VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF ELECTRONICS ENGINEERING (SENSE)

TARP

PROJECT REPORT

Digital watermarking using DWT-SVD technique for images

Karthik K – 17BEC0695

Slot: F2+TF2

Course code: ECE3999

Course faculty: Prof. SHAMBAVI

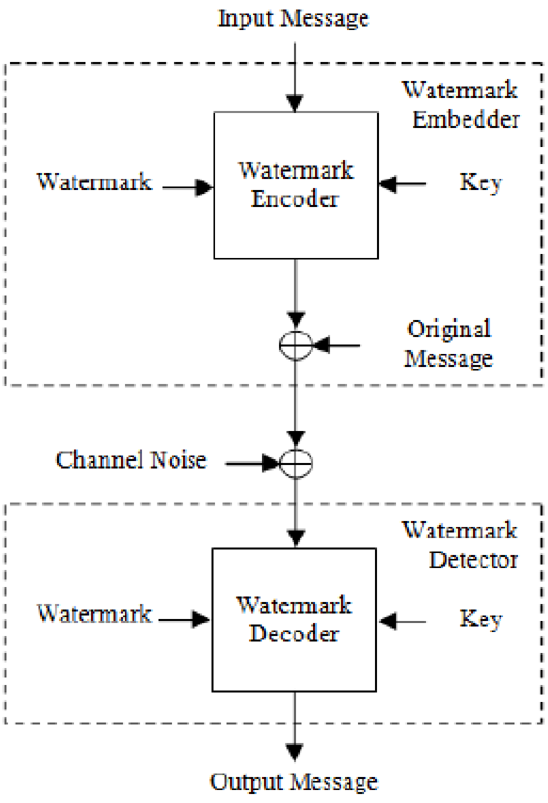
Digital watermarking using DWT-SVD TECHNIQUE for images

Abstract:

Nowadays huge amount of digital data is getting uploaded in internet. Documents across the internet expose them to various threats and this brings data security, duplication and authentication to a big question mark. Digital watermarking technique is a potential solution for this copyright protection issues. Idea behind digital watermarking is that information is inserted/ embedded into the digital media. This information can later be extracted and used for a lot of purposes like identification and authentication. Here we are using Discrete Wavelet transform (DWT) and Singular Value Decomposition (SVD) algorithm for embedding data into digital image. In this watermarking technique also uses a signature based authentication mechanism at the decoder which improves security. The method is subjected to various attacks and is valuated in terms of PSNR and correlation values. MATLAB environment is used for simulation.

Introduction:

A simple watermarking based communication model is shown in Fig 1. In this simple elaboration the input message or data is encoded using a watermark and a key. This is then added to the original data for transmission through the communication channel. While transmission the channel signals gets added to the transmitted signals as noise. The receiver uses a watermark detector to decode the original message from the noisy watermarked data using the key. With the widespread use of internet based applications, the concept of Image Watermarking got an importance in the transfer of digital information.



Digital image watermarking algorithms can be classified into two main categories according to the embedding domain: spatial and transform domain schemes. The spatial domain watermarking methods are simpler and are less robust against various geometric and non geometric attacks. The representative transform domain techniques embed the watermark by modulating the magnitude of coefficients in a transform domain, such as DWT and SVD. Transform domain methods can yield more information embedding and more robustness against many common attacks. But the computational cost is higher than spatial-domain watermarking methods.

Literature Survey:

Watermarking is a technique which is widely used and continuously developed by the use of various methods and implementations. Vast research work had already been done in this field that helped us to set a path for this work and contribute in the field of watermarking.

In[1] watermarking the image is done y DWT algorithm with key generation and this key is embedded within the cover image that is to be watermarked and during extraction process this algorithm checks if right key is given to extract the logo and also this paper also focuses on performance analyses of the algorithm on basis of PSNR and RMSE values

In [2] this paper watermarks a image and QR scanner logo, here DWT and SVD algorithm is applied on colour components blue, red, green(RGB) of the cover image both of them are recovered and analysed with different attacks to check the performance of the algorithm, calculating the PSNR and RMSE values with different attacks

In [3] A Robust watermarking scheme for digital images has been presented in literature. This scheme is based on nonnegative matrix factorization (NMF) and DWT. Gaussian pseudo-random code sequence is used as watermark. Watermark is inserted into factorized decomposition coefficients using NMF

In [4] proposed a Light weight Detection of Additive Watermarking in the DWT-Domain. They took a closer look at the computational requirements of watermark detectors. They showed that by switching to approximate host signal parameter estimates or even fixed parameter settings we achieve a remarkable improvement in runtime performance without sacrificing detection performance.

In [5] proposed a DWT-DFT Composite Watermarking Scheme which resists affine transform and JPEG compression attacks. Watermark is based on spread spectrum and embedded into LL band of discrete wavelet transform with a training sequence. A template is also inserted into middle frequency components of DFT. Robustness of watermarking method is improved by using new embedding strategy, watermark structure, 2-D interleaving, and synchronization technique.

Tools Used: MATLAB

Methodology:

Digital image watermarking algorithms can be classified into two main categories according to the

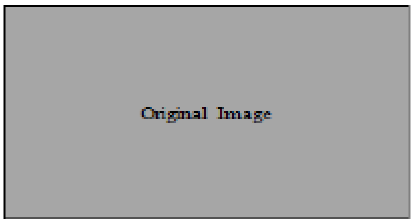
Embedding domain: spatial and transform domain schemes. The spatial domain watermarking methods are simpler and are less robust against various geometric and non geometric attacks. The representative transform domain techniques embed the watermark by modulating the magnitude of coefficients in a transform domain, such as DWT and SVD. Transform domain methods can yield more information embedding and more robustness against many common attacks. But the Computational cost is higher than spatial-domain watermarking methods.

Discrete Wavelet Transformation (DWT) -

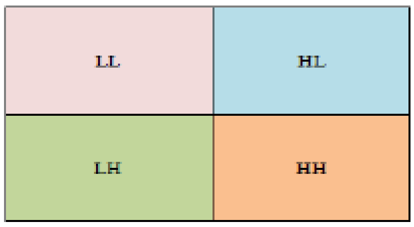
The DWT divides an image into four parts namely a lower resolution approximation component (LL) as well as horizontal (HL), vertical (LH) and diagonal (HH) detail components. The LL sub band is obtained after low-pass filtering both the rows and columns and contains a rough description of the image.

The HH sub band is high-pass filtered in both directions and have the high-frequency components along the diagonals. The HL and LH sub bands are the results of low-pass filtering on one direction and high-pass filtering in the other direction.

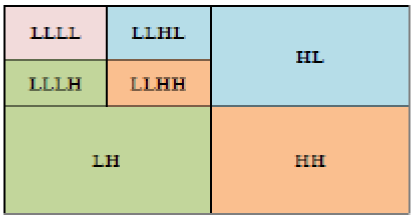
After the image is processed by the wavelet transform, most of the information contained in the host image is concentrated into the LL image. LH sub band contains mostly the vertical detail information which corresponds to horizontal edges. HL band represents the horizontal detail information from the vertical edges.



