Programiranje v Pythonu

Izdelava programa za steganografijo slik

Matej Marinko

februar 2017

Kazalo

1 Uvod

Kmalu potem, ko sem se naučil programirati me je začelo zanimati kriptiranje, varovanje podatkov in podobno. Pri kriptiranju pa eden ključnih problemov to, da ko želimo nekomu nekaj sporočiti, to vsi drugi opazijo. Kljub temu, da ne vedo o čem poteka pogovor, še vedno vedo kdo komunicira s kom. Steganografija pa odpravi točno ta problem. Sporočilo, ki ga pošljemo je zakrinkano, tako da sploh ni opazno da želimo komu kaj sporočiti.

Odločil sem se, da bom kljub temu, da že obstaja več programov za steganografijo slik, naredil svoj program. Izbral sem si programski jezik Python, saj imam v njem daleč največ izkušenj, v primerjavi z ostalimi programskimi jeziki.

2 Steganografija

Steganografija je znanost, ki omogoča skrivanje podatkov v navidezno nepomembnem prenosnem mediju. Beseda izhaja iz grščine, in pomeni "zakrito pisanje". Predstavlja skupek metod za skrivanje informacij v neke druge informacije. Za razliko od kriptografije oz. šifriranja, ki želi podatke narediti neberljive, steganografija poskuša prikriti, da ti podatki sploh obstajajo [4].

Šibkost kriptografije je, da je sporočilo sumljivo že na prvi pogled. Šibkost steganografije pa je, da ko nekdo enkrat odkrije, kje je sporočilo skrito, ga lahko enostavno prebere. V praksi se zato pogosto uporabljata obe metodi skupaj, saj druga drugi odpravita slabosti.

2.1 Tehnike

Primeri steganografije so znani že od Antike, različne tehnike pa so se ohranile vse do danes. Z razvojem računalništva se je steganografija razvila tudi na digitalnem področju.

Primeri fizičnih tehnik:

- V Stari Grčiji so sporočila skrivali v voščene tablice. Na les so napisali sporočilo, ki so ga nato prekrili z voskom. Na vosek pa so napisali nedolžno in nepomembno sporočilo.
- Drugi postopek, ki so ga uporabljali v Antični Grčiji je bil približno takšen: Sužnju so pobrili glavo in mu vtetovirali sporočilo. Ko so lasje zrasli sporočilo ni bilo več vidno. Očitna pomankljivost te metode je, da moramo počakati, da osebi lasje zrastejo nazaj.
- Sporočilo, ki je napisano z nevidnim črnilom, na nepopisanem delu pisma.
- Sporočilo je v Morsejevi abecedi, napisani na prejo, ki so jo potem vtkali kurirju v blago.
- Del črk v besedilu je napisan z drugačno pisavo kot druge črke (npr. ležeče). Te črke so tvorile skrivno sporočilo. [3]
- Nemci so med drugo svetovno vojno uporabljali mikropike (microdots). Podatke so skrivali v znakih, ki jih je bilo mogoče prebrati samo pod veliko povečavo. [4]
- Cardanovo rešeto - mreža z odprtinami, s katero prekrijemo besedilo, da se prikaže skrito sporočilo.

- Ameriški pilot Jeremiah Denton, ki ga je ujela vietnamska vojska, je bil med televizijsko konferenco prisiljen pričati, da z njim v ujetništvu ravnajo dobro. Hkrati pa je z mežikanjem v morsejevi abecedi črkoval besedo T-O-R-T-U-R-E (mučenje).[5]

Primeri digitalnih tehnik:

