Steganografske metode u obradi slike

Biondić, Enio

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:619334

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij računarstva

STEGANOGRAFSKE METODE U OBRADI SLIKE

Enio Biondić

Osijek, 2019.

Sadržaj

1. UVOD 1
1.1. Zadatak završnog rada 1
2. PRIMJENJENE TEHNOLOGIJE 2
2.1. Visual Studio 2015 2
2.2. C# 3
3. IZRADA APLIKACIJE 4
3.1. Instalacija neophodnog software-a 4
3.2. Stvaranje novog projekta 5
3.3. Kreiranje korisničkog sučelja 6
3.4. Učitavanje fotografija 8
3.5. Učitavanje tekstualne datoteke 9
3.6. Proces kriptiranja 10
3.7. Proces dekriptiranja..... 16
4. ANALIZA KVALITETE KRIPTIRANIH DATOTEKA 18
4.1. Razlika u veličini datoteka 20
4.2. Postotna razlika u promjenjenom sadržaju 21
5. **ZAKLJUČAK** 22

LITERATURA 23
SAŽETAK 24
ABSTRACT 24
ŽIVOTOPIS 25

1. UVOD

Razvojem tehnologije i društvenih mreža prijenos i brzina prijenosa informacija iz dana u dan rastu. Živimo u svijetu kada nam poslati informaciju na drugi kraj planeta u izrazito kratkom vremenu ne predstavlja ništa spektakularno već to smatramo egzistencijalnom potrebom. Usprkos svemu tome, zapitamo li se kako naša informacija stiže od polazišta do odredišta, i stoji li na tom putu netko tko ima pristup našoj informaciji i može li ju izmjeniti ili učiniti dostupnom publici za koju to nikad ne bi htjeli da bude. Vodeći se ovom problematikom ljudi su kroz povijest razvijali razne metode kriptiranja koje bi omogućile da samo primatelj može pristupiti poslanoj informaciji i nitko drugi osim njega. U današnje vrijeme postoji veliki broj algoritama za kriptiranje i dekriptiranje sadržaja poruke koji imaju za cilj spriječavanje neautoriziranog čitanja i/ili izmjene, a neki od njih se zasnivaju na steganografiji. Ovim završnim radom cilj mi je bio pokazati kako možemo primjenom steganografskih postupaka [1] omogućiti skrivanje naših tekstualnih informacija u slikovne datoteke te tako "osjetljive" informacije osigurati od doticaja neželjene publike

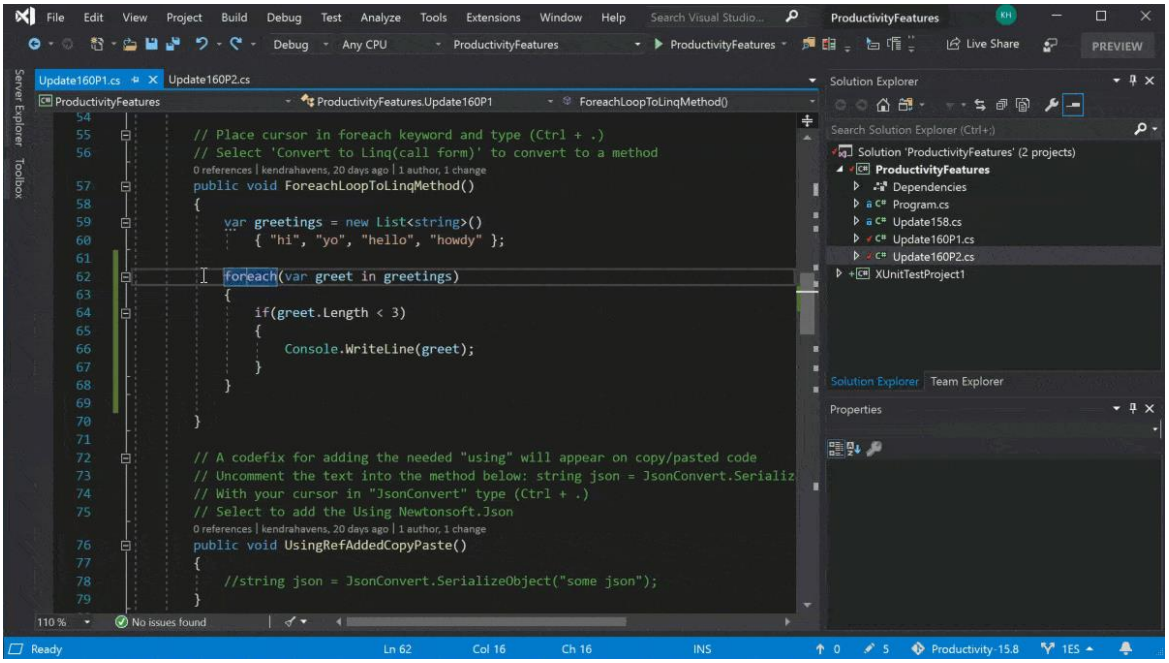
1.1. Zadatak završnog rada

Zadatak završnog rada je izraditi aplikaciju s grafičkim sučeljem primjenom programskog jezika C# koji omogućuje enkripciju i dekripciju grafičkih datoteka sa željenim tekstualnim dokumentom. Kada korisnik želi provesti enkripciju, odabire željenu grafičku datoteku formata PNG te željenu tekstualnu datoteku formata TXT. Pritiskom na odgovarajući button provodi se enkripcija i korisniku se predaje krptirana slika. Dobivena kriptirana grafička datoteka mora naočigled biti jednaka izvornoj fotografiji, ali ipak mora sadržavati kriptiranu poruku koju računalo pomoću aplikacije može ponovno dohvatiti. Odluči li se korisnik na dekripciju omogućuje mu se učitavanje grafičke datoteke za koju korisnik smatra da je kriptirana. Nakon učitavanja slike i pritiskom na button za dekripciju korisnik dobiva tekstualnu datoteku koja sadržava izdvojeni tekst.

2. PRIMJENJENJE TEHNOLOGIJE I ALATI

2.1 Visual Studio 2015

Visual Studio je itegrirano razvojno okruženje (IDE) koje je razvijeno od tvrtke Microsoft, a svoju primjenu prema [2] nalazi u razvoju i izradi računalnih programa za Windows, izradu web stranica, aplikacija i još mnogo toga. Visual Studio sadrži uređivač koda koji ima ugrađen IntelliSense čija je zadaća predviđanje i dovršavanje pisanja naredbi u pojedinim jezicima. Kako bi se programerima olakšao rad u ovom razvojnom okruženju, ugrađen je debuger koji radi na razini izvornog, ali i strojnog koda koji pomaže prilikom otklanjanja grešaka. Također je podržan veliki broj programskih jezika u kojima se može raditi.



Slika 1. *Visual Studio razvojno okruženje*

2.2 C# programski jezik

C# prema [3] jedan je od mlađih programskih jezika. Nastao je 2002. godine kao sastavni dio .NET framework-a. Njegova je prvenstvena namjena bila u potpunosti iskoristiti ono što taj framework pruža. Kao i svaki moderniji programski jezik, C# je objektno orijentiran programski jezik što znači da omogučava rad s objektima i klasama. Zbog svoje jednostavnosti i sličnosti programskim jezicima, kao što su Java, C++ i C, odabrao sam baš ovaj jezik za izradu aplikacije za završni rad. Osim toga, neki od jezika koji su slični C# jeziku obrađivani su na pojedinim kolegijima u prve dvije godine preddiplomskog školovanja te je bilo lako primijeniti naučeno na ovome projektu.

```
using System;

namespace HelloNameSpace
{
  public class HelloWorld
  {
   static void Main()
   {
     Console.WriteLine("Hello World!");
   }
  }
}
```

