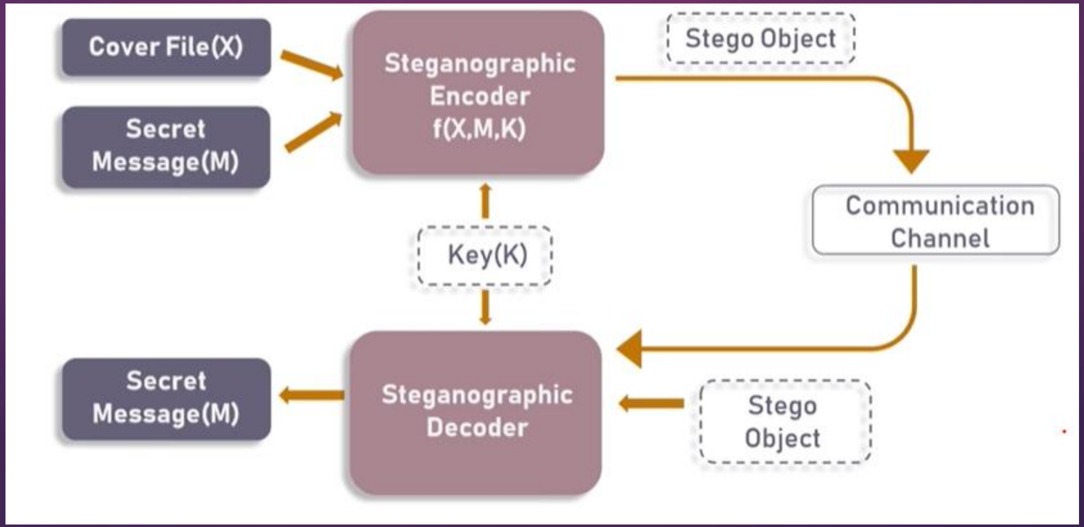
Image Steganography

Self project assigned by: Lilach Naor & Sapir Shemesh

What is Image Steganography?

Image Steganography is the technique of hiding data(including text, image, audio, video) inside an image in such a way that a human being generally won't be able to distinguish between the manipulated image and the original image. This adds an extra layer of security to the data being transmitted.

It is often used among hackers to hide secret messages or data within media files such as images, videos or audio files. Even though there are many legitimate uses for Steganography, malware programmers have also been found to use it to obscure the transmission of malicious code.



Representation of characters in the computer:

In order to represent characters in computer, we use Ascii table, which mapping every single character into it binary 8-bit representation.

For example, the word Dog in binary representation would look like that:

D = 01000100
o = 01101111
g = 01100111

Dog = 01000100 01101111 01100111

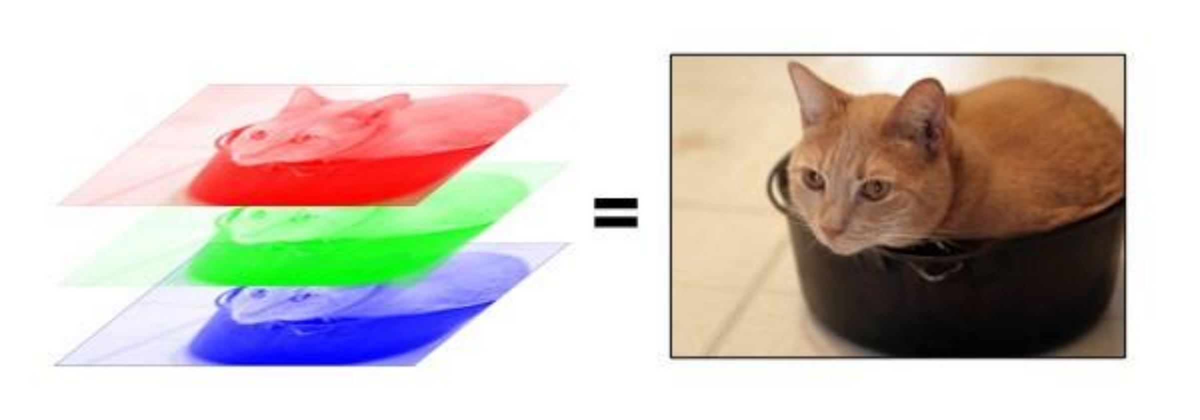
Decimal - Binary - Octal - Hex – ASCII
Conversion Chart

