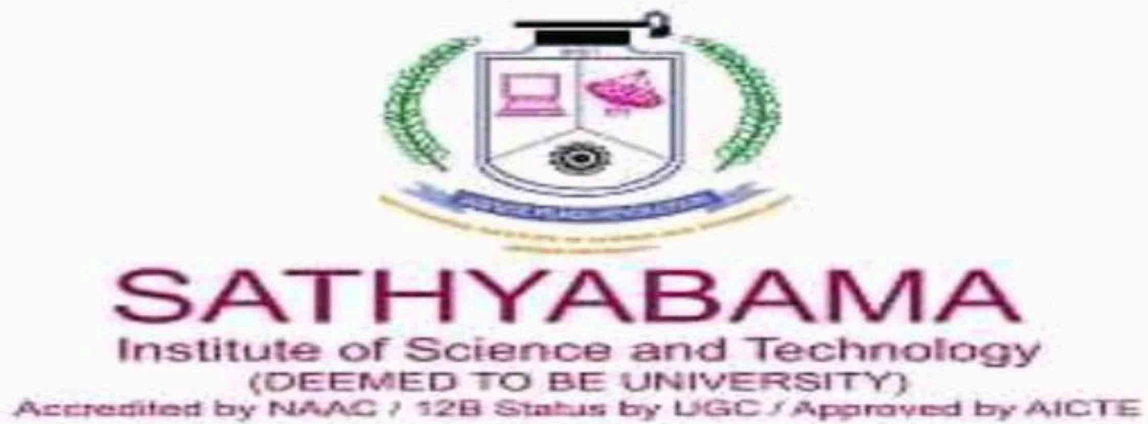




ENCRYPTING AN IMAGE UNDER IMAGE USING STEGANOGRAPHY

PROJECT REPORT

GUIDE: T.VIJAYASHREE (M.E,PhD)

TEAM MEMBERS:

PEKETI VEERA VENKATA SATYA TARUN(38130161)

POKURI DILLIP KUMAR(38130167)

Objective:

The main objective of the system is to secure the transmission of data with the help of LSB technique. LSB-Steganography is a steganography technique in which we hide messages inside an image by replacing Least significant bit of image with the bits of message to be hidden. So that we can share data anonymously without the fear of Data stealing.

Abstract:

- The development of high speed computer networks and Internet has increased the easiness of Information Communication. In contrast with Analog media and Digital media provide several different advantages such as high quality, simple editing, high loyalty copying and authenticity.
- But in the field of data communication this type of development has increased the fear of sneaking the data while sending data from the sender to the receiver. Due to this reason Information Security is main problem of Data Communication. Steganography plays an important role in field of Information Security. Video and images are very common choice for hiding data.
- It is very important for effective and successful embedding process to select appropriate pixels in the video frames, which are used to store the secret data. We use video based Steganography because of large size and memory requirements. Hiding information in a carrier file we use least significant bit (LSB) insertion technique.

- In Least significant bit (LSB) insertion technique, for hiding information we change LSB of video file with the information bits .This paper will focus on hiding information in specific frames of the video and in specific position of the frame by LSB substitution.

Introduction:

- Now a day's the uses of devices such as computer, mobile and many more other device for communication as well as for data storage and transmission has increases. As a result there is increase in no of user's also there is increase in no of unauthorized user's which are trying to access a data by unfair means.
- This arises the problem of data security. To solve this problem a data is stored or transmitted in the encrypted format.
- This encrypted data is unreadable to the un-authorized user. Cryptography is a science of information security which secures the data while the data is being transmit-ted and stored.
- Every encryption and decryption process has two aspects: the algorithm and the key use for the encryption and decryption. However, it is the key used for encryption and decryption that makes the process of cryptography secure. There are two types of cryptographic mechanisms: symmetric key cryptography in which the same key is use for encryption and decryption.
- In case of asymmetric key cryptography two different keys are used for encryption and decryption. Symmetric key algorithm is much faster and easier to implement and re-quired less processing power as compare to asymmetric key algorithm.

- The advance encryption standard (AES) specifies a federal information processing standards publication (FIPS) approved cryptographic algorithm that can be used to protect electronic data.

Literature survey:

1. Hybrid Approach to Text & Image Steganography using AES and LSB Technique

Author: Vikas M, Yashwanth E, Veeresh, Sanath Krishna S, Narender .M

For communication of secret messages or information from one place/source to another for different applications we use Steganography and Cryptography. Usually in Cryptography the content of secret message is scrambled while in Steganography secret message is embedded in a cover medium. In this paper we propose a secured model developed by combining Advanced Encryption Standard (AES) and Least Significant Bit (LSB) algorithms. Here, AES is used for Cryptography and LSB technique is used for Steganography. The system proposed encrypts a text or image inside a Cover image

[1] M Saritha, Sushravya M, Vishwanath M Kadabadi, “Image and Text Steganography with Cryptography using MATLAB”: This paper presents a system in which the text is encrypted using Symmetric XOR algorithm and Sequential algorithm is used to hide the data in cover image. The level of security provided is low. Also, the system defined is limited to low quality and to few types of images.

[2] Aman Arora, Manish Pratap Singh, Prateek Thakral, and Naveen Jarwal, “Image Steganography using Enhanced LSB Substitution technique”: This paper presents a system in which the bits are modified according to first bit set. If image has same pixel repeating over a region, then this method fails because the bit modification will bring changes to the entire pixels in the range, thus now the change in colour values will be noticeable to human eyes very easily.

[3] Utsav Seth and Shiva Saxena, “Image Steganography using AES Encryption and Least Significant Nibble”: This system provides a good security in comparison with

other papers above by using AES encryption. But there the quality of image is low and the after Steganography the final images have more noise compared to other papers this is because here least significant nibble is chosen over 1 or 2 least significant bits. Thus the human eyes can detect the fact that data is hidden in the image.

[4] Kamaldeep Joshi and Rajkumar Yadav, "New Approach toward Data Hiding using XOR for Image Steganography": This method assumes that both sender and receiver agree upon a cover image which is present at both the ends. But this method is very difficult as it will be difficult to maintain same image having same properties like size, resolution etc.

[5] Yambem Jina Chanu, T. Tuithung, k. Manglem Singh, "A short survey on image Steganography and steganalysis techniques": The paper describes a short survey on different types of Steganography techniques for image in spatial and transform domains and steganalysis techniques for the detection of secret message in the image.

2. Steganography Using AES and LSB Techniques

Author: Aishwarya Pandey, Prof. Jharna Chopra

With rapid growth in the digital market, Steganography is going to increase its importance by which the exponential development and secret communication of potential computer users are also increased over the internet. It can also be well defined as the study of secret invisible communication that generally deals with the different ways of concealing the existence of the communicated message. Usually, the data embedding is obtained in communication such as image, text, voice or multimedia content for copyright and also in military communication for authentication and many other different purposes. In image Steganography, the secret hidden communication is obtained through embed a message into a cover image which is used as the medium to embed message into the image and generate

a stego image which is a generated image which is carrying a secret hidden message. In this paper we propose a novel method hide secret information in combination of AES and LSB technique. The image quality are selected by the users and based on that the secret messages length are decided. Hence user has full right to select any size output based on requirements.

[1].Pixel-based algorithms generate the synthesized image pixel by pixel and use spatial neighborhood comparisons to choose the most similar pixel in a sample texture as the output pixel. Since each output pixel is determined by the already synthesized pixels, any wrongly synthesized pixels during the process influence the rest of the result causing propagation of errors.

[2].Otori and Kuriyama pioneered the work of combining data coding with pixel-based texture synthesis. Secret messages to be concealed are encoded into colored dotted patterns and they are directly painted on a blank image. A pixel-based algorithm coats the rest of the pixels using the pixel-based texture synthesis method, thus camouflaging the existence of dotted patterns. To extract messages the printout of the stego synthesized texture image is photographed before applying the data-detecting mechanism. The capacity provided by the method of Otori and Kuriyama depends on the number of the dotted patterns. However, their method had a small error rate of the message extraction.

[3].Patch-based algorithms paste patches from a source texture instead of a pixel to synthesize textures. This approach of Cohen et al. and Xu et al. improves the image quality of pixel-based synthetic textures because texture structures inside the patches are maintained. However, since patches are pasted with a small overlapped region during the synthetic process, one needs to make an effort to ensure that the patches agree with their neighbors.

[4].Liang et al. introduced the patch-based sampling strategy and used the feathering approach for the overlapped areas of adjacent patches. Efros and Freeman [17] present a patch stitching approach called “image quilting.” For every new patch to be synthesized and stitched, the algorithm first searches the source

texture and chooses one candidate patch that satisfies the pre-defined error tolerance with respect to neighbors along the overlapped region. Next, a dynamic programming technique is adopted to disclose the minimum error path through the overlapped region. This declares an optimal boundary between the chosen candidate patch and the synthesized patch, producing visually plausible patch stitching.

[5].Ni et al. proposed an image reversible data hiding algorithm which can recover the cover image without any distortion from the stego image after the hidden data have been extracted. Histogram shifting is a preferred technique among existing approaches of reversible image data hiding because it can control the modification to pixels, thus limiting the embedding distortion, and it only requires a small size location map, thereby reducing the overhead encountered. The current state-of-the-art for reversible image data hiding is the general framework presented by Li et al. [19].

3. Secure and hidden text using AES cryptography and LSB steganography

Author: may hattim abood*, zahraa khudhair taha

For secure data transmission over the internet, it is important to transfer data in high security and high confidentiality, information security is the most important issue of data communication in networks and the internet. Our main goal of this paper is to enhance the existing method of secure data communication possibly by combination cryptography and steganography. Cryptography and Steganography are two popular ways to transmit information in a secret way. In this paper, the Advanced Encryption Standard (AES) algorithm is utilized to change over content from its unique structure (plain text) to incoherent structure (figure content) then figure content is concealed in the picture by Least Significant Bit (LSB). The content is encoded with key 128 bit. Indeed, even with an alternate arrangement sorts and diverse sizes of the chosen pictures is utilized to cover up scrambled content. The experiments demonstrate the span of the Spread picture influences the nature of the stego picture; in these manner quality, PSNR increments and the

MSE decreases with the expansive size of spread picture. The outcomes demonstrate the suggested technique is powerful and add Security levels for information transfer.

Due to the increasing amount of data being exchanged over the internet, more secure messages is required. The vast number of clients requires security, especially since the numerous usage of PCs, networks, and the Internet with its worldwide availability. There are many techniques is used to protect the user data. One such technique is cryptography that is the art of conversion the readable original data from a state to evident babble. In the encryption process, confidential data is encrypted into the cypher form, which it became very hard (even impossible for unauthorized to recognize or read data. The second technique is Steganography, where the data is covered up inside any sight and sound substance like the picture, sound, video so that, just the authenticated individual knows the message content. In this paper, cryptography and steganography are utilised to implement a secure host for data transmission. AES-128 bit key is used to encrypt the message and LSB is used to encode the encrypted text into the image to be sent.

