International Journal of Scientific Research in Engineering
www.ijsrejournal.com
Volume 3 Issue 1ǁ Jan.- June 2018 ǁ PP. 09-18

Data Hiding For Captioned Images Using Reversible Image Transformation and Bit Encoding

Shreyans Sheth, Geetha Balan
Student at VIT University
Professor at VIT University
Corresponding Author: Shreyans Sheth

Date of Submission: 25-06-2018 Date of acceptance: 10-07-2018

I. INTRODUCTION

Images have transformed the way people share and document their lives. Moreover, captions and *hashtags* have become integral parts of images today, adding context and conveying vital information. With the storage of all these images being outsourced to the cloud, there are rising concerns over the security practices and reliability of image cloud services. There have been several security attacks such as the Dropbox hack , The iCloud 2017 data breach and more resulting in massive breach of privacy with sensitive images being exposed.

The threat of sensitive images being leaked in case of a data breach in today's time is a pertinent concern. Encrypting images through conventional means such as passwords and 3rd party software would mean attracting the *curious cloud* where attention is drawn to a particular encrypted image. For example, if a cloud provider were to attempt to embed certain metadata for it's storage purposes, the encrypted image would block such an attempt due owing to insufficient permission access, raising signals in the system.

This paper proposes a novel algorithm which helps overcome the aforementioned problems. Sensitive host images are transformed, or *encrypted* in a sense, into plaintext images which acts as decoys with the captioned message data embedded inside. Incase of a data breach, these images would seem like any other plain images, highly reducing exposure and unwanted attention. With plaintext images now being used, the cloud is free to write metadata information for storage purposes as well.

The algorithm uses a Reversible Image Transformation technique which takes a host image and a target image, converts the host image into a *encrypted* image, essentially a visually similar clone of the target image with metadata embedded within it. Following this, the captioned message string is embedded within the encrypted image using a Bit Encoding technique. A cryptographic key is generated and maintained. When this key is run on the encrypted image, it yields the hidden message. Following a series of steps, the original image is then retrieved from the by simply reversing the steps used to yield the encrypted image in the first place.

II. ALGORITHM

The Reversible Image Transformation(RIT) technique described here attempts to transform 2 spatial images, the target and the host images, which may be freely chosen, into a transformed image. For the purpose of demonstration, a single channel (grayscale) image is used but this can easily extended to RGB images by applying the same process on every channel.

Consider a Host Image H and a Target Image T. Both the images are broken down into n non-overlapping blocks of size $k * k$ where k would typically be a small, single digit number since breaking into more blocks yields visually accurate results.

After breaking the images into blocks, every block from the host image is paired with a block from the target image, yielding a pairing of blocks $B(H_1, T_1), B(H_2, T_2)\ldots\ldots B(H_n, T_n)$. Now, each of these pairings $B(H_i, T_i)$ are converted into a transformed block T'_i , which will be visually similar to the target block T_i . After repeating this process for all blocks, all of the T'_i are sequentially put together to yield the transformed image T'.

- Process of Block Pairing

Pairing of the blocks $B(H_i, T_i)$ is crucial to make sure that only those blocks are paired which ultimately yield a transformed block T'_i most similar to the target block. To ensure this, the host block and target block having the closest Standard Deviation(SD) are paired.

If B is a block of pixels $p_1, p_2, p_3 \ldots p_n$ then the Standard Deviation(SD) will be calculated as:

Mean: $u = (1/n) * \Sigma p_i$

SD: $sd = \sqrt{((1/n) * \Sigma(p_i - u)^2)}$

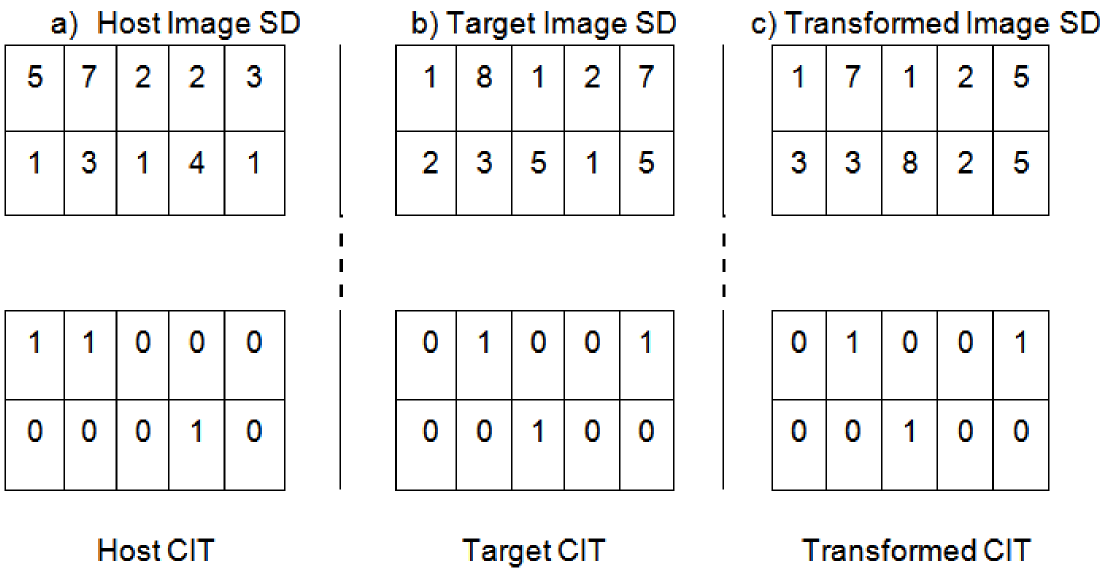
Note that to retrieve the host image from the transformed image, the position of the original blocks needs to be recorded and embedded in the transformed image as well. Simple calculation yields the fact that if an image would be divided into N blocks, $N * log\ (N)$ bits would be needed to record block indices. N is chosen such that a large segment of information is not recorded in small regions across the transformed image, distorting the latter.