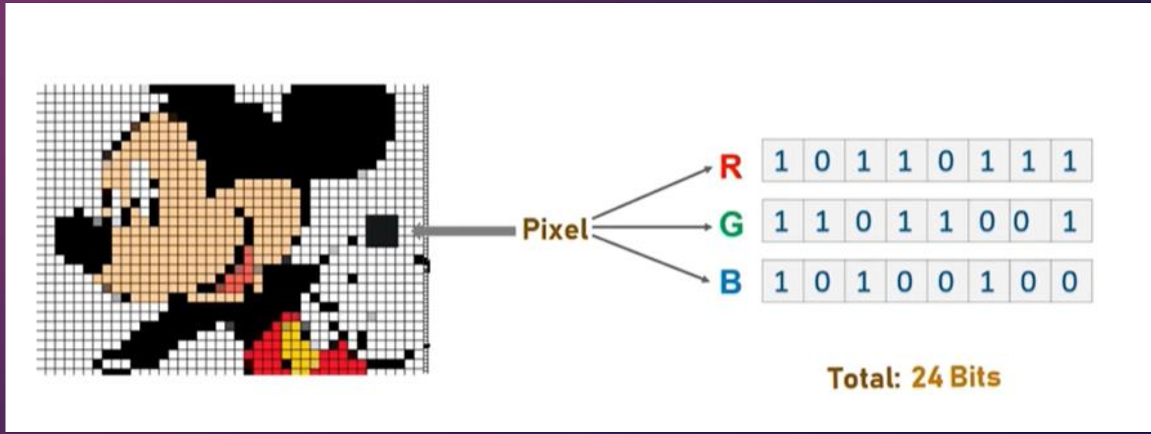
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

What is a digital image?

Before understanding how can we hide data inside image, we need to understand what a digital image is.

We can describe a digital image as a finite set of digital values, called pixels. Pixels are the smallest individual element of an image, holding values that represent the brightness of a given color at any specific point. So we can think of an image as a tensor (or a three-dimensional array) of pixels which contains a fixed number of rows, columns and color.
Each pixel describes a color value, there are 2 ways to describe colors: RGB (red-green-blue) and BGR (blue-green-red).