This section presents a review of the secure data transmission system.

[1].Sharma et al. proposed a framework that combines cryptography with steganography to improve the security of data. The message byte is XORing with a random key created by a pseudo-random generator then embeds plaintext in an image file. Sharma et al. proposed the BLOWFISH algorithm, which is utilized to encrypt the secret image and the LSB method for concealing the encrypted image in a video.

[2].Vijay and Swati implemented the Data Encryption Standard (DES) algorithm, which is used to encrypt the message and MD5 algorithm is used to compute the message digest, which is used to check the integrity of a message. Then it is hidden into image file. Singh and Attri suggested the dual layer of security to the data, the Least Significant Bit, in which, is used to encode data and encrypt the data using Advanced Encryption Standard Algorithm. Discrete Cosine. Transform is applied on the RGB layer (cover image).

[3].Varghese explained that the secret image is concealed into RGB layers.

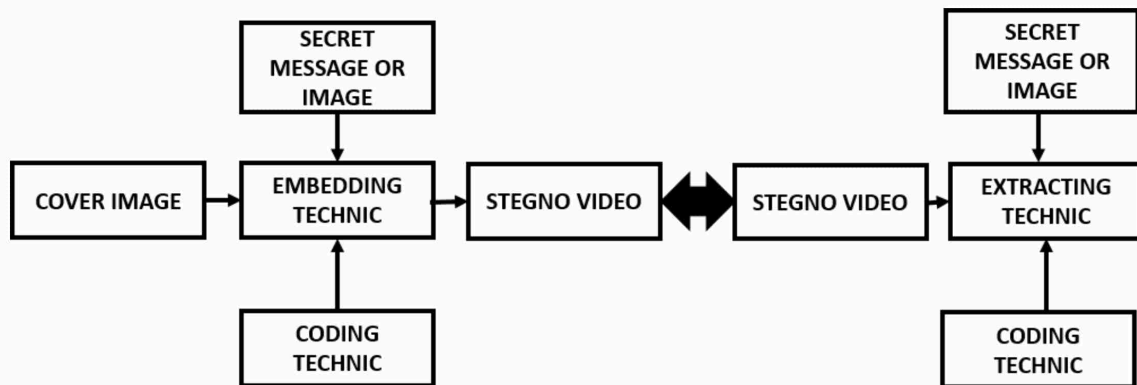
EXISTING SYSYTEM:

The scheme extracts binary bit planes from the plain-image and performs bit level permutation and confusion, which are controlled by a pseudo-random sequence and a random image generated by the Logistic map, respectively. As the rows and columns of the four LSBPs are permuted with the same pseudo-random sequence and the encryption process does not involve the statistical characteristics of the plain-image, In this paper to develop a tool that can encrypt/decrypt text using bit plane extraction.

Proposed system:

- This proposed approach is based on Image Steganography for hiding message in the Image, retrieving the hidden Image from the Image using LSB (Least Significant Bit) modification technique.
- LSB steganography techniques widely used technique in steganography and To examine under what conditions an observer can differentiate between stego-images and cover-images.
- LSB coding technique has the advantage of low computational complexity and very high watermark channel bit rate.
- To decrypt the stego images without loosing the Actual Data

Block diagram:



❏ INTRODUCTION

.1 STEGANOGRAPHY VS. CRYPTOGRAPHY AND WATERMARKING

Steganography is the process of hiding data in cover objects, the name comes from the Greek words “steganos” which means “covered” and “graphia” which means “writing” [1]. The aim of steganography is to hide the fact that communication happened at all which is different from the aim of cryptography. If communication happens encrypted the fact of communication is not hidden.

Watermarking uses similar techniques as steganography but with a different intention, a watermarked video could contain a hidden key which identifies the special copy of the video file. This would allow us to trace copies of the video. But in watermarking the absence of knowledge that a watermark is being used is not crucial, more importantly the watermark shall be robust against attacks like re-encoding and video filters. A steganographic system is broken as soon as the fact that communication took place is revealed, no matter if the messages are deciphered or not.

Steganography is a very old idea, in ancient times one method of data hiding was to write a message on the bald head of a slave and then to send him to the receiver once the hair has grown again. There are more recent methods especially in the area of online multimedia content which is becoming more present. Cisco Systems predicted in its VNI Forecast (February 2012) that by 2015 62% of all online traffic will be video, increasing from 40% in 2010¹. German authorities have detected and successfully extracted 141 documents containing detailed plans for future attacks embedded into a pornographic video found on an suspected al-Qaeda operative who was arrested in May 2011 in Berlin²³. It is important to investigate if video can be used for steganography and how steganography in videos can be detected.

.2 STEGANOGRAPHIC MEDIA

In this project we try to assist Alice and Bob who are imprisoned and trying to escape. They can exchange videos but all their communication is controlled by warden Eve who observes their communication.

There are three different approaches to digital steganography for Alice and Bob to use [1]: Steganography by cover ...

selection There is a fixed database of cover objects each assigned with a secret meaning.

Alice needs to pick the correct cover to transmit in order to convey a message.

synthesis Alice creates a cover with special properties to transmit a hidden meaning. This can be colours of clothes in a picture or the length of a text rather than the content.

modification Given a cover object Alice modifies it slightly so that the change is not at all or at least hardly noticeable. For example Alice could apply different video filters until the modified video contains the desired message. Or Alice can change the syntax elements inside the video file slightly so that Bob can extract a message from them.

Steganography by cover selection is not interesting to Alice and Bob, they would both have to synchronize a large collection of video files and agree on a meaning of each individual video. Cover synthesis is hard to do on video since it demands to re-shoot the same video multiple times. Therefore Alice and Bob will use cover modification. In this project we will discuss different approaches to how elements within a video bitstream can be tweaked to secretly convey a message and also investigate how detectable these modifications are.

Before we can start we need to introduce the h.264 video codec which is commonly used online at the time of writing and serves as cover object for our experiments. enCOding and DECOding

A video is a collection of images called frames each taken at a constant time interval. A CODEC is a pair of an encoder and a decoder which allow to convert a raw video into a bitstream (encoding) and also recover the raw video from the bitstream (decoding). As for many other codecs the h.264 standard only specifies the decoding process [2], any program that creates a valid h.264 bitstream can be called an encoder. Different encoders may yield different bitstreams but the decoded video is unique.



Figure 1: Video coding workflow. We modify only the quantized coefficients in the compressed video stream when simulating embedding.

When encoding a frame using h.264 first a prediction is made, this can either be an intra- or an inter prediction. Intra prediction uses data from the same frame that already has been encoded and inter prediction uses past and/or future frame(s). The prediction is subtracted from the pixel data, the residual gets transformed using an integer approximation to the Discrete Cosine Transform (DCT) and quantized. The quantized coefficients then get entropy-coded into the final bitstream.

3 PRACTICAL APPROACH

In this project we will simulate steganography by modification of quantised coefficients in an h.264 video stream and compare different approaches with respect to their detectability. The embedding simulation will happen on a set of DVDs that are transcoded to h.264 with the x264⁴ encoder and using ffmpeg⁵ to read the resulting video bitstreams.

A set of features will be proposed to represent a section of video as a high dimensional feature vector. The detectability of different embedding schemes with different parameters will be measured as distortion in these features using the *Kullback-Leibler Divergence (KL-D)* and *Maximum Mean Discrepancy (MMD)*. The distortion measures are described in detail in section 4.5.

THE H.264 ADVANCED VIDEO CODING (AVC) STANDARD

H.264 is a block-based video codec, the pipeline mentioned in section 2.3 is traversed for each 16x16 pixel macroblock in a video frame. Within each macroblock there are different channels representing luma and chroma components which are described in the following section.

3.1 COLOUR SPACES

3.1.1 RGB

RGB image or video data stores the relative proportion of red, green and blue colour of each pixel. It can be captured by arrays of different sensors that only capture one colour and displayed by illuminating the red, green and blue part of each pixel in the display accordingly. Figure 2 shows the RGB-decomposition of a Foreman⁶-snapshot.

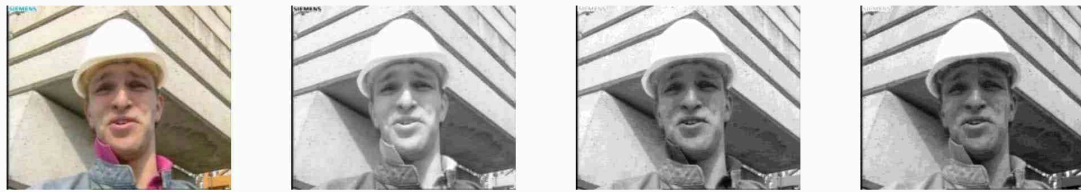


Figure 2: Original frame and RGB components

3.1.2 YCrCb

The human eye is more sensitive to luminance rather than to colour. That's the reason for most video codecs to convert the input video data into a different representation,

⁴[http://www.videolan.org/developers/x264.ht](http://www.videolan.org/developers/x264.html)

ml ⁵<http://www.ffmpeg.org/>

⁶A video commonly used for encoder testing (http://media.xiph.org/video/derf/y4m/foreman_cif.y4m).

YCrCb. The aim is to separate luminance and colour so that colour information can be encoded in lower resolution.

The luminance component can be calculated as follows:

$$Y = k_r R + k_g G + k_b B \quad (1)$$

The constants k_r , k_g , k_b are given by the h.264 specification [3].

Now only the difference from Y gets stored in the colour information:

$$\begin{aligned} Cr &= R - Y \\ Cb &= B - Y \\ Cg &= G - Y \end{aligned} \quad (2)$$

Cg can be expressed as a linear sum of the other variables [2], so only Y , Cr and Cb

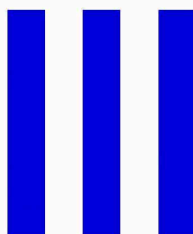
need to be stored. Figure 3 shows the decomposition of the same snapshot as above.



Figure 3: Original frame and YCrCb components

3.1.3 SAMPLING PATTERNS

There are three sampling patterns available in H.264 that allow to reduce the amount of stored chroma information: 4:4:4, 4:2:2 and 4:2:0. They are illustrated in Figure 4. 4:2:0 is most common since the human eye usually



does not notice any difference to the original image even though the number of chroma samples is reduced to a quarter.