The blocks are classified according to their Standard Deviation (SD) before pairing them up for the purpose of compression. For most images, it is found that the SD of most blocks is concentrated in a small range near zero and quickly drops thereafter. So for the purpose of demonstration, blocks shall be of size 4x4 and be classified into *category 1* or *category 2* based on the quantile of their SD.

All images having their SD in a range of $SD_0 - SD_\beta$ fall under category 1 and the ones having SDs in a range of $SD_\beta - SD_{100}$ fall under category 2. Following this, the blocks are scanned from left to right and assigned their respective categories on the basis of their SD.

Finally, the blocks of both the host and target images are scanned in parallel, pairing category 0 host blocks with category 0 target blocks and category 1 host blocks with category 1 target blocks in the order they are encountered.

Consider an example below where host and target blocks are assigned their SDs. $N_{\beta=70}$ is set with the last 3 images having large SDs assigned *category 1* (C_1) and the others assigned *category 0* (C_0). Assigning categories for all blocks yields a Category Index Table(CIT) for the host and target images.



<u>Pairings of blocks</u>

Block index of the host image	1	2	3	4	5	6	7	8	9	10
Block index of target image	2	5	1	3	4	6	7	9	8	10

The pairing in the example happens as follows - The 1st block of the host gets paired up with the 2nd block of the target (both are the 1st blocks in C_1, when scanned from left to right). Similarly, block 2 is paired with block 5, as they are the 2nd block to be encountered in C_1. Block 3 and 4 of the host are paired with block 1 and 3 of the target following the same logic and this goes for all the blocks.

The advantage of the CIT is that the entire original image can be constructed from the transformed image by just knowing the original CIT itself, which is embedded in the transformed image itself. Additionally, the CIT can be efficiently compressed using mostly conventional techniques owing to the bias between the number of $C_0$and C_1blocks.

- Process of Block Transformation

Let a host image block be represented as $B=\{p_1,p_2,p_3...p_n\}$and the target block as $T = p'_1,p'_2,p'_3...p'_n$, the the transformed block $T' = \{p_1'',p_2'',p_3''...p_n''\}$ can be constructed by applying *mean shifting* on every pixel of the host image. This is done to ensure that the transformed image and the target image blocks have the same mean. First, The the rounded difference between the means of the host and target blocks ($\square u$) is calculated added to the original host pixel.

$p''_i = p_i + \square u$ where $\square u = u_T - u_B$; $u_T = mean \quad (T)$; $u_B = mean \quad (B)$

p''_i should be a value between 0 and 255 as would be for any pixel. To handle the cases of underflow and overflow for pixel values, there are certain adjustments required. The maximum overflow pixel,OV_{max} , for $\square u >= 0$and the minimum overflow pixel, ON_{min} , for $\square u < 0$ are found for each block. Now, incase of over/under flow in blocks, it can fix this by adjusting the value of $\square u$ as follows:

$\square u = \square u + 255 - OV_{max}$ if $\square u >= 0$
$\square u = \square u - UN \quad _{min}$ if $\square u < 0$

This value of $\square u$ is further compressed to preserve precious bits in the transformed image which could be better utilised to encode the secret message later. a quantisation variable, λ is introduced and $\square u$ is modified as follows:

$\square u = \lambda * round \quad (\square u / \lambda)$ if $\square u >= 0$
$\square u = \lambda * floor \quad (\square u / \lambda) + \lambda / 2$ if $\square u < 0$

In which λ is an even parameter. Now, the variable to calculate and record simply becomes

$\square u' = 2|\square u| / \lambda$

where $\square u'$ is independent of the sign of $\square u$ and, reserving less space for bits, not recording the sign of $\square u$. The sign isn't required because when $\square u'$ is an even number, $\square u >= 0$ and when it is odd, it follows that $\square u' < 0$. There is also a decision to choose the value of λ for varying value of $\square u'$. The value $\lambda = 2$ has been set for demonstrating the idea and examples in this paper.