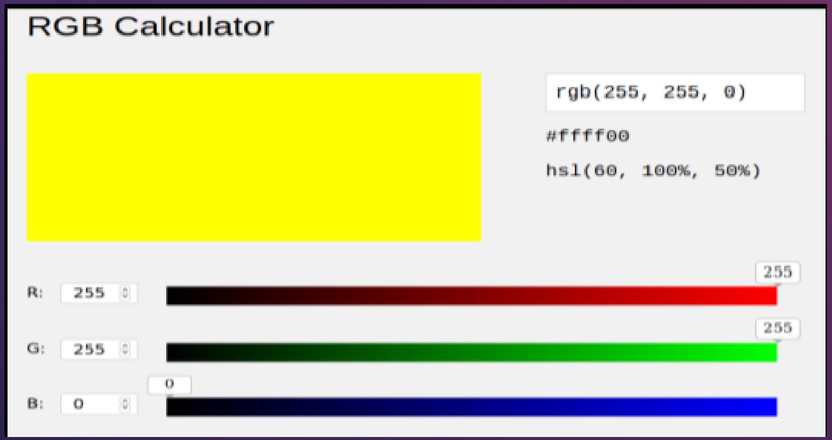


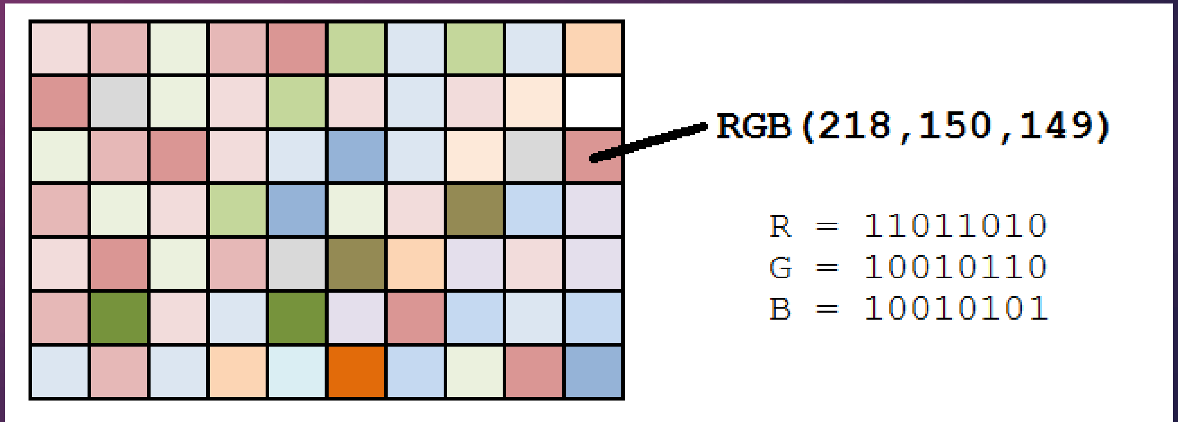
Pixel concept and color models:

combining the 3 layers in both color format will allow the human eye to enjoy all color variety, because this is how the human eye sees red, blue and green (when the higher sensitivity is to shades of green).

In the images below, we can see that each pixel ranges from 0-255, so that each RGB value represent different color. (e.g. RGB(255, 255, 0) will represent the color yellow, and so on).

In this project, we will focus on the RGB format.





Hiding text inside image - the LSB (least significant bit) technique

The leftmost bit is the most significant bit. If we change the leftmost bit it will have a large impact on the final value. On the other hand, the rightmost bit is the less significant bit. If we change the rightmost bit it will have less impact on the final value. Note that the rightmost bit will change only 1 in a range of 256 (it represents less than 1%)
You can understand that from the image below.

Note that this method only works on Lossless-compression images, which means that the files are stored in a compressed format, but that this compression does not result in the data being lost or modified.

How LSB technique works?

Let’s take an example of how this technique works, suppose you want to hide the message “hi” into a 4x4 image which has the following pixel values: [(225, 12, 99), (155, 2, 50), (99, 51, 15), (15, 55, 22),(155, 61, 87), (63, 30, 17), (1, 55, 19), (99, 81, 66),(219, 77, 91), (69, 39, 50), (18, 200, 33), (25, 54, 190)]

Using the ASCII Table, we can convert the secret message into decimal values and then into binary: 0110100 0110101.

Now, we iterate over the pixel values one by one, if the pixel is odd, and the bit of the char is ’0’ then modify the pixel by -1. else, if the pixel is even and the bit is ‘1’ – modify it by -1 (if the pixel is 0, then modify by +1). This will only modify the pixel values by +1 or -1 which is not noticeable at all. The resulting pixel values after performing LSBS is as shown below:

[(224, 13, 99),(154, 3, 50),(98, 50, 15),(15, 54, 23),(154, 61, 87),(63, 30, 17),(1, 55, 19),(99, 81, 66),(219, 77, 91),(69, 39, 50),(18, 200, 33),(25, 54, 190)]

Encoding algorithm

After we discussed the LSB technique, let's dive into the algorithm for encoding text into image.