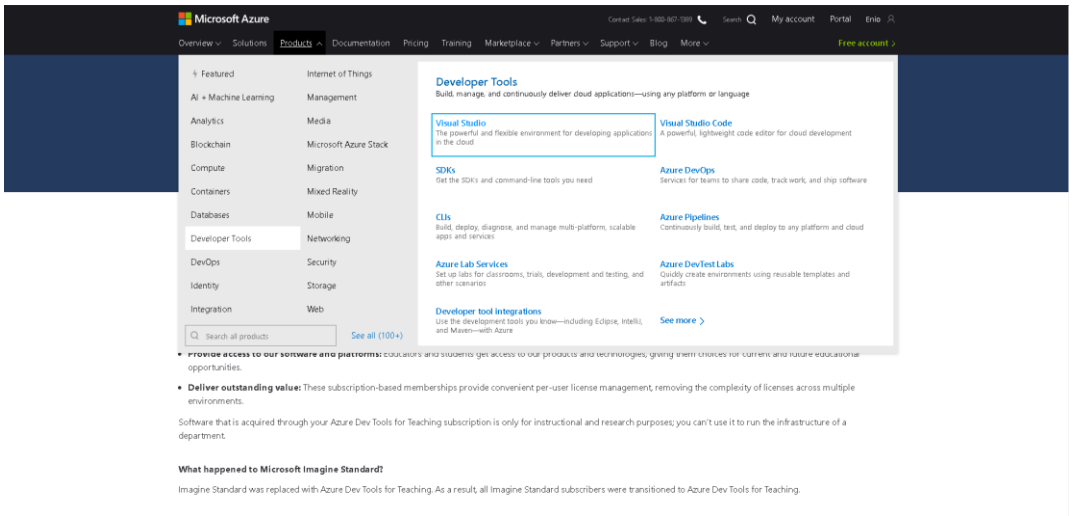
Slika 1.1 *Primjer koda u C# programskom jeziku*

3. IZRADA APLIKACIJE

U ovome ću poglavlju detaljno opisati način na koji sam izradio ovu aplikaciju. Detaljno će biti objašnjene i dokumentirane sve etape izrade, uključujući i izradu korisničkog grafičkog sučelja, ali i pozadnskih procesa čije je odvijanje neophodno za normalno funkcioniranje aplikacije.

3.1. Instalacija neophodnog software-a

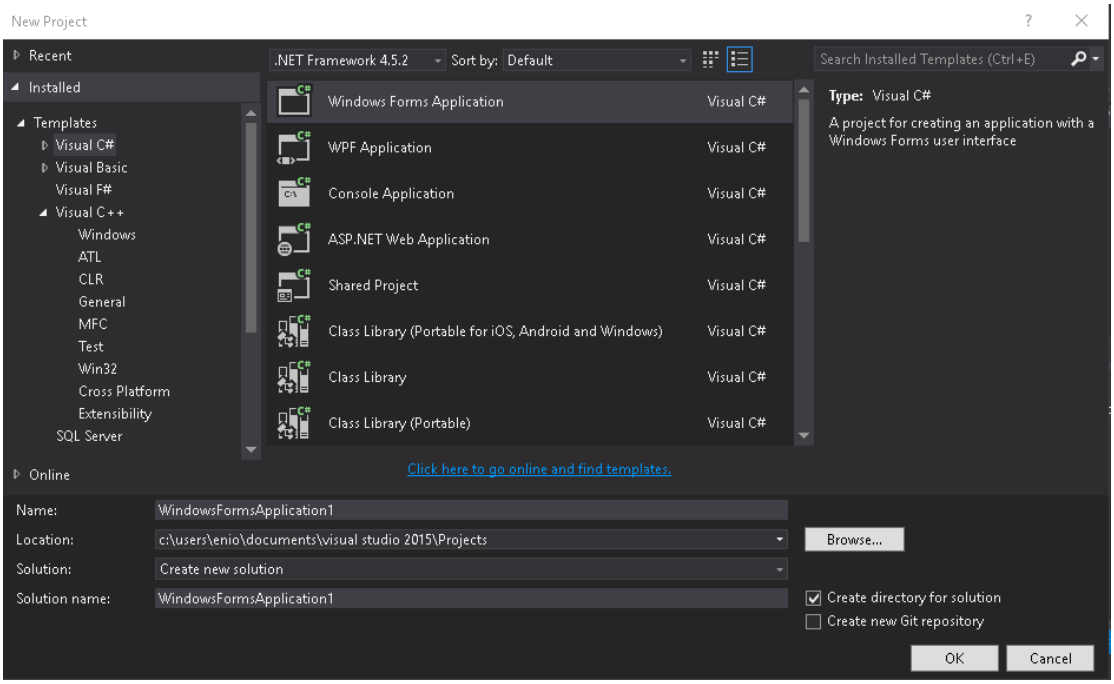
Prije samog početka pisanja aplikacije bilo je potrebno instalirati razvojno okruženje na računalo prema uputama [4]. Osobno sam izabrao rad u Visual Studio 2015 razvojnom okruženju iz razloga što je vrlo jednostavnoga sučelja, ali sadržava napredne alate koji mogu pomoći kod analiziranja rada aplikacije. Za instalaciju ovog software-a potrebno je nabaviti licencu za korištenje, ali za sve studente ta licenca je besplatna te se može kao i sam Visual Studio nabaviti na stranicama Microsoft-ovog azure-a uz prethodnu registraciju AAI identitetom. Nakon što smo pronašli traženi software, pritisnemo button download i slijedimo daljne upute čarobnjaka za instalaciju.



Slika 2. *Microsoft Azure web-site*

3.2. Stvaranje novog C# projekta

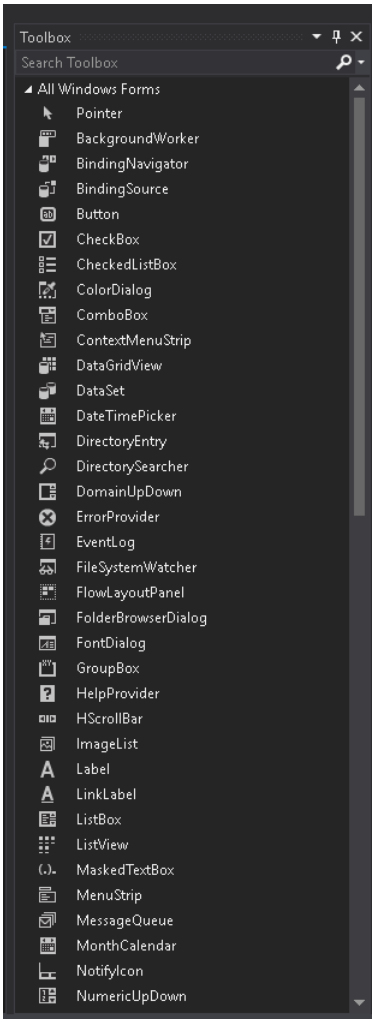
Kada smo završili s instalacijom razvojnog okruženja, potrebno je stvoriti novi C# projekt. Otvaramo Visual studio, zatim odabiremo izbornik "File", a iz njega odabiremo opciju "New" i podkategoriju "Project". Otvara nam se izbornik u kojemu biramo vrstu projekta kojega želimo napravit. Potrebno je odabrati "Visual C#" i podkategoriju "Windows Forms Application", zatim odabrati lokaciju na disku gdje želimo spremiti projekt te samo ime projekta. Ako u ponuđenim opcijama nemamo "Visual C#" kao opciju novog projekta, potrebno je preuzeti programski jezik iz "Online" kategorije koja se nalazi na desnoj strani dolje prikazanog prozora.