- Process of Block Rotation

To keep the target image as similar as possible, the transformed blocks are rotated in a direction θ, which is one of 0°, 90° 180° and 270°. This is done to compare the Root Mean Square Error(RMSE) of the transformed block(now rotated in either of these 4 directions) and the target block, picking the rotated transformed block giving *minimum* difference in RMSE.

- Generating the Encrypted Image

After completing the steps of Pairing, Transformation and Rotation, all the blocks T'_i are combined to yield the final transformed image T'. All the parameters, that are
$\square u$, rotation directions and CIT are compressed, encrypted and then embedded into the transformed image T' to output the encrypted image E(I), a visual clone of the target image.

- Embedding the Message into the Encrypted Image using a Bit Encoding Technique

The Message *M,* is now ready to be embedded inside of *E(I)* to yield the final *embedded image E'(I)*. Given that an image is represented as a matrix, certain pixels in the image shall be modified to embed *M.* Three parameters are analysed, given a bit from *M* and pixel to be modified from the image matrix:

1. Value of a bit in *M* (1 / 0)
2. Column number of the image matrix the pixel belongs to (odd / even)
3. The value of the image pixel (odd /even ; lies between 0 and 256)

Based on these parameters, it is decided to either add a 0 or 1 to the image pixel chosen, hence modifying it. Here is the *Bit Encoding Table(ET)* describing the encoding process for every permutation of these parameters.

S.No	Message Bit (1/0)	Column (O=odd / E=even)	Pixel Value (O=odd / E=even)	Value added (1 / 0)
1.	1	O	O	0
2.	1	E	E	0
3.	0	O	E	0
4.	0	E	O	0
5.	1	O	E	1
6.	1	E	O	1
7.	0	O	O	1
8.	0	E	E	1

Now, while embedding Message *M* inside the encrypted image, a *message decoding key(DK)* is generated which when applied to the encrypted image yields *M* back.

The key is easily constructed with the aid of the bit-encoding table. Consider a *DK* of size same as the original image (say *x* * *y*). Record *1* in the pixel entry where a pixel in the transformed image had been modified while embedding *M*, and a *0* otherwise. This yields a *DK* consisting of 0s and 1s. The pixel value, column number and whether the pixel had been modified or not is now known, corresponding to the first 3 columns in the table. This allows to accurately get the original bit by simply reading the entry in the 1st column, on locating the row corresponding to the given parameters.

For eg, given *Column no = 124 (even), Pixel Value = 245 (odd), Value added = 1* , it can be see that this combination corresponds to the *6th row* in the table and the value of the original bit is *1.*

Let us consider a concrete example. Take the word *hi* as the Message *M* which is to be embedded, represented as *0100100001101001* in binary. Now, a host and target (and following this, transformed) images of size *1 x 16* are picked. The following tabular example demonstrates embedding of *M* in *E(I)*:

1.	Initial Image	36	11	164	240	89	60	225	189	99	52	83	134	61	156	125	205
2.	Pixel (O/E)	E	O	E	E	O	E	O	O	O	E	O	E	O	E	O	O
3.	Column value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4.	Column (O/E)	E	O	E	O	E	O	E	O	E	O	E	O	E	O	E	O
5.	Binary message	0	1	0	0	1	0	0	0	0	1	1	0	1	0	0	1
6.	Decoding key / Add	1	0	1	0	1	0	0	1	0	1	1	0	1	0	0	0
7.	Modfied Image	37	11	165	240	90	60	225	190	99	53	84	134	62	156	125	205

Row 1 represents the values of the pixels of the Initial Image, which in our case would be *E(I)*. Row 7 represents the modified transformed image with *M* encoded. Row 6 tells us whether a bit was added and essentially acts as the decoding key *DK*.

It may be possible that a particular pixel value experienced overflow when 1 was added to it while modifying it. To counter this, one more *overflow key (OFK)* is maintained which stores the locations of the pixels that were modified but experienced overflow. The position of the overflow pixel is marked 1 in *OFK* or 0 otherwise.

To ensure security, *DK* and *OFK* are both encrypted and possessed independently by the sender and the receiver. Hence, even if an attacker knew that the image was a decoy, there is practically no way the attacker could reverse engineer the original image and *M*, owing to astronomical number of possibilities to check if a brute force attack had to be run on the image.