- Skrivanje podatkov v najnižje bite slikovnih datotek, z neopaznimi spremembami barv.
- Skrivanje podatkov v zvočni zapis, s spremembami, neslišnimi za človeško uho.
- Tehnika "pletja in presejanja" (chaffing and winnowing), kjer gre za to da paketkom, ki gredo čez nezavarovano povezavo dodamo lažne paketke, v katere lahko skrijemo sporočilo.
- Dodajanje podatkov v neuporabljene sektorje na disku, ali na konce sektorjev.
- Skrivanje z uporabo unicode znakov, ki izgledajo enako kot ASCII znaki.
- Nekateri moderni tiskalniki z težko vidnimi svetlo rumenimi pikami na vsakem listu označijo serijsko številko tiskalnika in čas tiska. [3]

2.2 Steganografija digitalnih fotografij

Večja kot je datoteka, v katero nameravamo skriti naše sporočilo, v primerjavi z tem sporočilom, lažje jo je skriti. Zato so slike primerne za steganografijo, saj vsebujejo velike količine podatkov. Tako lahko skrijemo podatke npr. na Internetu. Slika je vsem na očeh, venar se nihče ne zaveda, da so v njej skrite še dodatne informacije. Ni znano, kako pogosta je ta praksa, vendar vemo da obstaja.

V RGBA zapisu uporabimo 32 bitov za vsak piksel, to pomeni 8 bitov za vsako komponento. Samo rdeča barva ima 2^8 različnih intenzivnosti. (Glej poglavje: Zapis slik v računalniku) Razliko med $10111111_{(2)}$ in $10111110_{(2)}$ v inteziteti barve človeško oko zelo težko prepozna. Zato lahko najnižje bite (least significant bits) spremenimo in v njih skrijemo svoje podatke. Če v vsakem pikslu v vseh štirih komponentah spremenimo 2 najmanj pomembna bita, lahko v en piksel skrijemo 1 bajt (8 bitov) podatkov. V celotni sliki lahko, če je dovolj velika, skrijemo tudi več MB podatkov.

Pri tem je pomembno, da slika, v katero želimo skriti podatke ni enobarvna, oziroma sestavljena iz večjih enobarvnih ploskev. Če je slika takšna, obstaja večja možnost, da kdo opazi različne odtenke sosednjih pikslov na sliki. Zato so slike, ki se uporabljajo za steganografijo pogosto slike narave, živali, ipd.

2.3 Steganaliza

Steganaliza je veda, ki se ukvarja z zaznavanjem sporočil, skritih s pomčjo steganografije. Pogosto se povezuje s kriptoanalizo. Namen je torej najti sumljiva sporočila, ter ugotoviti ali se v njih skriva skrito sporočilo in, če je možno, prebrati to sporočilo.

Problema se ponavadi lotimo z statistično analizo. Analiziramo recimo slike, ki so bile posnete z enakim fotoaparatom, ali zvočne posnetke in ugotavljamo skupne značilnosti. Zaradi pogoste izgubne kompresije je predvidljivo, kakšni naj bi bili podatki. V JPEG sliki, naprimer, lahko precej dobro sklepamo, katere barve je piksel, če poznamo vse sosednje piksle. Ker so takšne razporeditve predvidljive, bodo slike, v katerih je skrito stenografsko sporočilo hitreje opazne.

Najlažje pa je odkriti skrito sporočilo, če imamo na voljo originalno sliko, v kateri ni skritih podatkov, saj bomo hitro opazili razliko in bomo posledično lažje izluščili podatke.

Še naprednejše metode predpostavijo, da so podatki, ki so skriti poleg tega še kriptirani. Posledica sodobne enkripcije je, da so podatki videti naključni. Večina metod skriva podatke v najmanj pomembne bite (least-significant bits). Če bo razporeditev 1 in 0 v najmanj pomembnih bitih skoraj popolnoma naključna, je velika verjetnost, da je v datoteki skrito kriptirano sporočilo.

3 Zapis slik v računalniku

Slika je v računalniku zapisana tako, da je razdeljena na drobne kvadratke, imenovane piksli. Vsak piksel ima podatke kako močno in v katerih barvah žari. Za zapis se uporablja več različnih formatov, med katerimi so najbolj znani JPEG, PNG, BMP, GIF... Med seboj se razlikujejo v različnih lastnostih, kot so način zgoščevanja, število možnih barv posameznega piksla...

