ABSTRACT

With the great popularity and expansion in digital world technologies and the rapid growth of the network, the Internet has turned out to be a commonly used channel for communication of all forms of data such as audio, video, image, and text, digitally. There are many cryptography and encryption algorithms available, however after a certain threshold they fail to deliver expected levels of encryption. Many have significant vulnerabilities. With steganography, most of these problems are eliminated. Steganography is the technique of hiding text, images, audio or even video in other images or any other digital content, such that it is very difficult or near impossible to detect that some data has been hidden into an image. The use of steganography algorithms can be embedded with encryption as an additional step for hiding or protecting sensitive data, thus ensuring information security.

PROBLEM STATEMENT AND OBJECTIVE

The technique of steganography is gaining importance in the fields of law enforcement, intelligence and military communication. Hence, it is important to develop techniques for steganography and steganalysis to protect the data in such images.

We aim to implement different steganography algorithms that hide the texts and images into other images and also recover the hidden data from the image. To find the vulnerability of these algorithms, we then aim to detect if text has been concealed in the images using methods of vulnerability analysis like histogram analysis, chi-square analysis, pairs analysis, compression and reformat attack. After this we will prepare a vulnerability assessment report highlighting the vulnerabilities and scope of improvement of each such algorithm.

LITERATURE REVIEW

S.No	Paper Details	Objective	Proposed Methodology	Limitations
1.	Nilizadeh, Amirfarhad, et al. "Adaptive Matrix Pattern Steganography on RGB Images." *Journal of Cyber Security and Mobility* (2022): 1-28. Publication 13 August **2021**	Traditional spatial steganography models are sensitive to processes such as randomly flipped bits, image filters, and image compression algorithms. This paper aims to find a method of steganography unaffected by these processes.	This paper presents an adaptive spatial domain image steganography algorithm for hiding digital media based on matrix patterns, named "Adaptive Matrix Pattern" (AMP). The AMP method increases the security of the steganography scheme of largely hidden messages since it adaptively generates a unique codebook matrix pattern for each ASCII character in each image block.	This method uses a pre-processing technique to identify the most suitable block to generate matrix patterns which can be very computationally expensive.
2.	Pal, A.K., Naik, K. and Agrawal, R., **2019.** A steganography scheme on JPEG compressed cover image with high embedding capacity. *Int. Arab J. Inf. Technol.*, *16*(1), pp.116-124.	To improve the embedding capacity of a given steganography algorithm.	The proposed scheme considers the jpeg bit stream of a cover image for embedding secret message bits. In order to improve the embedding capacity, they have increased the no. of non-zero quantized DCT by using a modified quantization table during the quantization phase of DCT and subsequently 2 bit secret message is embedded into each	Creating the modified table is itself a very computationally intensive process and it was observed that the proposed approach failed for larger images.

			quantized DCT coefficient. The secret message is itself encrypted using AES.The modified quantization table is obtained Q'(x,y) = Q(x,y)/SF where Q(x,y) and Q'(x,y) of the conventional quantization table and the modified one.	
3.	Weng, Shaowei, et al. "Dynamic improved pixel value ordering reversible data hiding." *Information Sciences* 489 (**2019**): 136-154	This paper aims to find an improved pixel value ordering steganography method which can flexibly identify the number of pixels in a block.	This paper proposes a dynamic IPVO RDH method that can flexibly modify the number of pixels in a block so that more data can be hidden in each block while keeping the distorting level low.	As the IPVO method depends on the smoothness of each block, the images must be of good quality to get best results.
4.	Yahya, Abid. *Steganography Techniques for Digital Images*. Springer, **2019.**	Most steganography methods suffer from having their embedded data destroyed by image manipulation techniques like image compression, image resizing etc. Also directly embedding data in the bytes of pixels is prone to image difference attacks. The objective of this method is to attain better imperceptibility and robustness.	The algorithm first detects feature points in the image using SURF feature detection algorithm. CDF 9/7 discrete wavelet transform is then applied to the image. Data is then hidden in the vertical and horizontal subbands of the transformed image.	The main limitation of this method is the large amount of complex computation needed to perform the algorithm. This will make the algorithm slow for large images.
5.	Wu, Da-Chun, and Wen-Hsiang Tsai. “A Steganographic Method for Images by Pixel-Value Differencing.” *Pattern Recognition Letters*, vol. 24, no. 9-10, 2003, pp. 1613–1626., doi:10.1016/s0167-8655(02)00402-6.	This paper proposes a new and efficient method of encoding secret messages into grey valued images.	In the process suggested by the authors, a cover image is partitioned into non-overlapping blocks of two consecutive pixels. A difference value is calculated from the values of the two pixels in each block. All possible	The selection of range intervals is critical and has to be done on the basis of human vision’s sensitivity to gray value variations from smoothness to contrast.

			difference values are classified into a number of ranges. The difference value then is replaced by a new value to embed the value of a sub-stream of the secret message.	
6.	Chan, Chi-Kwong, and L.m. Cheng. "Hiding Data in Images by Simple LSB Substitution." *Pattern Recognition*, vol. 37, no. 3, 2004, pp. 469–474., doi:10.1016/j.patcog.2003.08.007.	The paper focuses on a data hiding scheme(commonly used to hide data and images)known as Least Significant Bit Substitution(LSB). The above technique coupled with an optimal pixel adjustment scheme was used to improve the quality of stego-images	The proposed algorithm embeds a digital image into another digital image. The algorithm is as follows: An image was taken from the user and from the top-left pixel,pixels were scanned and 4 LSB bits were extracted. The values of 4 LSB bits of the cover image are overwritten by 4 MSB bits of the image to embed. A reverse process is applied for recovering messages from the image.	Since in this approach a basic LSB substitution is used. This technique is vulnerable to image difference attack. That is, taking the pixel-wise difference of the cover image and stego-image will reveal the hidden data.
7.	Liao, Xin, et al. "A Steganographic Method for Digital Images with Four-Pixel Differencing and Modified LSB Substitution." *Journal of Visual Communication and Image Representation*, vol. 22, no. 1, 2011, pp. 1–8., doi:10.1016/j.jvcir.2010.08.007	In order to improve the embedding capacity and provide an imperceptible visual quality, a novel steganographic method based on four-pixel differencing and modified least significant bit (LSB) substitution is presented.	The algorithm proposed is described below. The image is loaded and a message is taken from the user. The message is converted into bitstream. The image was traversed 4 pixels at a time 2 in vertical and 2 in horizontal direction. for each block of 4 pixels, the pixel with lowest grayscale value and tolerance(k) is determined(no. of LSB that can be	It was observed that image compression caused damage in the hidden message.Thus the proposed algorithm is not robust for large images.

			changed). data was embedded in each pixel using k-bit lsb. After applying modified k-bit lsb, a readjusting procedure is applied to obtain new values for the pixels.Replace original pixel values with new ones.	
8.	Lin, Yih-Kai. "High Capacity Reversible Data Hiding Scheme Based upon Discrete Cosine Transformation." *Journal of Systems and Software*, vol. 85, no. 10, 2012, pp. 2395–2404., doi:10.1016/j.jss.2012.05.032.	The main objective of the paper is to implement an algorithm which works in the frequency domain of the image. It takes the help of reversible discrete cosine transforms to implement the algorithm efficiently which retains the quality of the image.	This algorithm works in the frequency domain of the image. The image is required to be in grayscale.The block was converted into the frequency domain using reversible factorization of discrete cosine transform. Using the formula given in the paper the blocks were embedded with the bits.	The main limitation of this approach is that we are taking 8x8 blocks of pixels in the image and in the reversible discrete cosine function there are too many float value calculations and sometimes due to this floating point miscalculations happen because the values have to be very precise.
9.	Chen, Wen-Jan, et al. "High Payload Steganography Mechanism Using Hybrid Edge Detector." *Expert Systems with Applications*, vol. 37, no. 4, 2010, pp. 3292–3301.,	It is known that hiding data in the edge pixels of the image causes less deterioration in the quality of the image than if the same amount of data was hidden in the non-edge pixels. The object of this paper was to introduce a novel steganography algorithm that exploits this fact for better image quality while also retaining most of the capacity of the image.	The algorithm described in the paper first applies the fuzzy edge detector over the image to highlight the bold edges in the image. Then the canny edge detector is applied to detect small edges in the images. The resultant from the application of previous two algorithms will have most of the edges in the images. Finally the algorithm hides the data in the edge pixels using LSB substitution.	The limitation of this approach is that in most images, there aren't enough edge pixels to hide the data. This significantly reduces the capacity of most images. Hence images must be curated specifically for this approach to yield best results.

PROPOSED METHODOLOGY

The role of data encryption algorithms is to scramble the data such that the original content of the data can't be accessed by the unauthorised parties. Even though there are multiple data encryption algorithms that are virtually unbreakable with today's computing technology, they have a fundamental weakness that the attacker can still know if data is being shared between multiple parties even if the data is secured by a modern cipher. Steganographic techniques are used to supplement the encryption algorithms by hiding the data (preferably encrypted data) in such a manner that no one other than the sender and recipient recognize whether data is being transferred or not. Unlike encryption, the main purpose of steganography is defeated when the communication between two parties is detected, even if the actual data can't be deciphered by the attacker.