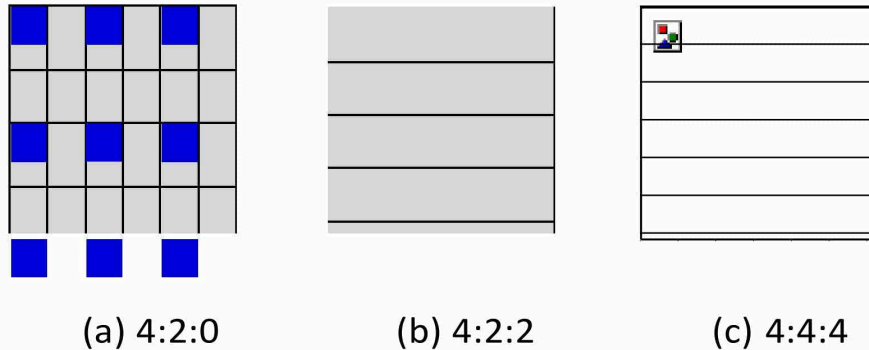


Figure 4: Different sampling patterns. Each cell represents a luma sample and coloured cells represent chroma samples.

There are two chroma channels, thus in 4:2:0 sampling half as many chroma as luma samples are being stored.

3.2 PREDICTION

A frame is divided into slices and each slice is divided into 16x16 pixel macroblocks. A macroblock can be further partitioned down to block size 4x4.

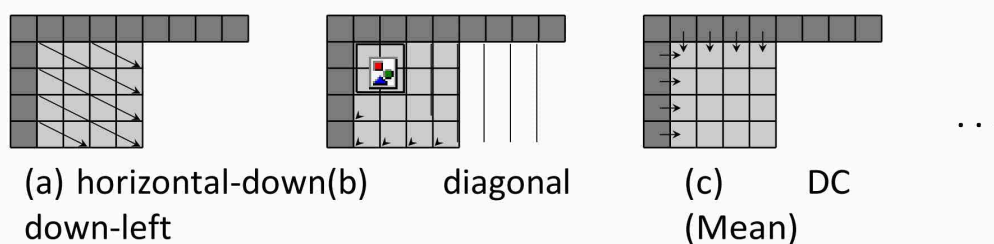


Figure 5: A few different Intra Prediction Modes. Dark grey Pixels are already decoded, the light grey ones need to be predicted from them.

There are three different kinds of frames, I, P and B frames, similarly there are I, P and B macroblocks. An I macroblock makes a prediction based on the pixels to its left and top in the same frame (intra prediction),

we can regard an I Frame as a picture with a compression technique that is similar to JPEG. P macroblocks use a portion of a past frame as prediction where the so-called motion vector usually has half pixel precision to capture fracture pixel movements. B macroblocks come with two motion vectors that can point in both future and past frames in order to make a more accurate prediction.

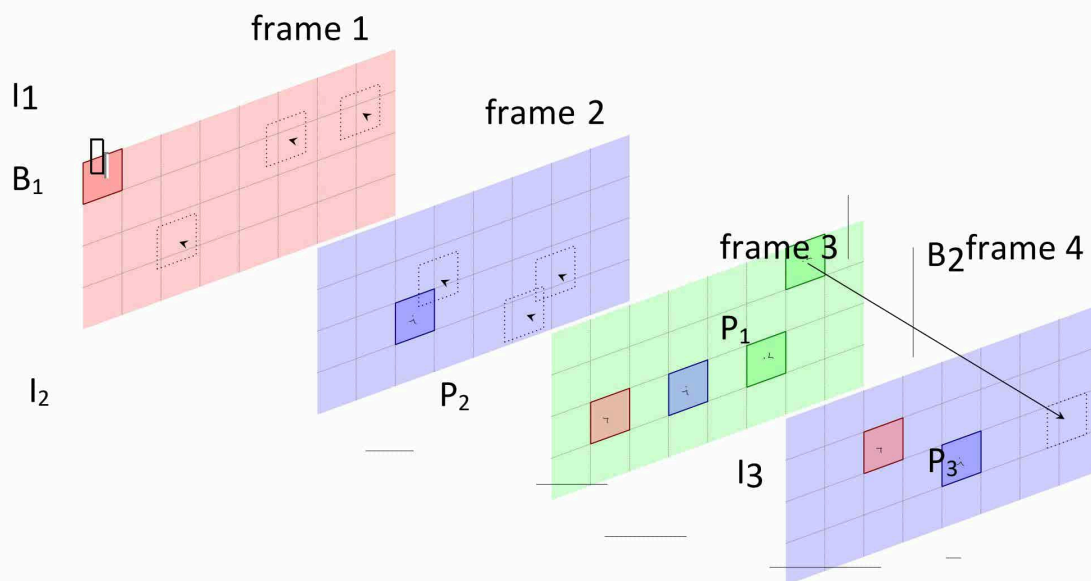


Figure 6: Inter Prediction. Showing I-, P- and B-Macroblocks.

The first frame in a video has to be an I frame since there are no possible reference frames present. There are two kinds of I frames in h.264, regular I frames and IDR (Instantaneous Decode Refresh) frames. IDR frames contain only I macroblocks as well and also they don't allow references by future P and B frames behind them. IDR frames partition the video into sections that can be decoded independently allowing the user to jump to a future position in the video stream without having to decode all frames in between. The first frame is an IDR frame.

The P macroblocks in figure 6 use frame 1 and frame 2 as reference frames. Frame 3 has a future reference on frame 4, hence frame 4 has to be decoded before frame 3. The macroblock P₃ is not allowed to reference frame 3 even though frame 3 is displayed before frame 4. P macroblocks in later frames are allowed to reference frame 3, there is no distinction between reference frame types.

P macroblocks use the point they reference to as prediction where coordinates have half- or quarter-pixel precision. B macroblocks combine both reference frames to a pre- diction using a weighted average or other combination methods. Prediction is a key part of each video codec.

Frames can also be partitioned into horizontal slices, there are I, P and B slices. Slices are separate units that can be decoded independently allowing the decoder to process a frame on parallel hardware. Blu rays for example have to contain at least 4 slices to ensure playability by relatively cheap hardware in blu ray players.

In order to visualize how an encoder might use the different frame types and how much they differ in size, Figure 7 shows the output of x264 when encoding foreman with QP

= 20. I frames are red, P frames blue and B frames green.

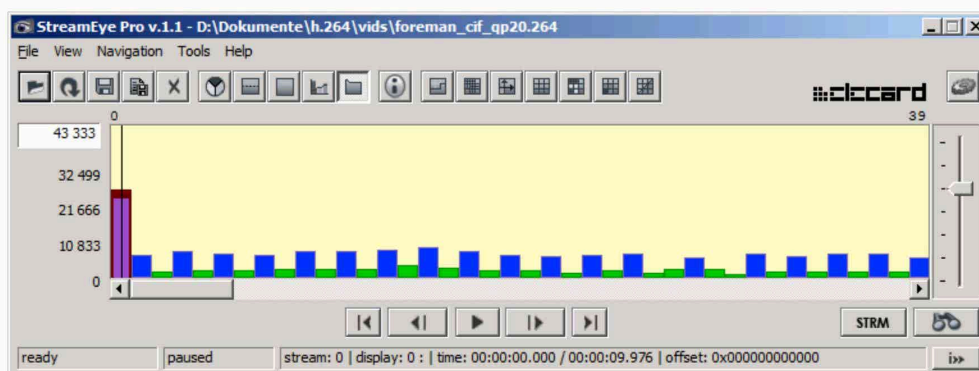


Figure 7: Example x264 output stream

3.3 TRANSFORMATION, QUANTIZATION AND ENTROPY CODING

H.264 supports two different luma transforms: the default and Intra 16x16 (AC/DC) transforms which are illustrated in figure 8 and figure 9. Chroma uses only the AC/DC transform, see figure 10.

The Quantisation Parameter (QP) is constant for each 4x4 block and the actual step size s is exponential in QP, it doubles as QP increases by 6 [2]. On input coefficient x we

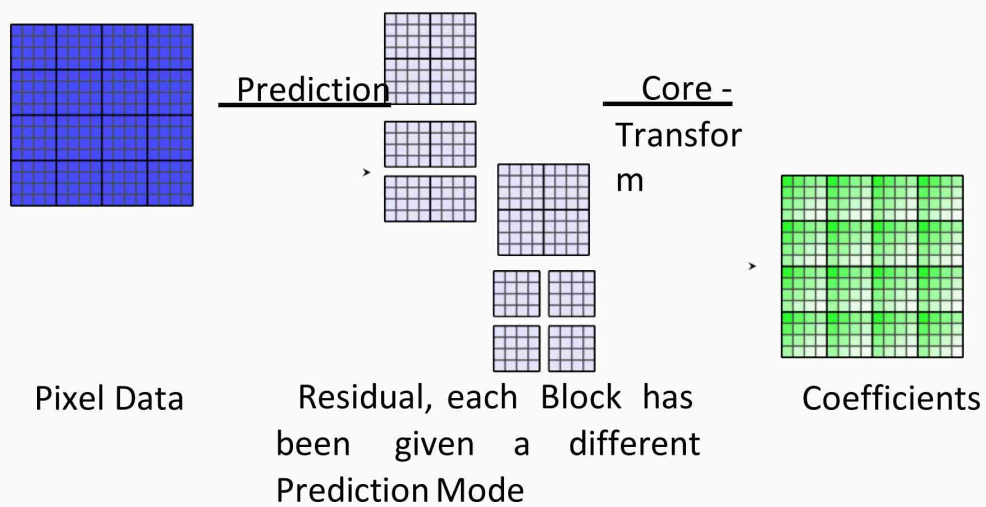
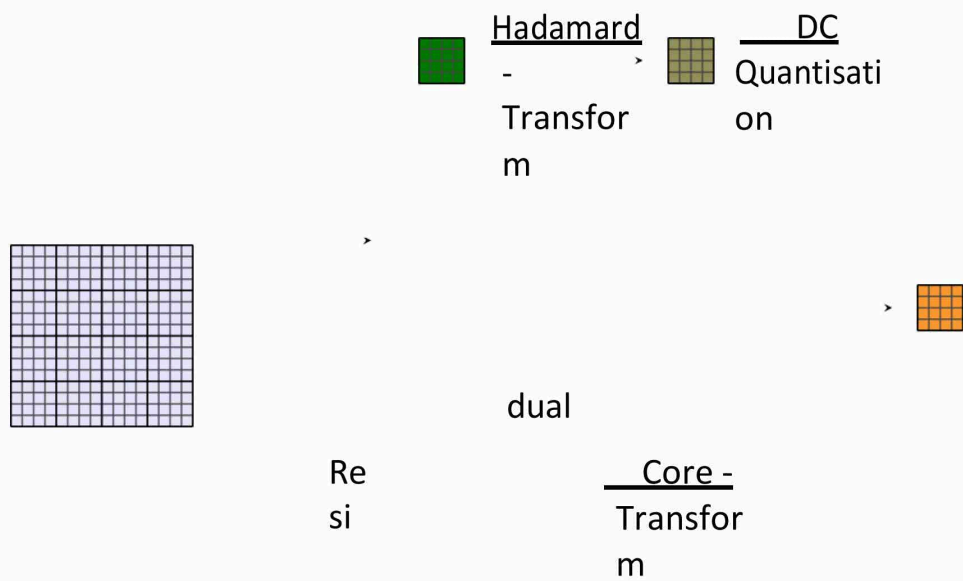


Figure 8: Luma default Transform. Lower intensity indicates lower magnitude of coefficients. Quantisation is omitted here.