Osnovna zgradba vsakega formata sestoji iz:

glave, ki vsebuje glavne podatke, kot so velikost, barvna globina in kompresijska tehnika.

slikovnih podatkov oziroma niza pikslov. Podatki so lahko kompresirani ali nekompresirani.

polj za metapodatke (metadata), kot so datum posnetka, avtor slike...

3.1 Format PNG

PNG (Portable Network Graphics) je razmeroma nov slikovni format, ki je popularen predvsem na spletu.

Format PNG uporablja brezizgubno kompresijo, kar nujno potrebujemo pri steganografiji, saj se zanašamo na to da bomo podatke lahko skrili tako, da jih človeško oko ne bo zaznalo. Izgubna kompresija pa izpušča podatke, ki jih človeško oko ne more zaznati, ter tako onemogoča pridobivanje skritega sporočila nazaj iz slike.

Ena izmed prednosti formata PNG pred drugimi slikovnimi formati je podpora več različnih barvnih tabel. Poleg standardnega RGB (Red Blue Green) zapisa podpira tudi RGBA (Red Green Blue Alpha) zapis, katerega bistvena razlika je dodaten alpha kanal. Le-ta se navadno uporablja za prosojnost slik. Če je vrednost kanala 0% je piksel popolnoma prosojen, če pa je vrednost 100% pa je piksel enak običajnim pikslom.

V primeru steganografije dodaten kanal veliko pripomore, saj lahko v sliko skrijemo kar $\frac{1}{4}$ več informacij, kot v zapisu RGB (Glej poglavje: Steganografija digitalnih fotografij).

4 AES kriptiranje

AES (Advanced Encryption Standard) je eden najbolj uporabljenih standardov za simetrično enkripcijo. Simetrična enkripcija pomeni, da imata pošiljatelj in prejemnik isti ključ, s katerim kriptirata oziroma dekriptirata sporočilo.

Algoritem sta razvila belgijska kriptografa Joan Daemen in Vincent Rijmen, ter ga poimenovala Rijndael. Pozneje je standard z manjšimi spremembami prevzela ameriška vlada kot naslednji standard po uporabi DES (Data Encryption Standard), saj so ključi DES postali prekratki in jih je bilo mogoče razbiti z močnejšimi računalniki.

AES je "substitution-permutation network", kar pomeni, da glavnino operacij, ki jih opravlja predstavljajo različne zamenjave in permutacije bitov.

Poleg višje varnosti je glavna prednost standarda AES hitrost. Operacije, ki jih opravlja so nezahtevne, v nasprotju z asimetričnimi kriptiranji. AES je možno celo zapisati v vezje v procesorju, tako da lahko današnji običajni namizni računalniki kriptirajo AES tudi z več TB/s.

4.1 Algoritem

Algoritem poenostavljeno sestoji iz štirih korakov, ki se naprej delijo na manjše korake.

1. `KeyExpansions` - razširitve ključa. Ključ, ki je vnaprej določene dolžine se razširi na več ključev, saj AES za vsak krog zahteva svoj ključ.

2. `InitialRound` - dodajanje ključa. Nastavjo se začetna stanja s pomočjo posameznih delov ključa.

3. `Rounds` - del, ki se večkrat ponovi, vsakokrat z novim ključem, ki smo ga razširili iz originalnega ključa. Če je ključ 128-biten se ponovi 10-krat, 12-krat pri 192-bitnih ključih in 14-krat pri 256-bitnih ključih.

   (a) `SubBytes` - preprosta substitucija znakov z uporabo tabele. Pri tem je pomembno, da ime tabela določene lastnosti, ki naredijo to preslikavo nelinearno, kar zelo oteži razbijanje šifre.