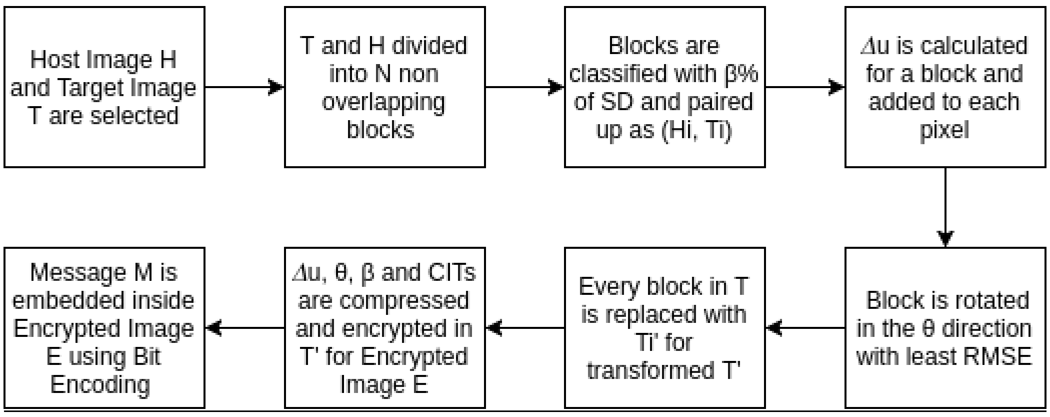
There are a few optimisations that can be made for the keys. JBIG2 compression can be applied on *DK* and *OFK* to reduce the space taken up by them.

- <u>Reconstructing the Host image from the Encrypted image</u>

The transformation procedures taking place in step 1 of the algorithm are all reversible. They need to be applied in the exact reverse order to obtain host image blocks and pair them up to yield the final host image.

<u>Procedure of Transformation</u>

Block Diagram for Transformation


Host Image H and Target Image T are selected
T and H divided into N non overlapping blocks
Blocks are classified with β% of SD and paired up as (Hi, Ti)
Δu is calculated for a block and added to each pixel
Block is rotated in the θ direction with least RMSE
Every block in T is replaced with Ti' for transformed T'
Δu, θ, β and CITs are compressed and encrypted in T' for Encrypted Image E
Message M is embedded inside Encrypted Image E using Bit Encoding

Input: Host Image *I*, Secret Message *M*, Standard Encryption Key *K*, Bit-Encoding Table *ET*
Output: Embedded Image *E'(I)*, Message Decoding Key *DK*

a. A Target Image *T* is selected from an Image Database having the same size as Host Image *H*
b. *H* and *T* are divided into non-overlapping blocks of size 4 x 4. Every image can be assumed to have N blocks with the the mean and SD calculate for each.
c. The blocks are classified with %β quantile of SDs and CIT is generated for *H* and *T*. Blocks of *H* and *T* are paired up correspondingly
d. For each block (H_i , T_i), the mean difference □u is calculated. □u is added to each pixel of H_i and then rotated into the optimal direction θi (θi ∈ {0◦ , 90◦ , 180◦ 270◦} yielding a transformed block T'_i
e. In the target image T , each block T_i is replaced with the corresponding transformed block T'_i for 1 ≤ i ≤ N and the transformed image T' is generated.
f. All the □us and θi s for the block pairs are collected and compressed together with the CIT of *H*. The compressed sequence and the parameter β are encrypted by a standard encryption scheme with the key *K*. This yields Encrypted Image *E(I)*.
g. The secret message *M* is embedded inside the *E(I)* using the Bit-Encoding Technique described with the aid of Bit Encoding Table *ET* following which final Embedded Image *E'(I)* and Message Decoding Key *DK* are generated.

<u>Procedure of Anti-Transformation</u>
Input: Embedded Image *E'(I)*, Decryption key *K'* for *K*, Message Decoding Key *DK*
Output: Host Image *I*, Secret Message *M*

a. The secret message *M* is retrieved from *E(I)'* using *DK* which denotes the location of modified bits, thus reverting the bits to their original state and retrieving *E(I)*.
b. The CIT of *H*, parameter β, □us and θi s, are decrypted from *E(I)* via *K'* and decompressed, yielding the transformed image T'' .
c. T'' is divided into N non-overlapping blocks of size of 4 × 4. The SDs of blocks are calculated and the the CIT of T'' is generated according to the %βquantile of SDs.
d. The blocks of T'' are rearranged as per the CITs of T'' and *H*.
e. For each block T''_i of T'' for $1 \leq i \leq N$, T''_i is rotated in the anti-direction of θi, then □ui is subtracted from each pixel of Ti and the original host image *H* is retrieved.

III. RESULTS

The similarity between the Target Image *T* and the final Embedded Image *E'(I)* is demonstrated as follows. From a dataset of images, random pairs of images are selected where one acts as the Host Image *H* and the other as the Target Image *T*. These pairs are now run through the algorithm described, yielding their respective *E'(I)*. Now with *T* and *E'(I)* ready, they are subjected to various standard image comparison techniques, namely Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), Structural Similarity (SSIM) Index and Mean Absolute Error (MAE).

MSE

Given an Image *I* and it's noisy approximation *K*, *MSE* is defined as

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

PSNR

Given the *MSE* and Max Intensity in *I* as MAX_I

$$PSNR = 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)$$

SSIM

The SSIM index is calculated on various windows of an image. The measure between two windows *x* and *y* of common size is

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$);
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.