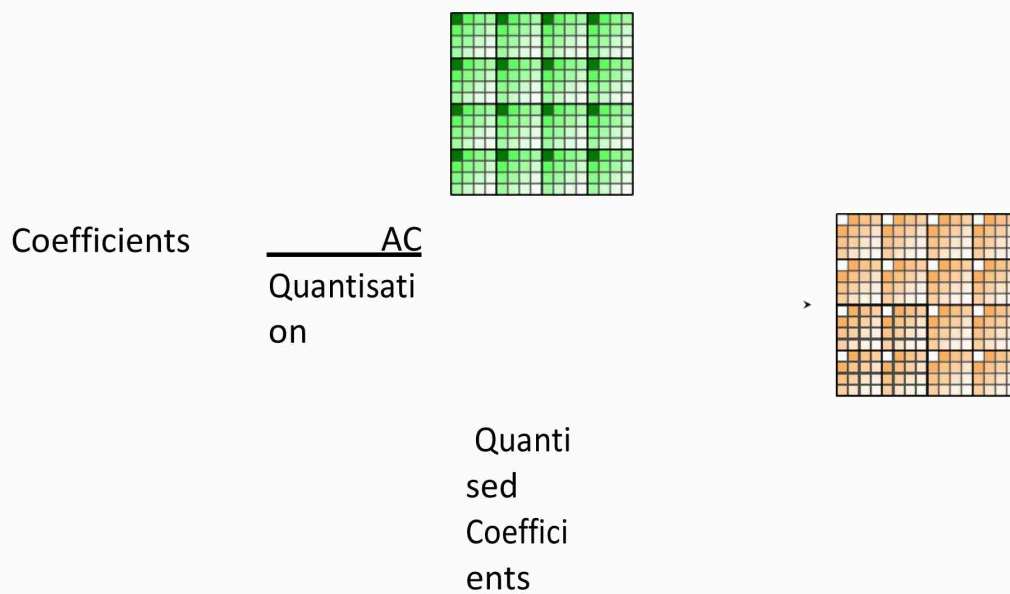


Figure 9: Luma Intra 16x16 Transform. Partitions are not allowed here.

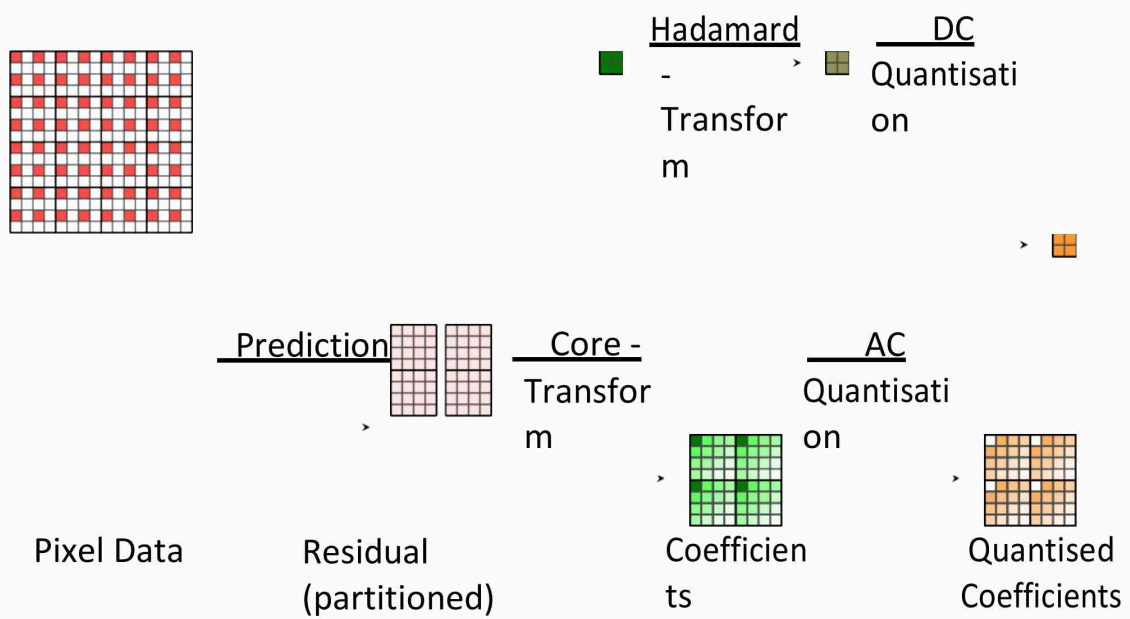


Figure 10: Chroma Transform.

compute quantized coefficient y by $y = \text{round}(\frac{x}{s})$. When decoding the video x will be approximated as $x' = y \cdot s$. Pixel values are integers, h.264 uses an integer approximation to the DCT as core transform, the Hadamard transform is an integer transform as well, quantization in h.264 takes integers as input and produces integers as output.

Each video file has a *profile* defining which encoding techniques need to be supported by the decoder to play the video. Higher profiles support more advanced features such as the 8x8 DCT transform or re-ordering of macroblocks within a frame. In this project we will use the main profile only since it supports all functionality that we require.

There are two different entropy coding modes, CAVLC (Context Adaptive Variable- Length Coding) and CABAC (Context Adaptive Binary Arithmetic Coding). Entropy coding is a lossless process whereas the quantisation is lossy, the decoded frame will differ from the original one. In CAVLC, the coefficients are scanned in the zig-zag pattern which is illustrated in figure 11

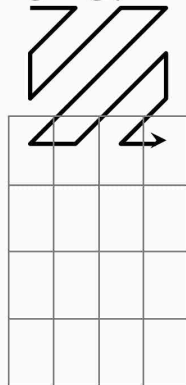


Figure 11: Zig-zag scan order

2 STEGANOGRAPHIC SECURITY

2.1 STEGANOGRAPHIC CHANNELS

A steganographic channel (or steganographic system / stegosystem) fully describes how two parties called Alice and Bob communicate secretly. It consists basically of the following elements:

- cover source

- embedding and extraction methods
 - source of stego keys (similar to cryptography)
 - source of messages
 - data exchange channel, observed by warden Eve
- The part of Figure 12 can be defined formally [1]:

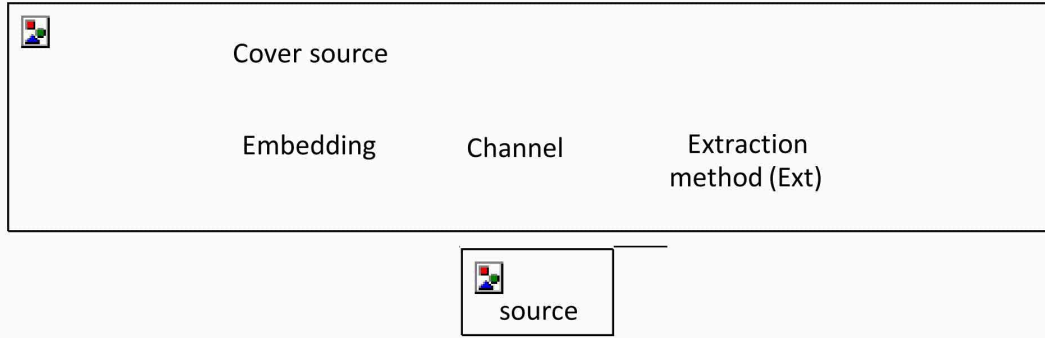


Figure 12: Steganographic channel

C set of cover objects $x \in C$, (3)

$K(x)$ set of stego-keys for x , (4)

$M(x)$ set of messages for x , (5)

$Emb : C \times K \times M \rightarrow C$, (6)

$Ext : C \times K \rightarrow M$, (7)

such that $Ext(Emb(x, k, m), k) = m$. (8)

The exchanged data is controlled by warden Eve who we assume to be a passive observer. Alice and Bob require a stegosystem that achieves maximum payload within a certain detectability. We will now present different embedding methods and approaches to estimate their performance.

2 EMBEDDING METHODS

All embedding methods discussed in this project share the extraction method which is to look at the least significant bits (LSB) in a certain subset of the encoded quantized transform coefficients. There are different

approaches to give the LSB the desired value which will be described in the next sections.

In the examples below we will assume that the hiding probability is 1 and that every coefficient is used for hiding. In practical steganography we would use a much smaller hiding probability and avoid hiding in zeros and ones since zeros receive special treatment in the entropy coding process, changing the number of zeros may have impact on the video file size and cause a desynchronization of codes. In CAVLC the number of non-zero coefficients in surrounding blocks determines the bitstring table that is used to store a coefficient, changing the number of non-zero coefficients may result in the decoder selecting the wrong table.

Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Why is Python so popular?

For the past few years, Python has witnessed incredible growth in the programming world. The key fact that drives Python's booming popularity is: it is an open-source language. Anyone can download and

use it to develop an application. It can be used on a wide variety of platforms like macOS, Microsoft Windows, etc. Also, it has been used for large scale projects at Google, YouTube, and many others.

Apart from the key fact, there are other reasons why Python is more popular. Here we go with them:

1. Easy to learn and code

Consistently, Python is ranking as one of the easiest programming languages to learn. This is due to its high reliability and simple syntax which is quite similar to plain text. So, you can write a Python code simply and execute it much faster than the other programming languages. Even, one can change the code base of Python for the quick experiment as it is an interpreted language. Due to the ease of use, Python is even more popular among all kinds of developers.

2. Has a Healthy, Active, and Supportive community

In general, the Programming language may lack documentation and developer support at some point. Python has neither of those issues. It has plenty of documentation, tutorials, and guides to learn.

Additionally, the developer community is active. That is, if someone needs any help or support while using Python, they can get instant support from the supportive community. The active community will ensure you with the developers from the basic skill level to the expert.

3. Support from great Corporate Sponsors

Usually, programming languages will grow faster especially when it has a great corporate sponsor at its back. For instance, Visual Basic and C# by Microsoft, PHP by Facebook, and Java by Oracle and Sun. Likewise, Python Programming language is heavily sponsored by Amazon web services, Facebook, and specifically by Google. Back in 2006, Google adopted python and have used it for many applications and various

platforms since then. Further, Google has created a dedicated portal for Python. This is an added benefit for Python and becomes more trending among the developers.