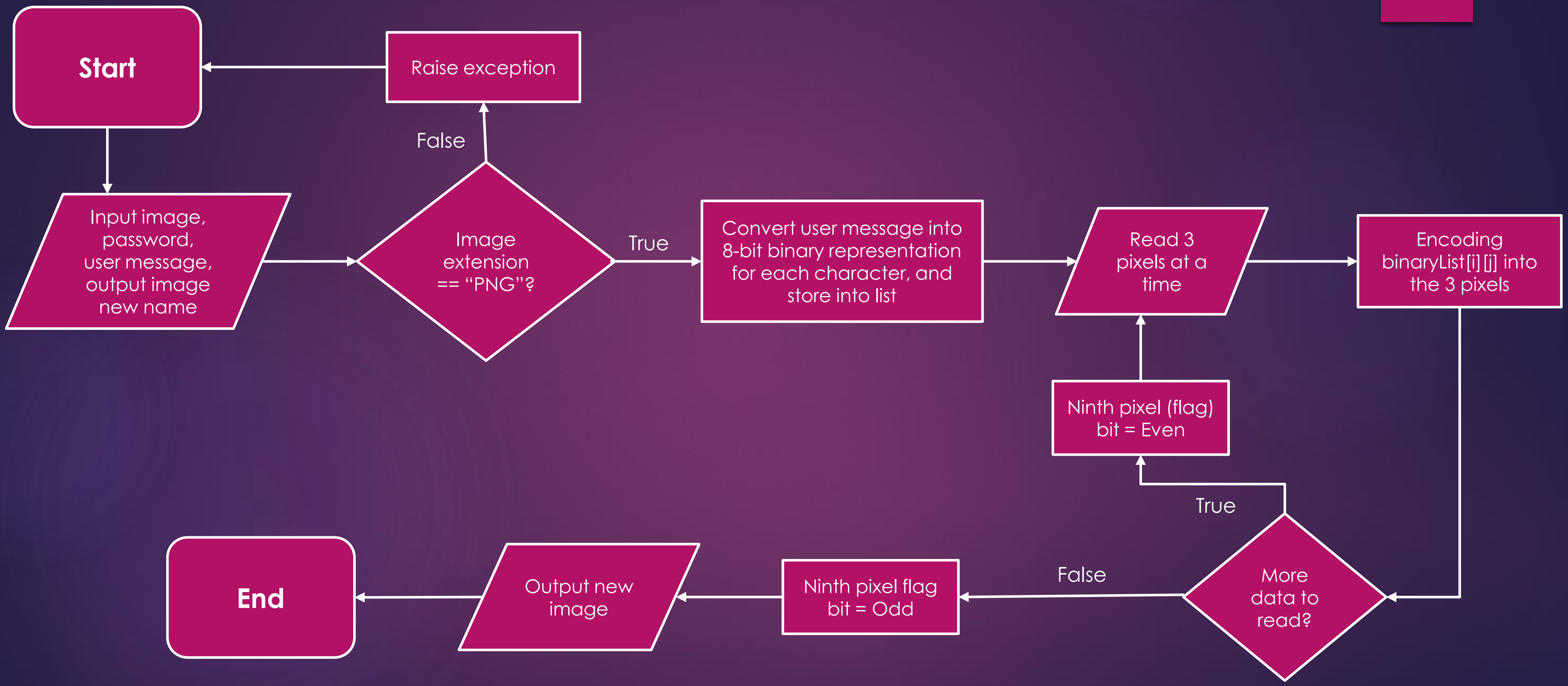
1. For each character in the data, its ASCII value is taken and converted into 8-bit binary

2. Three pixels are read at a time having a total of 3*3=9 RGB values. The first eight RGB values are used to store one character that is converted into an 8-bit binary.

3. The corresponding RGB value and binary data are compared. If the binary digit is 1 then the RGB value is converted to odd and, otherwise, even.

4. The ninth value determines if more pixels should be read or not. If there is more data to be read, i.e. encoded or decoded, then the ninth pixel changes to even. Otherwise, if we want to stop reading pixels further, then make it odd.

Flowchart - Encoding algorithm



Decoding algorithm

For decoding, we shall try to find how to reverse the previous algorithm that we used to encode data.