In this project we plan to implement a total of 3 algorithms, keeping in mind the various limitations discussed in the literature review. The algorithms are listed below:

- **Pixel Value Difference 2x substitution**
- **Reversible Discrete Cosine Transform**
- **Edge detection based steganography**

The project can be divided into 3 basic modules :

- **Identification**: In this module, we identify relevant research papers and shortlisted the algorithms which seemed implementable and viable keeping in mind the scope of the project.

- **Algorithm Implementation**: In this module, we are going to implement the 3 steganography algorithms mentioned above.
    - We will try to improve on the limitations of some of the implemented algorithms such as the limitation on the capacity of LSB using edge detection algorithm and precision problems in reversible DCT algorithm.

- **Vulnerability Analysis:** In this module, we aim to test each of our implement algorithms for vulnerabilities using the following methods of vulnerability analysis and make a detailed report highlighting the vulnerabilities of each.

    - **Histogram Analysis:** The histogram is made by the difference between two pixels of the block. Generally in normal image, the number of occurrences of the pixel difference, h(d) decreases with increasing |d| where d is the difference in the two pixels. After that we plot the difference between the number of occurrences of pixel difference to visually detect the presence of a secret message. The excessive high frequency components associated with the

steps of analysing using the histogram, and it will also provide a signature of the secret message. The vulnerability it uses is that the difference between the adjacent pixels can lead to the location of the hidden bytes in the pixels.

- **Chi Squared Analysis:** It is a common observation that the embedding of encrypted data changes the histogram of color frequencies in a particular way. The embedding process changes the least significant bits of the colors of the image. In this technique, the colors are addressed by their indices in the color table. To put it simply, the frequency difference between adjacent colors is reduced by the embedding process. Thus a chi-squared test can be used to determine whether color frequency distribution in an image matches a distribution that shows distortion when hidden data is embedded. The principle behind this attack is that in a normal image the frequencies of each of the two pixel values in each POV tend to lie far from the mean of the POV. However, when a text is embedded in the image, these tend to become nearly the same or in some cases, equal. In this way, we can identify whether a given image has embedded text in it and consequently state the above vulnerability.

- **RS Analysis:** The RS analysis is based on the fact that the content of each bit plane of an image is correlated with the remaining bit planes. We analyse how the number of R (regular) and S (singular) groups of pixels changes with the increase in the message length embedded in the LSB plane. To examine the image we define regular and singular groups of pixels based on how the smoothness of the pixels change when subject to flipping. We then find the relative frequencies of these groups in the given image, in the image obtained by flipping the LSBs of the original image and in the image obtained from randomising the LSBs of the original image. Using these three frequencies we try to predict the length of the embedded image.

- **Pairs Analysis:** It is used to detect the data hidden in palette images. It detects messages embedded in palette images as LSBs of indices to palette colors along a random walk. Its principle is based on color pairs. Let's say that colors c1, c2 are extracted from the image by scanning it by rows and columns. The sequence of colors is then converted to a binary vector by associating c1 with 0 and c2 with 1. After solving all the equations stated in the analysis, we can find out the length of the message hidden in the image. It shows that the function defined by the difference between the two homogeneity is quadratic in the amount of embedded data. It works on the weakness that natural images have no difference in homogeneity, one can obtain enough information to deduce the amount of embedded data in the image.

- **Compression and reformat attack:** In this attack the attacker tries to destroy the embedded image by performing image compression or changing the file format of the image. The principle behind this attack is image manipulation

like compression and reformatting changes the content of the image data without changing the image visually. If this attack is successful, then it will tell us that the steganography algorithm is vulnerable to image manipulation and work needs to be done to improve the robustness of the algorithm.

STEGANOGRAPHY ALGORITHMS IMPLEMENTED

Cover Image :

1. Pixel Value Difference 2x substitution

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include <tgmath.h>
#include <stdbool.h>
#include <string.h>
#include <stdarg.h>

#include "../constants.h"
#include "../image.h"


#include "pvd_greyscale.h"

typedef struct partitions{
    uint8_t size;
    uint8_t partitions[];
} Partitions;

//Paritioning scheme 8, 8, 16, 32, 64, 128
static u8_Pair find_difference_range(uint8_t d, const Partitions* restrict par){
    assert(d <= 255);

    uint8_t p = 0;
    for(uint8_t i = 0; i < par->size; i++){
        if(d >= p && d <= p +(par->partitions[i] - 1)){
            return (u8_Pair) {p, p + (par->partitions[i] - 1)};
        }

        p += par->partitions[i];
    }

    assert(0);
}

static u8_Pair embedding_func(u8_Pair i_pixel, int16_t d_old, int16_t d_new, bool *out_bounds){
    assert(out_bounds != NULL);
    int16_t diff = d_new - d_old;
    if(abs(d_old) % 2 == 0){
        int16_t x = (int16_t)(i_pixel.x) - floor(diff/2.0f);
        int16_t y = (int16_t)(i_pixel.y) + ceil(diff/2.0f);
        if( x < 0 || x > 255 ||  y < 0 || y > 255)  *out_bounds = true;
        else *out_bounds = false;

        return (u8_Pair){x, y};
    }
```

```
static u8_Pair embedding_func(u8_Pair i_pixel, int16_t d_old, int16_t d_new, bool *out_bounds){
    assert(out_bounds != NULL);
    int16_t diff = d_new - d_old;
    if(abs(d_old) % 2 == 0){
        int16_t x = (int16_t)(i_pixel.x) - floor(diff/2.0f);
        int16_t y = (int16_t)(i_pixel.y) + ceil(diff/2.0f);
        if( x < 0 || x > 255 ||  y < 0 || y > 255)  *out_bounds = true;
        else *out_bounds = false;

        return (u8_Pair){x, y};
    }

    int16_t x = (int16_t)(i_pixel.x) - ceil(diff/2.0f);
    int16_t y = (int16_t)(i_pixel.y) + floor(diff/2.0f);
    if( x < 0 || x > 255 ||  y < 0 || y > 255)  *out_bounds = true;
    else *out_bounds = false;

    return (u8_Pair){x, y};
}

static u8_Pair embed_data(u8_Pair old_vals, e_PVD_GREY st_data, bool * restrict skip)
{
    uint8_t num_bits;
    int16_t d;
    int16_t d_new;
    u8_Pair range = {};
    bool out_bounds = false;

    d = old_vals.y - old_vals.x;
    range = find_difference_range(abs(d), st_data.partitions);
    //Checking if the pixel is eligible or not for embedding
    embedding_func(old_vals, abs(d), range.y, &out_bounds);
    if(out_bounds){
        *skip = true;
        return (u8_Pair){0, 0};
    }

    //Embedding data
    num_bits = log2(range.y - range.x + 1);
    if (num_bits == 0){
        *skip = true;
        return (u8_Pair){0, 0};
    }
    *skip = false;

    d_new = range.x + get_bits(st_data.stream, num_bits);
    //printf("Bits written: %u value written: %u ", num_bits, bits_to_val(&msg[*msg_index], num_bits, *bit_num));
    d_new *= d >= 0 ? 1 : -1;

```

```
    d_new *= d >= 0 ? 1 : -1;


    return embedding_func((u8_Pair){old_vals.x, old_vals.y}, d, d_new, &out_bounds);
}

static void recover_data(u8_Pair old_vals, d_PVD_GREY st_data)
{
    uint8_t num_bits;
    uint8_t bits;
    int16_t d_new;
    u8_Pair range = {};
    bool out_bounds = false;

    d_new = old_vals.y - old_vals.x;

    range = find_difference_range(abs(d_new), st_data.partitions);
    embedding_func(old_vals, abs(d_new), range.y, &out_bounds);
    if(out_bounds) return;

    bits = d_new >= 0 ? d_new - range.x : -d_new - range.x;
    num_bits = log2(range.y - range.x + 1);

    write_bits(st_data.stream, bits, num_bits);

}

void pvd_grayscale_encrypt(e_PVD_GREY st_data){
    bool flip = false;
    bool skip_first_pixel = false;
    bool skip = false;

    Image * st_img = st_data.st_img;

    uint8_t channels = st_img->channels;
    uint8_t *g1 = NULL, *g2 = NULL;
    u8_Pair new_val;

    uint64_t i = 0;
    uint64_t width = st_img->width * st_img->channels;
    for(uint64_t j = 0; j < st_img->height; j++){
        if(!get_rBit_stream_status(st_data.stream)) break;

        if(!flip){
```

Encryption:

Recovery Key: 592

Decryption:

Recovered Message :

Various Germanic tribes have inhabited the northern parts of modern Germany since classical antiquity. A region named Germania was documented before AD 100. In the 10th century, German territories formed a central part of the Holy Roman Empire. During the 16th century, northern German regions became the centre of the Protestant Reformation. Following the Napoleonic Wars and the dissolution of the Holy Roman Empire in 1806, the German Confederation was formed in 1815. In 1871, Germany became a nation-state when most of the German states unified into the Prussian-dominated German Empire.

2. Reversible Discrete Cosine Transform

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include <tgmath.h>
#include <stdbool.h>
#include <string.h>

#include "../constants.h"
#include "../image.h"
#include "reversible_DCT.h"
#include "reversible_dct_mat_consts.h"

#define mat_width 8
#define mat_height 8

static void round_mat(const long double f[mat_height * mat_width], long double o[mat_height * mat_width]){
    assert(f != NULL);
    assert(o != NULL);

    for(uint8_t i = 0; i < mat_height * mat_width; i++){
        o[i] = round(f[i]);
    }
}

static void mat_mult(const long double a[mat_height * mat_width], const long double b[mat_height * mat_width],
                     long double o[mat_width * mat_height]){

    assert(a != NULL);
    assert(b != NULL);
    assert(o != NULL);
    assert(mat_height == mat_width);

    long double sum = 0.0f;
    for(uint8_t j = 0; j < mat_height; j++){
        for(uint8_t i = 0; i < mat_width; i++){

            for(uint8_t k = 0; k < mat_width; k++){
                sum += a[i_img(mat_width, k, i)] * b[i_img(mat_width, j, k)];
            }

            o[i_img(mat_width, j, i)] = sum;
            sum = 0.0;

        }
    }

}