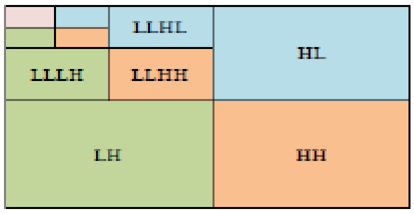
(a) Original Image



(b) Level-1 DWT



(c) Level-2 DWT



(d) Level-3 DWT

Singular Value Decomposition (SVD) -

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix. An image can be represented in the form of a matrix of scalar values. SVD decomposes an image represented by a matrix A of size M.N into a product of three matrices $A=USV^T$ where U and V^T are M.N and N.N orthogonal matrices, respectively. Here, S is an N.N diagonal matrix. The elements of S are only nonzero on the diagonal and are called the singular values of A.

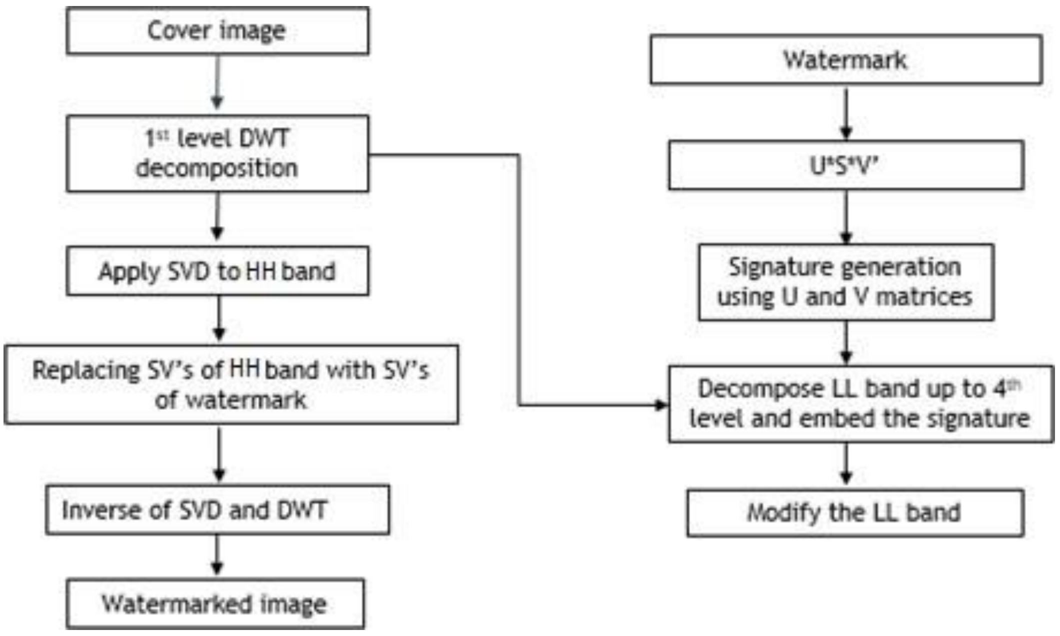
SVD is used because of the following properties:

1. Little disturbance added to the image does not cause high variation to the singular values of the image.
2. The singular value of the image represents the essential algebraic image properties.

Algorithm:

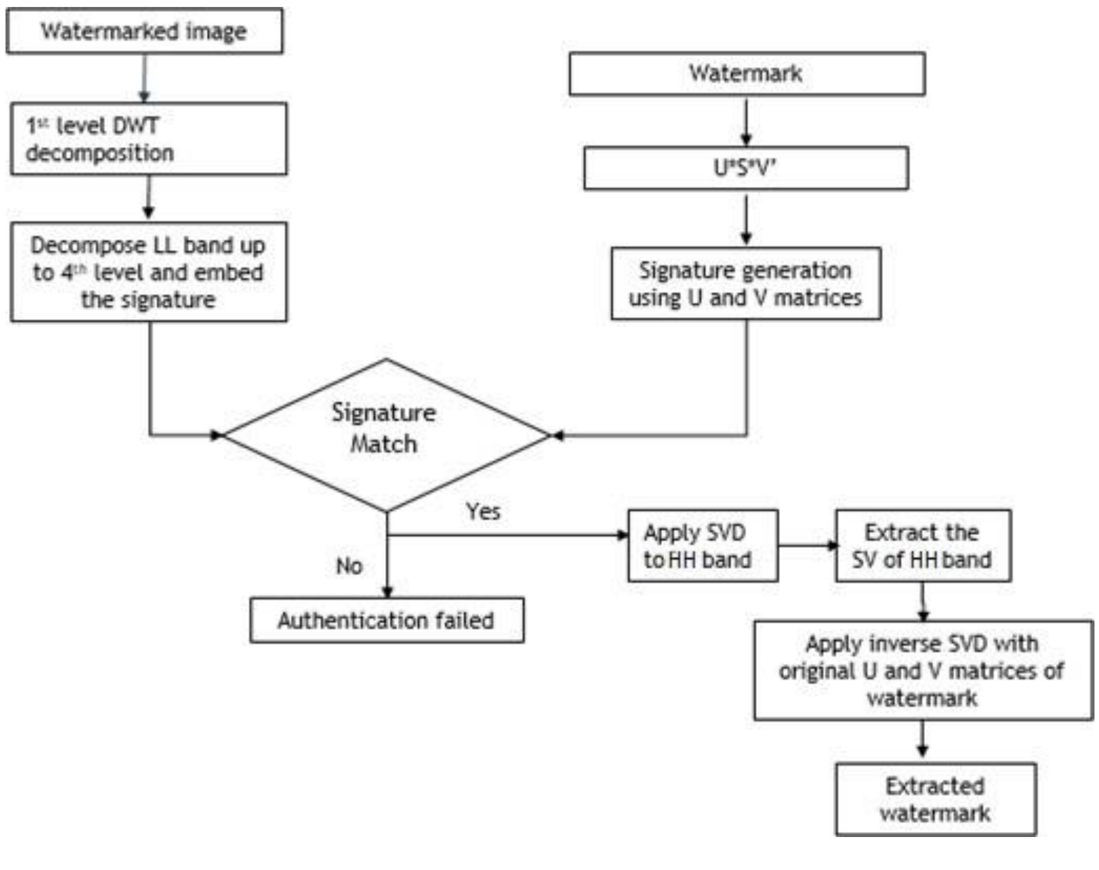
Watermark Embedding: 17BEC0695

1. Watermark W is decomposed using SVD. $W = U_W \times S_W \times V_W{}^T$
2. Using Haar wavelet performs first level decomposition of the cover image: LL, HL, LH, and HH. SVD is then applied to HH band. $H = U_L \times S_W \times V_L{}^T$
3. The singular values of the HH band are replaced with the singular values of the watermark. After applying inverse SVD we obtain modified HH band. $H' = U_L \times S_L \times V_L{}^T$
4. Then LL band is decomposed by 4 level DWT and the signature in generated and embedded in it. Thus we obtain modified LL band.
5. Inverse DWT is applied to produce the watermarked cover image.


