Математичка гимназија

Матурски рад
из предмета Програмирање и програмски језици

Стеганографија

Ученик
Вук Долијановић, IVд

Ментор
Станка Матковић

Београд, 24. мај 2023.

Садржај

1

Увод

Од када се писана реч усталила као средство комуникације, настала је потреба за очувањем тајности писаних порука. Поруке су се најпре шифровале у ратним околностима, нпр. Јулије Цезар је шифровао своје поруке померавши свако слово за неколико места у абецеди. Из тих првобитних система шифровања развила се Криптографија, наука која користи разне математичке методе како би направила системе који онемогућавају трећем лицу које чита поруку да разуме њен садржај.

Међутим, размотримо сада проблем две особе, Алисе и Боба које желе да међусобно безбедно комуницирају. Уколико би њих двоје комуницирало енкриптованим порукама, трећа особа која надзире њихов канал комуникације врло лако би могла да примети да је неки криптографски алгоритам коришћен, чиме би Алиса и Боб привукли пажњу на себе и потенцијално били означени као сумњиви по неком критеријуму. Такође, сам акт слања енкриптованих порука може довести до легалних проблема у неким државама.

Као решење за те проблеме, развила се засебна наука : Стеганографија. Стеганографија се бави уграђивањем информација у неку поруку, тако да се уопште не може приметити постојање сакривених информација. Дакле, информација се прво сакрије у неки медијум, који може бити слика, текст, аудио фајл .... Особа која шаље поруку се претвара да жели само да пошаље медијум, док је особа која прима поруку свесна стеганографског алгоритма којим је тајна порука уграђена и познаје алгоритам којим ће извући поруку из медијума. Као медијуми су се врло добро показали дигитални фајлови, јер је због њихове велике величине јако тешко приметити мале промене у њима.

У овом раду, изложићемо стеганографске алгоритме за уграђивање порука у неколико дигиталних формата фајлова :PNG слике, Unicode текст и JPEG

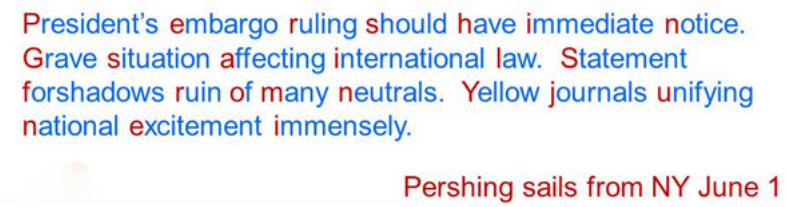
слике, као и методе стеганализе које служе за утврђивање постојања сакривене поруке. Такође, описаћемо имплементацију стеганографских алгоритама који раде на PNG и Unicode фајловима у програмском језику Python, чији се изворни код може наћи на следећем линку:
https://github.com/dolijan/steganografija.

2

Историјат и примене

Прву примену Стеганографије забележио је Херодот 440. године пре нове ере у свом делу „Херодотова историја", где наводи причу о владару Милета који је ошишао свог слугу и „урезао " му поруку на главу, а затим га послао на пут, у току кога му је коса порасла и открила поруку. Реч *стеганографија* први пут је употребио Јохан Тритемијус, отац модерне криптографије и стеганографије, у свом делу „Steganographia " почетком 17.века, где је, између осталог развио начин којим се произвољна порука убацује у молитву *Ave maria*.

До почетка дигиталног доба, примене стеганографије биле су ограничене на писање невидљивим мастилом и сакривање порука у сваком n-том слову реченице. На пример, на слици испод можемо видети поруку која је послата из немачке амбасаде у Првом светском рату. Иако на први поглед изгледа неважно, када се узме прво слово сваке речи добија се порука о њиховим плановима.