```

```c
static void transpose(const long double a[mat_height * mat_width], long double o[mat_height * mat_width]){
    for(uint8_t j = 0; j < mat_height; j++){
        for(uint8_t i = 0; i < mat_width; i++){
            o[i_img(mat_width, j, i)] = a[i_img(mat_width, i, j)];
        }
    }
}

static void copy_mat(const long double src[mat_height * mat_width], long double dest[mat_height * mat_width]){
    for(uint8_t i = 0; i < mat_height * mat_width; i++){
        dest[i] = src[i];
    }
}
/*
static void print_mat(const long double a[mat_width * mat_height]){
    for(uint8_t j = 0; j < mat_height; j++){
        for(uint8_t i = 0; i < mat_width; i++){
            printf("%.1f ", a[i_img(mat_width, i, j)]);
        }
        printf("\n");
    }
}*/

static void transform_subimage(const long double a[mat_height * mat_width], long double o[mat_height * mat_width]){
    long double temp[mat_height * mat_width];
    long double temp2[mat_height * mat_width];
    const long double * const transforms[8] = {s1, s2, s3, s4, s5, s6, s7, s8};

    copy_mat(a, temp2);
    //print_mat(temp2);
    for(uint8_t i = 0; i < 2; i++){
        //print_mat(s0);
        mat_mult(s0, temp2, temp);
        //print_mat(temp);
        round_mat(temp, o);
        //printf("0\n");
        //print_mat(o); printf("\n");

        for(uint8_t j = 0; j < 8; j++){
            mat_mult(transforms[j], o, temp);
            round_mat(temp, o);
            //printf("%u\n", j+1); print_mat(o);
        }

        mat_mult(p, o, temp);
        //printf("P\n"); print_mat(temp);
        transpose(temp, temp2);
        //printf("T\n"); print_mat(temp2);
```

```
static void invsere_transform_subimage(const long double a[mat_height * mat_width], long double o[mat_height * mat_width]){
    long double temp[mat_height * mat_width];
    long double temp2[mat_height * mat_width];
    const long double * const inverse_transforms[9] = {i_s0, i_s1, i_s2, i_s3, i_s4, i_s5, i_s6, i_s7, i_s8};

    copy_mat(a, temp2);
    for(uint8_t i = 0; i < 2; i++){
        mat_mult(i_p, temp2, o);
        for(uint8_t j = 8; j != (uint8_t)-1; j--){
            mat_mult(inverse_transforms[j], o, temp);
            round_mat(temp, o);
        }

        transpose(o, temp2);
    }

    transpose(temp2, o);
}

static void embed_data(long double dct[mat_height * mat_width], e_rDCT st_data)
{
    assert(dct != NULL);
    assert(st_data.stream != NULL);


    uint8_t bit;
    for(uint8_t j = 0; j < mat_height; j++){
        if(!get_rBit_stream_status(st_data.stream)) break;

        for(uint8_t i = 0; i < mat_width; i++){
            if(!get_rBit_stream_status(st_data.stream)) break;

            if((i + j) <= st_data.p) continue;

            if(st_data.q <= dct[i_img(mat_width, j, i)]) dct[i_img(mat_width, j, i)] += st_data.q;
            else if(-st_data.q >= dct[i_img(mat_width, j, i)]) dct[i_img(mat_width, j, i)] -= st_data.q;
            else {
                bit = get_bits(st_data.stream, 1);

                //printf("%u %f\n", bit, dct[i_img(mat_width, j, i)]);

                if(!bit) dct[i_img(mat_width, j, i)] *= 2;
                else dct[i_img(mat_width, j, i)] = 2*dct[i_img(mat_width, j, i)] + 1;

                //printf("%u %f\n", bit, dct[i_img(mat_width, j, i)]);

```

Encryption:

Recovery Key: 542

Decryption:

Recovered Message :

The Nazi seizure of power in 1933 led to the establishment Df a dictatorship, World War II, and the Holocaust. After the end of World War II in Europe and a period of Allied occupation, Germany was divided into the Federal Republic of Germany, generally known as West Germany, and the German Democratic Republic, East Germany. The Federal Republic of Germany was a founding member of the European Economic Community and the European Union, while the German Democratic Republic was a communist Eastern Bloc state and member of the Warsaw Pact.

3. Edge Detection Based Steganography

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include <tgmath.h>
#include <stdbool.h>
#include <string.h>

#include "../constants.h"
#include "../image.h"
#include "../util.h"
#include "edge_detect_lsb.h"

void edge_detect_encrypt(e_Edge_Detect st_data){
    Image * st_img = st_data.st_img;

    Image edge = hybrid_edge_detector(st_img);
    const uint8_t channels = st_img->channels;

    const uint64_t size = edge.image_size - (edge.image_size % st_data.block_size);

    uint8_t block_id = 0;
    uint8_t bits = 0;
    for(uint64_t i = 0; i < size; i += st_data.block_size * channels){
        if(!get_rBit_stream_status(st_data.stream)) break;

        for(uint8_t j = 1; j < st_data.block_size; j++){
            if(!get_rBit_stream_status(st_data.stream)) break;

            if(edge.img_p[i + j * channels] == 255){
                block_id += power_2(j - 1);
                bits = get_bits(st_data.stream, st_data.edge_bits);
                st_img->img_p[i + j * channels] = k_bit_lsb(st_img->img_p[i + j * channels], bits, st_data.edge_bits);
            }else{
                bits = get_bits(st_data.stream, st_data.non_edge_bits);
                st_img->img_p[i + j * channels] = k_bit_lsb(st_img->img_p[i + j * channels], bits, st_data.non_edge_bits);
            }
        }
        st_img->img_p[i] = k_bit_lsb(st_img->img_p[i], block_id, st_data.block_size - 1);

        block_id = 0;
    }

    recovery_key_msg(st_data.stream);

    free_image(&edge);
}
```

```c
void edge_detect_decrypt(d_Edge_Detect st_data){
    Image * st_img = st_data.st_img;
    const uint8_t channels = st_img->channels;
    const uint64_t size = st_img->image_size - st_img->image_size % st_data.block_size;

    uint8_t block_id = 0;
    uint8_t bits = 0;

    for(uint64_t i = 0; i < size; i += st_data.block_size * channels){
        if(!get_wBit_stream_status(st_data.stream)) break;

        block_id = recover_k_bit_lsb(st_img->img_p[i], st_data.block_size - 1);

        for(uint8_t j = 1; j < st_data.block_size; j++, block_id >>= 1){
            if(!get_wBit_stream_status(st_data.stream)) break;
            if((block_id & 1) == 1){
                bits = recover_k_bit_lsb(st_img->img_p[i + j * channels], st_data.edge_bits);
                write_bits(st_data.stream, bits, st_data.edge_bits);
            }else{
                bits = recover_k_bit_lsb(st_img->img_p[i + j * channels], st_data.non_edge_bits);
                write_bits(st_data.stream, bits, st_data.non_edge_bits);
            }

        }

    }
}

void destroy_e_edge_detect_struct(e_Edge_Detect * restrict st){
    assert(st != NULL);

    free_image(st->st_img);
    free(st->st_img);
    delete_read_bitstream(st->stream);
}

e_Edge_Detect construct_e_edge_detect_struct(const char * restrict img_path, uint32_t msg_len,
                                             const char * restrict msg, uint8_t block_size,
                                             uint8_t non_edge_bits,
                                             uint8_t edge_bits){

    assert(img_path != NULL);
    assert(msg != NULL);

    if(block_size == 0){
        fprintf(stderr, "Invalid block size, must be greater than 0.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }
```

```c
e_Edge_Detect construct_e_edge_detect_struct(const char * restrict img_path, uint32_t msg_len,
                                             const char * restrict msg, uint8_t block_size,
                                             uint8_t non_edge_bits,
                                             uint8_t edge_bits){

    assert(img_path != NULL);
    assert(msg != NULL);

    if(block_size == 0){
        fprintf(stderr, "Invalid block size, must be greater than 0.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }

    if(edge_bits > 8){
        fprintf(stderr, "Invalid value for number of bits to embed in edge pixels.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }

    if(non_edge_bits > 8){
        fprintf(stderr, "Invalid value for number of bits to embed in non-edge pixels.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }

    e_Edge_Detect st;