4. Amazing Libraries and Frameworks

When you work on bigger projects, libraries can help to cut down the initial development cycle and save time. Due to a supportive community and corporate sponsorship, Python has an excellent selection of libraries from SciPy to Django for web development. Furthermore, few libraries include some specific focus like nltk for natural language processing and scikit-learn for machine learning applications. Also, Python has library-like tools to offer cross-platform support to the developers.

Aside from these, other few libraries and frameworks are accessible in Python. Some of them are:

SciPy and NumPy for scientific computing.

Django for server-side Web development.

BeautifulSoup for XML and HTML parsing.

matplotlib for plotting graphical charts.

5. Big Data, cloud computing, and Machine Learning

The use of cloud computing and big data in this enterprise world is one of the main reasons for the popularity of Python. Next to the R language for data science, Python is the most widely used tool. Hence, many data processing workloads in enterprises are powered by Python. Also, Python is mostly used for research and development due to its ease of analyzing and usable data organizing.

6. Versatile, Reliable, and Speed

If you ask any developer, they will heartily agree that Python is reliable, efficient, and faster than other languages. Python language can be used in any kind of environment without facing any issue or loss. Also, it is versatile to use. Thus, you can use Python in various environments like web development, mobile applications, and much more. This makes it more attractive to use Python.

Why do we choose python for this project

We guess you might have a glimpse of why Python is so popular compared to other languages. Now, it's time to know why you should learn Python and the importance of learning Python programming. Below are some reasons why you should learn Python.

1. Versatile and multiple programming paradigms

Like other programming languages, Python supports multiple programming paradigms like object-oriented programming and structured programming. Also, the features of Python will support various concepts like aspect-oriented and functional programming. As well, Python features automatic memory management and a dynamic type system that help you for developing large and complex software applications. Furthermore, Python has an automatic memory management system to deal with dynamic storage for easy handling.

2. Web testing framework

Python has many in-built testing frameworks that cover the fastest debugging workflow. Also, like Selenium and Splinter, Python has many modules and tools to make things much easier. Well, we all know testing is a tedious task. Python is a booster for testing. Because it supports testing for cross-browser and cross-platform with the testing frameworks like PyTest and Robot. So, if you are a tester, you should go for Python.

3. Simplify complex development

The main goal of Python is to provide simplicity to get rid of complexity. So, Python is quite easier to read, write, and learn than other programming languages. Hence, you can use the Python programming language for developing both web and desktop applications. Further, Python is designed with some special features to facilitate visualization and data analysis. So, you can take the benefit of data analysis to create big data solutions without any effort and extra time.

4. Easy Access

Well, do you wish to access databases easily? Python will do it for you. Python has two strong points that allow the user to customize interfaces like Oracle, MySQL, etc. This allows you to access other databases rapidly. Furthermore, its integration and flexibility will allow it to run with other programming languages like CPython, Jython, etc.

Advantages of Python

Whenever you learn a new language, you must be aware of the advantages of that language. This will help you to get better knowledge to use the language powerfully while building a robust application. Below we have listed the advantages of Python. Make sure to keep your eyes on them.

1. High-level language

Python looks more similar to the human-readable language than other programming languages. This gives the ability to code at a faster rate than other languages. Python can be extended to other programming languages. So, you can code in languages like C or C++.

2. Simple Syntax

When compared to other programming languages. Python's syntax is easy to learn. Both the programmers and non-programmers can begin programming right away.

3. Do more with less code

As we mentioned previously, Python is easy to learn and write, you can do more with less coding. That is, you can construct a prototype and test out your ideas faster in Python. So, using Python, you can save valuable time.

4. User-friendly Data Structures

Python is designed with an in-built dictionary data structure that can be used to build fast runtime. Further, Python doesn't know the variable type, it automatically assigns the data type while executing the code. Hence, programmers need not worry about the variables and their data types.

5. Support libraries and third-party modules

Python has a large standard library. It includes areas like string operations, internet protocols, and operating system interfaces. The standard library of Python will reduce the length of the code for writing significantly. Additionally, the Python package has numerous third-party modules. This makes Python more capable while interacting with other languages.

6. Object-Oriented Programming

Python supports both procedural and object-oriented programming paradigms. Object-oriented programming allows you to build data structures that can be reused. This reduces the repetitive work. So, have you ever decided to switch your programming language from

Python? If yes, you have a significant chance to work with object-oriented programming.

7. Productivity

As Python has an object-oriented design, it offers enhanced process control, text processing, and integration. Also, the extensive library and language simplicity will render more productivity than other programming languages.

Libraries we used in python:

1)click

2)PIL

CLICK PACKAGE:

Click is a Python package for creating beautiful command line interfaces in a composable way with as little code as necessary. It's the "Command Line Interface Creation Kit". It's highly configurable but comes with sensible defaults out of the box.

It aims to make the process of writing command line tools quick and fun while also preventing any frustration caused by the inability to implement an intended CLI API.

Click in three points:

arbitrary nesting of commands

automatic help page generation

supports lazy loading of subcommands at runtime

Why Click?

There are so many libraries out there for writing command line utilities; why does Click exist?

This question is easy to answer: because there is not a single command line utility for Python out there which ticks the following boxes:

Is lazily composable without restrictions.

Supports implementation of Unix/POSIX command line conventions.

Supports loading values from environment variables out of the box.

Support for prompting of custom values.

Is fully nestable and composable.

Supports file handling out of the box.

Comes with useful common helpers (getting terminal dimensions, ANSI colors, fetching direct keyboard input, screen clearing, finding config paths, launching apps and editors, etc.).

There are many alternatives to Click; the obvious ones are `optparse` and `argparse` from the standard library. Have a look to see if something else resonates with you.

Click actually implements its own parsing of arguments and does not use `optparse` or `argparse` following the `optparse` parsing behavior. The reason it's not based on `argparse` is that `argparse` does not allow proper nesting of commands by design and has some deficiencies when it comes to POSIX compliant argument handling.

Click is designed to be fun and customizable but not overly flexible. For instance, the customizability of help pages is constrained. This constraint is intentional because Click promises multiple Click instances will continue to function as intended when strung together.

Python Imaging Library:

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

Let's look at a few possible uses of this library.

Image Archives

The Python Imaging Library is ideal for image archival and batch processing applications. You can use the library to create thumbnails, convert between file formats, print images, etc.

The current version identifies and reads a large number of formats. Write support is intentionally restricted to the most commonly used interchange and presentation formats.

Image Display

The current release includes Tk PhotoImage and BitmapImage interfaces, as well as a Windows DIB interface that can be used with PythonWin and other Windows-based toolkits. Many other GUI toolkits come with some kind of PIL support.

For debugging, there's also a `show()` method which saves an image to disk, and calls an external display utility.

Image Processing

The library contains basic image processing functionality, including point operations, filtering with a set of built-in convolution kernels, and colour space conversions.

The library also supports image resizing, rotation and arbitrary affine transforms.

There's a histogram method allowing you to pull some statistics out of an image. This can be used for automatic contrast enhancement, and for global statistical analysis.

Getting ready with mechane:

Before starting implenting the mechane should be installed with some software. As we choose python as our coding language latest version of the python should be downloaded into the mechane.if already installed we should check the version by opening command prompt and changing it into pytgon intrepetter by giving *python* command.

- Import sys
- Print("sys.version")

```
C:\windows\system32\cmd.exe - python
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.version)
3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)]
>>>
```

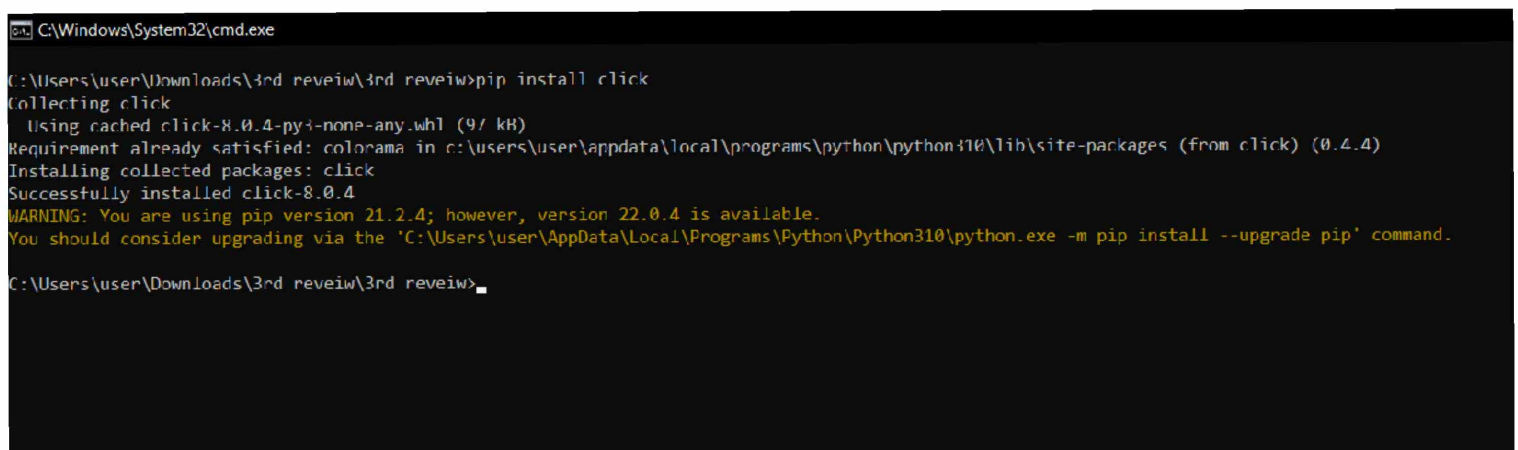
Installing the libraries:

Click:

Once we had the latest version of python. we should install *click* package so that we can give our own commands. install that following these steps.

- Type "*cmd*" in the search bar and hit Enter to open the command line.
- Type "*pip install click*" (without quotes) in the command line and hit Enter again. This installs click for your default Python installation.
- The previous command may not work if you have both Python versions 2 and 3 on your computer. In this case, try "*pip3 install click*" or "*python -m pip install click*".