   (b) `ShiftRows` - operacija na vrsticah trenutnega stanja. Vsak bit v neki vrstici se ciklično zamakne za neko število. Biti iz konca se premaknejo na začetek.

   (c) `MixColumns` - korak, kjer se stolpci zamenjajo z drugimi stolpci. Vsa stanja v novem stolpcu so neposredno odvisna od vsakega posameznega stanja v prvotnem stolpcu. Če spremenimo en znak, se popolnoma spremeni celoten novi stolpec.

   (d) `AddRoundKey` - korak skoraj enak postopku v InitialRound. Stanju se doda nov ključ, ki ustreza trenutnemu krogu, z operacijo `XOR` (ekskluzivni ali).

4. `Final Round` - še zadnja ponovitev, ki je skoraj enaka vsem ostalim krogom, edina razlika je, da ne vsebuje koraka `MixColums`.[2]

5 Python

Python je sodobno programski jezik, ki je primeren za razvoj najrazličnejših programov, od preprostih skript do numerično zahtevnih simulacij in sodobnih spletnih aplikacij. Zaradi svoje enostavnosti je postal eden najpriljubljenejših programskih jezikov vseh časov, ter je primeren za učenje programiranja. Python je tolmačen jezik, to pomeni, da se sproti med izvajanjem pretvarja v strojno kodo.[1] Zato je razmeroma počasnejši od prevajanih jezikov, kot so C++, Java in C#. V praksi se z uporabo različnih knjižnjic, kot je numpy, njegova hitrost izvajanja lahko približa hitrosti teh jezikov.

Prednost (in hkrati tudi slabosti Pythona) je uporaba dinamičnih tipov, kar pomeni, da imamo lahko v istu spremenljivki ob različnih časih različne podatkovne tipe. Sintaksa Pythona omogoča, da razvijalci prišejo kodo hitreje kot v drugih programskih jezikih, saj potrebujejo manj vrstic kode kot v konkurenčnih programskih jezikih.

5.1 Verzije Pythona

Python trenutno obstaja v dveh glavnih verzijah. To sta 2.x in 3.x (v nadaljevanju tudi Python2 in Python3). Verzija Python3 je novejša, bolj optimizirana, v njej so tudi popravili nekaj pomembnih "napak", ki so bile v Python2. Pomembnejše izboljšave v Python3 so:

- Podpora Unicode znakov. Unicode znaki so lahko vključeni v nizih in tudi v imenih spremenljivk.
- Pravilnejša implementacija nekaterih delov osrednjih jezika - `print` in `exec` nista več stavka (statements), temveč funkciji. Deljenje dveh celih števil vrne racionalno število.
- Optimizacije delovnega spomina - različne funkcije (`range()`, `map()`, `filter()` ...) vrnejo iteracijske objekte, namesto da bi ustvarile celotne sezname.
- Besedi `True` in `False` sta rezervirani in jih programer ne more več po nesreči spreminjati.

Python3 ima veliko prednosti in skoraj nobene slabosti v primerjavi z Python2. Ena od slabosti je, da je za majhna števila malo počasnejši kot Python2, saj ne uporablja tipov `int` ampak tipe `long`, ki zahtevajo več spomina in novejše procesorje.

5.2 Knjižnjice

Python-ova standardna knjižnjica je razmeroma velika, že vsebuje orodja za veliko različnih nalog. Vsebuje že knjižnjice za izdelavo preprostih internetnih aplikacij, grafičnih vmesnikov in tudi knjižnjici za poganjanje testov.

Veliko programov pa ni vključenih v Pythonovo standardno knjižnjico, vendar jih je veliko vključenih v PyPI (Python Package Index), kjer je trenutno (februarja 2017) 99610 različnih paketov. Z različnimi moduli si poenostavimo pisanje programa, saj nam ni treba ponovno implementirati celotnih funkcij, ki bi jih potrevali. Nekateri moduli pa nam omogočijo stvari, ki jih

v "čistem Pythonu" sploh ne moremo napisati, oziroma so napisani deloma v drugem programskem jeziku zato, da se naš program izvaja hitreje.