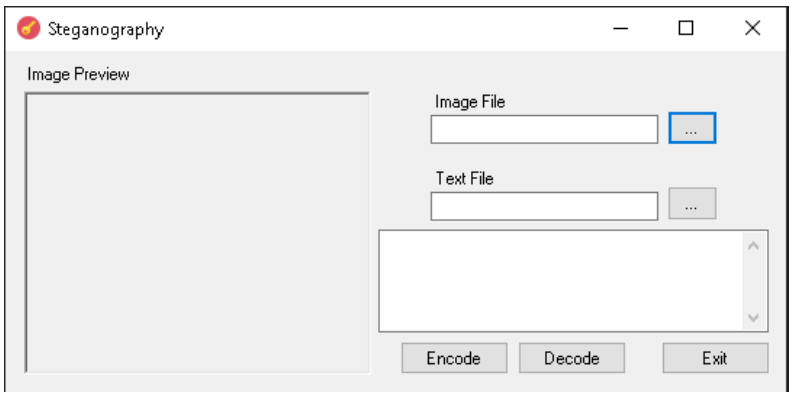
Slika 2.1. *Stvaranje novog C# projekta*

3.3. Kreiranje korisničkog sučelja

Kreiranje korisničkog sučelja u Visual Studio razvojnom okruženju vrlo je jednostavno. Cijeli princip slaganja elemenata sučelja zasniva se na "Drag & Drop" metodi kojom postavljamo elemente sučelja i raspoređujemo kako želimo da nam oni izgledaju u aplikaciji. Kod izrade sučelja za svoju aplikaciju želja mi je bila da ono bude što jednostavnije i intuitivnije kako bi aplikacija izgledala i bila jednostavna za korištenje. Važno je napomenuti kako je za one elemente sučelja koji sudjeluju u logici programa potrebno izabrati prikladno ime, dok se ostali atributi koji opisuju izgled elementa postavljaju i kod svih ostalih. Sve elemente sučelja povlačimo iz "ToolBox-a".

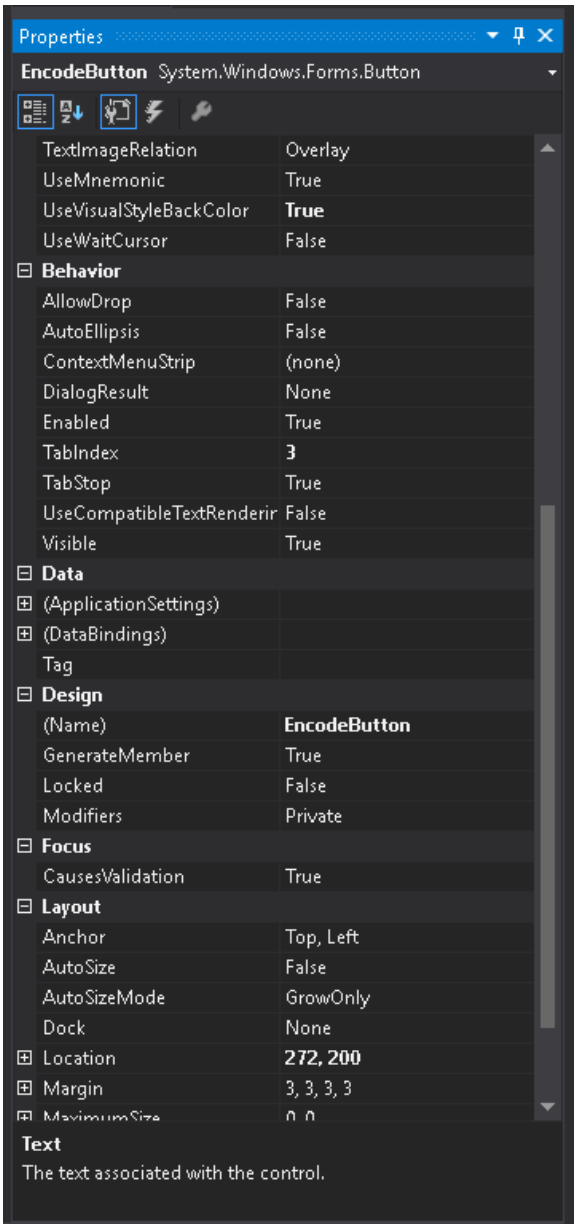


Slika 2.2. *ToolBox*



Slika 2.3. *Korisničko sučelje aplikacije*

Na slici 2.3. možemo vidjeti izgled gotovoga korisničkog sučelja s postavljenim elementima i atributima.



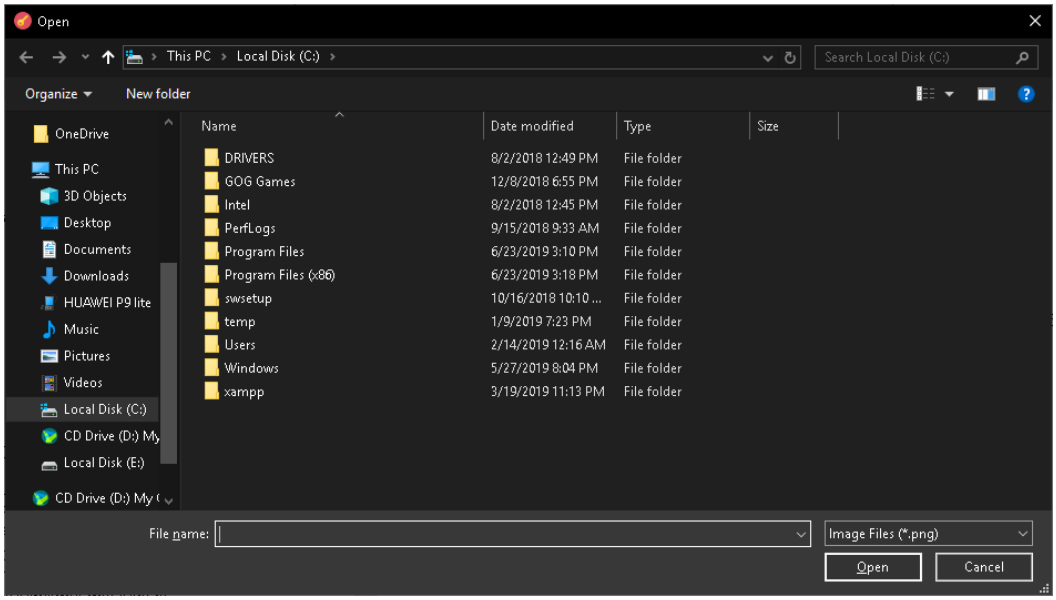
Slika 2.4. *Postavljanje atributa elementu*

Kada postavljamo elemente sučelja metodom “Drag & Drop“, Visual Studio postavlja predefinirane vrijednosti atributa. Ponekad nam vrijednosti tih atributa mogu stvarati probleme u pisanju programskoga koda zbog njihovog imenovanja. Na primjeru buttona “EncodeButton“ vidimo da će nam u kodu biti jasno o kojem se buttonu radi kada koristimo ime koje je deskriptivno. Predefinirane vrijednosti atributa mijenjamo u kartici “Propertis“.

3.4. Učitavanje fotografije

Kada u programu želimo učitati slikovnu datoteku na kojoj je potrebno izvesti kriptiranje ili dekriptiranje, to možemo učiniti tako da odaberemo button za učitavanje fotografija te zatim kada nam se otvori "File Explorer" navigiramo do tražene fotografije.

Slika 2.5. *Button za učitavanje fotografije*



Slika 2.6. *File Explorer za navigaciju do tražene datoteke*

U pozadinskom djelu aplikacije nakon pritiska na button za učitavanje fotografije pokreće se funkcija "OpenImageFileButton_Click", njezin kod prikazan je na slici 2.7.

```
private void OpenImageFileButton_Click(object sender, EventArgs e)
{
    OpenFileDialog Dialog = new OpenFileDialog();
    Dialog.Filter = "Image Files (*.png) | *.png;";
    Dialog.InitialDirectory = @"C:\";
    Dialog.RestoreDirectory = true;
    if (Dialog.ShowDialog() == DialogResult.OK)
    {
        ImageFilePath.Text = Dialog.FileName.ToString();
        pictureBox1.ImageLocation = ImageFilePath.Text;

    }
}
```

Slika 2.7. *Funkcija za učitavanje grafičke datoteke*

U danom kodu vidimo da se funkcija za učitavanje poziva tek kada korisnik pritisne traženi button. Kada se funkcija pokrene, ona započinje otvaranje "File Explorera" na način da inicijalizira novi "OpenFileDialog" te istom postavlja atribute koji specificiraju njegov način rada. Ovdje vidimo da će novootvoreni Dialog prikazivati samo datoteke formata PNG te da će nakon uspješnoga otvaranja datoteke njezinu lokaciju na disku zapisati u "ImageFilePath" u obliku stringa. Ovaj podatak ostaje pohranjen sve dok se aplikacija ne ugasi ili dok se ne učita nova datoteka.