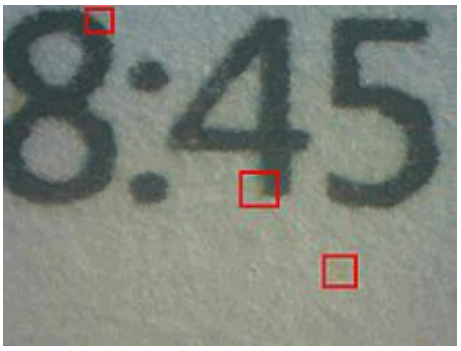


Слика 2.1. Порука немачке амбасаде из првог светског рата

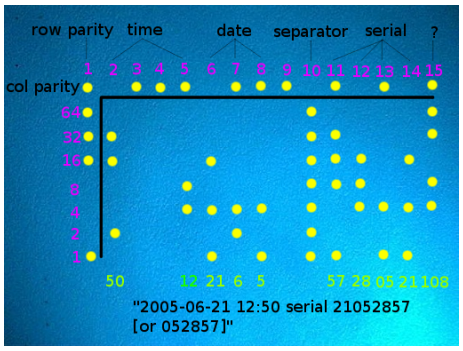
Појава напреднијих рачунара омогућила је нове примене стеганографије. За разлику од писаног текста, дигитални фајлови садрже много више информација у себи, те самим тим остављају велику количину простора за уграђивање порука. Као што ћемо касније видети, променом неких битова у слици можемо добити слику која се не може никако голим оком разликовати

од оригиналне, а у коју можемо уградити било какав фајл. Такође, могуће је уграђивати информације у текст, применом невидљивих карактера, у звук, малим променама одређених фреквенција, као и у физичке медијуме, нпр. папир одштампан специфичним штампачом.

Многи произвођачи штампача применили су стеганографију у сарадњи са њиховим матичним владама. Средином осамдесетих година прошлог века, амерички произвођач Xerox развио је алгоритам који штампа врло мале жуте тачке на папиру, видљиве само јако прецизним скенерима, које специфичним алгоритмом јединствено одређују машину којом је штампање извршено, као и датум штампања. На тај начин осигурано је спречавање штампања лажног новца и могућност да се форензички одреди порекло докумената. Јавност је о томе обавештена тек 2004. године, а 2005. године је група ентузијаста декодирала алгоритме већине великих произвођача (Слика 5.4б).



(а) Жуте тачке који производи штампач *HP Color LaserJet CP1515n*



(б) Алгоритам којим се енкодирају информације жутим тачкама

Слика 2.2

У последње време актуелан је такозвани *watermarking*, где се у фајл, најчешће слику, уграђује код који означава власништво над тим фајлом, како би се спречила повреда ауторских права. Такође се у циљу спречавања пиратерије, у сваку копију филма у почетним стадијумима продукције стеганографски уграђује другачија порука, како би се у евентуалном случају да филм процури знало који члан тима је за то одговоран.

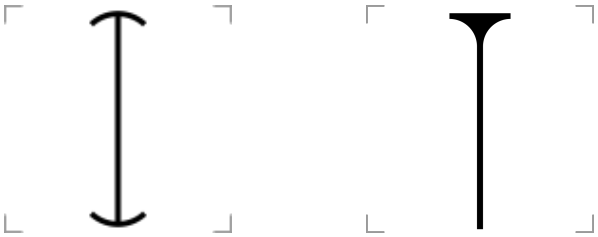
Нажалост, и хакери су нашли примену стеганографије - *stegomalware* је малициозан код уграђен у неки наизглед безазлен фајл како би прошао детекцију антивируса, да би касније био активиран од стране неког екстерног кода.

3

Једноставна текстуална стеганографија

За почетак, размотримо имплементацију једноставног алгоритма који у *Unicode* текст уграђује текстуалну поруку. Ову методу открио је ирачки научник 2009. године [4], и иако је са техничке стране јако једноставна, послужиће нам као добар увод у стеганографске алгоритме.

Прво, приметимо да *Unicode* стандард дефинише два карактера који примарно служе за спајање слова у арапском и још неким језицима : *Zero width joiner* и *Zero width non joiner*, који су означени *Unicode* кодовима *U+200D* и *U+200C*, респективно. Као што им само име каже, ови карактери имају ширину нула, то јест уколико се нађу у сред неког текста, биће невидљиви. Ту чињеницу ћемо искористити како би уградили нашу невидљиву поруку.



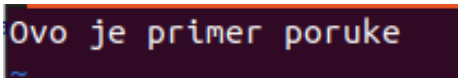
(а) *ISO* симбол за *ZWJ*

(б) *ISO* симбол за *ZWNJ*

Слика 3.1

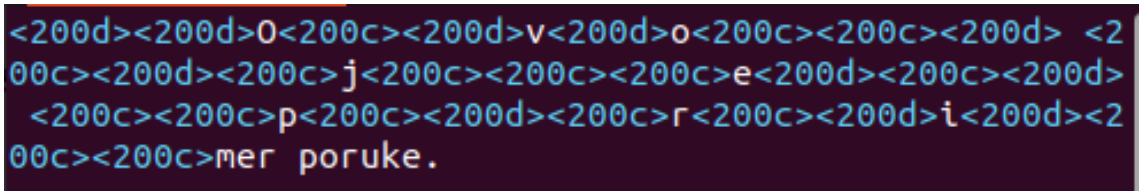
Како бисмо послали поруку, прво ћемо претворити свако слово у Морзеов код, који ће нам од сваког карактера дати низ нула и јединица максималне дужине четири. Када сам имплементирао пратећи програм уз овај рад, разма-