    Image* img = malloc(sizeof(Image));
    if(img == NULL){
        fprintf(stderr, "Unable to allocate memory for image.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }

    *(img) = load_image(img_path);
    if(img->img_p == NULL){
        fprintf(stderr, "Unable to allocate memory for image.\n");
        return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
    }

    //Convert to greyscale
    if(img->channels > 2){
        Image grey = convert_to_greyscale(img);

        if(grey.img_p == NULL){
            fprintf(stderr, "Error in greycale conversion.\n");
            return (e_Edge_Detect){NULL, NULL, 0, 0, 0};
        }

        free_image(img);
        *(img) = grey;
```

Encryption:

Recovery Key: 669

Decryption:

Recovered Message :

Germany is a great power with a strong economy; it has the largest economy in Europe, the world's fourth-largest economy by nominal GDP, and the fifth-largest by PPP. As a global leader in several industrial, scientific and technological sectors, it is both the world's third-largest exporter and importer of goods. As a developed country, which ranks very high on the Human Development Index, it offers social security and a universal health care system, environmental protections, and a tuition-free university education. Germany is a member of the United Nations, NATO, the G7, the G20, and the OECD. It has the third-greatest number of UNESCO World Heritage Sites.

VULNERABILITY TESTS IMPLEMENTED

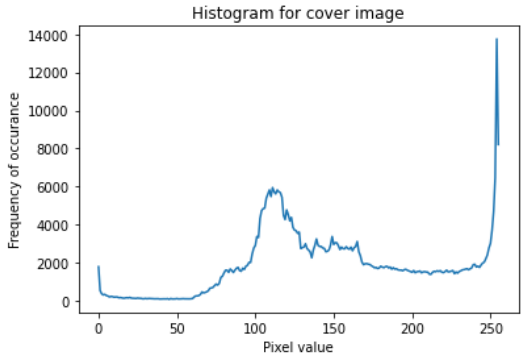
1. Histogram Analysis

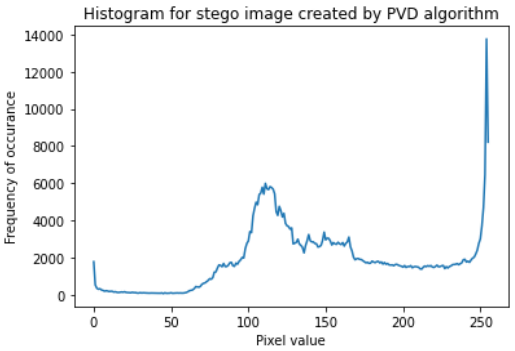
```python
from cv2 import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

img_post=cv2.imread('/content/pup.jpg')
img_post_stego=cv2.imread('/content/er_pup.jpg')

img_post=cv2.resize(img_post, (800,600))
img_post_stego=cv2.resize(img_post_stego, (800,600))
print(img_post.shape) #rgb image so 600 rows, 800 cols, 3 channels

#converting the img into binary
img_gray=cv2.cvtColor(img_post, cv2.COLOR_BGR2GRAY)
img_gray_stego=cv2.cvtColor(img_post_stego, cv2.COLOR_BGR2GRAY)
print(img_gray.shape)

#creating a numpy array of 0s
arr=np.zeros(shape=(256,1))
rows=img_post.shape[0]
cols=img_post.shape[1]
for i in range(rows):
  for j in range(cols):
    #read kth position and add 1 in it
    k=img_gray[i,j]
    arr[k,0]=arr[k,0]+1

```

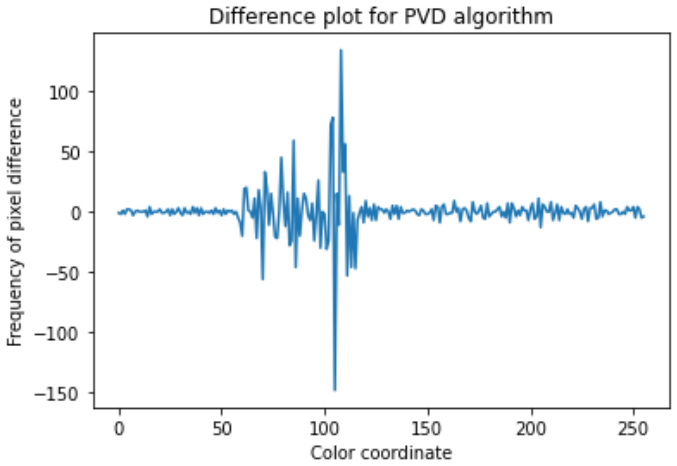
```
#we get total no of values for each pixel in the image in normal image
print(arr)


#for stego image
arr_stego=np.zeros(shape=(256,1))
for i in range(rows):
  for j in range(cols):
    #read kth position and add 1 in it
    k=img_gray_stego[i,j]
    arr_stego[k,0]=arr_stego[k,0]+1

#we get total no of values for each pixel in the image in normal image
print(arr_stego)

plt.plot(arr)
plt.title("Histogram for cover image")
plt.xlabel("Color coordinate")
plt.ylabel("Frequency")
plt.show()

plt.plot(arr_stego)
plt.title("Histogram for stego image")
plt.xlabel("Color coordinate")
plt.ylabel("Frequency")
plt.show()

#finding the difference between the 2 frequency arrays
diff_arr=arr-arr_stego
```

```
diff_arr=np.round(diff_arr, 2)

plt.plot(diff_arr)
plt.title("difference plot")
plt.xlabel("Color coordinate")
plt.ylabel("Frequency of difference between cover and stego image")
plt.show()
```

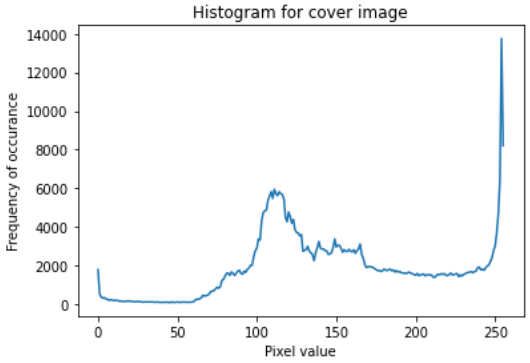
Histogram Analysis on Pixel Value Difference 2x Substitution Algorithm generated Stego Image

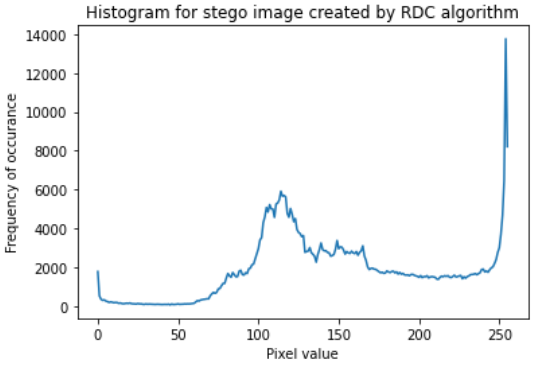


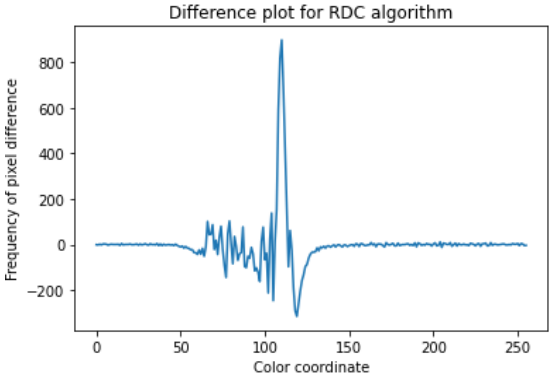




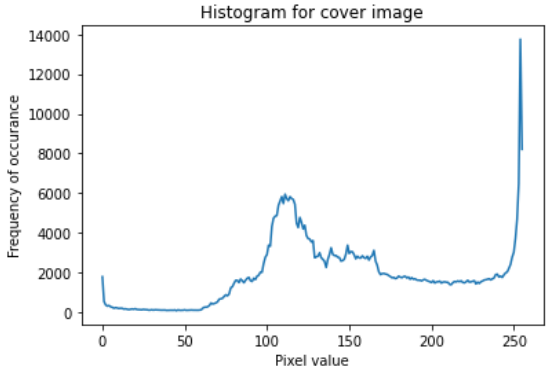
Histogram Analysis on Reversible Discrete Cosine Transform Algorithm generated Stego Image

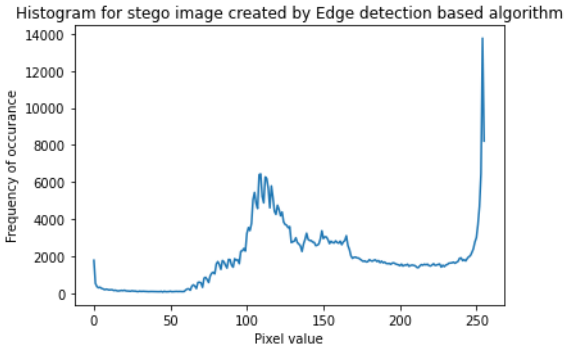


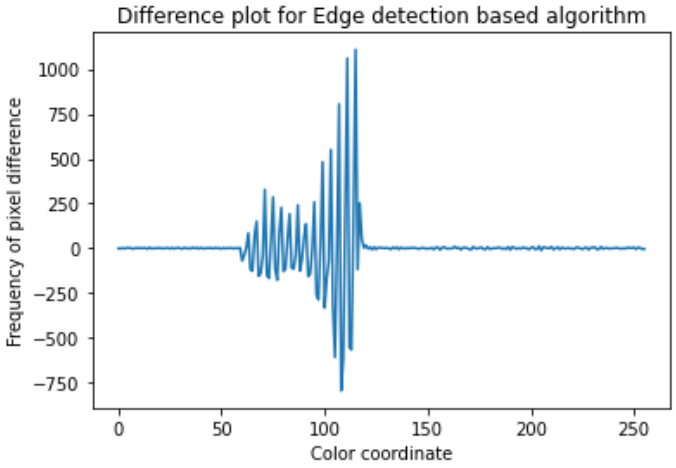




Histogram Analysis on Edge Detection Based Steganography Algorithm generated Stego Image







2. RS Analysis