Pomembnejše knjižnjice za izdelavo programa steganografije slik:

Pillow: Knjžnjica za delo z različnimi formati slik. Povzeta po knjižnjici `PIL` (Python Imaging Library), ki je napisana za Python verzije 2.x, medtem ko je Pillow namenjena za verzije 3.x.

PyCrypto Zbirka varnih hash funkcij in različnih kriptirnih algoritemov (RSA, AES, DES...).

Tkinter Pythonova standardna knjižnjica za izdelavo grafičnega uporabniškega vmesnika (GUI).

6 Izdelava programa

6.1 Priprava okolja

Sam sem pri programiranju uporabljal operacijski sistem Ubuntu, popularno distribucijo Linuxa, ker je po mojem mnenju programiranje na Linuxu veliko lažje in bolj praktično kot na Windowsih. Tudi programski jezik Python je že prednaložen na večini Linux distribucijah, tako verzija 2.x kot 3.x. Program je napisan v Pythonu verzije 3.5.

Najprej naložimo knjižnjici, ki jih potrebujemo:

```
$ pip3 install pillow
$ pip3 install pycrypto
```

Za pisanje kode sem uporabljal odprtokodni program Atom (`https://atom.io/`), za upravljanje verzij pa program Git, v povezavi z shrambo v oblaku Github (`www.github.com`).

6.2 Pridobivanje podatkov

Program od uporabnika pridobi podatke, v katero sliko želi skriti podatke in datoteko s podatki, ki jo želi skriti. Po mojem mnenju je veliko bolje, če lahko v sliko skrijemo katere koli podatke, ne samo besedila. Ko program pridobi ime datoteke, jo odpre kot `bytes` objekt z imenom `secret`.

```
with open(secret_name, 'rb') as f:
    secret = f.read()
```

V tem primeru je bolj "varno" uporabiti stavek `with`, saj nam Python samodejno zapre datoteko, ko program konča delo v tistem scopu. Tako ne pride do nenamerne prekomerne porabe spomina, saj imamo datoteko odprto samo toliko časa kot jo res potrebujemo.

6.3 Pridobivanje gesla

Program uporabnika vpraša za geslo, s katerim želi zakriptirati datoteko. Ker v običajnem načinu terminala lahko vidimo, kaj uporabnik vpisuje v terminal, ampak ne želimo, da se vidi tudi geslo, moramo v brezgrafičnem načinu uporabiti posebno Pythonovo knjižnjico `getpass`.

```
import getpass
geslo = getpass.getpass('Vpišite geslo: ')
```

Ko bo uporabnik vpisoval geslo, se ga ne bo videlo, a bo še vedno spravljeno v spremenljivki geslo. Ker so Pythonove spremenljivke globalno dostopne, je bolje, če gesla ne shranimo v ločeno spremenljivko, saj načeloma lahko nek drug del našega programa dostopa do uporabnikovega gesla. Bolje je če geslo takoj podamo funkciji, ki bo iz gesla naredila `sha256 hash`.

```
key = crypto.create_key(getpass.getpass('Password: '))
```

V grafičnem vmesniku `tkinter` pa objektu `Entry` dodamo lastnost, da namesto gesla, ki ga vpisuje uporabnik prikaže samo nek znak, v tem primeru `*`.

```
self.label1 = tk.Label(self, text="Password:")
self.password = tk.Entry(self, show="*")
```

Ko programu s pritiskom na gumb `Hide` ali `Find` povemo, naj med drugim pridobi geslo, ga moramo potem izbrisati iz polja, da ne bi prišlo do kakšne zlorabe gesla.

```
password = self.password.get()
self.password.delete(0, tk.END)
key = crypto.create_key(password)
del(password)
```

Pythonova funkcija `del()` izbriše določeno spremenljivko. Za stvari kot so gesla v nizih, je ponavadi bolje, če jih izbrišemo takoj ko jih ne potrebujemo več, namesto da čakamo, da to namesto nas naredi Python.