MAE

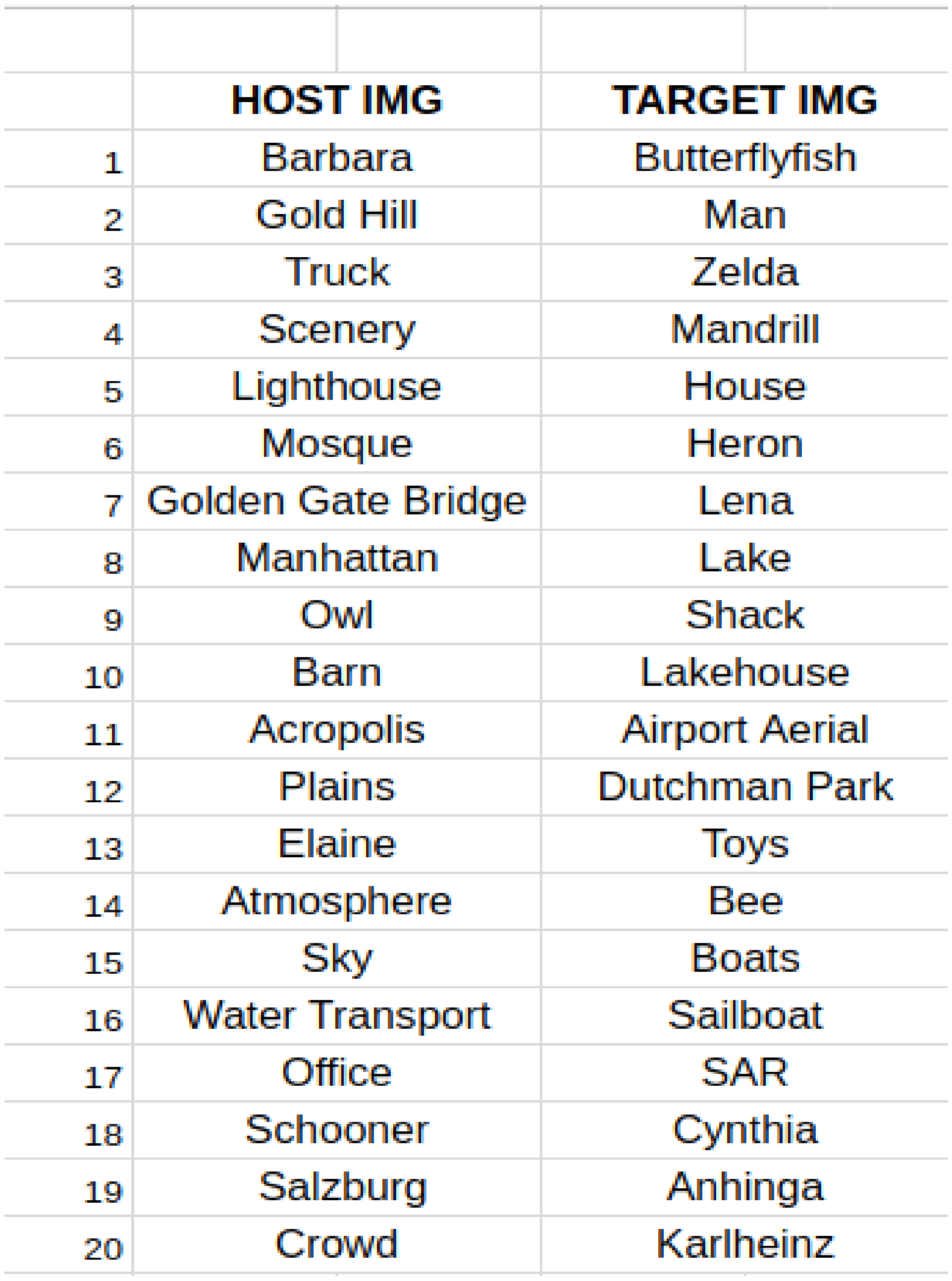
Given pixels of image x and y as x_i and y_i

$$\text{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n}$$

- Image Dataset

A heterogeneous dataset of grayscale images having dimensions 512 x 512 is constructed by randomly picking images from various, freely available standard datasets 1, 2, 3, 4, 5. The dataset constructed for the purpose of demonstration is available here.