Cover image
1st level DWT decomposition
Apply SVD to HH band
Replacing SV's of HH band with SV's of watermark
Inverse of SVD and DWT
Watermarked image
Watermark
U*S*V'
Signature generation using U and V matrices
Decompose LL band up to 4th level and embed the signature
Modify the LL band

Watermark Extraction: 17BEC0777

1. Using Haar wavelet, the noisy watermarked image is decomposed.
2. After level 1 decomposition of watermarked image, it is the decomposed by 4 level DWT and extract the signature.
3. If Signature matches then watermarked image is extracted.
4. SVD is applied to HH band. $H = U_L \times S_L \times V_L{}^T$
5. Then extract the singular values from HH band.
6. The watermark is constructed using singular values and orthogonal matrices UW and VW obtained using SVD of original watermark. $W_E = U_W \times S_L \times V_W{}^T$


Watermarked image
1st level DWT decomposition
Decompose LL band up to 4th level and embed the signature
Watermark
U*S*V'
Signature generation using U and V matrices
Signature Match
Yes
No
Authentication failed
Apply SVD to HH band
Extract the SV of HH band
Apply inverse SVD with original U and V matrices of watermark
Extracted watermark

Signature Authentication:

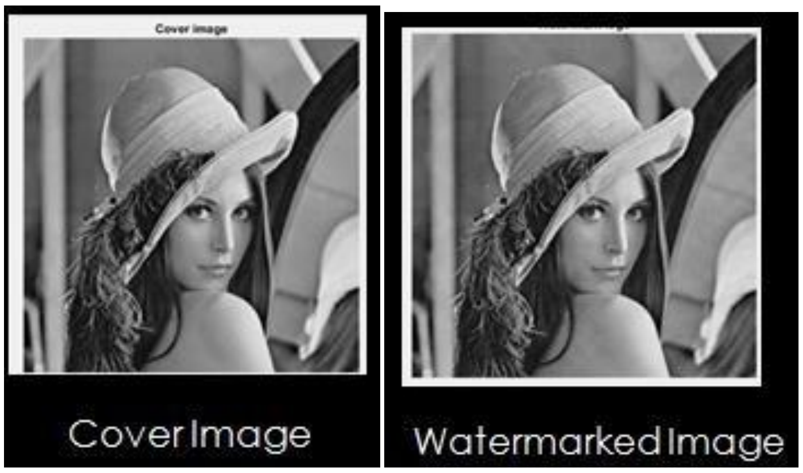
The U and V orthogonal matrices are authenticated before extracting the watermark. A unique signature is generated using the U and V matrices and it is embedded into the cover image along with the watermark. There is one to one correspondence between the SV's and orthogonal matrices.

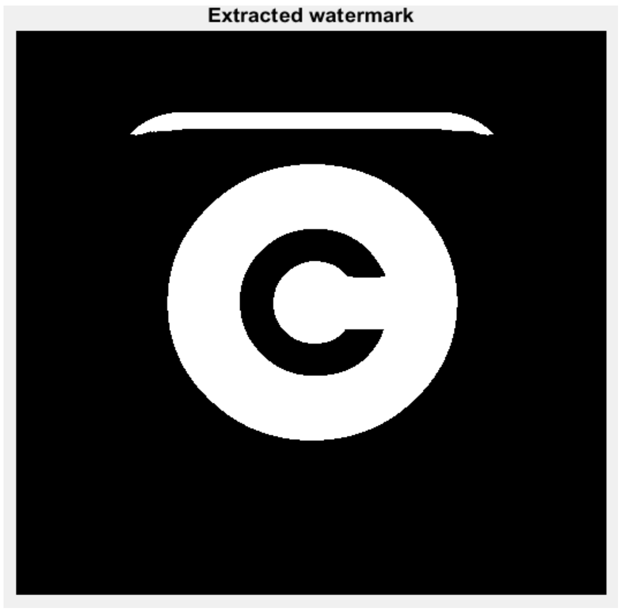
Digital signature for the U and V matrices is generated as follows:

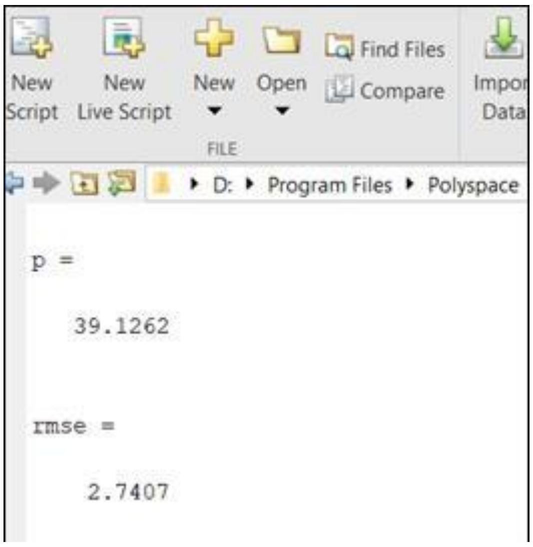
1. Create an array by summing the column of the U and V orthogonal matrices.
2. Map the values of the array obtained with the corresponding binary digits based on threshold.
3. XOR the binary digits to obtain the signature.

DWT is very suitable to identify areas in the cover image where as a watermark can be imperceptibly embedded due to its excellent spatio – frequency localization properties. One of the important mathematical properties of SVD is that slight variations of singular values do not affect the visual perception of the host image, which motivates the watermark embedding procedure to achieve good transparency and robustness.

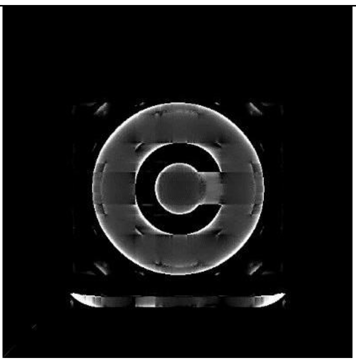
Observation & Results:

