ):

:= mod+

0 := mod±

**ako je**  − 0 = – 1:

1 := 0

0 := 0 − 1

**inače:**

1 := ( − 0)/

**vrati** (1, 0)

#### HighBits*q*(, ) i LowBits*q*(, )

Pomoćne funkcije koje služe za dohvaćanja viših i nižih bitova koje vraća funkcija Decompose*q*(, ).

HighBits*q*(, ):

(1, 0) := Decompose*q*(, )

**vrati** 1

LowBits*q*(, ):

(1, 0) := Decompose*q*(, )

**vrati** 0

### 3.1.6 Hintovi

#### MakeHintq(, , )

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 sa .

MakeHint*q*(, , ):

1 := HighBits*q*(, )

1 := HighBits*q*( + , )

**ako** 1 == 1:

**vrati 0**

**inače:**

**vrati 1**

#### UseHint*q*(, , )

Funkcija UseHint*q*(, , ) na temelju *hinta* koji može biti 0 ili 1 ispravlja i vraća više bitove broja .

UseHint*q*(, , ):

:=

(1, 0) := Decompose*q*(, )

**ako**  = 1 i 0 > 0:

**vrati** (1 + 1) mod+

**inače ako**  = 1 and 0 ≤ 0:

**vrati** (1 − 1) mod+

**inače:**

**vrati** 1

#### Leme vezane uz korištenje i stvaranje *hintova*

1. UseHint*q*(MakeHint*q*(, , ), , ) = HighBits*q*( + , ).
   1. Stvara se *hint* na temelju zbroja i i koristi se na visokim bitovima što je jednako visokim bitovima zbroja i .
2. **Ako** ≤ i LowBits*q* (, ) < /2 − **vrijedi**   
   HighBits*q*(, ) = HighBits*q*( + , )
   1. Ova lema daje dovoljan uvjet za slučaj da se neće mijenjati visoki bitovi kada mu se pribroji .
3. **Neka je** (1, 0) = Decompose*q*(, ), (1,0) = Decompose*q*( + , ) i ≤ **vrijedi:** + 0 < /2 − **ako i samo ako** 1 = 1 i 0 < /2 −

## 3.2 Važni parametri

Slika na kojoj se prikazuje tekst, stol

Opis je automatski generiran

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

## 3.3 Generiranje ključeva

GenerirajKljučeve():

← {0, 1}256

(, , ) ∈ {0, 1}256×3 := H()

(**s**1, **s**2) ∈ × := H()

**Â** := ExpandA()

**t** :=NTT-1(**Â** · NTT(**s**1)) + **s**2

(**t**1, **t**0) := Power2Round*q*(**t**, )

∈ {0, 1}384 := CRH( ‖ **t**1)

*pk* := (, **t**1)

*sk* := (, , , **s**1, **s**2, **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 , nasumični 256-bitni broj. Na temelju hasha od se generiraju brojevi , i . Hashiranjem generiraju se vektori: **s**1, duljine i **s**2, duljine . Koeficijenti ovih vektora su polinomi iz . Iz broja generira se i privremeno sprema **Â** (oblik NTT matrice **A**). Svako polje matrice **A** polinom je iz prstena polinoma , a sama matrica veličine je , dakle **A** ∈ . Računa se vektor polinoma **t** koristeći pretvorbu NTT te se potom dijeli na više bitove **t**1 i niže bitove **t**0 funkcijom Power2Round*q*(**t**, ). Hash koji na ulazu prima konkatiniran broj i **t**1 (više bitove vektora **t**) na izlazu daje broj .

Javni ključ *pk* sastoji se od broja i **t**1 (viših bitova vektora **t**). Privatni ključ *sk* sastoji se od brojeva , i i vektora **s**1, **s**2 i **t**0 (viših bitova vektora **t**).

Funkcija vraća javni i privatni ključ.