Entire code for algorithm, implementation and generation results: https://github.com/bholagabbar/image-research-impl



	HOST IMG	TARGET IMG
1	Barbara	Butterflyfish
2	Gold Hill	Man
3	Truck	Zelda
4	Scenery	Mandrill
5	Lighthouse	House
6	Mosque	Heron
7	Golden Gate Bridge	Lena
8	Manhattan	Lake
9	Owl	Shack
10	Barn	Lakehouse
11	Acropolis	Airport Aerial
12	Plains	Dutchman Park
13	Elaine	Toys
14	Atmosphere	Bee
15	Sky	Boats
16	Water Transport	Sailboat
17	Office	SAR
18	Schooner	Cynthia
19	Salzburg	Anhinga
20	Crowd	Karlheinz

Consider **P** as the maximum size for message *M*. Following are the results gotten by comparing these images on the aforementioned metrics using various payload values. Calculation Sheet available here. PSNR measured in *dB.*

	0.1P			
	MSE	SSIM	PSNR	MAE
1	490.4311	0.5793	20.6622	0.0525
2	203.5047	0.7207	25.0451	0.0351
3	66.0135	0.7768	27.2405	0.0218
4	490.6448	0.6096	20.3269	0.0596
5	194.3744	0.7955	24.535	0.031
6	105.6849	0.7623	25.9098	0.0248
7	91.8183	0.8343	28.154	0.0214
8	338.7721	0.6488	21.4701	0.0467
9	310.4014	0.6509	22.5757	0.0469
10	190.7171	0.6254	25.05	0.0363
11	233.8674	0.6803	23.8418	0.0386
12	704.2222	0.5151	19.6537	0.0698
13	98.8995	0.7423	27.7244	0.0253
14	74.2324	0.8077	28.8256	0.019
15	168.4978	0.7504	25.302	0.0303
16	187.0196	0.7586	24.7761	0.0331
17	915.4351	0.3894	18.5145	0.0887
18	128.1274	0.7947	25.4503	0.0266
19	366.8176	0.7732	22.4863	0.037
20	251.1946	0.8265	24.0623	0.0321

	0.2P			
	MSE	SSIM	PSNR	MAE
1	490.6262	0.5783	20.6605	0.0526
2	203.5873	0.7206	25.0433	0.0351
3	66.0966	0.7766	27.235	0.0218
4	490.7643	0.6096	20.3258	0.0596
5	194.4772	0.7952	24.5327	0.0311
6	105.7707	0.7622	25.9063	0.0248
7	91.8994	0.8342	28.1502	0.0214
8	338.8432	0.6487	21.4691	0.0467
9	310.5048	0.6509	22.5743	0.047
10	190.8194	0.6252	25.0477	0.0363
11	233.9283	0.6803	23.8407	0.0386
12	704.2514	0.515	19.6535	0.0698
13	98.9438	0.7421	27.7224	0.0254
14	74.3437	0.8074	28.8191	0.0191
15	168.5986	0.7501	25.2994	0.0303
16	187.1161	0.7585	24.7739	0.0331
17	915.6691	0.3894	18.5134	0.0887
18	128.2081	0.7946	25.4476	0.0267
19	366.9056	0.7729	22.4853	0.0371
20	251.2913	0.8262	24.0606	0.0321

	0.4P			
	MSE	SSIM	PSNR	MAE
1	490.9925	0.5767	20.6572	0.0527
2	203.8528	0.7197	25.0376	0.0351
3	66.2284	0.7764	27.2264	0.0219
4	490.9026	0.6095	20.3246	0.0596
5	194.6796	0.7951	24.5282	0.0312
6	105.9003	0.762	25.9009	0.0249
7	92.0976	0.8337	28.1408	0.0215
8	338.9503	0.6483	21.4678	0.0468
9	310.6901	0.6507	22.5717	0.047
10	190.9812	0.6251	25.044	0.0363
11	234.0916	0.6801	23.8377	0.0387
12	704.3437	0.5149	19.653	0.0699
13	99.0481	0.7418	27.7178	0.0254
14	74.5304	0.8071	28.8082	0.0192
15	168.7997	0.7496	25.2942	0.0304
16	187.2673	0.7582	24.7703	0.0331
17	915.7505	0.3894	18.513	0.0887
18	128.5775	0.7943	25.4351	0.0267
19	367.0825	0.7723	22.4832	0.0372
20	251.5069	0.8253	24.0569	0.0322

	0.5P			
	MSE	SSIM	PSNR	MAE
1	491.209	0.576	20.6553	0.0527
2	203.9625	0.7196	25.0353	0.0352
3	66.3234	0.7763	27.2202	0.0219
4	491.0252	0.6094	20.3235	0.0596
5	194.7508	0.795	24.5266	0.0312
6	105.9994	0.7619	25.8969	0.0249
7	92.1847	0.8336	28.1367	0.0215
8	338.9937	0.6481	21.4672	0.0468
9	310.8025	0.6507	22.5701	0.047
10	191.0619	0.625	25.0422	0.0363
11	234.1545	0.68	23.8365	0.0387
12	704.3891	0.5148	19.6527	0.0699
13	99.086	0.7417	27.7162	0.0254
14	74.6381	0.8069	28.8019	0.0193
15	168.9098	0.7496	25.2914	0.0304
16	187.3673	0.7582	24.768	0.0332
17	915.9089	0.3893	18.5123	0.0888
18	128.4851	0.7944	25.4382	0.0267
19	367.1525	0.772	22.4823	0.0373
20	251.6506	0.8251	24.0544	0.0322