3.5. Učitavanje tekstualne datoteke

Kao i na primjeru učitavanja slikovne datoteke, tekstualna datoteka u aplikaciji učitava se na način da kliknemo na button za učitavanje tekstualne datoteke. Nakon klika otvara se "File Explorer" (slika 2.6.) kojim navigiramo do željene datoteke. Ovaj button koristimo samo kada radimo kriptiranje i biramo tekst koji "*utiskujemo*" u sliku.

Slika 2.8. *Button za učitavanje tekstualne datoteke*

Logika funkcioniranja učitavanja tekstualnih datoteka skriva se iza funkcije "OpenTextFileButton_Click" čiji je kod vidljiv na slici 2.9.

```
private void OpenTextFileButton_Click(object sender, EventArgs e)
{
    OpenFileDialog Dialog = new OpenFileDialog();
    Dialog.Filter = "Txt Files (*.txt) | *.txt";
    Dialog.InitialDirectory = @"C:\";
    Dialog.RestoreDirectory = true;
    if (Dialog.ShowDialog() == DialogResult.OK)
    {
        TextFilePath.Text = Dialog.FileName.ToString();

    }
}
```

Slika 2.9. *Funkcija za učitavanje tekstualne datoteke*

Slično prethodno opisanoj funkciji za učitavanje fotografija i ova funkcija otvara novi "OpenFileDialog" čijim se inicijaliziranjem pokreće "File Explorer". Za razliku od prethodnoga, ovaj Dialog prikazuje samo datoteke TXT formata, a nakon uspješnog učitavanja putanja do tražene datoteke zapisuje se u obliku stringa u "TextFilePath". Isto kao i prethodna, ova funkcija svoje izvršavanje započinje kada korisnik klikne na button za učitavanje tekstualne datoteke (slika 2.8.).

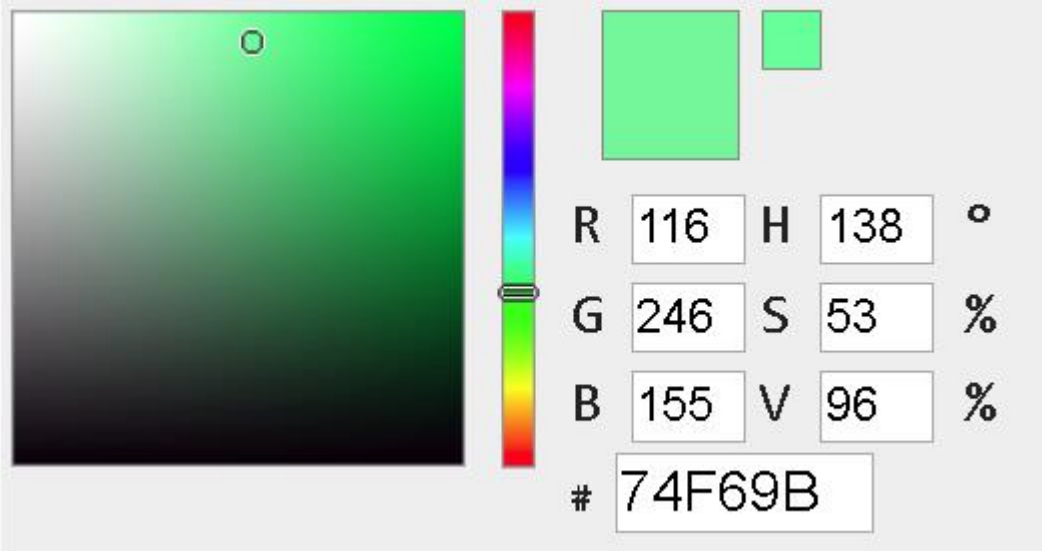
3.6. Proces kriptiranja

Kompletna složenost ove aplikacije zasniva se na dvije funkcije: one za kriptiranje i one za obavljanje reverznog posla, odnosno za dekriptiranje. Te su dvije funkcije dosta složene i zato im želim posvetiti najviše pažnje prilikom njihove deskripcije. Prije nego li krenem u objašnjavanje koda, volio bih napisati nešto više o samim slikovnim datotekama s kojim ova aplikacija i radi.

Svaka slika prema [5] (neovisno kojeg datotečnog formata) ima svoju rezoluciju. Rezolucija se definira količinom pixela koji su sadržani u toj fotografiji i to na način da navedemo koliko pixela ima u jednome retku te slike, a koliko ih je u jednome stupcu. Na primjer, slika čija je rezolucija 1920x1080 sadrži 1080 linija gdje je svaka linija sastavljena od 1920 pixela. Važno je dodatno naglasiti kako se svaki pixel slike sastoji od 3 boje (RGB odnosno red, green, blue). Boje u pixelima definirane su odnosom zastupljenosti pojedine RGB komponente. Svaka komponenta može biti zastupljena u intervalu [0, 255] gdje 0 predstavlja potpuni izostanak te boje, a 255 punu zastupljenost.

Na primjer, ako pixelom želimo prikazati plavu boju, njegove komponente moramo postaviti na sljedeći način: R = 0, G=0, B=255. U ovome slučaju komponente za crvenu i zelenu boju su u potpunosti isključene dok je komponenta za plavu boju u potpunosti zastupljena. Jednostavnom

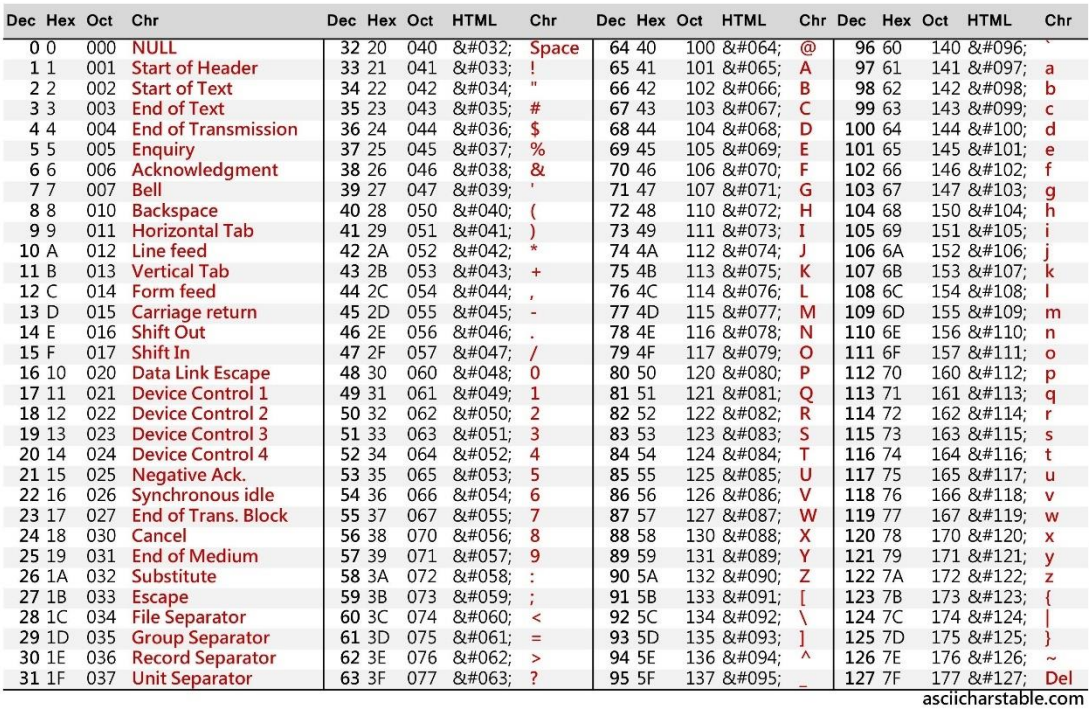
računicom možemo zaključiti da ovim načinom zapisivanja boja možemo ostvariti 255^3 kombinacija, odnosno 16 581 375 nijansi.