6.4 Kriptiranje podatkov

Glavni del AES kriptiranja bo za nas opravila knjižnjica `PyCrypto`, saj je njihova implementacija kriptiranja zagotovo bolj pravilna od takšne, ki bi jo napisal jaz. Sama koda je popolnoma preprosta, vse se zgodi v funkciji `encrypt(message, key)`, pri čemer je `message` sporočilo, ki ga želomo zakriptirati, `key` pa je `sha256 hash` gesla.

```
def encrypt(message, key):
    AESchiper = AES.new(key)
    return AESchiper.encrypt(message)
```

Paziti moramo le na to, da je dolžina celotnega sporočila, ki ga želimo zakriptirati z AES algoritmom deljiva z 16. Tudi ustvarjanje hasha je preprosto, saj skoraj vse za nas naredi knjižnjica `PyCrypto`.

```
def create_key(string):
    hashed = sha256(bytes(string, 'utf8')).digest()
    return hashed
```

6.5 Skrivanje podatkov v sliko

Knjižnjica `PIL` (Pillow) nam omogoča, da s sliko delamo podobno, kot z dvodimenzionalnim seznamom. Najprej ustvarimo novo sliko, in potem za vsako vrstico in vsak stolpec prekopiramo piksel iz originalne slike `image_data`. Dokler še nismo skrili celotnega sporočila (`if index < len(secret):`), vsakemu kanalu na sliki (`RGBA`) spremenimo zadnja dva bita v vrednost, ki jo potrebujemo.

```
def hide_core(image_data, secret, size):
    ''' Zaradi preglednosti je v tej funkciji izpuščenih več
    vrstic, ki za samo razumevanje koncepta niso pomembne. '''
    new_image = Image.new('RGBA', size)
    new_image_data = new_image.getdata()
```

```
    index = 0
    for y in range(size[1]):
        for x in range(size[0]):
            r, g, b, a = image_data.getpixel((x, y))

            if index < len(secret):
                r &= ~3
                g &= ~3
                b &= ~3
                a &= ~3
                r |= secret[index] & 3
                g |= (secret[index] & 12) >> 2
                b |= (secret[index] & 48) >> 4
                a |= (secret[index] & 192) >> 6
                index += 1

            new_image_data.putpixel((x,y), (r, g, b, a))

    return new_image
```

Zato, ker je pikslov v posamezni sliki veliko, uporabimo bitne operacije, ker so malo hitrejše, kot npr. operacije na celih številih. Cela števila si lahko predstavljamo v binarnem zapisu. Če želimo dobiti določene bite nekega števila (v tem primeru vse bite razen zadnjih dveh) uporabimo bitno operacijo and (`&`) z enicami na tistih mestih, kjer želimo dobiti bite tistega števila in ničlo na tistih mestih, kjer želimo 0 v vsakem primeru. Ker želimo dobiti vse bite razen zadnjih dveh, to pomeni, da želimo izvesti operacijo z številom $11111100_{(2)}$ oziroma negiranim številom $0\ldots011_{(2)} = 3$. Znak za negacijo v Pythonu pa je `~`.

```
r &= ~3
```

Sedaj želimo na zadnji dve mesti, ki sta zagotovo 0 skriti en bajt naših podatkov, kar pomeni, da bomo v vsako "barvo" skrili po dva bita. Če si predstavljamo, da naš bajt izgleda takole: $11001001_{(2)}$ in mi želimo dobiti po dva bita, to izgleda približno tako:

$$11001001_{(2)} \wedge 00000011_{(2)} = 00000001_{(2)}$$
$$11001001_{(2)} \wedge 00001100_{(2)} = 00001000_{(2)}$$

$$11001001_{(2)} \wedge 00110000_{(2)} = 00000000_{(2)}$$
$$11001001_{(2)} \wedge 11000000_{(2)} = 11000000_{(2)}$$