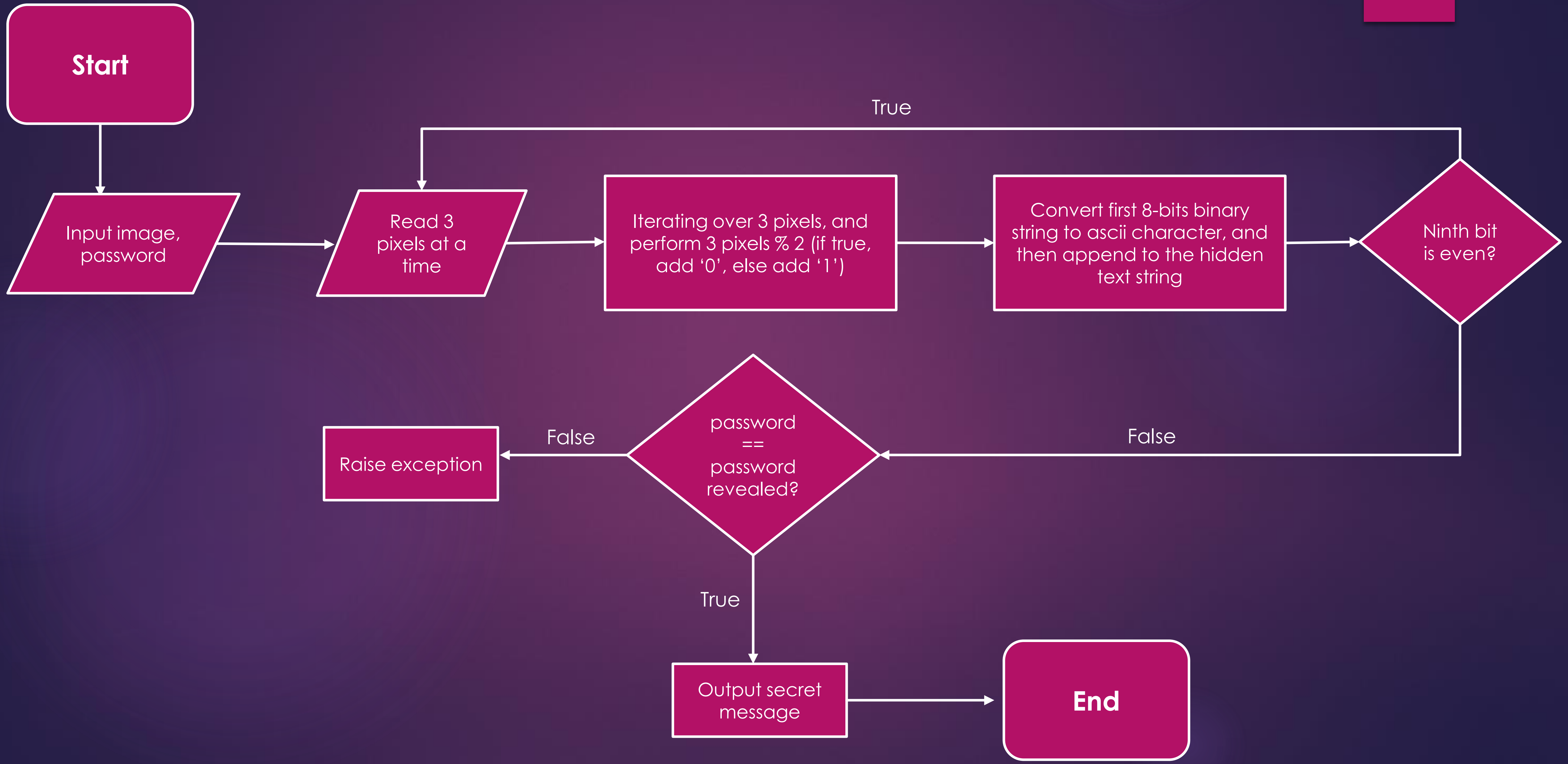
1. Three pixels are read at a time. The first 8 RGB values give us information about the secret data, and the ninth value tells us whether to move forward or not.

2. For the first eight values, if the value is odd, then the binary bit is 1, otherwise it is 0.

3. The bits are concatenated to a string, and with every three pixels, we get a byte of secret data, which means one character.