трао сам могућност да свако слово претворим у његову ASCII репрезентацију у бинарном формату, међутим то би захтевало пуно меморије, те сам од те идеје одустао, одлучивши да смањим опсег поруке на слова енглеске абецеде, како бих смањио могућност детекције поруке (велика порука би се много више истицала у тексту). Затим, сваку нулу заменићемо са *zero width joiner*-ом, а јединицу са *zero width non joiner*-ом. Након тога ћемо слово по слово додавати у оригинални *Unicode* текст. Прилично је јасно да је овај процес лако реверзибилан и да врло лако можемо екстрактовати поруку из текста у коме се она налази. Детаљи имплементације могу се пронаћи у коду који прати овај рад.



(а) Порука у коју убацујемо текст

(б) Текст који убацујемо



(ц) Порука са убаченим текстом, невидљиви карактери приказани

Слика 3.2

Иако се карактери са ширином нула не приказују у већини апликација са широком употребом (email, *instant messaging* апликације...), њих врло лако може детектовати специјализовани софтвер. Уз то, уколико се порука шаље као фајл, његова величина (карактери са ширином нула су величине два бајта) може посматрачу са истанчаним оком указати на постојање сакривене поруке. Самим тим, потребно је наћи бољи медијум, што ћемо и урадити у следећем поглављу.

4

Стеганографија у PNG сликама

Сада ћемо размотрити начин на који стеганографски можемо уградити поруку у PNG слику. Како су PNG слике генерално јако велики фајлови, нећемо се ограничавати само на текстуалне поруке, већ ћемо уграђивати произвољне фајлове. При томе ћемо користити технику такозване *least significant bit*, то јест LSB стеганографије [5], која се заснива на уграђивање поруке у најмање значајни бит сваког бајта фајла. Алгоритам који описујемо такође ради и за BMP и GIF фајлове, који служе са складиштење слика и у принципу нису никако фундаментално другачији од PNG фајлова. На крају ћемо видети и како се овакав тип стеганографије детектује.

4.1 LSB стеганографија

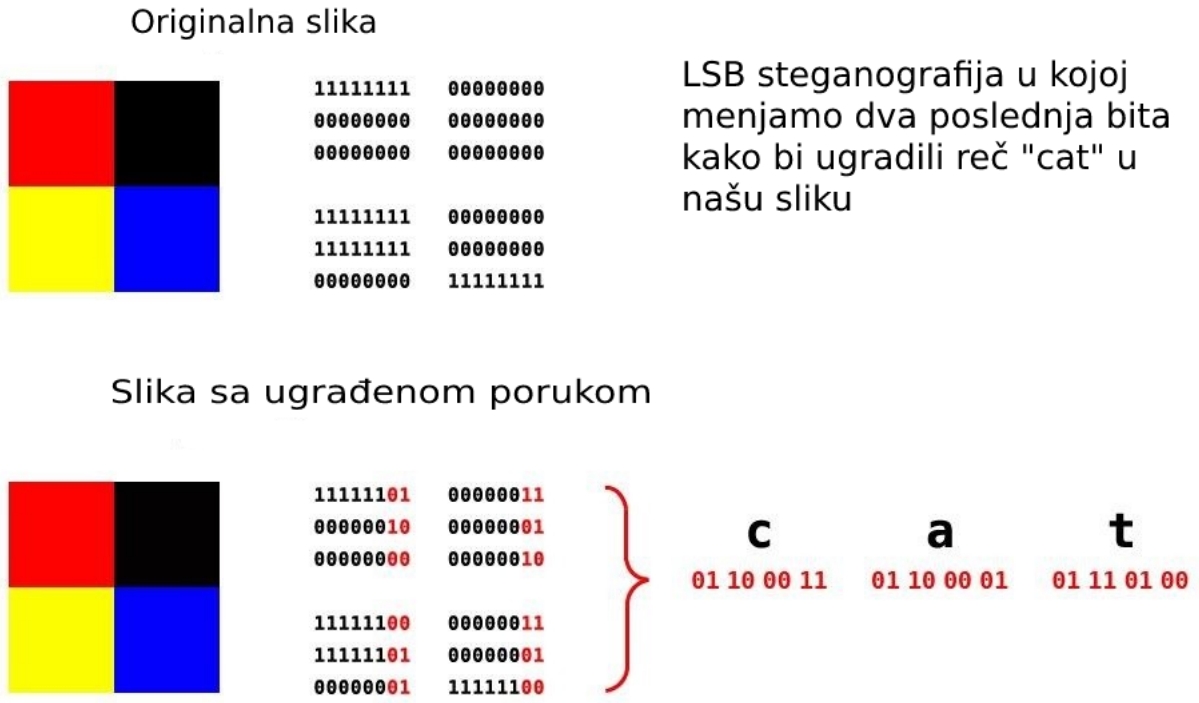
Својство PNG слика које нам дозвољава да примењујемо LSB стеганографију на њих је непостојање компресије (каже се да је PNG *lossless* формат података). PNG формат функционише тако што за сваки пиксел слике чува вредност црвене, зелене и плаве боје (њих зовемо RGB вредности), као један број из интервала $[0, 255]$. Такође, PNG слике подржавају и опционо чување четвртог броја за сваки пиксел, који се зове *алфа канал*, који означава прозирност сваког пиксела (вредност 0 означава тоталну прозирност, а вредност 255 означава потпуну непрозирност, то јест потпуно видимо боју о којој је реч). Алфа вредност најчешће се користи када имамо композитну слику која се састоји од две слике, како бисмо израчунали која од оригиналне две слике треба да се види у којој мери. Када користимо алфа канал у PNG слици, онда уместо RGB вредности говоримо о RGBA вредностима.