Wait for the installation to terminate successfully. It is now installed on your Windows machine.



```
C:\Windows\System32\cmd.exe

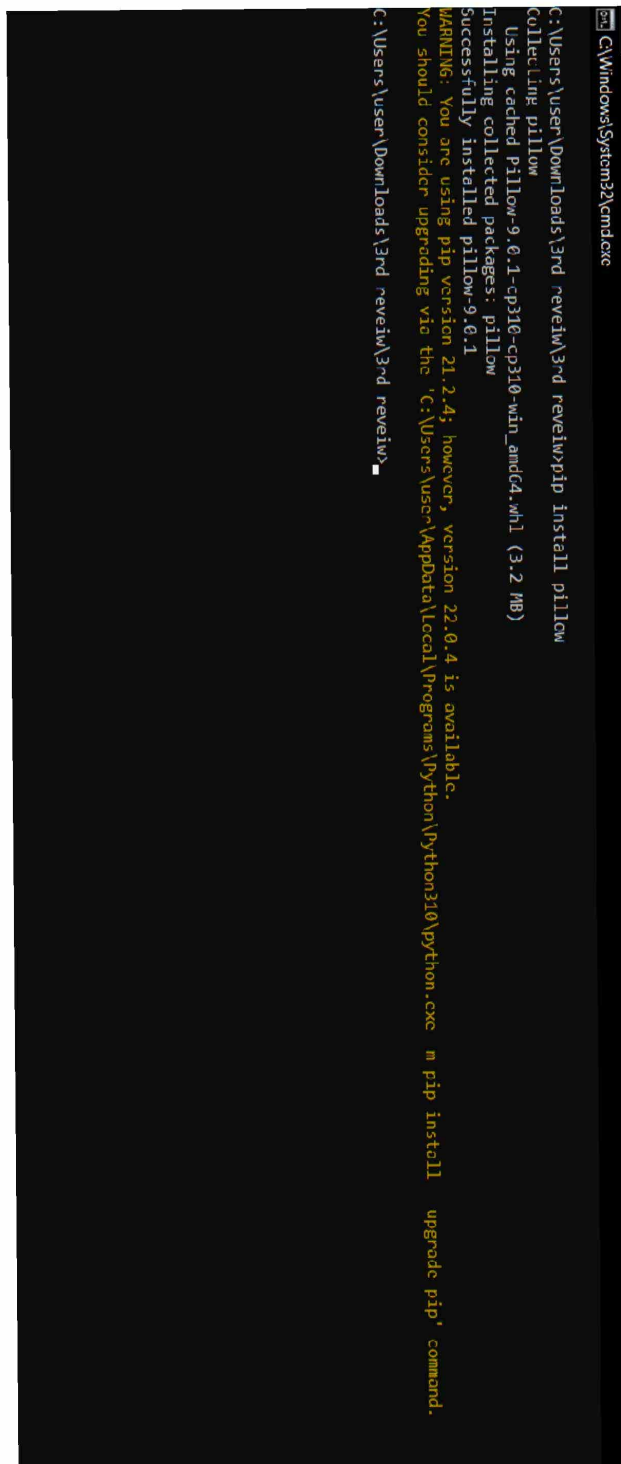
C:\Users\user\Downloads\3rd reveiw\3rd reveiw>pip install click
Collecting click
  Using cached click-8.0.4-py3-none-any.whl (97 kB)
Requirement already satisfied: colorama in c:\users\user\appdata\local\programs\python\python310\lib\site-packages (from click) (0.4.4)
Installing collected packages: click
Successfully installed click-8.0.4
WARNING: You are using pip version 21.2.4; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\user\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.

C:\Users\user\Downloads\3rd reveiw\3rd reveiw>
```

Python image library:

To install the updated Pillow framework on your Windows machine, run the following code in your command line or Powershell:

```
python3 -m pip install --upgrade pip
```



```
C:\Windows\System32\cmd.exe
C:\Users\User\Downloads\3rd reveiw\3rd reveiw>pip install pillow
Collecting pillow
  Using cached pillow-9.0.1-cp310-cp310-win_amd64.whl (3.2 MB)
Installing collected packages: pillow
Successfully installed pillow-9.0.1
WARNING: You are using pip version 21.2.4; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\User\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.
C:\Users\User\Downloads\3rd reveiw\3rd reveiw>
```

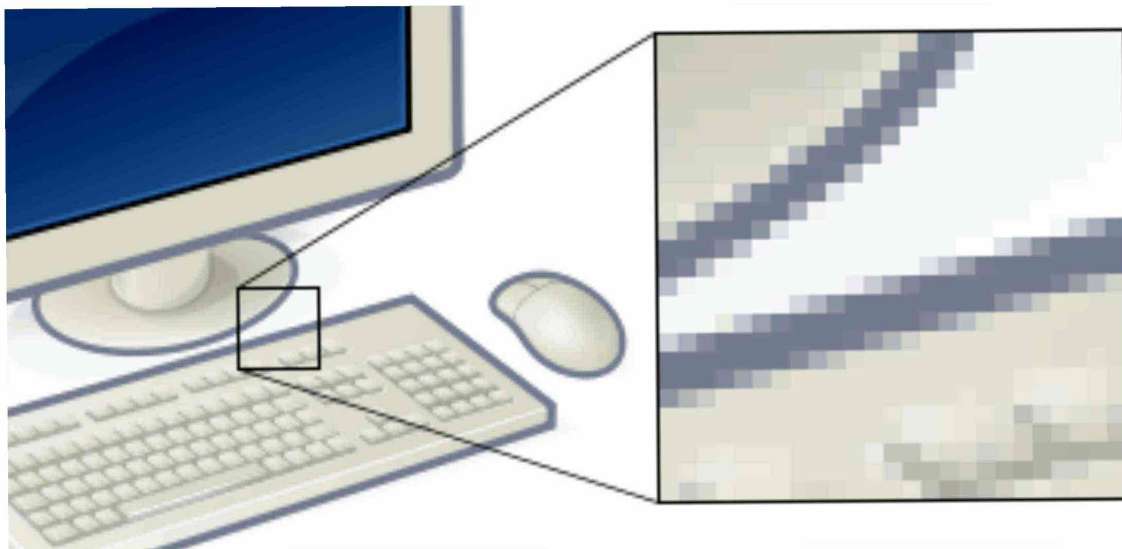
In other words, steganography is more discreet than cryptography when we want to send a secret information. On the other hand, the hidden message is easier to extract.

What is a digital image?

Ok, now that we know the basics of steganography, let's learn some simple image processing concepts.

Before understanding how can we hide an image inside another, 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 matrix (or a two-dimensional array) of pixels which contains a fixed number of rows and columns.



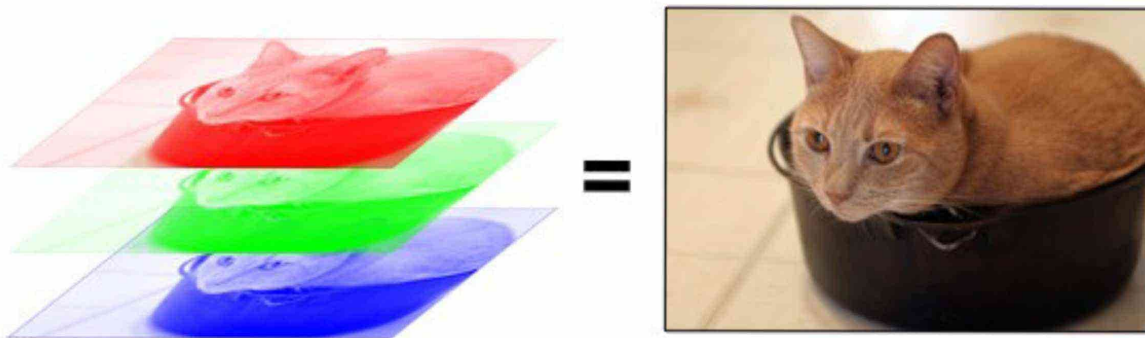
In other words, steganography is more discreet than cryptography when we want to send a secret information. On the other hand, the hidden message is easier to extract.

What is a digital image?

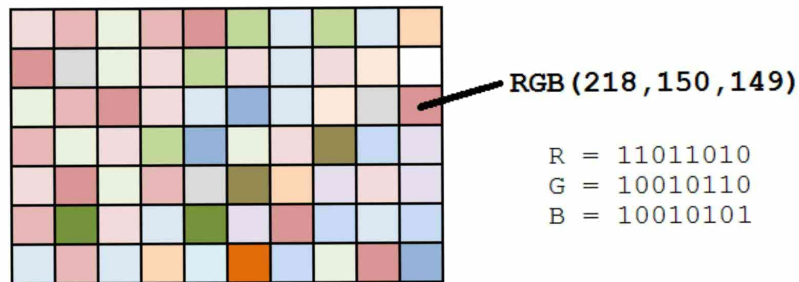
Ok, now that we know the basics of steganography, let's learn some simple image processing concepts.

Before understanding how can we hide an image inside another, 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 matrix (or a two-dimensional array) of pixels which contains a fixed number of rows and columns.



So, each pixel from the image is composed of 3 values (red, green, blue) which are 8-bit values (the range is 0–255).



As we can see in the image above, for each pixel we have three values, which can be represented in binary code (the computer language).

When working with binary codes, we have more significant bits and less significant bits, as you can see in the image below.

	128	64	32	16	8	4	2	1
8 bit binary digit	1	0	1	1	0	0	0	1
	128 + 32 + 16 + 1 = 177							

The leftmost bit is the most significant bit. If we change the leftmost bit it will have a large impact on the final value. For example, if we change the leftmost bit from 1 to 0 (11111111 to 01111111) it will change the decimal value from 255 to 127.

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. For example, if we change the leftmost bit from 1 to 0 (11111111 to 11111110) it will change the decimal value from 255 to 254. Note that the rightmost bit will change only 1 in a range of 256 (it represents less than 1%).

Each pixel has three values (RGB), each RGB value is 8-bit (it means we can store 8 binary values) and the rightmost bits are less significant. So, if we change the rightmost bits it will have a small visual impact on the final image. This is the steganography key to hide an image inside another. Change the less significant bits from an image and include the most significant bits from the other image.

Pixel from Image 1

R(11001010)
G(00100110)
B(11101110)

Pixel from Image 2

R(00001010)
G(11000001)
B(11111110)

New pixel from the new Image

R(11000000)
G(00101100)
B(11101111)

The KL-D has shown to be not computable in practice, see section 5.6 for details.

🔖 IMPLEMENTATION

There is more than one step involved in the practical portion of this project:

- Extracting a pixel map for both images.
- Forming a new blank image with black colour
- Layering the two images without disturbing the first image

The secret data can be data of any format like text or even a file. In a nutshell, the main motive of steganography is to hide the intended information within any file, usually an image, audio, or video, without actually changing the external appearance of the file, i.e. it should look the same as before.

In this blog, we will be focussing on learning image-based steganography, i.e. hiding secret data in an image.

But before diving a little deeper into it, let's look at what an image comprises of.

- 🔖 Pixels are the building blocks of an image.
- 🔖 Every pixel contains three values: (red, green, blue) also known as RGB values.
- 🔖 Every RGB value ranges from 0 to 255.

This much information is enough to get started.

Now, let's look at how we can encode and decode data into our image.

Encoding:

There are a lot of algorithms that can be used to encode data into the image, and in fact, you can also make one yourself. The one being used in this blog is easy to understand and implement, as well.

The algorithm is as follows:

1. For each character in the data, its ASCII value is taken and converted

into 8-bit binary [1].