## 3.4 Potpisivanje

Potpiši(*sk*, *M*):

**Â** := ExpandA()

∈ {0, 1}384 := CRH( *M*)

:= 0

(**z**, **h**) := ⊥

∈ {0, 1}384 := CRH() (ili ← {0, 1}384)

**ŝ**1 := NTT(**s**1)

**ŝ**2 := NTT(**s**2)

**t̂**0 := NTT(**t**0)

**dok je** (**z**, **h**) == ⊥:

**y** ∈ := ExpandMask(, )

**w** := NTT−1(**Â** · NTT(**y**))

**w**1 := HighBits*q*(**w**, 2)

∈ {0, 1}256 := H( **w**1)

:= NTT(SampleInBall())

**z** := **y** + NTT-1( · **ŝ**1)

**r**0 := LowBits*q*(**w** − NTT-1( · **ŝ**2), 2)

**ako je z** ≥ - **ili** **r**0 ≥ - :

(**z**, **h**) := ⊥

**inače:**

**h** := MakeHint*q*(−NTT-1( · **t̂**0), **w** − NTT-1( · **ŝ**2) + NTT-1( · **t̂**0), 2γ2)

**ako je** NTT-1( · **t̂**0) ≥ **ili**  (broj jedinica u **h**) > :

(**z**, **h**) := ⊥

= +

**vrati** *σ* = (**z**, **h**, )

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

Iz broja ρ generira se i privremeno sprema **Â** (oblik NTT matrice **A**). μ se generira kao 384-bitni broj iz hasha otpornog na kolizije (CRH) koji na ulazu prima konkatinirani broj iz tajnog ključa i poruku *M*. Broj postavlja se na 0, a par vektora (**z**, **h**) postavljaju se na vrijednost ⊥ (*none*/ništa). U slučaju determinističkog potpisivanja se generira kao 384-bitni broj iz hasha otpornog na kolizije (CRH) koji na ulazu prima konkatinirani broj iz tajnog ključa i prethodno generirani broj . U slučaju nedeterminističkog potpisivanja se generira kao nasumični niz bitova.

Izračunaju se oblici NTT vektora polinoma **s**1, **s**2 i **t**0.

Provodi se petlja dok je par vektora (**z**, **h**) jednak ⊥. Na temelju i generira se vektor polinoma duljine , a svaki njegov polinom ima koeficijente manje od . Vektor polinoma **w** izračuna se kao umnožak matrice **A** i vektora **y** koristeći pretvorbu NTT. Vektor **w**1 dobije se primjenjujući HighBits*q* funkciju uz = 2 na vektor **w**. , 256-bitni broj dobije se hashiranjem kontkatiniranog broja s vektorom polinoma **w**1. Polinom dobiva se funkcijom SampleInBall kojoj se kao ulazni argument prosljeđuje broj . Vektor polinoma **z** dobiva se sumiranjem vektora polinoma **y** i umnoška polinoma s vektorom polinoma **s**1. Množenje se provodi koristeći pretvorbe NTT. Vektor polinoma **r**0 jednak je razlici funkciji nižih bitova razlike **w** - **s**2 (množenje pretvorbama NTT) uz = 2.

Ako je najveći koeficijent polinoma u vektoru polinoma z veći ili jednak – ili je najveći koeficijent polinoma u vektoru polinoma **r**0 veći ili jednak – , postavljamo par (**z**, **h**) na početnu vrijednost ⊥.

Inače stvaramo vektor *hintova* **h** funkcijom MakeHint*q* za zbroj −**t**0 i **w** − **s**2 + **t**0. Ako je pak najveći koeficijent polinoma u vektoru polinoma umnoška **t**0 veći ili jednak ili je broj jedinica u vektoru *hintova* **h** veći od , postavljamo par (**z**, **h**) na početnu vrijednost ⊥.

Na kraju svakog ponavljanja broju κ dodajemo broj .

Kada se uspije stvoriti zadovoljavajući par (**z**, **h**), vraća se potpis *σ* koji se sastoji od vektora polinoma **z**, vektora **h** i broja .