Разматрајмо од сада па надаље, без умањења општости, слику која се састоји само од RGB вредности. Наш алгоритам лако ће се генерализовати на

слике са RGBA вредностима, које се обрађују у коду који прати овај рад (тај код заправо ради са оба типа записа PNG слика, уколико наиђе на RGB слику, он дода алфа канал при чему је вредност алфа канала сваког пиксела 255).

Као што смо већ видели, свака од вредности интензитета црвене, зелене и плаве боје налази се у опсегу $[0, 255]$, то јест представљена је са осам битова. Уколико би се свака од тих вредности променила за један, наше очи биолошки не би могле да примете никакву разлику у односу на оригиналну слику. Дакле, од осам битова за сваку од три боје, последњи, најмање значајан бит, можемо мењати произвољно - управо у најмање значајном биту сваке од RGB вредности за сваки пиксел можемо сакрити нашу поруку.

Наравно, можемо мењати и два бита са најмањом вредношћу у свакој од RGB компоненти слике. Ни таква промена неће се видети голим оком, а уштедеће нам на простору, јер можемо уградити дупло већу поруку у исти број пиксела.



Слика 4.1. Пример уграђивања поруке у RGB вредности слике

Дакле, да бисмо имплементирали овај алгоритам, прво претворимо нашу поруку у низ битова које ћемо уграђивати један по један. Затим, итерирамо

по пикселима и последњи бит сваког пиксела променимо у бит који је на тренутном месту у поруци. У наставку видећемо релевантан Python код који убацује поруку у PNG слику са RGBA вредностима.

```
#input_tuple je torka RGBA vrednosti, ova funkcija uzima
    poruku
#i ugradjuje odgovarajuca 4 bita nje u RGBA vrednosti
    promenljive input_tuple
def embed(input_tuple,message,k):
    output_tuple=list(input_tuple)
    for i in range(4):
        if input_tuple[i]%2!=ord(message[k+i])-ord('0'):
            if ord(message[k+i])-ord('0')==0:
                output_tuple[i]=input_tuple[i]-1
            else:
                output_tuple[i]=input_tuple[i]+1

    return tuple(output_tuple)
```

```
def embed_message(pixels,message,n,m,mode,size):
    #pre ovoga imamo nerelevatan kod u nasem projektu

    k=0
    #ugradjujemo poruku bit po bit
    #koristimo RGBA, pa imamo 4 mesta u svakom pikselu
    for i in range(n):
        for j in range(m):
            if len(message)<=k:
                return encoded_picture
            encoded_pixels[i,j]=embed(pixels[i,j],message,k)
            k+=4

    print(type(encoded_picture))
    return encoded_picture
```