Slika 2.10. *Vrijednosti RGB komponenti za danu boju*

Zbog jednostavnosti svog formata i izostanka gubitka informacija, za rad ove aplikacije odabrana je PNG vrsta grafičkih datoteka. Ovaj format omogućuje nam pristupanje svakome pixelu slike i dohvaćanje podataka o zastupljenosti pojedine RGB komponente.

Drugi važan dio kompletnog kriptiranja oslanja se na tekstualne datoteke. Za rad ove aplikacije odabran je TXT format zapisa teksta jer je predviđeno da se u slike "*utiskuje*" samo običan tekstualni dokument. Prednost ovoga formata je što za svaki znak koji se može pojaviti postoji odgovarajući ASCII kod.



Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	Del

asciicharstable.com

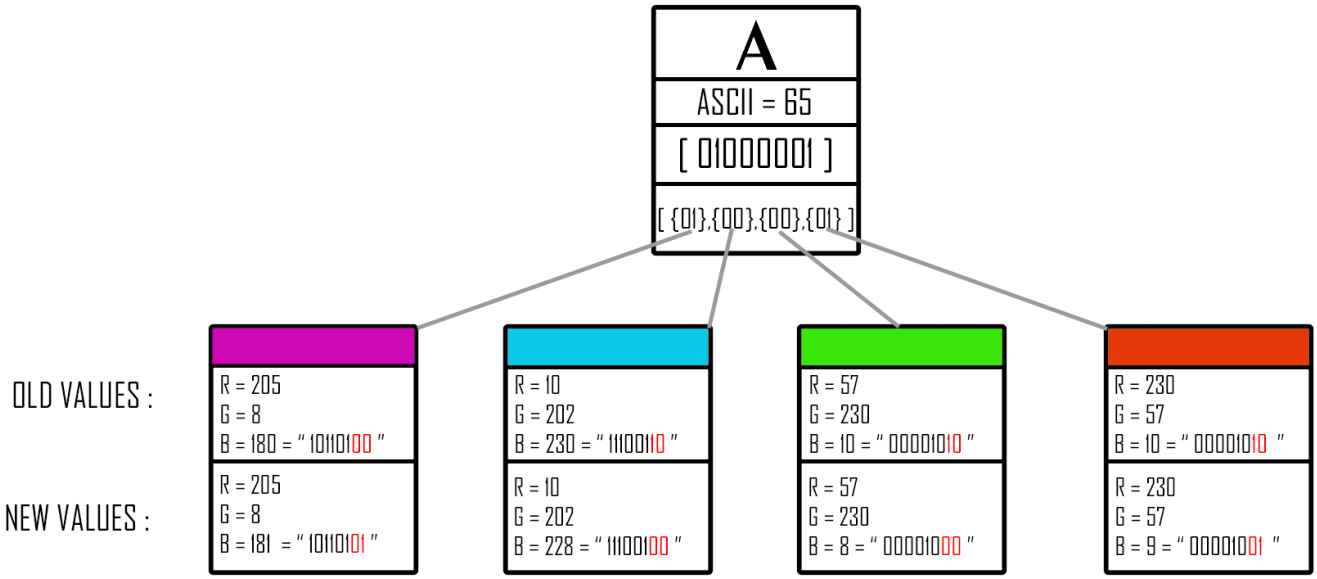
Slika 2.11. *ASCII tablica*

Oslanjajući se na ASCII tablicu (Slika 2.11.) prema [6] lako možemo tekstualni dokument pretvoriti u niz brojeva. Na primjer, želimo li pretvoriti veliko slovo "A" u odgovarajući (ASCII) broj, služeći se tablicom, vidimo da je njegova odgovarajuća dekadska vrijednost 65. Analogno tome, svaki znak koji se može pojaviti u tekstu imat će svoj kod u dekadskom intervalu [0,127].

Čitav princip kodiranja zasniva se na manipulaciji zapisa teksta i pixela. Način kako program "*utiskuje*" tekst u sliku je sljedeći.

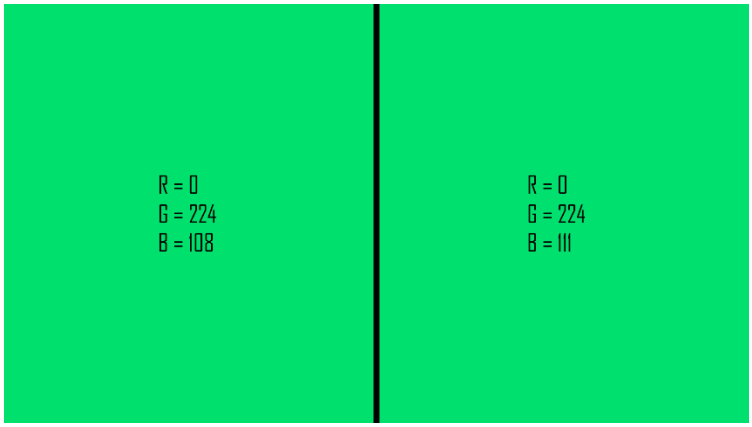
Čitav tekst koji je sadržan u tekstualnoj datoteci pretvara se u odgovarajući ASCII dekadski kod. Svaki se dekadski zapis pretvara u binarni zapis nadopunjen na 8-bita, što znači da će naše veliko slovo "A" biti prvo pretvoreno u dekadsku vrijednost 65, a nakon toga u binarni zapis "01000001". Kada je to učinjeno, sa slike se dohvaćaju vrijednosti 4 pixela, koliko je i potrebno da se jedan znak iz teksta "*utisne*" u sliku. Od dohvaćena 4 pixela promatramo dekadske vrijednosti segmenata plave boje i njih pretvaramo u polje od 4 dekadske vrijednosti pa kasnije svaku od te 4 vrijednosti u binarni zapis. U ovom trenutku imamo jedan 8-bitni zapis znaka i četiri 8-bitna zapisa plavoga segmenta pixela sa slike. Sljedeće što je potrebno napraviti je rastaviti 8-bitni zapis znaka na četiri 2-bitna elementa krenuvši od najznačajnijega bita prema najmanje značajnome bitu. Na primjeru slova "A" nakon rastavljanja dobivamo polje ["01", "00", "00", "01"]. Prolazeći kroz ovo polje binarnih vrijednosti prolazimo simultano i kroz polje binarnih zapisa segmenata plave boje pixela. Za svaki 2-bitni zapis u polju provjeravamo vrijednosti zadnja dva najmanje značajna bita kod zapisa pixela. Ako se

vrijednosti zadnja dva najmanje značajna bita razlikuju od odgovarajučeg 2-bitnog zapisa, vrijednost tog pixela se mijenja tako da ta zadnja dva bita budu identična dobivenim 2-bitnim zapisima.



Slika 2.12. *Grafički prikaz "utiskivanja" jednog znaka u sliku*

Ovim načinom modulacije vrijednosti pixela maksimalna promjena koja se može dogoditi na pixelu je promjena za 3 vrijednosne veličine u nijansi plave boje i to u slučaju kada su zadnja dva najmanje značajna bita pixela "00", a očekivana vrijednost je "11" ili obratno. Ova je količina promjene ljudskome oku gotovo neprimjetna, a ako još tome pridodamo veličinu pixela na modernim zaslonima, usudio bih se reći da je u potpunosti neprimjetna. Za primjer pogledajte sliku 2.13. kako biste se uvjerili u razmjer promjene.