## 3.5 Provjera potpisa

ProvjeriPotpis(*pk*, *M*, *σ* = (**z**, **h**, )):

**Â** := ExpandA()

∈ {0, 1}384 := CRH(CRH( ‖ **t**1) ‖ *M*)

:= SampleInBall()

**w**'1 := UseHintq(**h**, NTT-1(**Â** · NTT(**z**) – NTT() · NTT(**t**1 · )), 2)

**ako je z**< – **i**  == H( **w**'1) **i** broj jedinica u **h** ≤ :

**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 (koji se izračunava iz i **t**1 kao u funkciji GenerirajKljučeve()) i poruke *M*. se izračuna funkcijom SampleInBall iz . Iskorištava se vektor hintova **h** iz potpisa na razliku **Az** – **t**1 · funkcijom UseHint*q* te se rezultat sprema kao **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 **w**'1 se provjerava preko hash funkcije iz čega se donosi odluka o ispravnosti potpisa.

# 4. Program za testiranje

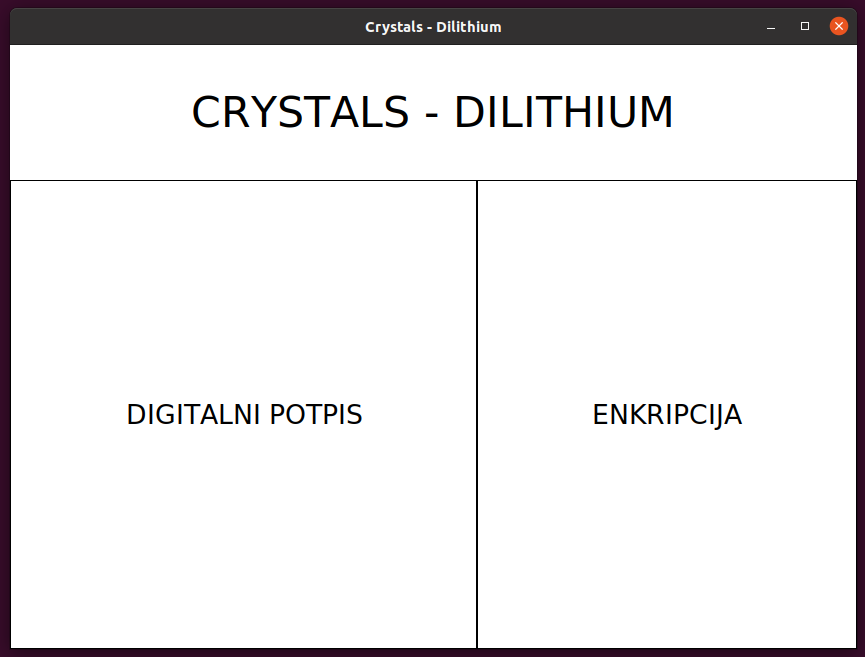
## 4.1 Uvod

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

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

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.

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

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

Slika na kojoj se prikazuje tekst

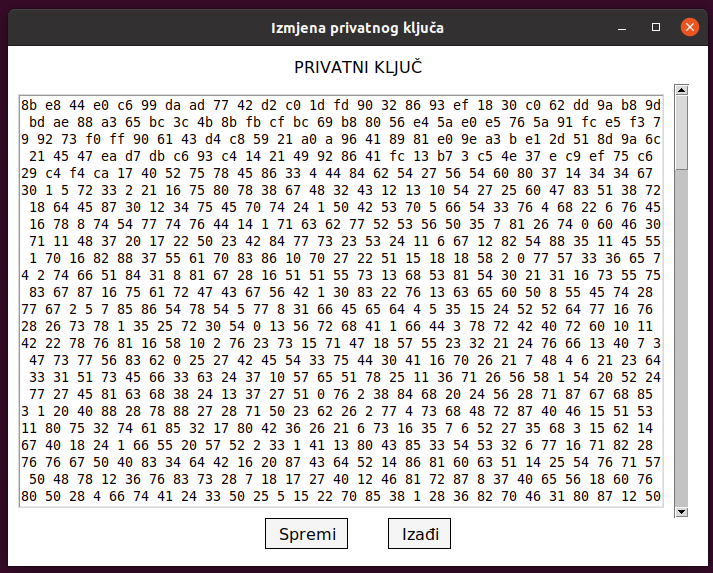
Opis je automatski generiran

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.

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

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.

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

## 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“.

Slika na kojoj se prikazuje tekst, zid, srebrno

Opis je automatski generiran

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

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

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>)