Интересантно је да се не морамо ограничити само на текстуалне поруке - ако боље погледамо наш алгоритам, видећемо да је једино што уграђујемо јесте низ битова. Сваки тип дигиталног фајла је у суштини ништа друго него низ битова, а програмски језик Python даје нам могућност да врло лако добијемо све битове неког фајла , које затим можемо да уградимо у нашу сли-

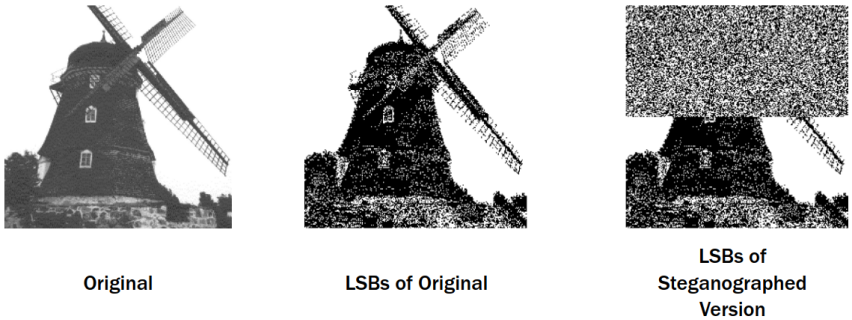
ку, што сам ја у имплементацији пратећег програма и урадио. Као резултат те опсервације, имамо могућност да у PNG слику уградимо било који фајл: другу PNG слику, текстуални фајл, програм, аудио фајл итд. С обзиром да је врло лако екстрактовати фајл уграђен на овај начин (тај поступак није потребно детаљније објашњавати), видимо да LSB стеганографија има велике могућности које су јако често коришћене у пракси.

Треба напоменути да овај стеганографски метод ради за све такозване *bitmap* фајлове, фајлове који чувају слике без компресије, чувајући одређене вредности за сваки пиксел. Поред PNG формата, значајни bitmap формати су GIF и BMP.

4.2 Стеганализа

Фајлови уграђени на претходно описани начин узрокују промене у слици које нису видљиве голим оком. Природно се намеће питање: Да ли је могуће на било којии начин детектовати да је слика промењена? Одговор на то питање је потврдан.

Испоставља се да последњих неколико битова у свакој од RGB вредности нису потпуно независни један од другог. Уколико изолујемо само последње битове слике и прикажемо их, у непромењеној слици (без поруке) требало би да добијемо слику на којој се виде „обриси" оригиналне слике. Наравно, уколико је слика промењена неким стеганографским алгоритмом, уместо „обриса" оригиналне слике добићемо насумичну шару. Ово је основа за такозвани *visual attack*, најчешћи начин којим се примећује присуство стеганографије у слици.



Слика 4.2. Видљиве промене узроковане променом последња 2 LSB бита

Уколико смо променили само један LSB бит и не можемо да визуелно приметимо никакву разлику, просечна вредност LSB битова по целој слици може нам бити индикатор да је она стеганографски промењена - у слици са стеганографском поруком њихова просечна вредност биће око 0.5 (сваки бит биће ефективно насумичан), што није случај у класичним сликама где има више међузависности међу битовима.

Поред тога, PNG слике су врло некомпактни фајлови и у пракси се користе врло ретко, најчешће само за слике екрана, те на интернету њихово слање само по себи може бити сумњиво. На пример, слика која изгледа као да је сликана фотоапаратом или камером са телефона скоро па никада неће бити у PNG формату, те би њено слање у PNG формату могло да наведе особу која посматра канал комуникације да посумња да су коришћене стеганографске методе.

Дакле, иако је LSB историјски први тип стеганографије у дигиталним фајловима и уз то далеко најпознатији, немогућност примене на формате који компресују податке, који су у данашње време домиантни, и рањивост на *visual attack* нападе наводе нас да пронађемо бољи начин да сакријемо наше поруке. То ћемо и урадити у следећем поглављу.

5

Стеганографија у JPEG сликама

JPEG, срађеница за Joint Photographic Experts Group, је стандард којим се дефинишу процедуре за компресију дигиталних слика [7]. Због могућности да смањи величину слике и до 10 пута без видљивих промена у њој , JPEG је врло брзо после свог настанка 1993. године постао најкоришћенији стандард за компресију слика на свету, као и најкоришћенији формат за чување истих на интернету. Због његове широке распрострањености, врло је атрактивно направити алгоритам који уграђује стеганографску поруку у JPEG слику.