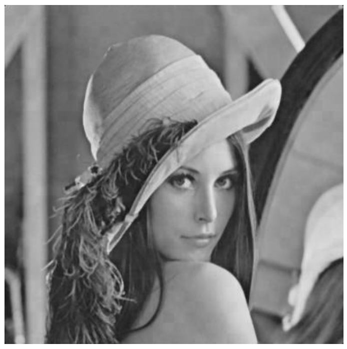
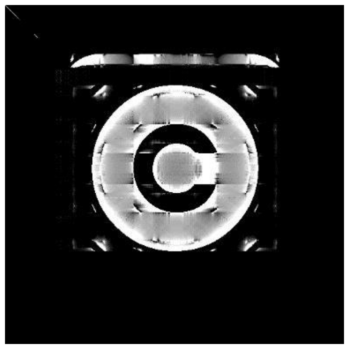
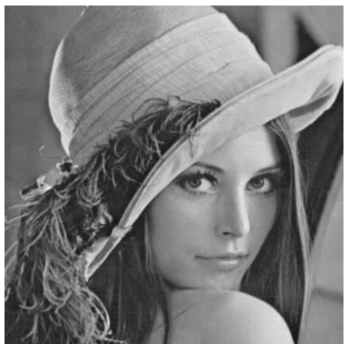
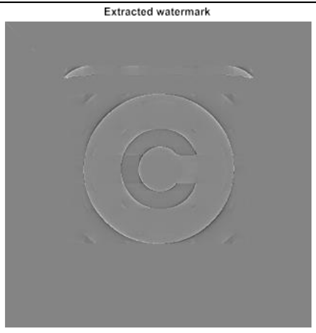
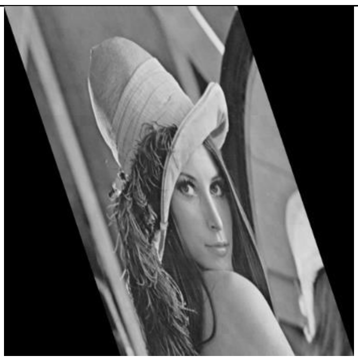
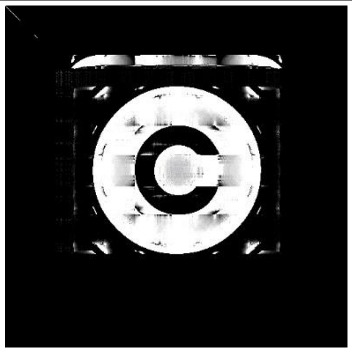
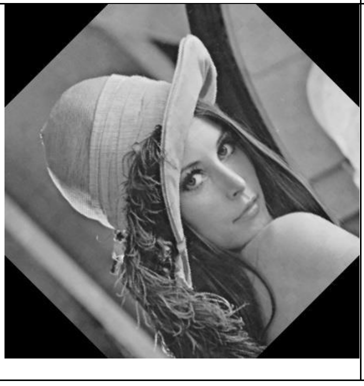
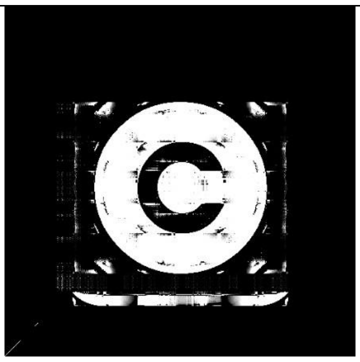


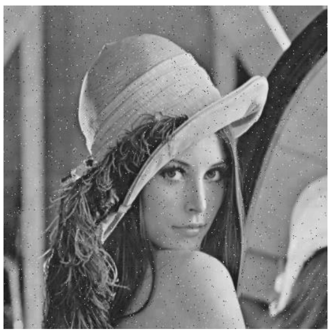
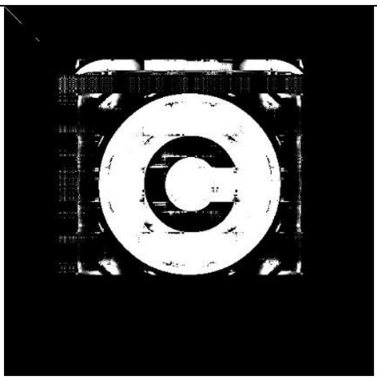


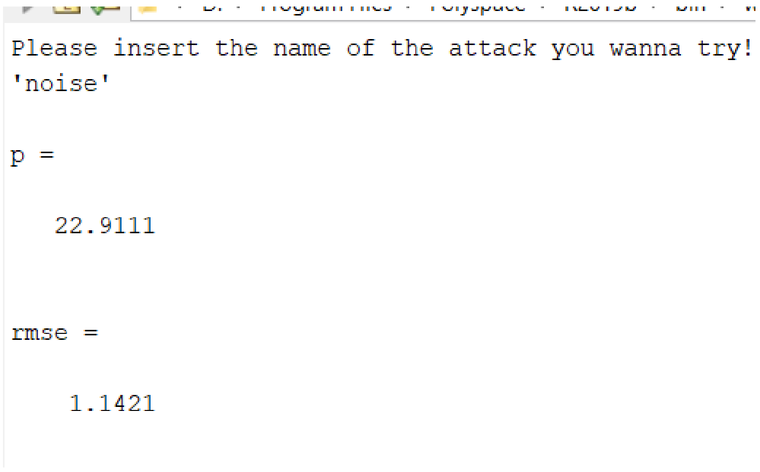


```
p =

   39.1262


rmse =

    2.7407
```

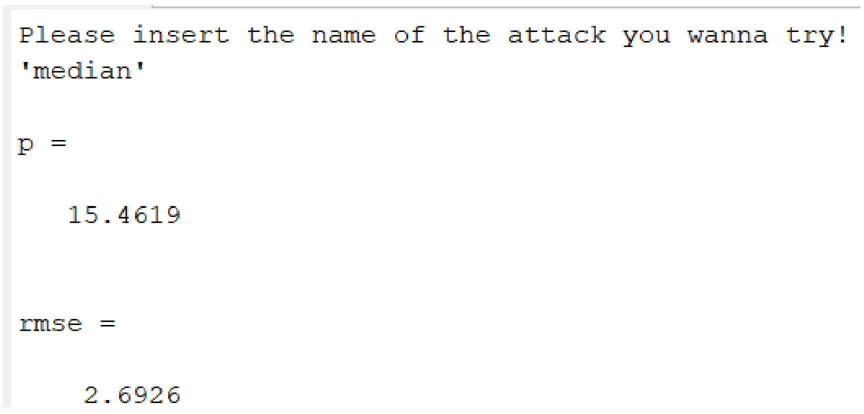
Attack	**Water marked image**	**Attacked image**	**Extracted logo**
Mean			