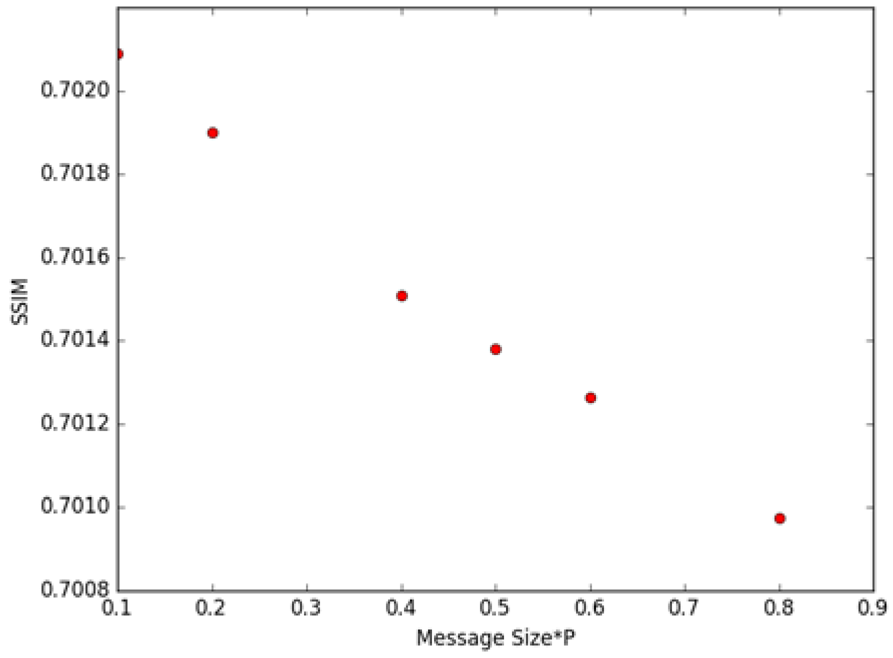
	0.6P			
	MSE	SSIM	PSNR	MAE
1	491.4791	0.5759	20.6529	0.0527
2	204.0617	0.7195	25.0332	0.0352
3	66.3951	0.7761	27.2155	0.0219
4	491.0855	0.6094	20.323	0.0596
5	194.874	0.7949	24.5238	0.0312
6	106.0666	0.7619	25.8941	0.0249
7	92.307	0.8334	28.131	0.0216
8	339.1227	0.648	21.4656	0.0468
9	310.8666	0.6506	22.5692	0.047
10	191.1827	0.625	25.0395	0.0363
11	234.2046	0.6799	23.8356	0.0387
12	704.4995	0.5148	19.652	0.0699
13	99.1567	0.7415	27.7131	0.0254
14	74.7269	0.8068	28.7968	0.0193
15	168.9952	0.7495	25.2892	0.0305
16	187.4349	0.758	24.7665	0.0332
17	915.685	0.3894	18.5133	0.0888
18	128.742	0.7942	25.4295	0.0268
19	367.2092	0.7716	22.4817	0.0373
20	251.8788	0.8249	24.0505	0.0323

	0.8P			
	MSE	SSIM	PSNR	MAE
1	491.8759	0.5753	20.6494	0.0528
2	204.3787	0.7183	25.0264	0.0353
3	66.5478	0.7758	27.2055	0.022
4	491.2973	0.6093	20.3211	0.0597
5	195.0406	0.7948	24.5201	0.0313
6	106.2189	0.7616	25.8879	0.025
7	92.4679	0.8332	28.1234	0.0216
8	339.3232	0.6479	21.463	0.0469
9	311.0497	0.6505	22.5667	0.047
10	191.2733	0.6248	25.0374	0.0364
11	234.3432	0.6797	23.833	0.0387
12	704.6273	0.5148	19.6512	0.0699
13	99.283	0.7414	27.7076	0.0254
14	74.932	0.8064	28.7849	0.0195
15	169.1436	0.7493	25.2854	0.0305
16	187.6103	0.7578	24.7624	0.0332
17	916.0609	0.3893	18.5116	0.0888
18	128.9064	0.794	25.424	0.0268
19	367.1014	0.7713	22.4829	0.0374
20	252.4231	0.824	24.0411	0.0323

As expected, the PSNR and SSIM go on decreasing as the message size increases as this would mean greater manipulation of the target image matrix. However, these changes remain constant up to 2 decimal places and the decrease in quality is diminutive.