2. Three pixels are read at a time having a total of $3 \times 3 = 9$ 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.

Repeat this process until all the data is encoded into the image.

Example

Suppose the message to be hidden is 'Hii'.

The message is of three bytes, therefore, the pixels required to encode the data are $3 \times 3 = 9$. Consider a 4×3 image with a total of 12 pixels, which are sufficient to encode the given data.

**[(27, 64, 164), (248, 244, 194), (174, 246, 250), (149, 95, 232),
(188, 156, 169), (71, 167, 127), (132, 173, 97), (113, 69, 206),
(255, 29, 213), (53, 153, 220), (246, 225, 229), (142, 82, 175)]**

Step 1

The ASCII value of H is 72, whose binary equivalent is 01001000.

Step 2

Read the first three pixels.

(27, 64, 164), (248, 244, 194), (174, 246, 250)

Step 3

Now, change the pixel value to odd for 1 and even for 0 as in the binary equivalent of data.

For example, the first binary digit is 0 and the first RGB value is 27, it needs

to be converted to even, which implies 26.

Similarly, 64 gets converted to 63 because the next binary digit is 1 so the RGB value should be made odd.

So, the modified pixels are:

(26, 63, 164), (248, 243, 194), (174, 246, 250)

Step 4

Since we have to encode more data, the last value should be even. Similarly, i can be encoded in this image.

While making the pixel values odd/even by doing +1 or -1, you should take care of binary conditions. I.e., the pixel value should be more than or equal to 0 and less than or equal to 255.

The new image will look like:

**[(26, 63, 164), (248, 243, 194), (174, 246, 250), (148, 95, 231),
(188, 155, 168), (70, 167, 126), (132, 173, 97), (112, 69, 206),
(254, 29, 213), (53, 153, 220), (246, 225, 229), (142, 82, 175)]**

While dealing with the images the principle will be:

- Let $\text{img1}[i][j][l]$ and $\text{img2}[i][j][l]$ be some pixel value of each image.
- Let $v1$ be 8 bits binary representation of $\text{img1}[i][j][l]$ and $v2$ be 8 bits binary representation of $\text{img2}[i][j][l]$.
- Therefore, $v3 = v1[:4] + v2[:4]$, where, $v3$ is the first 4 bits of $v1$ and $v2$. Then we assign $\text{img1}[i][j][l]$ to $v3$.



+



1. -> Here I choose python programming language to code the required process.
2. -> I installed click package to write my own commands
3. -> I imported image package from the PIL library which stands for python image library.
4. For example, suppose we have to hide img2 in img1, where both img1 and img2 are numpy nd array 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 $\text{img}[i][j][l]$ be the pixel value at location (i, j) and of channel l where i varies from 0 to width and j varies from 0 to height and l varies from 0 to 2.

The screenshot shows a Visual Studio Code editor window titled 'steganography.py - Visual Studio Code'. The editor is in 'Restricted Mode' as indicated by a blue banner at the top. The file explorer on the left shows a project structure with a folder named 'steganography.py' and a file named 'steganography.py'. The search bar is empty. The main editor area displays the following Python code:

```

1  #!/usr/bin/env python
2
3  import click
4  from PIL import Image
5
6
7  class Steganography:
8
9      @staticmethod
10     def __int_to_bin(rgb):
11         """Convert an integer tuple to a binary (string) tuple.
12
13         :param rgb: An integer tuple (e.g. (200, 110, 96))
14         :return: A string tuple (e.g. ("00101010", "1101011", "00010110"))
15         """
16         r, g, b = rgb
17         return (f'{r:08b}',
18                 f'{g:08b}',
19                 f'{b:08b}')
20
21     @staticmethod
22     def __bin_to_int(rgb):
23         """Convert a binary (string) tuple to an integer tuple.
24
25         :param rgb: A string tuple (e.g. ("00101010", "1101011", "00010110"))
26         :return: Return an int tuple (e.g. (200, 110, 96))
27         """
28         r, g, b = rgb
29         return (int(r, 2),
30                 int(g, 2),
31                 int(b, 2))
32
33     @staticmethod
34     def __merge_rgb(rgb1, rgb2):
35         """Merge two RGB tuples.
36

```

The status bar at the bottom shows 'Restricted Mode', '0.0.0', 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'L', 'Python', and the date '2017 03 03 2022'.

```
41
42 r1, g1, b1 = rgb1
43 r2, g2, b2 = rgb2
44 rgb = (r1:a1 + r2:a1,
45        g1:a1 + g2:a1,
46        b1:a1 + b2:a1)
47
48 return rgb
```

```
@staticmethod
def merge(img1, img2):
```

```
    """Merge two images. The second one will be merged into the first one.
```

```
    :param img1: First image
    :param img2: Second image
    :return: A new merged image.
    """
```

```
    # Check the images dimensions
```

```
    if img2.size[0] > img1.size[0] or img2.size[1] > img1.size[1]:
        raise ValueError("Image 2 should not be larger than Image 1")
```

```
    # Get the pixel map of the two images
```

```
    pixel_map1 = img1.load()
    pixel_map2 = img2.load()
```

```
    # Create a new image that will be outputted
    new_image = Image.new(img1.mode, img1.size)
```

```
    pixels_new = new_image.load()
```

```
    for i in range(img1.size[0]):
```

```
        for j in range(img1.size[1]):
```

```
            rgb1 = steganography._int_to_bin(pixel_map1[i, j])
```

```
            # Use a black pixel as default
```

```
            rgb2 = steganography._int_to_bin(0, 0, 0)
```

When we run our images through this piece of code it will generate a new png format image which looks exactly like image one.



+



The output image will be:



- After successfully merging two image now comes to the key part of the project that is decrypting.it follows this process
- **Decoding:**
- For decoding, we shall try to find how to reverse the previous algorithm that we used to encode data.
- Again, 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.
- For the first eight values, if the value is odd, then the binary bit is 1, otherwise it is 0.
- The bits are concatenated to a string, and with every three pixels, we get a byte of secret data, which means one character.
- Now, if the ninth value is even then we keep reading pixels three at a time, or otherwise, we stop.
- **For example**
- Let's start reading three pixels at a time.
- Consider our previously encoded image.

```
[(26, 63, 164), (248, 243, 194), (174, 246, 250), (148, 95, 231),
(188, 155, 168), (70, 167, 126), (132, 173, 97), (112, 69, 206),
(254, 29, 213), (53, 153, 220), (246, 225, 229), (142, 82, 175)]
```

- **Step 1**
- We first read the three pixels:
- [(26, 63, 164), (248, 243, 194), (174, 246, 250)]
- **Step 2**
- Reading the first value: 26, which is even, therefore the binary bit is 0. Similarly, for 63, the binary bit is 1 and for 164 it is 0. This process continues until the eighth RGB value.
- **Step 3**
- We finally get the binary value: 01001000 after concatenating all individual binary values. The final binary data corresponds to decimal value 72, and in ASCII, it represents the character H.
- **Step 4**

- Since the ninth value is even, we repeat the above steps. We stop when the ninth value encountered is odd.
- As a result, we get our original message back which was Hii

In case of images :

- Let $\text{img}[i][j][l]$ be the pixel value of the image. Let $v1$ be 8 bits binary representation of $\text{img}[i][j][l]$. Let $v2 = v1[:4] + 4$ random bits and $v3 = v1[4:] + 4$ random bits. Then we assign $\text{img1}[i][j][l]$ to $v2$ and $\text{img2}[i][j][l]$ to $v3$.
- Take the encrypted image as in input then it read the first four digits of an bit tuple .
- Creates a new image and form the image with those bits.

```

95
96 # load the pixel map
97 pixel_map = img.load()
98
99 # Create the new image and load the pixel map
100 new_image = Image.new(img.mode, img.size)
101 pixels_new = new_image.load()
102
103 # tuple used to store the image original size
104 original_size = img.size
105
106 for i in range(img.size[0]):
107     for j in range(img.size[1]):
108         # Get the RGB (as a string tuple) from the current pixel
109         r, g, b = Steganography.__int_to_bin(pixel_map[i, j])
110
111         # Extract the last 4 bits (corresponding to the hidden image)
112         # Concatenate 4 zero bits because we are working with a bit
113         rgb = (r[-4:] + '0000',
114              g[-4:] + '0000',
115              b[-4:] + '0000')
116
117         # Convert it to an integer tuple
118         pixels_new[i, j] = Steganography.__bin_to_int(rgb)
119
120         # If this is a 'valid' position, store it
121         # as the last valid position
122         if pixels_new[i, j] != (0, 0, 0):
123             original_size = (i + 1, j + 1)
124
125 # Crop the image based on the 'valid' pixels
126 new_image = new_image.crop((0, 0, original_size[0], original_size[1]))
127
128 return new_image
129
130

```

steeganography.py x READ/Write

C:\Users> python3 develop > and reverse > steeganography.py

124 # Crop the image based on the 'value' property

125 new_image = new_image.crop((0, 0, original_size[0], original_size[1]))