```
from PIL import Image
import math
import sys
from matplotlib import pyplot as plt
import cv2
import scipy.stats

def splitpixels(img):
    pixRow = []
    pix = []
    pixNum = 0

    pixels = img.getdata()

    for pixel in pixels:
        pixRow.append(pixel[0])
        pixNum += 1

        if pixNum % img.size[0] == 0:
            pix.append(pixRow)
            pixRow = []

    print(pixNum, "pixels")

    return pix

def groupmask(gmask, mask):

    # initialise the new mask variable
    newgroupmask = []

    # adds to the list for the new mask
    for line in gmask:
        newgroupmask.append(list(line))

    # sets the row and column values
    totalrow = len(newgroupmask)
    totalcolumn = len(newgroupmask[0])

    # adds/subtracts from the group depending on if it is divisible by 2 (mod2)
    for row in range(0, totalrow):
        for column in range(0, totalcolumn):
            if newgroupmask[row][column] % 2 == 0:
                newgroupmask[row][column] += mask[row][column]
            else:
                newgroupmask[row][column] -= mask[row][column]
```

```
for row in range(0, imageRow):
    for column in range(0, imageCol):
        if discriminator_overlap:
            if row <= imageRow - maskRow:
                if column <= imageCol - maskCol:
                    pos = [row, column]
                    numCount += 1
                    breakimagebox = breakimage(imageBox, mask, pos)

                    flip_box = []
                    for line in breakimagebox:
                        flip_box.append(list(line))
                    for fliprow in range(0, len(breakimagebox)):
                        for flipcolumn in range(0, len(breakimagebox[0])):
                            if breakimagebox[fliprow][flipcolumn] % 2 == 0:
                                flip_box[fliprow][flipcolumn] += 1
                            elif breakimagebox[fliprow][flipcolumn] % 2 == 1:
                                flip_box[fliprow][flipcolumn] += -1

                    discr_breakimagebox = discrimination_function(breakimagebox)
                    discr_mask_breakimagebox = discrimination_function(groupmask(breakimagebox, mask))
                    discr_neg_mask_breakimagebox = discrimination_function(groupmask(breakimagebox, neg_mask))
                    discr_flip_box = discrimination_function(flip_box)
                    discr_mask_flip_box = discrimination_function(groupmask(flip_box, mask))
                    discr_neg_mask_flip_box = discrimination_function(groupmask(flip_box, neg_mask))



                    if discr_breakimagebox > discr_mask_breakimagebox:
                        s_p2 += 1
                    elif discr_breakimagebox < discr_mask_breakimagebox:
                        r_p2 += 1

                    if discr_breakimagebox > discr_neg_mask_breakimagebox:
                        neg_s_p2 += 1
                    elif discr_breakimagebox < discr_neg_mask_breakimagebox:
                        neg_r_p2 += 1

                    if discr_flip_box > discr_mask_flip_box:
                        s_1p2 += 1
                    elif discr_flip_box < discr_mask_flip_box:
                        r_1p2 += 1

                    if discr_flip_box < discr_neg_mask_flip_box:
                        neg_r_1p2 += 1
```

```
    # displays the size of the chosen mask
    print("")
    print("Mask size = ", len(chosen_mask[0]), "x", len(chosen_mask))

    # analyses the red pixels to determine what percent of them may contain embedded content
    print("")
    print("Analysing Red LSBs")
    gpercent = analyseLSBs(pix, chosen_mask, neg_mask, discriminator_overlap)

    # controls any errors within the calculations
    print("")
    if gpercent == 0:
        print("Unable to calculate the percent of the pixels")

        print("")
        encodedpercent = "?"
    else:
        # calculates and displays the total percentage of pixels that are likely to be encoded
        encodedpercent = gpercent
        print("")
        print("Decimal value probability of pixels likely to be encoded: ", encodedpercent)
        totalpercent = (encodedpercent * 100)
        print("Total Percent of pixels likely to contain embedded data: ", round(totalpercent, 2), "%")
        width, height = img.size
        totalpix = int(width) * int(height)
        # the size of the file is calculated by multiplying the percent of encoded pixels by the total number of pixels
        # then this is multiplied by 3 as each pixel requires 3 bits (rgb)
        data = ((encodedpercent * totalpix) * 3)
        print("Approximately ", round(data, 2), " bits of data (", round((data/8000), 2), "KB)")

    return encodedpercent

filename = "./stego_img_flwr.jpeg"
image_filename = Image.open(filename)

m0 = [[0, 1, 0]]
discrimination_overlap = 0


image_analyser(image_filename, m0, discrimination_overlap)
```

RS Analysis on Original Image

```
(Steganalysis) $ python3 RSAnalysis.py
99600 pixels

Mask size =  3 x 1

Analysing LSBs


Total Percent of pixels likely to contain embedded data:  0.75 %
Approximately  743.37  bits of data
```

RS Analysis on Pixel Value Difference 2x Substitution Algorithm generated Stego Image

```
(Steganalysis) $ python3 RSAnalysis.py
99600 pixels

Mask size =  3 x 1

Analysing LSBs


Total Percent of pixels likely to contain embedded data:  0.74 %
```

RS Analysis on Reversible Discrete Cosine Transform Algorithm generated Stego Image

```
(Steganalysis) $ python3 RSAnalysis.py
99600 pixels

Mask size =  3 x 1

Analysing LSBs


Total Percent of pixels likely to contain embedded data:  21.4 %
Approximately  21316.32  bits of data
```

RS Analysis on Edge Detection Based Steganography Algorithm generated Stego Image

```
(Steganalysis) $ python3 RSAnalysis.py
99600 pixels

Mask size =  3 x 1

Analysing LSBs


Total Percent of pixels likely to contain embedded data:  59.13 %
Approximately  58896.84  bits of data
```

3. Pairs Analysis

```
import numpy as np
import cv2 as ocv
import scipy.stats as scs
import matplotlib.pyplot as plt
import cmath

def quadratic_solution(a, b, c):
    d = (b**2) - (4 * a * c)
    sol1 = ((-b - cmath.sqrt(d)) / (2*a))
    sol2 = ((-b + cmath.sqrt(d)) / (2*a))

    return sol2.real, sol1.real

img_r = ocv.imread("./er_pup.jpg", ocv.IMREAD_GRAYSCALE)
threshold = 0

dimensions = img_r.shape
img_height = dimensions[0]
img_width = dimensions[1]

P = []

for i in range(0, img_height):
    for j in range(0, img_width-1):
        c1 = img_r[i, j]
        c2 = img_r[i, j + 1]
        P.append((c1, c2))

x = y = kp = 0
```

```
for k in range(0, (img_height * (img_width-1))):
    (r, s) = P[k]
    if(((s % 2 == 0) and (r < s)) or ((s % 2 != 0) and (r > s))):
        x += 1
    elif(((s % 2 == 0) and (r > s)) or ((s % 2 != 0) and (r < s))):
        y += 1
    elif((s // 2) == (r // 2)):
        kp += 1

if kp == 0:
    print("SPA Failed because k = 0")

a = 2 * kp
b = 2 * (2 * x - img_height*(img_width-1))
c = y - x

betaP, betaM = quadratic_solution(a, b, c)
beta = min(betaP, betaM)
print("Change Rate of Stego Image: " + str(beta))
```

Pairs Analysis on Original Image

```
(Steganalysis) $ python3 simplePairAnalysis.py
Change Rate of Stego Image: 0.0054439192803363914
```

Pairs Analysis on Pixel Value Difference 2x Substitution Algorithm generated Stego Image

```
(Steganalysis) $ python3 simplePairAnalysis.py
Change Rate of Stego Image: 0.01356954685188131
```

Pairs Analysis on Reversible Discrete Cosine Transform Algorithm generated Stego Image

```
(Steganalysis) $ python3 simplePairAnalysis.py
Change Rate of Stego Image: 0.10064105972240023
```

RS Analysis on Edge Detection Based Steganography Algorithm generated Stego Image

```
(Steganalysis) $ python3 simplePairAnalysis.py
Change Rate of Stego Image: 0.24191330207118739
```

4. Chi Square Analysis

```
#freq of each 0's,(black) 1's ...255's(white) is plotted
from cv2 import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

img_post=cv2.imread('./pup.jpg')
img_post_stego=cv2.imread('./edg_img.png')

img_post=cv2.resize(img_post, (800,600))
img_post_stego=cv2.resize(img_post_stego, (800,600))