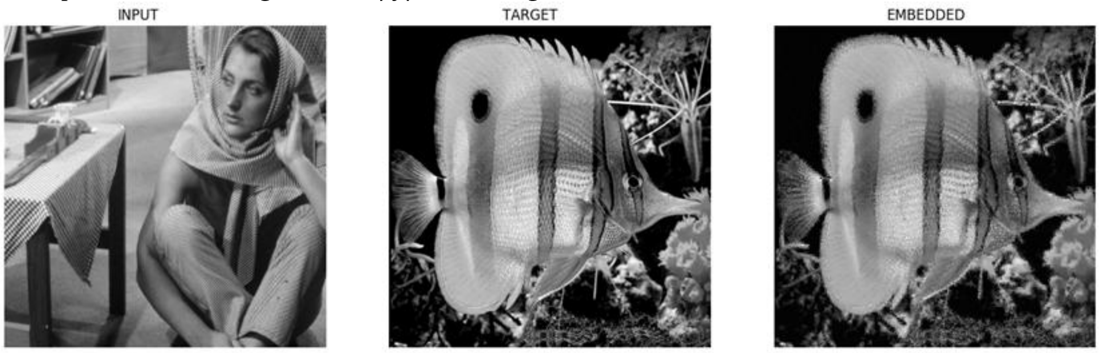
In any case, it can be seen that the Encrypted Images are able to maintain satisfactory visual quality. The average PSNR for these 20 randomly selected pairs having message size embedded 0.5P is 24.080315 dB which is acceptable.

A plot between the average SSIM against message size can be seen decreasing linearly. Again, these changes are miniscule nearly indistinguishable visually.

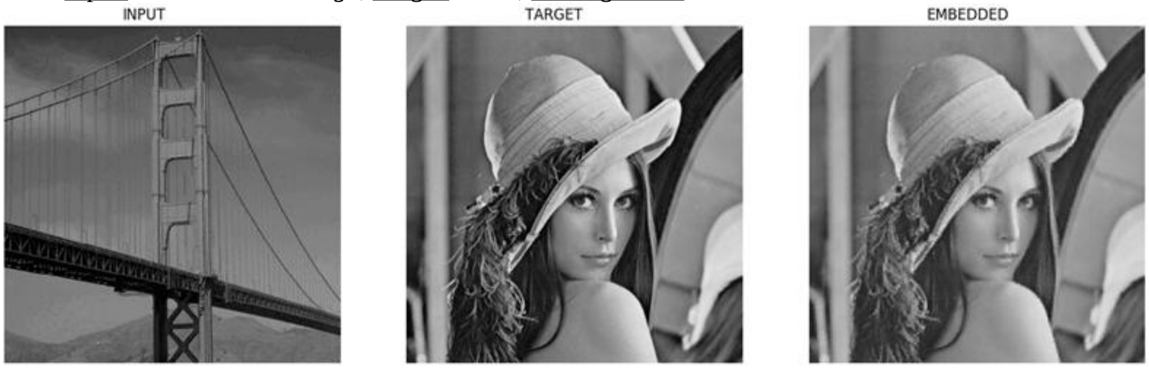


- Visual Samples

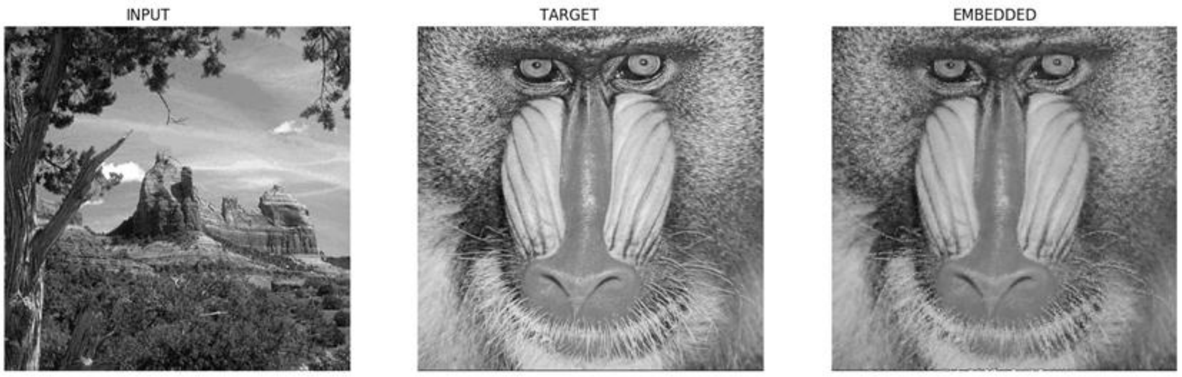
1. Input - *Barbara*, Target - *Butterflyfish*, Message Size - 0.2*P*



2. Input - *Golden Gate Bridge*, Target - *Lena*, Message Size - 0.5*P*



3. Input - *Scenery*, Target - *Mandrill*, Message Size - *0.8P*



IV. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated how sensitive images having additional textual data can be concealed discreetly in case of a data breach. It is nearly impossible for attackers to recognize the camouflage of sensitive images due to the decoy being an ordinary looking image and is also free from the notion of the curious cloud. With more and more services using the cloud for storing various assets and images, it is a necessity for them to employ all means possible and protect sensitive data and this algorithm fits the bill.

It would be interesting to see how the algorithm can be refined to further minimise artifacts and also implement a more advanced bit encoding technique.This algorithm could be extended to other multimedia formats such as audio, video as well.