JPEG је *lossy* формат, што значи да користи компресију и да не можемо једноставно мењати вредности одређених пиксела у слици јер, за разлику од bitmap фајл формата, вредност неког пиксела зависи од вредности његових суседа. Испоставља се да је ипак могуће направити алгоритам који уграђује поруку у JPEG формат, међутим пре тога прво морамо разумети како он функционише.

5.1 JPEG стандард

JPEG стандард је комплексан алгоритам од више корака, који се у суштини заснива на томе да људи не могу да примете велике промене у фреквецији боја на малом простору и још неким особинама нашег визуелног система.

5.1.1 Трансформација простора боја

Слику из RGB формата треба претворити у YC_BC_R систем боја. Y компонента наива се *лума компонента* и представља колика је светлина (осветљеност) сваког пиксела, док C_B и C_R компоненте представљају удео плаве и

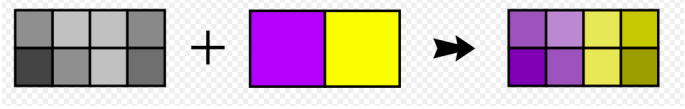
црвене боје у сваком пикселу. Овај систем боја еквивалентан је са RGB системом у смислу да се не губе никакве информације. Предност YC_BC_R система је да смо информацију о осветљености пиксела, која је људима биолошки најбитнија за перцепцију слике, свели на један канал. На тај начин знатно олакшавамо компресију.

Слика 5.1. Слика и њене YC_BC_R компоненте

5.1.2 Смањено узорковање

Због начина на које је људско око грађено, људи могу да виде много више детаља у осветљености слике (Y компонента) него у боји (C_Bи C_R компоненте). Због тога, можемо на нашој слици извршити смањено узорковање (енглески *downsampling*), где ћемо смањити у C_Bи C_R компонентама број пиксела 2 пута и у хоризонталном и у вертикалном правцу. Дакле, од 4 пискела у C_Bи C_R компонентама добићемо један пиксел, који има вредност њиховог просека, чиме смањујемо величину две од три компоненте четвороструко - без приметне разлике у квалитету. Од сада па надаље, сваку од наше три компоненте компресоваћемо независно једну од друге.

(а) Слика без смањеног узорковања



(б) При смањеном узорковању, Y компонента остаје непромењена, док од четири пиксела C_Bи C_R компоненти добијамо по један

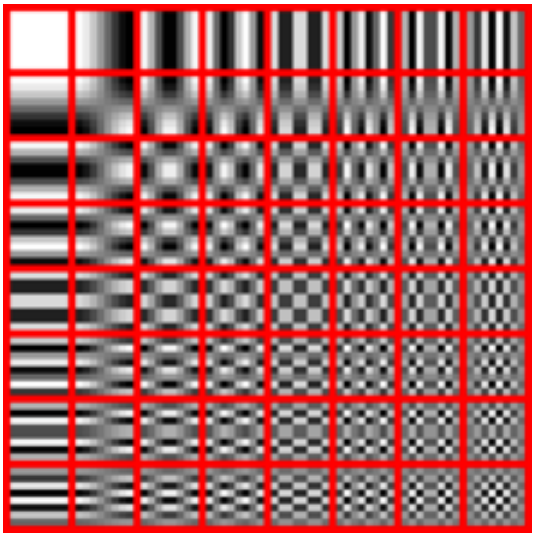
Слика 5.2

5.1.3 Дискретна косинусна трансформација

За почетак, поделићемо сваки од наша три канала на 8×8 блокове (уколико ширина или дужина нису дељиви са 8, додаћемо неке насумичне податке како би постигли дељивост са 8 у обе димензије).Знамо да се наше вредности у сваком 8×8 блоку налазе у опсегу $[0, 255]$. За дискретну косинусну трансформацију требају нам вредности у опсегу $[-128, 127]$, па ћемо од сваке вредности одузети 128, као што можемо видети на примеру испод.

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} \xrightarrow{-128} \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

Желимо да наш 8×8 блок представимо као линеарну комбинацију неких основних 8×8 блокова. Уведимо појам *DCT базне функције* (DCT долази од појма Discrete Cosine Transformation). DCT базна функција је 8×8 матрица, у којој је свако поље 8×8 блок пиксела. Први ред и прва колона представљају косинусни талас у хоризонталном и вертикалном правцу, респективно, а остале колоне добијамо композицијом косинусних таласа са различитим фреквенцијама.