#converting the img into binary
img_gray=cv2.cvtColor(img_post, cv2.COLOR_BGR2GRAY)

img_gray_stego=cv2.cvtColor(img_post_stego, cv2.COLOR_BGR2GRAY)


#enhancing the whites to be lighter and the greys to darker
#img_gray=cv2.convertScaleAbs(img_gray, alpha=1.10, beta=-20) #alpha is contrast, beta is brightness

#creating a numpy array of 0s
arr=np.zeros(shape=(256,1))
rows=img_post.shape[0]
cols=img_post.shape[1]
for i in range(rows):
  for j in range(cols):
    #read kth position and add 1 in it
    k=img_gray[i,j]
    arr[k,0]=arr[k,0]+1


#we get total no of values for each pixel in the image in normal image

#for stego image
arr_stego=np.zeros(shape=(256,1))
for i in range(rows):
  for j in range(cols):
    #read kth position and add 1 in it
    k=img_gray_stego[i,j]
    arr_stego[k,0]=arr_stego[k,0]+1

#we get total no of values for each pixel in the image in normal image
```

```
#finding the difference between the 2 frequency arrays
diff_arr=arr-arr_stego
diff_arr=np.round(diff_arr, 2)

#minr=abs(math.floor(np.amin(diff_arr)))
#maxr=abs(math.floor(np.amax(diff_arr)))

#freq_diff=np.zeros(shape=(minr,maxr))
#creating histogram of difference
#for i in range(diff_arr.shape[0]):
#  k=diff_arr[i]
#  k=math.floor(k)
#  k=abs(k)
#  freq_diff[k,0]=freq_diff[k,0]+1

count_pairs=0

for i in range(diff_arr.shape[0]-1):
  p1=diff_arr[i]
  p2=diff_arr[i+1]
  if(p1 != p2 and p1==abs(p2)):
    count_pairs=count_pairs+1

dist=0

for i in range(arr.shape[0]):
p=arr[i]
  q=arr_stego[i]
  dist=dist+(pow((p-q),2))/(pow((p+q),2))

print("Probability of being a steganoimage: " + str(dist*100))
```

Chi Square Analysis on Pixel Value Difference 2x Substitution Algorithm generated Stego Image

```
(Steganalysis) $ python3 chisquaredist.py
Probability of being a steganoimage: [3.68987415]
```

Chi Square Analysis on Reversible Discrete Cosine Transform Algorithm generated Stego Image

```
(Steganalysis) $ python3 chisquaredist.py
Probability of being a steganoimage: [38.93056882]
```

Chi Square Analysis on Edge Detection Based Steganography Algorithm generated Stego Image

```
(Steganalysis) $ python3 chisquaredist.py
Probability of being a steganoimage: [68.41254583]
```

5. Compression Reformat Attack

```python
from cv2 import cv2
imgpath = "./edg2_img.png"
img = cv2.imread(imgpath)

img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def save(path, image, jpg_quality=None, png_compression=None):
  if jpg_quality:
    cv2.imwrite(path, image, [int(cv2.IMWRITE_JPEG_QUALITY), jpg_quality])
  elif png_compression:
    cv2.imwrite(path, image, [int(cv2.IMWRITE_PNG_COMPRESSION), png_compression])
  else:
    cv2.imwrite(path, image)

# save the image in JPEG format with 85% quality
outpath_jpeg = "./compressed_edg2_jpg.jpg"
save(outpath_jpeg,img,jpg_quality=40) #higher the no,better the quality
outpath_png = "./compressed_edg2_png.png"
# save the image in PNG format with 4 Compression
save(outpath_png, img,png_compression=9)#0-9, compression value
```

Original Text:

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated, they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. A small river named Duden flows by their place and supplies it with the necessary regelialia. It is a paradigmatic country, in which roasted parts of sentences fly into your mouth. Even the all-powerful Pointing has no control about the blind texts; it is an almost orthographic life. One day however a small line of blind text by the name of Lorem Ipsum decided to leave for the far World of Grammar. The Big Oxmox advised her not to do so, because there were thousands of bad Commas, wild Question Marks and devious Semikoli, but the Little Blind Text didn’t listen. She packed her seven versalia, put her initials into the belt and made herself on the way. When she reached the first hills of the Italic Mountains, she had a last view back on the skyline of her hometown.

Compression Reformat Attack on Pixel Value Difference 2x Substitution Algorithm generated Stego Image

```
Recovered message: Far far away, behind the word mountains, far from the countries Vokalia and Conson
antia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Se
mantics, a large language ocean. A small river named Duden flows by their place and supplies it with
the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into
 your mouth. Even the all-powerful Pointing has no control about the blind texts it is an almost unor
thographic life One day however a small line of blind text by the name of Lorem Ipsum decided to leav
e for the far World of Grammar. The Big Oxmox advised her not to do so, because there were thousands
of bad Commas, wild Question Marks and devious Semikoli, but the Little Blind Text didn’t listen. She
 packed her seven versalia, put her initial into the belt and made herself on the way. When she reach
ed the first hills of the Italic Mountains, she had a last view back on the skyline of her hometown$@
H
```

Compression Reformat Attack on Reversible Discrete Cosine Transform Algorithm generated Stego Image

```
Recovered message: Far far away, behind the word mouztains, far from the countries Vokalia and Conson
antia, there live the blind texts. Separated they live in BoDkmarksgrove right at the coast of the Se
mantics, a large language ocean. A small river named Duden flows by their place and supplies it with
the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into
 your mouth. Even the all-powerful Pointing has no control about the blind texts it is an almost unor
thographic life One day however a small line of blind text by the name of Lorem Ipsum decided to leav
e for the far World of Grammar. The Big Oxmox advised her not to do so, because there were thousands
of bad Commas, wild Question Marks and devious Semikoli, but the Little Blind Text didn’t listen. She
 packed her seven versalia, put her"-f□Cc□□…幀\ǔC□□C$x4ɻxь□Ç履ʔ□□
```

Compression Reformat Attack on Edge Detection Based Steganography Algorithm generated Stego Image