4. Now, if the ninth value is even then we keep reading pixels three at a time, or otherwise, we stop.

Flowchart - Decoding algorithm



Example

Pixels values before encoding:

Pixel 1			Pixel 2			Pixel 3		
175	220	137	136	215	120	195	123	152
D = 0	1	0	0	0	1	0	0	

End of string flag 0/1

Pixels values after encoding:

Pixel 1			Pixel 2			Pixel 3		
174	219	136	136	214	119	194	122	151

Pixels values before decoding:

Pixel 1			Pixel 2			Pixel 3		
174	219	136	136	214	119	194	122	151

Pixels values % 2:

Pixel 1			Pixel 2			Pixel 3		
0	1	0	0	0	1	0	0	1

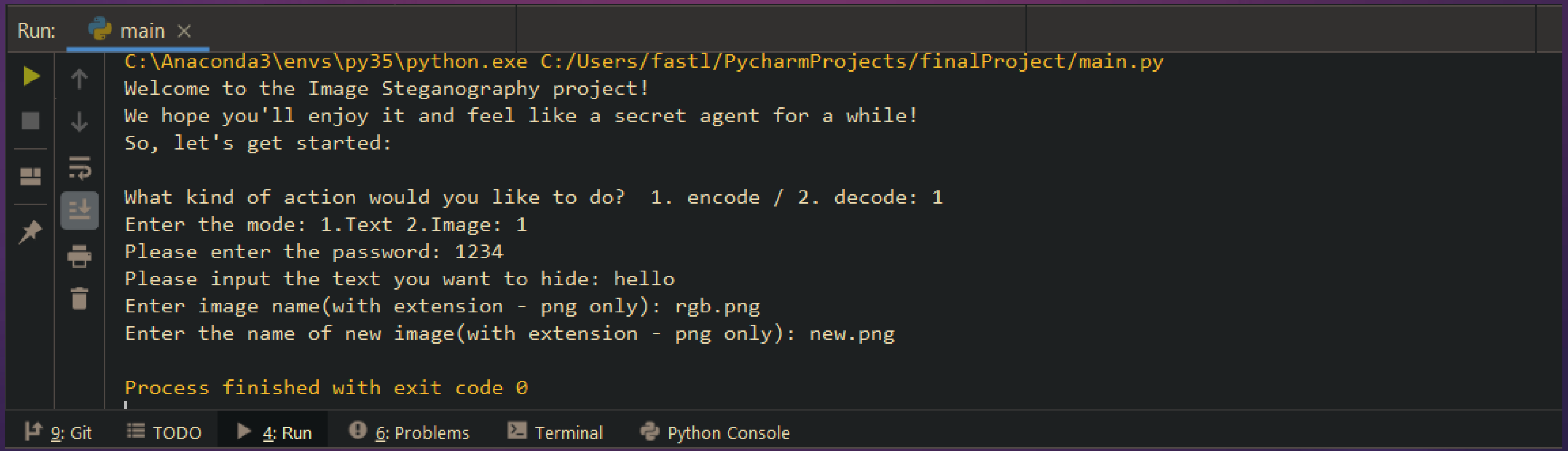
Total = 010001001

D = 01000100

Flag = 1

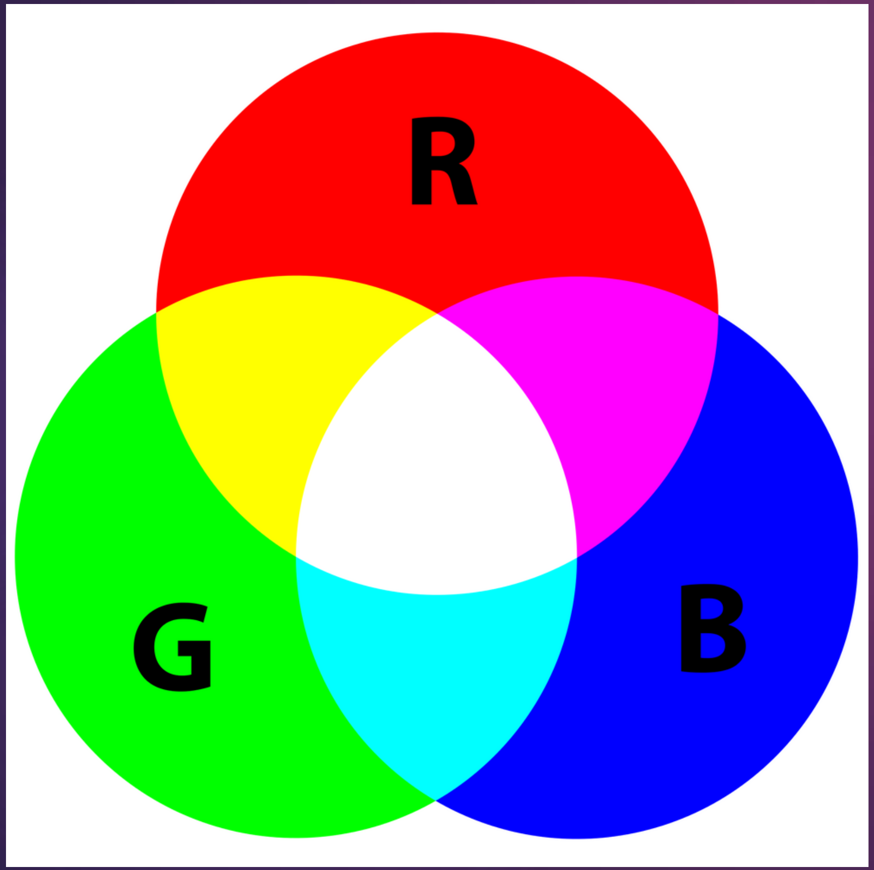
Running example