Median			
Crop			 Extracted watermark
Shear			
Rotation			

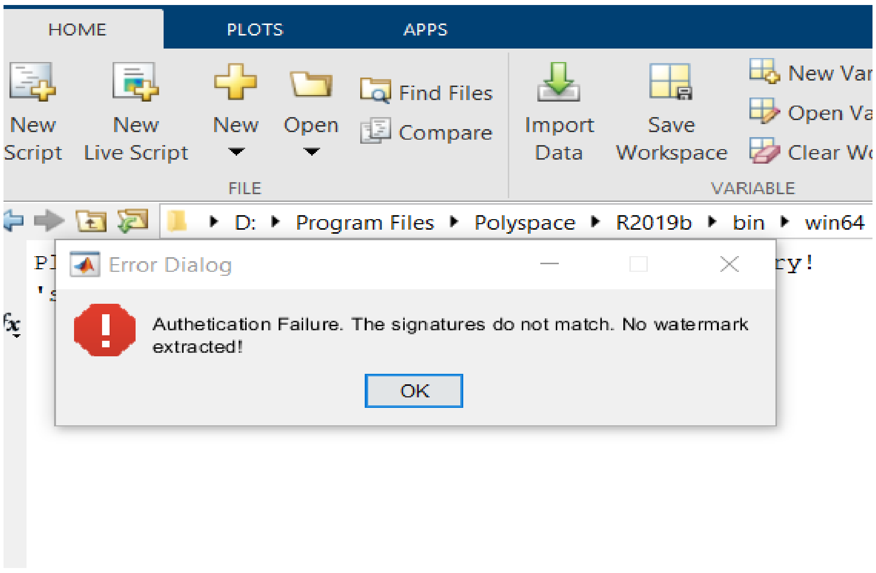
Noise			



```
Please insert the name of the attack you wanna try!
'noise'

p =

   22.9111


rmse =

    1.1421
```



```
Please insert the name of the attack you wanna try!
'median'

p =

   15.4619


rmse =

    2.6926
```



Authetication Failure. The signatures do not match. No watermark extracted!

Base papers / References:

1. Ranjeet Kumar Singh, Dillip Kumar Shaw, Sudhanshu Kumar Jha and Manish Kumar, "A DWT-SVD based multiple watermarking scheme for image based data security", Journal of Information and Optimization Sciences, Volume 39:1, 67-81, 2018. DOI: 10. 1080 /02522667.2017.1372153

2. Ankita Pareek and Poonam Sinha, "Image Data Authentication using Watermarking Scheme by DWT based Data Embedding Approach", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJIRCCE), Volume 4, Issue 12, 2015. DOI: 10.15662/IJAREEIE.2015.0412014.

3. Singh, Gulroz, and Mankirat Singh Lamba. "Efficient hardware implementationof image watermarking using DWT and AES algorithm." Systems Conference(NSC), 2015 39th National. IEEE, 2015.

4. Kwitt, Roland, Peter Meerwald, and Andreas Uhl. "Lightweight detection ofadditive watermarking in the DWT-domain." IEEE transactions on imageprocessing 20.2 (2011): 474-484.

5. Kang, Xiangui, et al. "A DWT-DFT composite watermarking scheme robust toboth affine transform and JPEG compression." IEEE transactions on circuitsand systems for video technology 13.8 (2003): 776-786.

6. Alifa D'Silva , Nayana Shenvi "Data Security Using SVD Based Digital Watermarking Technique" International Conference on Trends in Electronics and Informatics (ICEI 2017)

Digital watermarking Using DWT-SVD TECHNIQUE for images

DWT_SVD CODE

```
function [psnr_values, psnr2dB_values] = dwt_svd()
% clear workspace clear
all;
close all; clc;

%change directory
folder_name = uigetdir(pwd, 'Select Directory Where the .m Files Reside'); if( folder_name ~= 0
)
  if( strcmp(pwd, folder_name) == 0 )
                cd(folder_name);
end else
return; end

% read cover image & watermark logo
[cover_fname, cover_pthname] = ...
uigetfile('*.jpg; *.png; *.tif; *.bmp', 'Select the Cover Image'); if (cover_fname ~= 0)
cover_image = strcat(cover_pthname, cover_fname); cover_image = double(
rgb2gray( imread( cover_image ) ) ); cover_image = imresize(cover_image,
[512 512], 'bilinear'); else
return; end

[watermark_fname, watermark_pthname] = ...
uigetfile('*.jpg; *.png; *.tif; *.bmp', 'Select the Watermark Logo'); if (watermark_fname
~= 0)
watermark_logo = strcat(watermark_pthname, watermark_fname); watermark_logo = double(
im2bw( rgb2gray( imread( watermark_logo ) ) ) ); watermark_logo =
imresize(watermark_logo, [512 512], 'bilinear');
else return;
end

% Set constant variables
numOfKeys = 20; gaussian_plot
= true; print_figures = false;

if (gaussian_plot == false) secret_key = 3; %
random secret key

watermarked_image = watermark_embedding(cover_image, watermark_logo, ... secret_key,
print_figures);

watermark_extraction(watermarked_image, watermark_logo, secret_key, ... print_figures,
true);

else
% Pre-allocate matrices psnr_values =
zeros(1, numOfKeys);
        psnr2dB_values = zeros(1, numOfKeys);

decoded_correct = zeros(1, numOfKeys); decoded_wrong =
zeros(1, numOfKeys);
```

```
%           [m, n] = size(watermark_logo);
%           correctWatermarkDiff = zeros(1, m*n); wrongWatermarkDiff
= zeros(1, length(watermark_logo));

for key = 1:numOfKeys
% Embedding with the right key
              [watermarked_image, original_signature] = ...
watermark_embedding(cover_image, watermark_logo, key, print_figures);

watermarked_images_dir = strcat(pwd, '\watermarked_images\', num2str(key), '.png');
imwrite(uint8(watermarked_image), watermarked_images_dir, 'png');

% Transform the original watermrak (i.e. the original signature) to
% the interval [-1, 1]
original_signature(find(original_signature == 0)) = -1;

% Measure the degree of distortion of the original and the watermarked
% image using PSNR (dB)
  psnr_values(key) = psnr(cover_image, watermarked_image);
                psnr2dB_values(key) = pow2db(psnr_values(key));

              [~, ~, recon_sig_corr, LL4, HH4] = ...
watermark_extraction(watermarked_image, watermark_logo, key, ... print_figures,
true);

% transform the watermark_correct(i.e. the reconstructed_signature)
% and the watermark_wrong(i.e. the wrong reconstructed signature)
% to the interval [-1, 1]
recon_sig_corr(find(recon_sig_corr == 0)) = -1;

                LL4 = LL4(:)';
                HH4 = HH4(:)';
                LL4 = LL4 - mean(LL4); HH4 =
                HH4 - mean(HH4);
                comb_HH4_LL4 = [LL4 HH4];
[~, watermark_wrong, recon_sig_wrng, LL4wr, HH4wr] = ...
watermark_extraction(watermarked_image, watermark_logo, key+11, ... print_figures,
false);

  watermark_wrong(find(watermark_wrong == 0)) = -1; LL4wr =

                LL4wr(:)';
                HH4wr = HH4wr(:)';
                LL4wr = LL4wr - mean(LL4wr); HH4wr
                = HH4wr - mean(HH4wr);
                combwr_HH4_LL4 = [LL4wr HH4wr];


clear('watermark_correct', 'comb_HH4_LL4', 'watermark_wrong', ...
'combwr_HH4_LL4', 'original_signature', ...
'LL4', 'HH4', 'LL4wr', 'HH4wr');

% Clear workspace
clear('watermark_logo_row', 'watermark_logo_extracted_correct', ...
'watermark_logo_extracted_wrong', 'watermarked_image', ... 'watermark_correct',
'watermark_wrong', 'message_correct', ... 'message_wrong', 'watermarked_image');
```

```
end end

end
```