```
Recovered message: Far far away, behind the word mountains, far from the countries Vokalia and Conson
antia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Se
mantics, a large language ocean. A small river named Duden flows by their place and supplies it with
the necessary regelialia. It is a paradisematic country, in which roasted parts of sentences fly into
 your mouth. Even the all-powerful Pointing has no control about the blind texts it is an almost unor
thographic life One day however a small line of blind text by the name of Lorem Ipsum decided to leav
e for the far World of Grammar. The Big Oxmox advised her not to do so, because there were thousands
of bad Commas, wild Question Marks and devious Semikoli, but the Little Blind Text didn’t listen. She
 packed her seven versalia, put her initial into the belt and made herself on the way. When she reach
ed the first hills of the Italic Mountains, she had a last view back on the skyline of her hometown$@
H
```

Appendix 1: Vulnerability Assessment Report of Steganographic Algorithms

VULNERABILITY ASSESSMENT REPORT OF STEGANOGRAPHIC ALGORITHMS

Version 1.0

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. EXECUTIVE SUMMARY

1.1 BACKGROUND

Digital steganography provides potential for private and secure communication that has become the necessity of most of the applications in today's world. Cryptosystems have various challenges that render them vulnerable. Since the cryptosystem ensures data security and integrity, the weaknesses of such systems are very critical. Various steganographic algorithms are used for hiding such secret information and a large variety of image steganography techniques are available with their respective advantages and disadvantages. Some algorithms provide more payload capacity while some provide more robustness against attack. Steganography can have two types based on its purposes-protection against detection(data hiding) and protection against removal(document marking by fingerprinting or watermarking). In our report, we mainly focus on the first category. Steganalysis is the technology that attempts to defeat steganography, by detecting the hidden information and extracting or destroying it.

1.2 SCOPE AND OBJECTIVES

Following is the indicative list of activities envisaged to meet the scope of work:

1. **Vulnerability Assessment & Penetration Testing(Steganalysis)** - White Box testing testing of the following algorithms: Pixel Value Difference 2x Substitution, Reversible Discrete Cosine Transform and Edge based detection technique.

2. **The penetration testing is performed** by carrying out the following attacks on each of the algorithms : Histogram analysis, Chi-Square analysis, RS Analysis, Sample pair analysis and lastly, Compression testing.

The objective of performing this Vulnerability Assessment is to create an overview of the security risks to the various steganography algorithms and then use these as a guideline to resolve those threats.

Further, we aim to provide certain guidelines/recommendations and remediations as to how one can enhance the security of these algorithms against the 5 attacks we have

tested them against. We further discussed some recommendations for the use of these algorithms in information security use cases.

1.3 INTENDED AUDIENCE

This Vulnerability report generated by the security staff, development and operations teams is intended for any developers, project manager or documentation writer that needs to understand the various vulnerabilities associated with the steganography algorithms implemented.

- **Developer:** The developer who wants to read, change, modify or add new requirements into the existing program, must firstly consult this document and update the requirements with appropriate manner so as to not destroy the actual meaning of them and pass the information correctly for future development and improvements in the proposed algorithm.

- **Tester:** The tester needs this document to validate the security of these algorithms by using various steganalysis (penetration testing) technique listed in the document so as to secure the system of attack.

For each one of the reader types to better understand this document, here is a suggestion to read in this document:

- Types of Steganography algorithms
- Challenges of Steganography
- Methodology
- Vulnerability assessment findings
- Recommendations and Remediation

1.4 TYPES OF IMAGE STEGANOGRAPHIC ALGORITHMS

Digital images are the most widely used cover objects for steganography. Due to the availability of various file formats for various applications the algorithm used for these formats differs accordingly. In the report, we have implemented only those algorithms that support a number of image types so as to increase the scope of this vulnerability assessment.

Large amounts of data can be encoded into 24-bit images as it is compared to 8-bit images. The drawback of 24-bit digital images is their size which is very high and this

makes them suspicious due to their heavy size when compared to 8-bit images. Depending on the type of message and type of the image different algorithms are used.

The following are some common types of stego algorithms:

- Least significant bit insertion
- Masking and filtering
- Redundant Pattern Encoding
- Encrypt and Scatter
- Algorithms and transformations
- Least significant bit insertion

Several domains exist for image steganography. Important steganographic domains are:

- Spatial Domain Techniques :
  Spatial domain techniques include bitwise manipulation of intensity of pixels and noise manipulation. There are various approaches to embed data in spatial domains. Most commonly used and simple techniques for spatial domain are Least Significant Bit (LSB) Methods.

- Transform domain Techniques: Transform domain techniques are also known as frequency domain techniques. Transform domain techniques first convert images from spatial domain to frequency domain and then a secret message is embedded. These techniques hide data by using mathematical functions. We often use these techniques in compression algorithms and transformations involve hiding secret messages in the transform space of the cover object. In Frequency domain schemes, the secret data will be embedded into transform coefficients which are transformed first into frequency domain by various frequency domain methods like Discrete Cosine Transformation (DCT), Discrete Wavelet Transform (DWT), Discrete Fourier Transform (DFT) etc then secret data will be embedded into transform coefficients.

- Distortion techniques: Here the secret message is embedded by distortion of cover and measuring deviation between original cover and stegos at decoding stage. Distortion techniques are less secure and are not used in various applications because original cover objects may be available to steganalysis for comparison. Text based steganography techniques generally use distortion type for embedding.

- Masking and Filtering: This technique masks secret data over original data by changing the luminance of particular areas. It embeds the message within significant bits of the cover image. Unlike LSB, masking is not susceptible to lossy techniques because image manipulation does not affect the secret message because masking adds redundancy to the hidden information. This makes the masking technique more suitable than LSB with lossy JPEG images. It may also help to protect against some image processing operations such as cropping and rotating.

1.5 CHALLENGES OF STEGANOGRAPHY

The challenges of steganalysis are:

- Steganographic techniques are very sensitive to various modifications in cover medium like Image processing operations (smoothing, filtering, image transformations etc.) compression techniques, removing and filtering digital noise techniques because these techniques lead to removal or modifications of secret embedded information too.

- Hidden messages must be secure both from perceptual and statistical attacks. There is a requirement to design more robust steganography algorithms and there is need to pay special attention for the presence of active and malicious attacks.

- Steganography algorithms have difficulty providing high payload capacity and imperceptibility. If a technique provides high payload capacity then it may become less robust and vice versa. Requirements for higher capacity and secure communication are often contradictory . Depending upon the specific application this trade off needs to seek out and at the same time there is also a need to produce a high quality stego algorithm by achieving high value of PSNR (Peak Signal to Noise Ratio).

- It is also a challenge to embed message into group images, which are highly intercorrelated and often manipulated in compressed form.

2. METHODOLOGY

To test the security of the implemented steganographic algorithms, we have taken **6 sample cover images** titled [1] pup [2] baboon [3] bear [4] barbara [5] lena [6] critter. Secret messages were hidden in each of these images using the three implemented steganographic algorithms. To exploit vulnerabilities in each of these algorithms, steganalysis was carried out on each of these images. The stego image is made by embedding 50% pixels of the cover image with some message.

The results were noted to find the number of times each algorithm was breached which in turn is used to determine whether the algorithm is immune against the attack.

An algorithm was deemed to be "breached" by a particular attack if and only if more than 3 images i.e 50% of sample space has values greater than the thresholds of the respective attacks (specified below).

In the end, these results were analysed to provide suitable recommendations for improvement. Their application in each use case was also identified and documented.

Fig.1 Pup

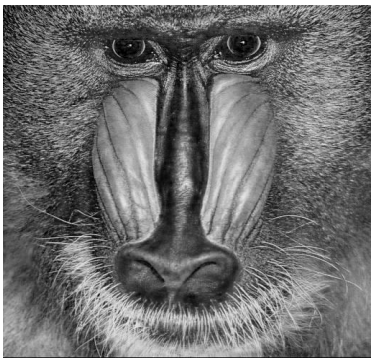


Fig.2 Baboon

Fig.3 Bear

Fig.4 Barbara

Fig.5 Lena

Fig.6 Critter

2.1 RS Analysis

The RS Analysis basically checks the regular and singular groups of pixel change with the increase in the length of the message in the LSB plane. We can say that this analysis is effective for stegano images that have the LSB embedding. The RS Analysis tries to predict the length of the message by analysing the frequencies of these regular and singular groups. We have applied the RS Analysis for different stego images for each of the algorithms that have been implemented. Based on the observations, we have found out that if the result is coming out to be **more than 7%,** then we can say that there is some major data embedded in the image.

2.2 Histogram Analysis

To find the vulnerability of the steganographic algorithms using histogram analysis, we find the number of color coordinates that differ between the cover image and the stego image. We count the number of coordinates that show a difference more than 10% from the original coordinates. Finally, this number is converted to per cent value. Based on our experiments, **an optimal threshold of 70%** was set to find if the attack was successful or not.

2.3 Chi Square Analysis

The idea of the chi-square attack is to compare the theoretically expected frequency distribution in the cover image with some possibility of distortion caused by the hidden message. To do this, the chi-square distance between corresponding color coordinates in the cover and stego images was calculated. **The greater the total chi-square distance value, the greater the probability of a hidden message**.

2.4 Pair Analysis

The pair analysis basically detects the data in images as LSBs of indices. The function defined by the difference between the LSBs is quadratic in the amount of the embedded data. After solving the equation given according to the analysis, we can find out the rate at which there is change in the stego image due to the embedded data. **If the rate is greater than 0.08%** then we can say that there is some major data embedded in the stego image.

2.5 Compression Attack

The main logic behind this attack is that if the image is compressed then the embedded data will be changed. It doesn't change the appearance of the image but the hidden content is distorted. This attack was done on various stego images with data embedded with each of the different algorithms. It was observed that the compression attack does not work on the Edge LSB algorithm but we can see that none of the messages could be recovered from the stego images generated from Reversible DCT. We have put in the table the **percentage of values that could be recovered from the stego image after compression it for each of three algorithms**.

3. VULNERABILITY ASSESSMENT FINDINGS

3.1 PIXEL VALUE DIFFERENCE 2x SUBSTITUTION

Cover Image	Histogram Analysis	RS Analysis	Pairs Analysis	Compression Attack	Chi Square Analysis
pup	55.85%	2.46 %	0.014 %	0.52 %	0.05
baboon	36.71%	4.43 %	0.008 %	0.45 %	1.22