1. First, the user will be asked to choose between encode/decode, then choose between text or image.
2. Password will be taken from user (without encrypt it with the text).
3. The data will include the password + '@' + text ('@' for flag when the password ends). The encrypted text will be converted into raw binary data using ASCII table
4. This modified image will contain the secret data, which can only be read if someone has the right password.

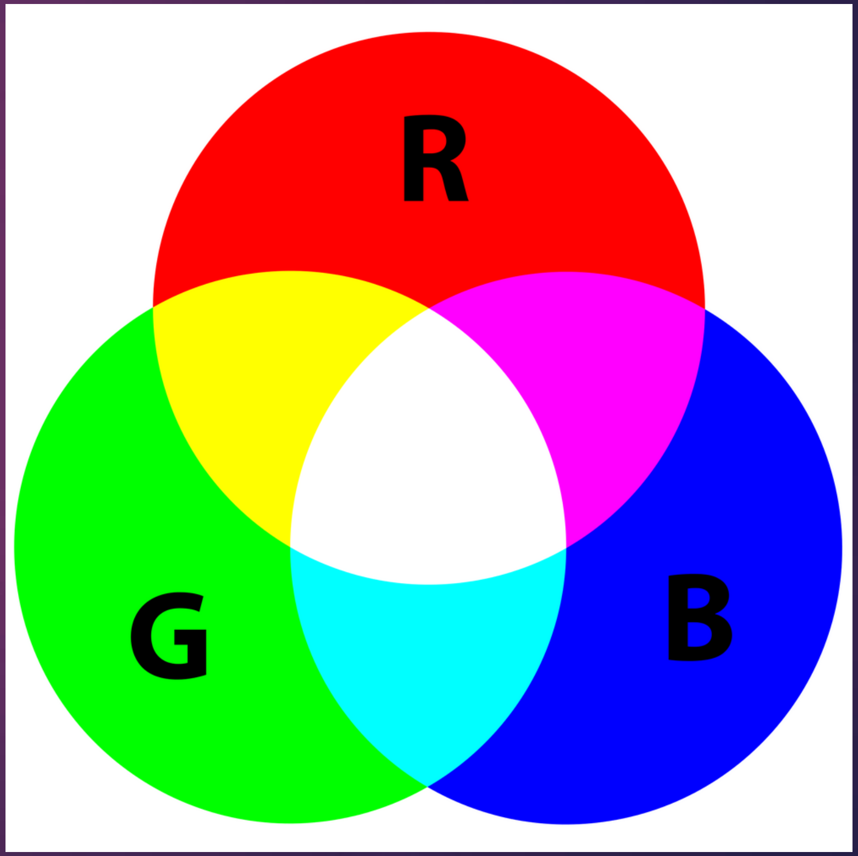


Running example – comparing two images

Input image



output image



Hiding image inside image

Since we understood the pixel concept and color models, we can talk about the procedure of hiding an image inside another.