Slika 2.13. *Razlika u bojama za najveće odstupanje*

Na ovaj način aplikacija prolazi kroz dani tekstualni dokument uzimajući znak po znak i slijedeći gore opisani algoritam vrši promjenu na slici, odnosno "*utiskuje*" željeni tekst u sliku. Važno je još spomenuti kako aplikacija prije početka "utiskivanja" izračuna koliko znakova tekstualna datoteka ima te taj podatak na sličan način dodatno "*utisne*" u prvih nekoliko pixela slike. Ovaj korak je nužan za kasniji proces dekripcije, ali o njemu nešto kasnije.

Sada kada sam objasnio algoritam koji aplikacija treba koristiti, slijedi programsko rješenje:

```
private void EncodeButton_Click(object sender, EventArgs e)
{
    StreamReader sr = new StreamReader(TextFilePath.Text);
    string text = sr.ReadToEnd();
    SaveFileDialog saveImage = new SaveFileDialog();
    saveImage.ShowDialog();
    saveImage.Filter= "Png Files (*.png) | *.png";
    saveImage.DefaultExt = "png";
    saveImage.AddExtension = true;
    Bitmap image = new Bitmap(ImageFilePath.Text);
    GenerateHeader(text.Length, image);
    EncodeMessage(image);
    image.Save(saveImage.FileName);

}
```

Slika 2.14. *Poziv funkcije EncodeButtonClick*

Kada korisnik klikne na "Encode Button", dolazi do poziva funkcije "EncodeButton_Click" (slika 2.14.). Ova funkcija pokreće "StremReader", odnosno objekt zadužen za manipulaciju datotekama u C# okruženju. Stream reader-u predaje se TXT datoteka koju je korisnik prethodno odabrao kroz "File Explorer" i on sav sadržaj datoteke sprema u string varijablu text. Nakon toga otvara se "SaveFileDialog", odnosno upit korisniku gdje želi spremiti novu sliku koja će biti rezultat "*utiskivanja*" teksta u sliku. Kada je to sve obavljeno, poziva se funkcija "EncodeMessage" kojoj se predaje učitana slika, a ona radi kriptiranje prema algoritmu koji je opisan ranije u tekstu.

Izvorni kod funkcije dan je na slici 2.15.

```
public void EncodeMessage(Bitmap image)
{
    StreamReader sr = new StreamReader(TextFilePath.Text); //   D O H V A C A N J E
    string textFromData = sr.ReadToEnd();                  //      T E K S T A
    //textBox1.Text=textFromData;                          //   I Z   D A T O T E K E
    sr.Close();
    StreamWriter sw = new StreamWriter("bynaryOutput.txt");    // G E N E R I R A N J E
    for (int i = 0; i < textFromData.Length; i++)
    {                                                      //    B I A R N E
        string block = getBinNum(textFromData[i]);         //   D A T O T E K E
        sw.Write(block);

    }
    sw.Close();
    StreamReader BinLine = new StreamReader("bynaryOutput.txt");
    string line = BinLine.ReadToEnd();
    BinLine.Close();
    int n = line.Length;
    StreamWriter sw1 = new StreamWriter("bynaryOutput.txt");    // G E N E R I R A N J E
    for (int k= 0; k < textFromData.Length; k++)
    {                                                      //    B I A R N E
        string block = getBinNum(textFromData[k]);         //   D A T O T E K E
        sw1.WriteLine(block);
    }
    sw1.Close();
    int Counter = image.Width;
    int PxCounter = 0;
    textBox1.AppendText("Duljina bin datoteke: ");
    textBox1.AppendText(n.ToString());
    textBox1.AppendText(" znakova");
    textBox1.AppendText(Environment.NewLine);
    StreamReader sr1 = new StreamReader("bynaryOutput.txt");
    int z = image.Width * image.Height;
    textBox1.AppendText(Environment.NewLine);
    textBox1.AppendText("WRITING TO BODY");
    textBox1.AppendText(Environment.NewLine);
    textBox1.AppendText("--------------------------");
    textBox1.AppendText(Environment.NewLine);
    for (int i=0; i<n/8; i++)
    {
        string binValueOfCharacter = sr1.ReadLine();
        for (int j = 0; j < 4; j++)
        {
            int x = Counter % image.Width;
            int y = Counter / image.Width;
            Color Pixel = image.GetPixel(x, y);
            string binValueOfPixel = getBinNum(image.GetPixel(x, y).B);
            //textBox1.AppendText("Org: " + binValueOfPixel.ToString());
            var ColorBuilder = new StringBuilder(binValueOfPixel);
            ColorBuilder.Remove(6, 1);
            ColorBuilder.Insert(6, binValueOfCharacter[2 * j]);
            ColorBuilder.Remove(7, 1);
            ColorBuilder.Insert(7, binValueOfCharacter[2 * j+1]);
            //textBox1.AppendText("    New: " + ColorBuilder.ToString());
            //textBox1.AppendText(Environment.NewLine);
            int NewValueOfColor = getIntFromBin(ColorBuilder.ToString());
            image.SetPixel(x, y, Color.FromArgb(Pixel.R,Pixel.G, NewValueOfColor));
            Counter++;
            PxCounter++;
        }
    }
    textBox1.AppendText(" Obradeno "+PxCounter.ToString()+" px");
}
```

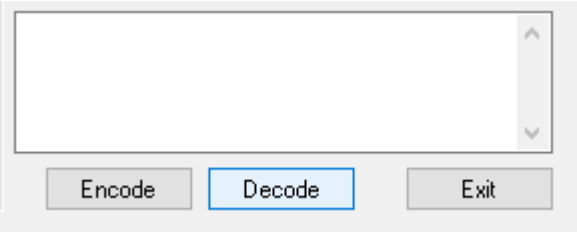
Slika 2.15. *Izvorni kod funkcije EncodeMessage*

3.7. Proces dekriptiranja

Druga najvažnija funkcija ove aplikacije je ona za dekriptiranje kriptiranih slika. Njezino ponašanje slijedi gotovo isti algoritam kao i za kriptiranje, samo u suprotnom smijeru, naravno uz neke izmjene. Kod pokretanja ove funkcije ona prvo provjerava tzv. header slike. Header slike je skup pixela koji se nalaze u prvom retku slike, a njega sam koristio kako bih mogao spremiti koliko je pixela promjenjeno, odnosno koliko je znakova "*utisnuto*" u sliku. Kada funkcija pročita podatak iz headera, ona prema toj informaciji postavlja maksimalan broj ponavljanja petlji koje prolaze kroz sliku i vrše dekodiranje. Svrha headera nije samo za postavljanje broja ponavljanja petlji već nam omogučava da izvršimo dekodiranje, odnosno dohvaćanje kriptiranoga teksta iz slike bez posjedovanja originalne slike. Način na koji ovo funkcionira je sljedeći.

Kada se pročita podatak o tome koliko je znakova "*utisnuto*" u fotografiju iz headera, može se izračunati jednostavno koliko je pixela zahvaćeno promjenom. Za svaki "*utisnuti*" znak potrebno je 4 pixela pa na primjer ako je tekstualna datoteka sadržavala 100 znakova, na slici treba promotriti 400 pixela. Nakon izračunatoga broja zahvaćenih pixela promjenom potrebno je "proći" kroz te pixele i grupirati ih u skupine od 4 komada. Kada imamo formiranu skupinu, odnosno polje od 4 pixela, generiramo binarni broj za svaku vrijednost plave komponente pojedinoga člana. Od svakoga člana polja uzimamo zadnja dva najmanje značajna bita i "spajamo" ih u novi 8-bitni binarni broj krenuvši od najznačajnijega prema najmanje značajnome bitu. Tada dobiveni broj prebacujemo u dekadski sustav i uspoređujemo ga s ASCII tablicom gdje tražimo odgovarajući znak. Tako dobivene znakove spremamo jedan za drugim u novu tekstualnu datoteku sve dok se ne "obiđu" svi zahvaćeni pixeli.