Ker želimo te znake dodati na konec barv, moramo dobljena števila spremeniti še z operacijo `right shift`, v Pythonu `>>`. Ta operacija vse bite v številu premakne za določeno število v desno.

$$11011100 \gg 2 = 00110111$$

Ko naredimo to, samo še z operacijo or (`|`) dodamo bite barvnim kanalom in vse skupaj shranimo v piksel na novi sliki.

```
g |= (secret[index] & 12) >> 2
```

6.6 Iskanje podatkov

Iskanje podatkov v sliki je precej podobno skrivanju podatkov. Prav tako najprej od uporabnika potrebujemo podatke, v kateri sliki želi iskati podatke, kam bo shranil najdene podatke in geslo. Ena od razlik je, da ko iščemo podatke v sliki, najprej ne vemo, koliko jih je, oziroma do katerega piksla so naši podatki. Zato so v slikah, v katerih so bili skriti podatki s tem programom v prvih 4 pikslih podatki o dolžini, datoteke, ki jo želimo skriti. To pomeni, da je najdaljši možen zapis, ki ga lahko skrijemo v sliko:

$$(2^8)^4 = 256^4 = 4294967296 > 4 \cdot 10^9$$

Pri velikosti slik, ki so danes nekaj čez 10 mio pikslov je torej dovolj, da si podatke o dolžini zapomnimo v samo 4 pikslih.

```
def find_core(image_data, size, key):
    ''' Zaradi preglednosti je v tej funkciji izpuščenih več
    vrstic, ki za samo razumevanje koncepta niso pomembne. '''
    secret = bytearray()
    index = 0
    real_index = 0
    num = 4
    finished = False

    for y in range(size[1]):
        for x in range(size[0]):
```

```
            if index > 0 or num > 0:
                r, g, b, a = image_data.getpixel((x, y))

                value = a & 3
                value <<= 2
                value |= b & 3
                value <<= 2
                value |= g & 3
                value <<= 2
                value |= r & 3

                if num > 0:
                    num -= 1
                    index <<= 8
                    index |= value
                else:
                    secret.append(value)
                    index -= 1
            else: break

    secret = crypto.decrypt(bytes(secret), key)
    return secret
```

Potem, ko imamo podatke o dolžini skritega sporočila, samo dodajamo najdene podatke na seznam `secret`. Ko pridobimo vse podatke, pa moramo seznam še dekriptirati in dobimo sporočilo. Ker sporočilo ni nujno samo tekstovno, ga namesto, da bi ga izpisali na zaslonu, shranimo v datoteko, ki jo je poimenoval uporabnik. Zato lahko na ta način v eno sliko lahko skrijemo ne le preprostega besedila, ampak tudi druge dokumente ali celo druge slikovne datoteke.

6.7 Primer uporabe programa

Leva fotografija je samo fotografija čebele na cvetu, velika 400x300 slikovnih točk. Desna slika ji je na prvi pogled enaka, vendar je v njej skrito 10000 besed vzorčnega teksta (*Lorem ipsum dolor sit amet...*), zakriptiranih z geslom "`geslo`". Velikost datoteke, ki smo jo skrili je bila 68,1 kB. Prva slika je velika 254,5kB, druga pa 273,3 kB. Sliki se razlikujeta v velikosti, saj zaradi skritih podatkov v najnižjih bitih stiskanje podatkov ni bilo več tako zelo učinkovito. Razlika v velikosti v tem primeru ni bistvena, saj je prva slika že dovolj "nepravilna", da že prej ni bilo mogoče slike popolnoma stisniti. Če bi podatke skrivali v sliko, ki bi bila enobarvna, potem bi se velikost spremenila še bolj, poleg tega pa bi bilo s prostim očesom možno opaziti, da je v sliki nekaj skrito (slika ne bi bila več enbarvna, temveč bi bili sosednji piksli med seboj rahlo različnih barv).