For example, suppose we have to hide img2 in img1, where both img1 and img2 are numpy ndarray of pixel values. The size of img2 must be less than the size of img1. We are using color images, hence both will have 3 values (red, green, blue).

Each pixel value varies from 0 to 255, so each pixel value is of 1 byte or 8 bits. Let new_image[i][j][l] be the pixel value at location (i, j) and of color layer l where i varies from 0 to width and j varies from 0 to height and l varies from 0 to 2.

Note: The quality of the new images is a little bit less than the old images.

Encoding algorithm

1. Iterating 3-nested for loops over image1 and image2 (rows, columns, and channel of color).

2. For each pixel in image1[i][j][k] we will take the 4 MSB of image1 and 4 MSB of image2, then concatenating them so that the image1 MSB'S will be the MSB in the new binary representation, and the image2 MSB'S will be the LSB in the new image – as shown below.

3. Then, we will output the new image – image1, with the pixels modified.

Decoding algorithm

1. First of all, we need to create 2 blank images, by numpy.zeros function.

2. Then, we will iterate over each pixel of the input image, and convert it into 8-bit binary representation.

3. For each location source[i][j][k] we take the first MSB bits, and add 4 bits of 0 or 1 randomly, which effects the original pixel value (of the original images), by maximum 16 in RGB values, because we have no way to restore the bits that were before.

4. Then we will write the new ndarray images into images using CV2 module.

Running example

1. First, the user will be asked to choose between encode/decode, then choose between text or image.
2. User will input 2 images for merging them, and will choose a new name for the merged image.
3. This modified image will contain the source image, and the hidden image.

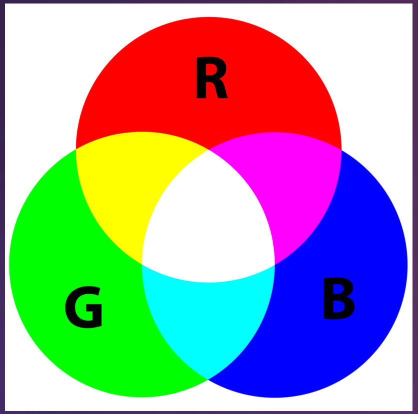
Run: main

```
C:\Anaconda3\envs\py35\python.exe C:/Users/fastl/PycharmProjects/finalProject/main.py
Welcome to the Image Steganography project!
We hope you'll enjoy it and feel like a secret agent for a while!
So, let's get started:

What kind of action would you like to do?  1. encode / 2. decode: 1
Enter the mode: 1.Text 2.Image: 2
Enter name of the original image with png extension only: rgb.png
Enter name of the hidden image with png extension only: sky.png
Enter the name for merged images - with png extension only: new.png

Process finished with exit code 0
```

9: Git | TODO | 4: Run | 6: Problems | Terminal | Python Console

Running example – comparing two images

Input image1

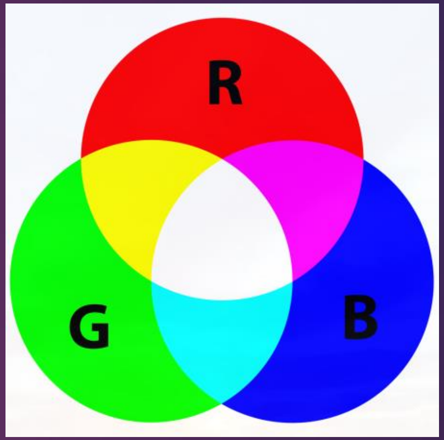


+

Input image2

=

output image



Project limitations

Text inside image:

1. Will work only with PNG images because the LSB method only works on Lossless-compression images, which means that the files are stored in a compressed format, but that this compression does not result in the data being lost or modified, PNG, TIFF, and BMP as an example, are lossless-compression image file formats.

2. The text to be hidden should be less or equal to number of image's pixels / 3.

Image inside another:

1. The 2 images must be at the same size (for optimal decoding).

2. For best result, please use pictures with the same shades as possible.