bear	97.26%	1.88 %	0.007 %	94.54 %	3.46
barbara	53.51%	1.97 %	0.010 %	94.54 %	1.67
lena	36.71%	0.4 %	0.005 %	1.95 %	0.012
critter	90.62%	2.32 %	0.011 %	3.2 %	0.97
Avg.	61.77 %	2.24 %	0.009 %	32.53 %	0.08

Table. 1. Results from different analysis on PVD stego images

It is clearly observable from the above table that PVD is most vulnerable to statistical attacks like Histogram Analysis and Compression attack.

This can be attributed to the following **vulnerabilities** in the algorithm:

1. **Raster Scanning:** In the PVD algorithms implemented, a raster scanning order is typically employed for dividing the embedding units, which means that only vertical edges can be used. At the same time, such fixed division will leave some revealing clues for detectors. For instance, we can analyze some statistical artifacts in the PVD histogram to expose the stego images and estimate the size of hidden data.

2. **Contiguous ranges:** The above approach employs fixed contiguous ranges to classify the differences between the pixel values in the embedding unit, which will lead to undesired steps appearing at the PVD histogram thus breaking its imperceptibility.

3. **Contamination of Smooth regions:** On experimentation, it was found that the PVD algorithm tends to embed data in “busy” regions of the image. However

due to this trait, smoother regions of the image were affected which are found to be more exploitable by the attacks instead of less detectable regions like “edges”.

3.2 REVERSIBLE DISCRETE COSINE TRANSFORM

Cover Image	Histogram Analysis	RS Analysis	Pairs Analysis	Compression Attack	Chi Square Analysis
pup	91.26 %	18.32 %	0.091 %	0 %	21.67
baboon	97.65 %	44.45 %	0.159 %	0.01 %	35.66
bear	95.70 %	5.36 %	0.020 %	0.05 %	2.81
barbara	94.14 %	8.93 %	0.136 %	0 %	64.11
lena	97.26 %	6.63 %	0.046 %	0 %	41.51
critter	95.31 %	4.99 %	0.035 %	0.07 %	5.06
Avg.	95.22 %	14.78 %	0.081 %	0.02 %	28.47

Table. 2. Results from different analysis on Reversible DCT stego images

Reversible discrete cosine transform is vulnerable to histogram analysis and resistant to RS analysis and pairs analysis. Its vulnerability to histogram analysis is explained by:

- **Drastic changes in Pixel values:** This algorithm causes a lot of changes in the colour of pixels. This change in pixel colours is easily observable when comparing the histogram of the cover and stego images.
- Another possible vulnerability is due the fact that the image is processed in 8x8 blocks and between the edges of two blocks there can be artifacts that are causing significant differences between the histogram of cover and stego images.

3.3 EDGE DETECTION BASED STEGANOGRAPHY

Cover Image	Histogram Analysis	RS Analysis	Pairs Analysis	Compression Attack	Chi Square Analysis
pup	67.57 %	53.2 %	0.222 %	94.54 %	1.87
baboon	36.32 %	4.12 %	0.045 %	94.54 %	0.17
bear	97.65 %	15.15 %	0.091 %	94.54 %	3.53
barbara	68.75 %	33.52 %	0.153 %	94.54 %	8.51
lena	64.84 %	32.91 %	0.143 %	94.54 %	0.15
critter	87.89 %	17.73 %	0.089 %	94.54 %	1.15
Avg.	70.50 %	26.05 %	0.123 %	94.54 %	2.56

Table. 3. Results from different analysis on Edge LSB stego images

Edge detection based steganography is most susceptible to RS analysis and least susceptible to compression attack.

This vulnerability can be attributed to the following factors:

- **Uneven embedding:** Using this method, only 2 or 3 bits can be hidden per pixel. In the cases where the payload is increased beyond this threshold, the image quality is reduced noticeably.
- Another evident weakness is that this type of encoding can easily be decoded by converting the stego image in binary format and extracting the edge bits as specified.

3.4 SUMMARY OF FINDINGS

Based on the averages calculated, below is the summary that can be derived from the analysis conducted above:

1. RDCT was breached by Histogram analysis attacks in almost all instances suggesting that it was the least secure algorithm against statistical attacks. On the other hand, PVD performed the best against Histogram analysis.

2. Edge detection based steganography algorithm had the message detection percentage against RS analysis while PVD held against this attack.

3. Edge based detection had the highest percentage of message recovered which indicates its weakness against Image manipulation attacks.

The algorithm that best performs against all the attacks conducted was found to be pixel value difference(PVD) 2x substitution. However, there is still some scope of improvement which is described in the below sections.

4. COMPARATIVE ASSESSMENT

It is important to consider the following terminologies during the comparative assessment of the 3 algorithms. These include:

- **Payload Capacity:** Payload capacity indicates the amount of secret data that can embed to the cover media.

- **Robustness against statistical attack:** Statistical steganalysis is the practice of detecting hidden information through applying statistical tests on image data. In this report, histogram analysis and chi-square analysis are the statistical attacks that have been used to test the algorithms.

- **Peak- Signal-to-Noise Ratio (PSNR)** will be calculated. If the PSNR ratio is high then it indicates less distortion of image after embedding secret data that will increase robustness against statistical analysis.

- **Complexity:** The complexity of encoding and decoding the message is another consideration. If a more complex algorithm is used, then more time is required for both encoding and decoding operation.

- **Image Type Supported:** There are different types of image formats like GIF, JPEG, JPG, PNG, BMP etc. There is a need for a steganography technique which supports different types of image format.

ALGORITHM	**COMPLEXITY**	**PSNR VALUE (Db)**	**IMAGE TYPE SUPPORTED**
1.Pixel value difference	Low	55.584	PNG, JPG, Bitmap (Greyscale image)
2. Reversible DCT	High	41.274	PNG, JPG, Bitmap (Greyscale image)
3. Edge Detection based Steganography	Medium	52.444	PNG, JPG, Bitmap (Greyscale image)

Table. 4. Comparison of the algorithms

5. RECOMMENDATIONS AND REMEDIATIONS

5.1 EDGE BASED LSB

- To reduce the effectiveness of RS analysis, we can randomise the plane in which the data bit is hidden. For example, in one edge pixel we can hide data in the 8th bit of the pixel and in another edge pixel we can hide the data in the 4th bit of the pixel.

- We can use advanced edge detection algorithms based on deep learning to better identify pixels in which data should be embedded.

5.2 PIXEL VALUE DIFFERENCE

Inorder to increase the security of PVD algorithm, following are some promising improvements that can be adopted:

- Instead of adopting Raster scanning for division of embedded units, we can first divide the cover images into non-overlapping squares, and rotate them by a degree of 0, 90, 180 or 270 randomly. In such a way, both horizontal and vertical edges in images can be used. Moreover, tracking the embedding units from the stego images becomes impossible without the rotation key, which makes the detection more difficult.

- To defeat the histogram based steganalysis, we can employ a pseudo-random dithering to the fixed ranges. The experimental results show that the improved approach can avoid the occurrence of the undesired steps in the histogram effectively.

- To ensure that smooth regions of the image are not extensively used we can employ an optional threshold T to measure the edge strength for each embedding unit. When performing data embedding, our method can first estimate whether the embedding units whose strength is larger than T are enough for a given secret message. If not, the method decreases the threshold T to $T - 1$ to release more spaces, until all of the secret message can be embedded completely. In such a way, most smooth/flat regions in cover images can be well preserved.

5.3 REVERSIBLE DISCRETE COSINE TRANSFORM

Inorder to increase the security of reversible DCT, following are some promising improvements that can be adopted:

- We can decrease the size of the blocks which are applied to the image. This will decrease the number of pixels that are being affected by the application of the algorithm and reduce the difference between adjacent blocks in the stego image.

- We can reduce the amount of data stored per block to lessen the changes in the values of pixels. This will reduce the difference between the histograms of cover and stego images.

5.4 RECOMMENDATIONS FOR APPLICATIONS IN INFORMATION SECURITY

5.1 Password Manager

A password manager is used to store and generate passwords automatically. Most of the password managers encrypt the password and store them in the database. Instead of directly storing the encrypted password in the database, they can be stored in an image using steganography as an added layer of protection.
For such applications, we recommend the use of **Reversible Discrete Cosine Transform**, as for this algorithm the message was not recovered at all in most of the cases even if the presence of a secret message was detected.

5.2 Annotated Images in Photo Application

Media data today are usually associated with other information like metadata. A photo picture, for instance, may have the following:

- The title of the picture and some physical object information.
- The date and the time when the picture was taken.
- The camera and the location at which the picture was taken.

To find a specific shot in the piles of pictures, a photo album software usually sorts the pictures using these metadata words in each photo. However, the annotation data in such software are not unified with the target pictures. Each annotation only has a link to the picture. Therefore, when you transfer the pictures to a different album software for backup, all the annotation data is lost.

This problem is also known as "Metadata (e.g., annotation data) in a media database system (a photo album software) are separated from the media data (photo data) in the database management system (DBMS)." Steganography can solve this problem because a steganography program unifies two types of data into one by way of embedding operation. So, metadata can easily be transferred from one system to another without a hitch.

We recommend the use of **Pixel Value Difference Algorithm** for this use case as the requirements are more of faster algorithm and less computationally expensive algorithm. PVD algorithm as per our experiments turned out to be more perceptible to detection but faster and less complex in terms of computation.

5.3 Access Control System for Digital Content Distributed

Today, digital contents are commonly distributed over the Internet freely instead of individual distribution channels like emails to save time and labor. For example, music companies release new albums on their webpage in a free or charged manner. However, in this case, all the contents are equally distributed to the people who can access the webpage and an ordinary web distribution scheme is not suited for such open but selective distribution.

The use of a steganographic scheme can help solve this problem by uploading that content on the Internet in some covert manner. A special access key can be issued to extract the content selectively. In this area the embedded data is hidden, but is explained to publicize the content.

We recommend the use of the **Reverse DCT algorithm** because while it easily shows the presence of hidden information, its implementation is complex and cannot be decrypted easily.

5.4 Digital Watermarking

Secure steganographic algorithms can be used alongside encryption algorithms for digital watermarking in an information alteration protective system such as a **Digital Certificate Document System**. In this application, customers can send their digital certificate data to any place in the world through the Internet. No one can forge, alter, nor tamper such certificate data. If forged, altered, or tampered, it is easily detected by the extraction program.

For this use case, the use of edge based LSB algorithm is recommended as it was observed from our experiments that image quality is best preserved in this case.

6. CONCLUSION

All in all, each algorithm had properties and methods that led to the images generated by them being immune against certain attacks and weak against others. From the vulnerability analysis conducted, certain noteworthy points can be derived:

- In case of statistical attacks, RDCT proved to be the least secure among all algorithms tested.
- Edge detection based steganography proved weak against RS analysis and image manipulation attacks(in this case, compression).
- PVD was found to be the most versatile algorithm of the 3.
- However, each algorithm had certain vulnerabilities which could potentially be exploited and thus certain recommendations were provided to increase the security of these algorithms. Also, certain use cases were provided towards the end for better clarity in this respect.