WATERMARK EMBEDDING

```
function [watermarked_image, signature] = watermark_embedding(cover_image,
watermark_logo, key, print_figures)

  % Apply Haar wavelet and decompose cover image into four sub-bands: [LL, HL,
    LH, HH] = dwt2(cover_image, 'haar');

% Apply SVD to HH (high frequency) band [Uh
ShVh] = svd(HH, 'econ');

% Watermark logo W is decomposed using SVD
[UwSwVw] = svd(watermark_logo, 'econ');

% Replace singular values of the HH band with the singular values of watermark Sh_diag = diag(Sh);
Sw_diag = diag(Sw);
if (length(watermark_logo) >= 256) Sh_diag(1:length(Sh), :) =
Sw_diag(1:length(Sh), :); elseif(length(watermark_logo) < 256)
Sh_diag(1:length(watermark_logo), :) = Sw_diag(1:length(watermark_logo), :); end
Sh(logical(eye(size(Sh)))) = Sh_diag;

% Signature generation algorithm
% Generate signature
signature = signature_generation(Uw, Vw, key);

% Signature embedding algorithm
% Embedd signature to cover image
LL_inv = signature_embedding(LL, signature, print_figures);

% Apply SVD to HH band which now holds the SV's of watermark logo
HH_modified = Uh * Sh * Vh';

% Apply inverse DWT with modified LL(LL_inv) & HH(HH_modified) band to
% obtain the watermarked image
watermarked_image = idwt2(LL_inv, HL, LH, HH_modified, 'haar');

if (print_figures == true) figure(1)
imshow(cover_image, []);
title('Cover image'); figure(2)
imshow(watermarked_image, []);
title('Watermarked image signed with secret key'); figure(3)
imshow(watermark_logo, []);
title('Watermark logo'); end

clear('LL', 'HL', 'LH', 'HH','LL_1', 'HL_1', 'LH_1', 'HH_1','LL_2', 'HL_2',
'LH_2', 'HH_2','LL_3', 'HL_3', 'LH_3', 'HH_3','LL_4', 'HL_4', 'LH_4', 'HH_4');

clear('Uh', 'Sh', 'Vh', 'Uw', 'Sw', 'Vw', 'Sh_diag', 'Sw_diag'); end
```

WATERMARK EXTRACTION

```
function [watermark_logo_extracted, generated_signature, reconstructed_signature, LLw_4, HHw_4] =
watermark_extraction(watermarked_image, watermark_logo, key, print_figures,
signature_authentication)

  % Using Haar wavelet, decompose the noisy watermarked image into four
   [LLwHLwLHwHHw] = dwt2(watermarked_image, 'haar');

% Further decompose LL band to the 4th level.
[LLw_1, HLw_1, LHw_1, HHw_1] = dwt2(LLw, 'haar');            % 1st step DWT
[LLw_2, HLw_2, LHw_2, HHw_2] = dwt2(LLw_1, 'haar');          % 2nd step DWT
[LLw_3, HLw_3, LHw_3, HHw_3] = dwt2(LLw_2, 'haar');          % 3rd step DWT
[LLw_4, HLw_4, LHw_4, HHw_4] = dwt2(LLw_3, 'haar');          % 4rth step DWT

% Clear workspace
clear('LLw_1', 'LLw_2', 'LLw_3','HLw_1', 'HLw_2', 'HLw_3', 'HLw_4', 'LHw_1',
'LHw_2', 'LHw_3', 'LHw_4','HHw_1', 'HHw_2', 'HHw_3','LLw', 'HLw', 'LHw');

% Apply SVD to watermark logo
[Uw_xSw_xVw_x] = svd(watermark_logo, 'econ');

% Generate signature using Uw&Vw matrices generated_signature =
signature_generation(Uw_x, Vw_x, key);

% Extract signature from LLw_4 & HHw_4 bands using all of the 512 coefficients
reconstructed_signature = signature_extraction(LLw_4, HHw_4, length(watermark_logo));

if (signature_authentication == true)

% Compare the 2 signatures
if( reconstructed_signature == generated_signature |
corr2(reconstructed_signature, generated_signature) > 0.7 )

% Apply SVD to HH band
              [UcwScwVcw] = svd(HHw, 'econ');

% Extract the singular values from HH band HH_singularValues
= zeros(length(watermark_logo)); Shh_diag =
diag(HH_singularValues);
Scw_diag = diag(Scw);
if (length(watermark_logo) >= 256)
Shh_diag(1:length(Scw), :) = Scw_diag; elseif
(length(watermark_logo) < 256)
Shh_diag(1:length(watermark_logo), :) = Scw_diag(1:length(watermark_logo), :); End

HH_singularValues(logical(eye(size(HH_singularValues)))) = Shh_diag;

% Construct the watermark using singular values and orthogonal matrices
% Uw and Vw obtained using SVD of original watermark
watermark_logo_extracted = Uw_x * HH_singularValues * Vw_x';

clear( 'Uw_x', 'Sw_x', 'Vw_x','Ucw', ',Scw', 'Vcw', 'HH_singularValues', 'Shh_diag',
'Scw_diag');

if (print_figures == true)
```

```
                    figure;
imshow(watermark_logo_extracted, []);
title('Extracted watermark');
end

else
errordlg('Authetication Failure. The signatures do not match. No watermark extracted!');
watermark_logo_extracted = zeros(length(watermark_logo),
length(watermark_logo));
return; end
else

% Apply SVD to HH band
        [UcwScwVcw] = svd(HHw, 'econ');

% Extract the singular values from HH band HH_singularValues
= zeros(length(watermark_logo)); Shh_diag =
diag(HH_singularValues);
Scw_diag = diag(Scw);

if (length(watermark_logo) >= 256)
Shh_diag(1:length(Scw), 1) = Scw_diag; elseif
(length(watermark_logo) < 256)
Shh_diag(1:length(watermark_logo), :) = Scw_diag(1:length(watermark_logo), :); end
HH_singularValues(logical(eye(size(HH_singularValues)))) = Shh_diag;

% Clear workspace
clear('Ucw', 'Scw', 'Vcw', 'Shh_diag', 'Scw_diag', 'Scw_random_diag');

% Construct the watermark using singular values and orthogonal matrices
% Uw and Vw obtained using SVD of original watermark
watermark_logo_extracted = Uw_x * HH_singularValues * Vw_x';

  % Clear workspace
        clear('HH_singularValues');

if (print_figures == true)
             figure;
imshow(watermark_logo_extracted, []);
title('Extracted watermark');
end end

end
```