Слика 5.3. Желимо да нашу слику представимо као линеарну комбинацију ових 64 основних блокова

Сада нам је циљ да направимо DCT матрицу, која ће нам дати коефицијенте којима треба помножити поља DCT базне функције како би добили оригинални блок. Ако са B обележимо базну функцију, са G обележимо DCT матрицу, а са A обележимо оригинални блок, онда треба да важи:

$$A = \sum_{i=0}^{7} \sum_{j=0}^{7} B_{i,j} * G_{i,j}$$

DCT матрицу рачунамо на следећи начин:

$$G_{i,j} = \frac{1}{4}\alpha(i)\alpha(j) \sum_{x=0}^{7} \sum_{y=0}^{7} A_{x,y} \cos\left[\frac{(2x+1)i\pi}{16}\right] \cos\left[\frac{(2y+1)j\pi}{16}\right]$$

Где је:

- $0 \leq i \leq 7$ хоризонтална фреквенција косинусног таласа
- $0 \leq j \leq 7$ вертикална фреквенција косинусног таласа
- $\alpha(x)$ је нормализациони фактор, како би наша трансформација била ортонормална

Уколико применимо дискретну косинусну трансформацију на матрицу из почетног примера (претходна страна) добијамо :

$$\begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

Приметимо да за сада нисмо изгубили никакве информације, јер је линеарна комбинација DCT базне функције са коефицијентима из претходне матрице апсолутно иста као оригинални блок.

5.1.4 Квантизација

Наше очи нису добре у детектовању компоненти на сликама са великом фреквенцијом, то јест не можемо у потпуности да приметимо разлику између вредности суседних пиксела које драстично варирају. Дакле, у нашој DCT

табели, вредности које одговарају високофреквентним косинусним таласима могу често бити занемарене.

То се постиже коришћењем Квантизационих табела, које дефинишу коефицијенте којима треба поделити вредности у DCT табели. Дакле, уколико је квантизациона табела Q, поља наше нове табеле C дефинисана су са:

$$C_{i,j} = \frac{G_{i,j}}{Q_{i,j}}$$

Пример квантизационе табеле дефинисане у JPEG стандарду за компресију од 50 процената је следећа матрица:

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

Приметимо да су вредности које одговарају високофреквентним таласима много веће. После дељења, наша матрица постаје:

$$C = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Видимо да је већина вредности у нашој матрици постала нула: њих више не морамо памтити, и на тај начин смо постигли јако велику уштеду у меморији. Последњи корак је да вредности матрице C сместимо у меморију (смештаћемо их у низ), где ћемо применити још један алгоритам *lossless* компресије (заснива се на чињеници да не морамо памтити све нуле), међутим то нам тренутно није релевантно за сврхе овог рада.

Слика записана у том формату (Матрица C записана у низ, са још мало компресије), назива се JPEG фајл. Када приказујемо JPEG слику, обрнутим процесом од овога долазимо до матрице пиксела, који приказујемо на екрану.

5.2 Стеганографија у JPEG сликама

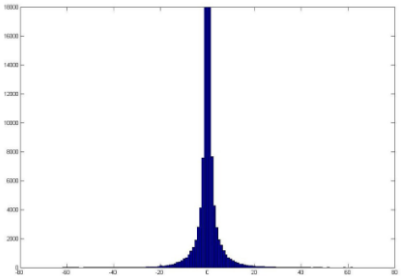
Због компресије која се дешава у JPEG сликама, није било могуће користити методе LSB стеганографије на самим пикселима. Међутим, сада знамо да су JPEG слике ништа друго него квантизиране вредности DCT табеле записане у низ. Дакле, ми можемо да у најмање значајне битове квантизиране DCT табеле уградимо нашу поруку, на идентичан начин као што смо то урадили и у PNG фајловима у претходном поглављу.

Треба имати у виду да се обично порука не уграђује у поља матрице чија је апсолутна вредност мања од 2 , као ни у прво поље у матрици (горње лево), јер је његова вредност велика, те то поље има највећи утицај на изглед коначне слике.

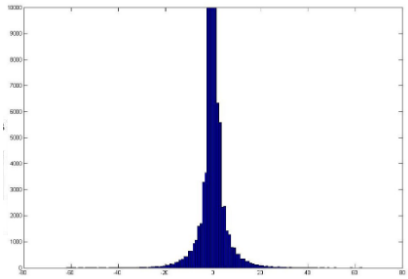
На основу ових обзервација настали су први алгоритми за JPEG стеганографију, од којих су дефинитивно најпознатији *JSTEG* и *F5*.