Funkcija koja slijedi navedeni algoritam poziva se kada korisnik pritisne "Decode Button", a izvorni kod ove funkcije može se vidjeti na slici 2.17.



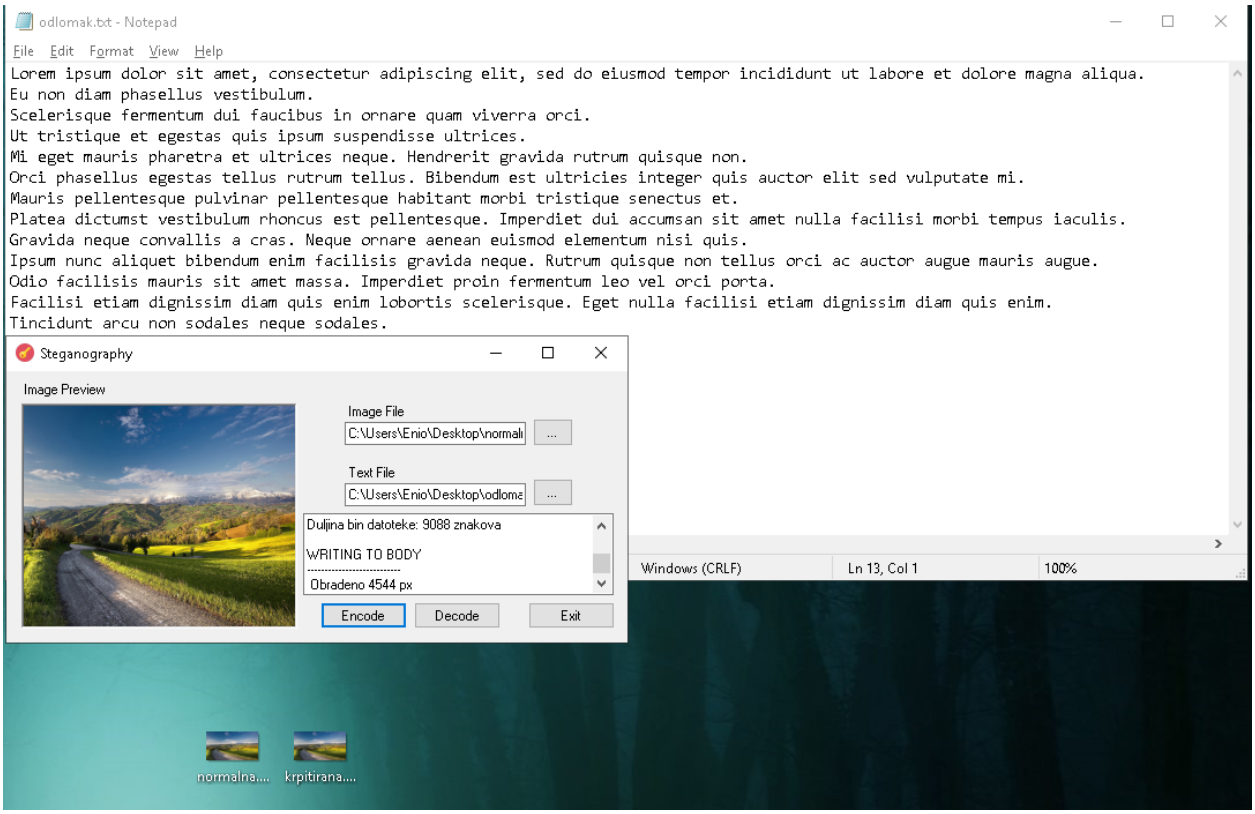
Slika 2.16. *Decode Button*

```
private void DecodeButton_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.ShowDialog();
    sfd.Filter = "Txt Files (*.txt) | *.txt";
    sfd.DefaultExt = "txt";
    sfd.AddExtension = true;
    Bitmap image = new Bitmap(ImageFilePath.Text);
    textBox1.AppendText("--------------------------");
    textBox1.AppendText(Environment.NewLine);
    string BinHeader = "";
    for (int i = 0; i < 8; i++) {
        string binPixel = getBinNum(image.GetPixel(i, 0).B);
        BinHeader = BinHeader.Insert(BinHeader.Length, binPixel[6].ToString());
        BinHeader = BinHeader.Insert(BinHeader.Length, binPixel[7].ToString());
    }
    string message = "";
    int n = getIntFromBin(BinHeader);
    textBox1.AppendText("INFO FROM HEADER:");
    textBox1.AppendText(Environment.NewLine);
    textBox1.AppendText("This image contains ");
    textBox1.AppendText(getIntFromBin(BinHeader).ToString());
    textBox1.AppendText(" characters");
    textBox1.AppendText(Environment.NewLine);
    textBox1.AppendText("--------------------------");
    textBox1.AppendText(Environment.NewLine);
    int counter = image.Width;
    StreamWriter sw = new StreamWriter(sfd.FileName);
    string character = "";
    for(int i=0; i<n; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            int x = counter % image.Width;
            int y = counter / image.Width;
            string binValueOfPixel = getBinNum(image.GetPixel(x, y).B);
           character=character.Insert(character.Length, binValueOfPixel[6].ToString());
           character=character.Insert(character.Length, binValueOfPixel[7].ToString());
            counter++;

        }

        int AsciiCharacter = getIntFromBin(character);
        char c = (char)AsciiCharacter;
        message=message.Insert(message.Length, c.ToString());
        sw.Write(c);
        character = "";
        //textBox1.AppendText(c.ToString());

    }
    sw.Close();
}
```

Slika 2.17. *Funkcija DecodeButton_Click*

4. ANALIZA KVALITETE KRIPTIRANIH DATOTEKA

U ovome poglavlju fokusirat ću se na analizu kvalitete obavljanja kriptiranja svoje aplikacije. Tijekom pisanja aplikacije želja mi je bila da svi postupci koji se obavljaju nad učitanim datotekama ne ostavljaju previše vidljivih "tragova" već da kriptirane fotografije izgledaju naočigled identično onima koje to nisu. Vodeći se ovom željom morao sam žrtvovati pojedine stvari. Za "*utiskivanje*" jednoga slova morao sam iskoristiti 4 pixela kako sam već ranije naveo. Prednost tomu je što tada promjena na boji nije velika i nije ju lako primjetiti. Glavni nedostatak zapisa jednoga znaka u 4 pixela je velika potrošnja pixela za zapis, no uzimajući u obzir da je danas standardna minimalna rezolucija slike 1920x1080, dolazimo do zaključka da je u jednu fotografiju te veličine moguće pohraniti preko pola milijuna znakova. Na slici 3.1. i slici 3.2. možemo vidjeti primjer nekriptirane fotografije i fotografije u koju je "*utisnut*" slučajno generirani tekst.