SIGNATURE GENERATION FUNCTION

```
function signature = signature_generation(U, V, key)

% Proposed algorithm
% Sum the column of orthogonal matrices and create 1-D array

Usum = sum(U);
Vsum = sum(V);
```

```
% Based on the threshold, map the array values into corresponding binary digits Usum_threshold =
median(Usum); % threshold based on median value of each Uw&Vw matrix
Vsum_threshold = median(Vsum);

% transform Usummartix to binary using above threshold
Usum(find(Usum>Usum_threshold)) = 1;
Usum(find(Usum<Usum_threshold)) = 0;

% transform Vsummartix to binary using above threshold Vsum(find(Vsum>Vsum_threshold)) = 1;
Vsum(find(Vsum<Vsum_threshold)) = 0; clear('Usum_threshold',

'Vsum_threshold');

% XOR the 2 matrices to obtain 1-D array of dimension 1x512 UV_XOR =
bitxor(uint8(Usum), uint8(Vsum));

% Generate a PSRNG sequence using the key of dim. 1x512 and XOR it with the UV_XOR 1-D
array above
rand('seed', key);
% produce binary sequence to perform XOR with UVsum
binary_seq = randi([0 1], 1, length(UV_XOR));
signature = double( bitxor(uint8(UV_XOR), uint8(binary_seq)) ); clear('Usum',
'Vsum', 'UV_XOR', 'binary_seq');
end
```

SIGNATURE EMBEDDING FUNCTION

```
functionLL_inv = signature_embedding(LL, signature, print_figures)

% Using Haar wavelet, further decompose LL band to the 4th level. [LL_1, HL_1,
LH_1, HH_1] = dwt2(LL, 'haar');
[LL_2, HL_2, LH_2, HH_2] = dwt2(LL_1, 'haar'); [LL_3,
HL_3, LH_3, HH_3] = dwt2(LL_2, 'haar'); [LL_4, HL_4,
LH_4, HH_4] = dwt2(LL_3, 'haar');

% choose all in total 512 coefficients from LL_4 & HH_4
% reshape them to row vectors of size 1x256 LL_4 =
reshape(LL_4, 1, length(LL_4)^2); HH_4 =
reshape(HH_4, 1, length(HH_4)^2);

% concatenate into a larger row vector of size 1x512
combined_LL4_and_HH4_coef = [LL_4 HH_4];

% keep record of the index position of negatives to put back the sign in inverse process
negative_idxs = combined_LL4_and_HH4_coef(logical(combined_LL4_and_HH4_coef)) < 0;

% keep only the positive integer parts & separate the integer from the decimal
combined_LL4_and_HH4_coeff_pos = abs(combined_LL4_and_HH4_coef);
integer_part = fix(combined_LL4_and_HH4_coeff_pos);
fraction_part = abs(combined_LL4_and_HH4_coeff_pos - integer_part);

% Convert the integer part into the binary code of L=16 bits.
binary_coefficients = {};

for p = 1:length(integer_part)
```

```
binary_coefficients{p} = decimalToBinaryVector(integer_part(p), 16); end

% Replace the n-th bit (10th bit) position of the coefficient with signature bit for m =
1:length(signature)
for n = 1:16 if (n
== 10)
binary_coefficients{1, m}(n) = signature(m); end
end end
% and then convert the binary code to its decimal representation bin2decimal =
zeros(1, length(binary_coefficients));
for x = 1:length(binary_coefficients)
        bin2decimal(x) = binaryVectorToDecimal(double(binary_coefficients{1, x}));
end

% reconstruct orignal array from integer and decimal fraction parts bin2decimal =
bin2decimal + fraction_part;

% put back the negative signs
bin2decimal(find(negative_idxs == 1)) = -bin2decimal(find(negative_idxs == 1));

% Clear workspace
clear('LL_1', 'LL_2', 'LL_3', 'LL_4', 'HH_4',
'combined_LL4_and_HH4_coef','negative_idxs', 'combined_LL4_and_HH4_coeff_pos',
'integer_part','fraction_part', 'binary_coefficients', 'signature');

% reshape the modified LL_4 & HH_4 sub-bands to their original size 16x16 LL_4_modified
= bin2decimal(1:256);
HH_4_modified = bin2decimal(257:end); LL_4_modified =
reshape(LL_4_modified, 16, 16);
HH_4_modified = reshape(HH_4_modified, 16, 16);

% Clear workspace
clear('bin2decimal');

% Apply the inverse DWT with modified LL4 and HH4 band coefficients. LL_3_inv =
idwt2(LL_4_modified, HL_4, LH_4, HH_4_modified, 'haar'); LL_2_inv =
idwt2(LL_3_inv, HL_3, LH_3, HH_3, 'haar');
LL_1_inv = idwt2(LL_2_inv, HL_2, LH_2, HH_2, 'haar'); LL_inv =
idwt2(LL_1_inv, HL_1, LH_1, HH_1, 'haar');

% Clear workspace
clear('LL_1_inv', 'LL_2_inv', 'LL_3_inv', 'LL_4_modified','HL_1', 'HL_2', 'HL_3',
'HL_4','LH_1', 'LH_2', 'LH_3', 'LH_4','HH_1', 'HH_2', 'HH_3',
'HH_4_modified'); end
```

SIGNATURE EXTRACTION

```
functionreconstructed_signature = signature_extraction(LLw_4, HHw_4,
lengthOfWatermark)

% 2. Select all coefficient from LL4 and HH4 band reshape them to row vectors of size 1x256
LLw_4 = reshape(LLw_4, 1, length(LLw_4)^2);

HHw_4 = reshape(HHw_4, 1, length(HHw_4)^2);
```

```
% concatenate the above row vectors into a larger row vector of size 1x512
combined_LLw_4_and_HHw_4_coeff = [LLw_4 HHw_4];

% keep record of the index position of negatives to put back the sign in
% inverse process
negative_watermarked_idxs =
combined_LLw_4_and_HHw_4_coeff(logical(combined_LLw_4_and_HHw_4_coeff)) < 0;

% separate the integer from the decimal fraction combined_LLw_4_and_HHw_4_coeff_pos =
abs(combined_LLw_4_and_HHw_4_coeff); integer_part_of_watermarked_image =
fix(combined_LLw_4_and_HHw_4_coeff_pos); fraction_part_of_watermarked_image =
abs(combined_LLw_4_and_HHw_4_coeff_pos - integer_part_of_watermarked_image);

% Clear workspace
clear('LLw_4', 'HHw_4', 'combined_LLw_4_and_HHw_4_coeff',
'negative_watermarked_idxs', 'fraction_part_of_watermarked_image');

% Convert the integer part of selected coefficient into the binary code of L bits
binary_watermarked_coefficients = {};
for y = 1:length(combined_LLw_4_and_HHw_4_coeff_pos)
binary_watermarked_coefficients{y} =
decimalToBinaryVector(integer_part_of_watermarked_image(y), 16); end

% Clear workspace
clear('combined_LLw_4_and_HHw_4_coeff_pos','integer_part_of_watermarked_image');

% 3. Extract the n-th (10-th) bit from the coefficient to extract the signature. reconstructed_signature =
zeros(1, lengthOfWatermark);
for u = 1:lengthOfWatermark for v =
1:16
if (v == 10)
reconstructed_signature(u) = binary_watermarked_coefficients{1, u}(v); end
end end

% Clear workspace
clear('binary_watermarked_coefficients');

end
```