126 return new_image

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

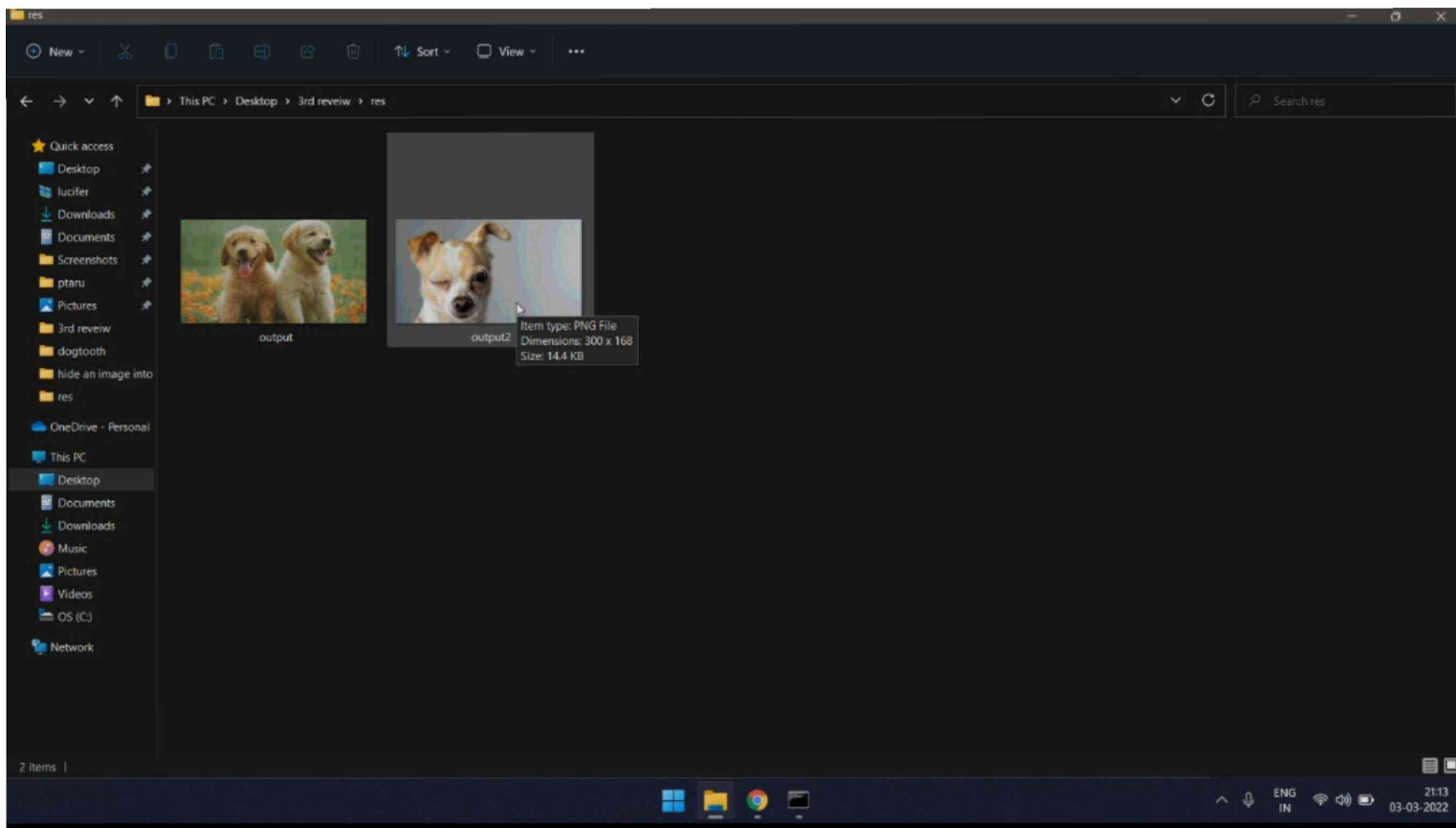
439

440

441

442

After giving stegno image as input to this code .



The ouptput image will be obtained at the same folder.

Result:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22557.1]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ptaru\Desktop\3rd reveiw>python steganography.py
Usage: steganography.py [OPTIONS] COMMAND [ARGS]...

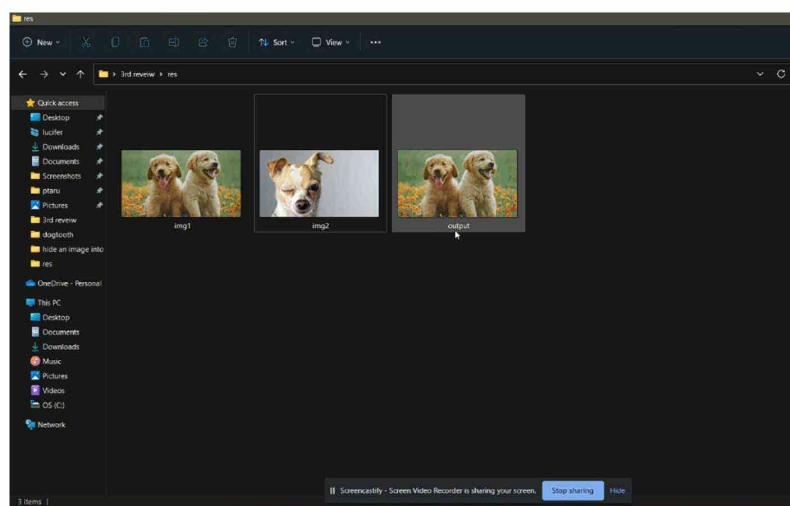
Options:
  --help Show this message and exit.

Commands:
  merge
  unmerge

C:\Users\ptaru\Desktop\3rd reveiw>python steganography.py merge --img1=res/img1.jpg --img2=res/img2.jpg --output=res/output.png
C:\Users\ptaru\Desktop\3rd reveiw>
```

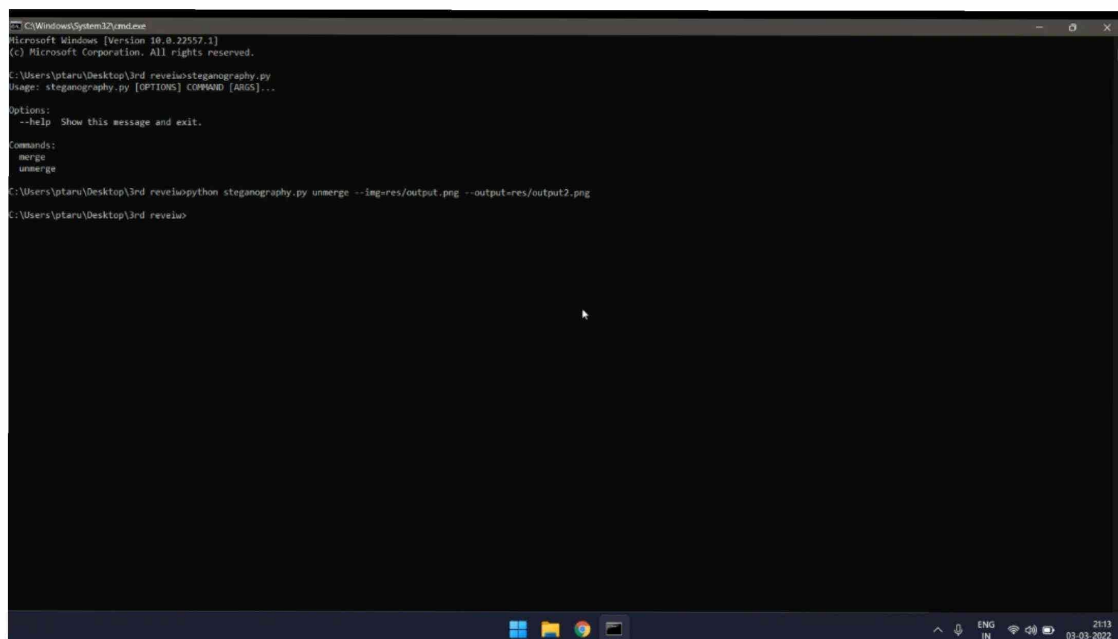
->As our code run successfully without any errors.

- The second image will be encrypted under the first image
- The stegno image will be obtained at the same location



Decrypting the hidden image:

- This should be done carefully otherwise the hidden data will be lost forever.
- We should select the unmerge command to decrypt the image
- The stegno image will be fed as input to the program
- After successfully running the code the output image will be obtained at the same folder with the name output2



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22557.1]
(c) Microsoft Corporation. All rights reserved.

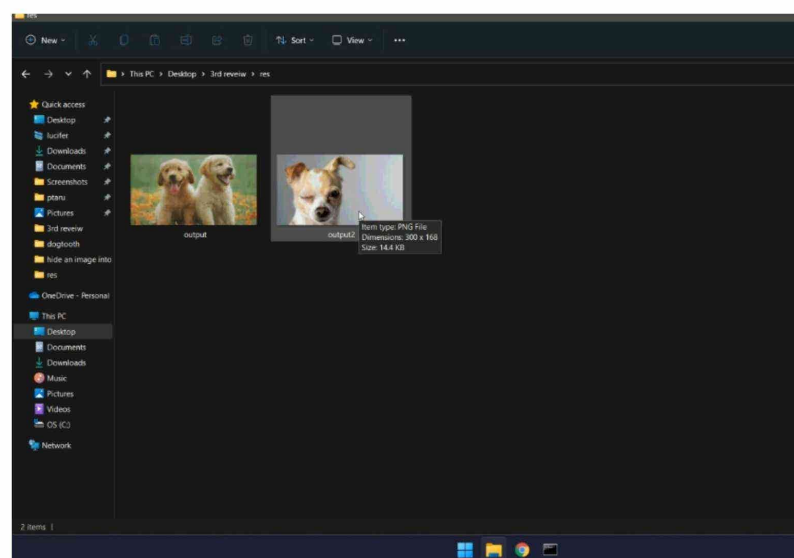
C:\Users\ptaru\Desktop\3rd review\steganography.py
Usage: steganography.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  merge
  unmerge

C:\Users\ptaru\Desktop\3rd review>python steganography.py unmerge --img-res/output.png --output-res/output2.png
C:\Users\ptaru\Desktop\3rd review>

```



Conclusion:

With this project I have learned a lot, especially about bit operations and bit masking, something that I never understood before. This project was fun from the start and only got more interesting as I went on developing it. I became more interested in the subject the more I researched it.

I have learned that while implementing Image Steganography is important, thinking of how to detect and attack it and the methods to do so are far more complex than actually doing the Steganography itself.

There is a lot of research that is beginning to discover new ways to detect Steganography, most of which involves some variation of statistical analysis.

We have tried the image hiding technique efficiently to make an image secure.

It is interesting to see what other methods will be developed and how accurate they will be at detecting Steganography.

Future work:

Steganography is a really interesting subject and outside of the mainstream cryptography and system administration that most of us deal with day after day. "You never know if a message is hidden", this is the dilemma that empowers steganography. As more emphasis is placed on the areas of copyright protection, privacy protection, and surveillance, we believe that steganography will continue to grow in importance as a protection mechanism. This project deals with Steganography in Image and Audio files using Least Significant Bit (LSB) coding. This project can be uplifted by considering following measures: A more sophisticated approach can be implemented by using a pseudo-random number generator to spread the message over the image file in a random manner. This project can be extended by using other media files like video and other complex formats of audio and image

REFERENCE:

- [1] Niels Provos and Peter Honeyman, "Hide and Seek: An Introduction to Steganography", University of Michigan, IEEE 2003
- [2] Daniela Stanescu, Mircea Stratulat, Voicu Groza, Joana Ghergulescu and Daniel Borca, "Steganography in YUVcolor space", IEEE International Workshop on Robotic and Sensors Environments (ROSE 2007), Ottawa-Canada, pp.1-4, October 2007
- [3] P.Ramesh Babu, Digital Image Processing. Scitech Publications., 2003.
- [4] Johnson, N. F. and Jajodia, S. (1998). Exploring steganography: Seeing the unseen.

Computer, 31(2): 26–34.

[5]. Jonathan Cummins, Patrick Diskin, Samuel Lau and Robert Parlett, "Steganography and digital watermarking" School of Computer Science, The University of Birmingham. 2003.
www.cs.unibo.it/people/phdstudents/scacciag/home_files/teach/datahide.pdf.

[6] Debnath Bhattacharyya, P. Das, S. Mukherjee, D. Ganguly, S.K. Bandyopadhyay, Tai-hoon Kim, "A Secured Technique for Image Data Hiding", Communications in Computer and Information Science, Springer, June, 2009, Vol. 29, pp. 151-159.