Program lahko uporabljamo na dva različna načina: v grafičnem načinu, ter v načinu ukazne vrstice.

6.7.1 Z ukazno vrstico

Recimo da imamo datoteko `slika.png`, v katero želimo skriti naše podatke, ki jih imamo spravljene v datoteki `skrivnost.txt`. Ime nove slike, v kateri bodo skriti podatki po `slika2.png`. Program iz ukazne vrstice pokličemo tako:

```
$ cd src/
$ python3 steganography.py hide slika.png slika2.png skrivnost.txt
```

Program nas vpraša za geslo s katerim želimo kriptirati naše podatke, potem pa začne s skrivanjem podatkov v sliko. Ker je slika lahko velika in lahko skrivanje traja nekaj 10 sekund, program sproti prikazuje kako daleč je prišel.

Če potem želimo najti podatke v sliki z imenom `slika.png` in jih shraniti v datoteko `nova_skrivnost.txt`

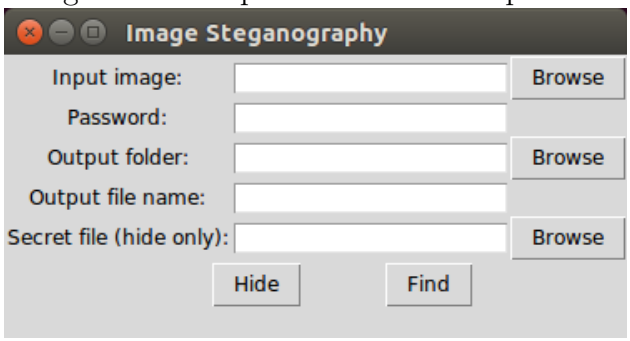
```
$ python3 steganography.py find slika.png nova_skrivnost.txt
```

6.7.2 Z grafičnim vmesnikom

Prav tako je vse mogoče narediti s preprostim grafičnim vmesnikom. Da ga zaženemo, v mapi `src/` izvedemo naslednji ukaz:

```
$ python3 GUI.py
```

Izberemo sliko, v katero želimo skriti podatke, vpišemo geslo za kriptiranje, ime nove slike in datoteko, ki jo želimo skriti. Prav tako lahko s pomočjo grafičnega vmesnika pridobivamo skrite podatke iz slik.



7 Zaključek

Menim, da program zadovoljivo deluje, ter da je steganografija implementirana pravilno, tako da ni očitno opazno, da so v sliki skriti podatki. Ker sem programu dodal tudi grafični uporabniški vmesnik, menim da je dovolj preprost za uporabo za večino ljudi. Programu bi lahko dodal še več možnosti, kot je npr. skrivanje podatkov samo v najnižji bit za še večjo varnost. Poleg tega je še veliko prostora za optimizacijo programa, ampak če bi želel, da program deluje hitreje, bi ga verjetno prepisal v nek drug programski jezik (verjetno C++).

Celotna koda je dostopna na `https://github.com/matejm/steganography`

Literatura

[1] Andrej Brodnik, Luka Fürst, Alenka Krapež in sod. *Računalništvo in Informatika 1*. 2015. URL: lusy.fri.uni-lj.si/ucbenik/.

[2] Wikipedija The Free Encyclopedia. *Advanced Encryption Standard*. URL: en.wikipedia.org/wiki/Advanced_Encryption_Standard (pridobljeno 2017).

[3] Wikipedija The Free Encyclopedia. *Steganography*. URL: en.wikipedia.org/wiki/Steganography (pridobljeno 2017).

[4] Marko Hölbl. "Skrivanje podatkov - steganografija". V: *Monitor* (2008). URL: www.monitor.si/clanek/skrivanje-podatkov-steganografija.

[5] Mojca Kumerdej. "Doma sem nikjer in hkrati povsod. Moja identiteta je steganografska". V: *DELO* (feb. 2017).