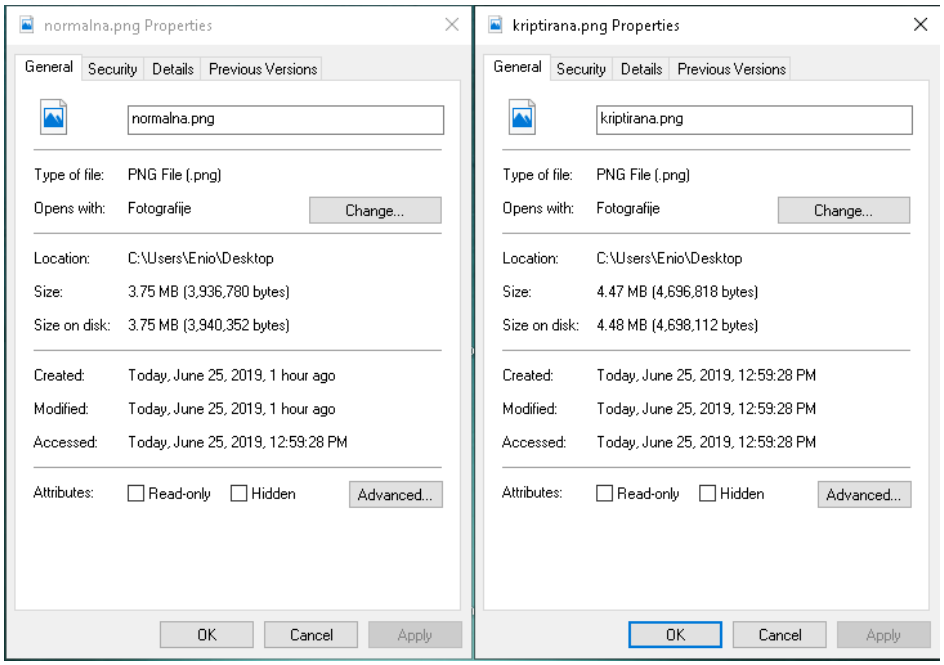
Slika 3. *Kriptiranje sa aplikacijom*

Slika 3.1. *Normalna slika*

Slika 3.2. *Kriptirana slika*

4.1. Razlika u veličini datoteka

Promotrimo postoji li razlika u veličini na disku između kriptirane i obične slike i ako postoji, razjasnimo zbog čega je nastala.

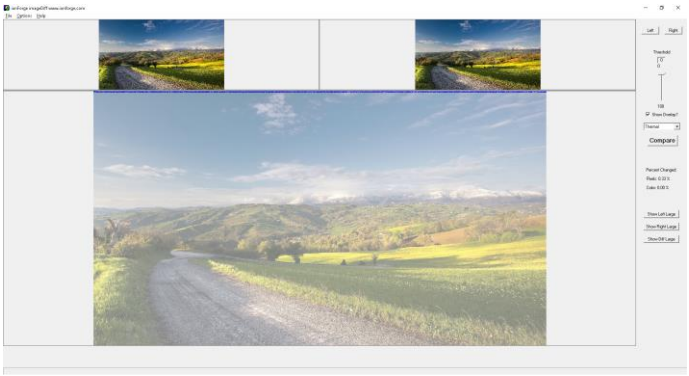


Slika 3.3. *Razlika u veličinami slikovnih datoteka*

Primjećujemo kako novonastala slika zauzima malo više prostora na disku. Razlog ovome nije rad nad sadržajem slike nego način na koji Visual Studio barata sa slikovnim datotekama. Veličina nove slike ne bi trebala biti promijenjena dodavanjem teksta u nju iz razloga što aplikacija ne dodaje novi sadržaj na sliku već modulira postojeći i tako ostvaruje zapis. Ovaj maleni porast veličine je konstantan za bilo koju količinu upisanog teksta što znači da ako "utisnemo" jedan znak ili čitavi odlomak, porast veličine bit će isti. Porast na veličini "ne odaje" sliku da je ona kriptirana jer mi ne možemo znati točnu veličinu datoteke koju je potrebno da slika ima ako izgleda tako kako izgleda.

4.2. Postotna razlika u promjenjenom sadržaju

Kako bih mogao ispitati kolika je postotna razlika između kriptirane i obične slike, preuzeo sam program s interneta koji radi usporedbe nad slikama. Riječ je o programu zvanom “ImageDiff“ kojega se može preuzeti sa stranice [7] Program je jednostavnog sučelja i omogućava učitavanje dvije fotografije na kojima se radi usporedba. Učitao sam kriptiranu i običnu fotografiju i zatražio od programa usporedbu.



Slika 3.4. *ImageDiff program za usporedbu slika*

Program je nakon usporedbe generirao novu fotografiju na kojoj je naznačio promjene koje su napravljene od strane moje aplikacije. Promjene se mogu vidjeti na slici 3.5. Osim toga, izračunao je postotnu promjenu od 0.33% promijenjenih pixela.

Slika 3.5. *Usporedna slika na kojoj su naznačene promjene plavom bojom*

5. **ZAKLJUČAK**

Čuvanje povjerljivih informacija i/ili datoteka u današnje vrijeme postalo je vrlo teško zbog globalne povezanosti. Ova činjenica dovodi nas do zaključka da datoteka koja je pohranjena na vašem disku nije vidljiva samo vama već možda i osobama koje se bave cyber kriminalom, a nalaze se na drugoj strani planeta. Uzmemo li primjerice u obzir da ne želite tu datoteku dijeliti, zaključujemo da ste u problemu. Što ako postoji način da unatoč globalnoj povezanosti vaše osjetljive datoteke poprime oblik nevažnih datoteka, a uz to budu dostupne samo vama? Steganografski postupci omogućuju nam uparvo navedeno, a ova aplikacija o kojoj ste do sada čitali primjer je steganografije na djelu. Koristeći algoritme za manipulaciju sadržajem tekstualnih i slikovnih datoteka aplikacija uspješno provodi kriptiranje, ali i dekriptiranje i to u vrlo kratkom vremenskom razdoblju.

LITERATURA

[1] Wikipedia: Steganography, dostupno na:

https://en.wikipedia.org/wiki/Steganography

lipanj (2019)

[2] Wikipedia: Visual Studio, dostupno na :

https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

lipanj (2019)

[3] Wikipedia : C# (programming language), dostupno na :

https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

lipanj (2019)

[4] Microsoft docs: Instaling Visual Studio, dostupno na :

https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019

lipanj (2019)

[5] Wikipedia: Pixel , dostupno na :

https://en.wikipedia.org/wiki/Pixel

lipanj (2019)

[6] Wikipedia : ASCII, dostupno na :

https://en.wikipedia.org/wiki/ASCII

lipanj (2019)

[7] SoftPedia: ImageDiff , dostupno na :

https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Others/ImageDiff.shtml

lipanj (2019)

SAŽETAK

U ovom završnom radu nakon istraživanja i proučavanja steganografskih postupaka napravio sam aplikaciju koja oslanjajući se na njih provodi kriptiranje slika. Također, kriptirane slike možemo dekriptirati i izvući iz njih "*utisnuti*" tekst uz pomoć iste. Proces pisanja programskoga rješenja za ovaj projekt nije bio posve lagan iz razloga što sam odlučio napraviti da kriptirane fotografije budu neovisne o originalnoj fotografiji, a to je zahtjevalo osmišljavanje dodatnog headera zapisanog u slici. Logika koja je upotrebljena u ovoj aplikaciji može se iskoristiti u nekim budućim i većim projektima kod kojih je želja zaštiti osjetljive informacije ili čak izvršiti skrivanje drugih formata datoteka u fotografije, naravno uz dodatne izmjene u kodu.

Ključne riječi : Steganografija, aplikacija, kriptiranje, dekriptiranje

ABSTRACT

In this bachelor's thesis, after researching and studying steganographic procedures, I made an application that relays on those procedures to encrypt the images. Also encrypted the images can be decrypted and inserted text can be extracted. The process of writing the software solution for this project was not entirely easy because I decided to make the encrypted images independent of the original ones, which required the design of an additional header written in the image. The logic used in this application can be used in some future and major projects where the desire is to protect sensitive information, or even hide other file formats into images, of course, with additional code changes.

Key words: Steganography, application, encryption, decryption

ŽIVOTOPIS

Enio Biondić rođen je 28. studenoga 1997. godine u Virovitici. Nakon završenog osnovnoškolskog obrazovanja u Osnovnoj školi Ivane Brlić Mažuranić u Virovitici 2012. godine upisuje Srednju tehničku školu, smjer elektrotehničar, također u Virovitici. U svome srednjoškolskom obrazovanju sudjeluje u natjecanjima iz matematike i raznim projektima, a 2016. odlazi na Erasmus+ projekt razmjene studenata u Veliku Britaniju. Nakon državne mature, iste godine, upisuje tadašnji Fakultet elektrotehnike i računarstva u Osijeku, preddiplomski smjer računarstvo.

Enio Biondić