5.3 Стеганализа

Слике које имају поруку уграђену у квантизоване DCT коефицијенте отпорне су на визуелни напад описан у претходном поглављу. Међутим, испоставља се да ће хистограм вредности свих DCT коефицијената у произвољној слици бити отприлике симетричан у односу на y осу. У слици у коју је уграђена порука то неће бити случај, те из дистрибуције DCT коефицијената можемо закључити да ли је дошло до промене слике.



(а) Дистрибуција DCT коефицијената у оригиналној слици



(б) Дистрибуција DCT коефицијената у слици са поруком

Слика 5.4

Тренутно истраживање у области стеганографије фокусирано је на начине на које се може очувати статистика DCT коефицијената при уграђивању

поруке. На ту тему осмишљено је много алгоритама чији су детаљи изван оквира овог рада.

Међутим, у скорије време проширила се употреба машинског учења и класификационих алгоринтама, који на основу тренинг података (који се у овом случају могу врло лако генерисати), могу са врло великом тачношћу да предвиде да ли је дошло до стеганографске промене у слици или не. Алгоритам који може да заобиђе такав начин стеганализе још увек није осмишљен и на њему се увелико ради.

6

Имплементација у програмском језику Python

За потребе овог рада имплементиран је програм у програмском језику Python, који се налази на репозиторијуму поменутом у уводу. Програм прво детектује тип фајла са којим ради на основу његових *магичних бајтова*, као и тип фајла који желимо да уградимо, а затим позива одговарајућу методу која уграђује поруку/вади уграђену поруку из фајла (Када кажемо порука не мислимо само на текстуалне поруке, већ на било који фајл). Тренутно су као медијум подржани само PNG и Unicode фајлови, међутим изворни код је постављен на Github и свако може да се прикључи даљем развоју овог програма.

Уграђивање поруке се из терминала/конзоле позива на следећи начин:

```
python3 steganografija.py -e fajl1 -m poruka
```

У претходном примеру, „fajl1" је назив PNG или Unicode фајла у који желимо да уградимо поруку, а „poruka " је назив фајла који уграђујемо у оригинални фајл)

Дохватање поруке врши се на следећи начин:

```
python3 steganografija.py -d fajl_sa_porukom
```

За манипулацију слика коришћена је библиотека *Pillow*. За контролисање верзија коришћен је *Github*, а програм је тестиран на Windows и Linux оперативним системима.

7

Закључак

За разлику од криптографије, стеганографија није широко позната, иако су јој пре неколико векова приписивали чак и магијско порекло. Овим радом хтео сам да представим неке од значајних идеја у дигиталној стеганографији. Оне се могу даље генерализовати за примену стеганографије у аудио и видео фајловима. Такође, као што је описано у раду, стеганографија има веома широку примену, и сматрам да ће се у будућности још више примењивати. Разлог томе је све веће ширење тзв. *deepfake* слика и снимака, тј. слика и снимака генерисаних вештачком интелигенцијом. Сматрам да ће се комбинацијом стеганографије и асиметричне криптографије у будућности гарантовати аутентичност дигиталних фајлова.

За крај, желео бих да се захвалим ментору Станки Матковић за све што ме је научила у протекле четири године.

Литература

[1] Wikipedia, *Steganography* `https://en.wikipedia.org/wiki/Steganography` сајту приступљено 24-05-2023.

[2] Wikipedia, *Machine Identification Code* `https://en.wikipedia.org/wiki/Machine_Identification_Code` сајту приступљено 24-05-2023.

[3] Wikipedia, *Stegomalware* `https://en.wikipedia.org/wiki/Stegomalware` сајту приступљено 24-05-2023.

[4] Akbas E. Ali, *A New Text Steganography Method By Using Non-Printing Unicode Characters*, Eng. And Tech. Journal, Vol.28, No.1, Багдад 2010.

[5] C.Kurak, J.McHugh, *A cautionary note on image downgrading*, Proceedings Eighth Annual Computer Security Application Conference, IEEE, 1992

[6] Wikipedia, *JPEG* `https://en.wikipedia.org/wiki/JPEG` сајту приступљено 24-05-2023.

[7] The International Telegraph and Telephone Consultative Committee, *Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines*, Terminal Equipment and Protocols for Telematic Services Recommendation T.81, 1993.