ATTACK

```
functionattacks()

clear   all;   close
all; clc;

watermarked_images_dir = fullfile( pwd, 'watermarked_images\*.png'); attacked_dir =
fullfile(pwd, 'attacked\');
logos_dir = fullfile(pwd, 'logos\');

numOfKeys = length( dir( watermarked_images_dir ) );
% numOfKeys = 100;

decoded_correct = zeros(1, numOfKeys);
decoded_wrong = zeros(1, numOfKeys); corr_coef
```

```
= zeros(1, numOfKeys);

% Set constant variables
print_figures = false;
authentication = false;
attack = input('Please insert the name of the attack you wanna try!\n'); if( ~isempty( dir(

watermarked_images_dir ) ) )

        [watermark_logo_fname, watermark_pthname] = ...
uigetfile('*.jpg; *.png; *.tif; *.bmp', 'Select the Watermark Logo'); if
(watermark_logo_fname ~= 0)
watermark_logo = strcat(watermark_pthname, watermark_logo_fname); watermark_logo =
double( im2bw( rgb2gray( imread( watermark_logo ) ) ) ); watermark_logo =
imresize(watermark_logo, [512 512], 'bilinear');
else return;
end

for key = 1:numOfKeys
watermarked_image = imread( fullfile( pwd, '\watermarked_images\',
strcat(num2str(key), '.png') ) );

switch( lower(attack) )
case'mean'
attackedImage = meanAttack(watermarked_image);
%p1 = psnr(watermarked_image,attackedImage)
imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');

case'median'
attackedImage = medianAttack(watermarked_image);
%p1 = psnr(watermarked_image,attackedImage)
imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');

case'noise'
attackedImage = noiseAttack(watermarked_image);
%p1 = psnr(watermarked_image,attackedImage)
imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');

case'rotation'
attackedImage = rotationAttack(watermarked_image, 45);
%p1 = psnr(watermarked_image,attackedImage)
imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');

case'shear'
attackedImage = shearAttack(watermarked_image); attackedImage
= imresize(attackedImage, [512 512]);
%p1 = psnr(watermarked_image,attackedImage)
imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');
%p1 = psnr(watermarked_image,attackedImage)
case'crop'
attackedImage = cropAttack(watermarked_image); attackedImage
= imresize(attackedImage, [512 512]);
%p1 = psnr(watermarked_image,attackedImage)

imwrite(uint8(attackedImage), fullfile( attacked_dir, strcat(num2str(key), '.png') ), 'png');

otherwise
```

```
errordlg('Please specify an attack mode!'); end
              [message_correct, ~, recon_sig_corr, LL4, HH4] = ...
watermark_extraction(double(attackedImage), watermark_logo, key, ... print_figures,
authentication);

imwrite(message_correct, fullfile( logos_dir, strcat(num2str(key), '.png') ), 'png');

% Check correlation coefficient between original message (watermark_logo)
% and the message_correct (watermark_logo after attack) after attack correlation =
corrcoef(watermark_logo, message_correct); corr_coef(key) = correlation(2, 1);

  recon_sig_corr(find(recon_sig_corr == 0)) = -1; LL4 = LL4(:)';
                HH4 = HH4(:)';
                LL4 = LL4 - mean(LL4); HH4 =
                HH4 - mean(HH4);
                comb_HH4_LL4 = [LL4 HH4];

% Detection with the wrong key
              [~, watermark_wrong, recon_sig_wrng, LL4wr, HH4wr] = ...
watermark_extraction(double(attackedImage), watermark_logo, key+11, ... print_figures,
authentication);

  watermark_wrong(find(watermark_wrong == 0)) = -1; LL4wr =

                LL4wr(:)';
                HH4wr = HH4wr(:)';
                LL4wr = LL4wr - mean(LL4wr); HH4wr
                = HH4wr - mean(HH4wr);
                combwr_HH4_LL4 = [LL4wr HH4wr];


% Clear workspace
clear('meanAttackedImage', 'medianAttackedImage', 'noiseAttackedImage', ...
'rotationAttackedImage', 'shearAttackedImage', 'cropAttackedImage', ... 'message_correct',
'recon_sig_corr', 'LL4', 'HH4', 'comb_HH4_LL4', ... 'watermark_wrong', 'recon_sig_wrng',
'LL4wr', 'HH4wr', 'combwr_HH4_LL4');

end end

end
```

PSNR FUNCTION

```
function p = psnr(x, y, vmax)
% psnr - compute the Peack Signal to Noise Ratio, defined by :
%            PSNR(x,y) = 10*log10( max(max(x),max(y))^2 / |x-y|^2 ).
%
ifnargin<3
         m1 = max( abs(x(:)) );
       m2 = max( abs(y(:)) ); vmax =
max(m1,m2);
end
```

```
d = mean( (x(:)-y(:)).^2 ); rmse =
sqrt(d);
p = 10*log10(double(vmax^2)/d );
display(p)
display(rmse) end
```

MEAN ATTACK FUNCTION

```
functionmeanImageAttacked = meanAttack(watermarked_image) h = fspecial('average',
[5, 5]);
meanImageAttacked = filter2(h, watermarked_image); end
```

MEDIAN ATTACK FUNCTION

```
functionmedianImageAttacked = medianAttack(watermarked_image) medianImageAttacked
= medfilt2(watermarked_image);
p1 = psnr(watered_image, medianImageAttacked) end
```

CROP ATTACK FUNCTION

```
functioncropImageAttacked = cropAttack(watermarked_image) cropImageAttacked =
imcrop(watermarked_image, [75 68 340 380]); end
```

NOISE ATTACK FUNCTION

```
functionnoiseImageAttacked = noiseAttack(watermarked_image) noiseImageAttacked =
imnoise(watermarked_image, 'salt & pepper', 0.01); p1 = psnr(watered_image, noiseImageAttacked)
end
```

SHEAR ATTACK FUNCTION

```
functionshearImageAttacked = shearAttack(watermarked_image) tformImage =
maketform('affine', [1 0 0; 0.5 1 0; 0 0 1]); shearImageAttacked =
imtransform(watermarked_image, tformImage); end
```

ROTATION ATTACK FUNCTION

```
functionrotationImageAttacked = rotationAttack(watermarked_image, angle)
rotationImageAttacked = imrotate(watermarked_image, angle, 'crop');
end
```