A THESIS REPORT

ON

# "Local Prediction Based Difference Expansion Reversible Watermarking"

Submitted in partial fulfillment of the requirement for the award of the degree

of

Master of Technology

In

Signal Processing & Digital Design

Submitted by

VINOD KUMAR SISODIA

(2K13/SPD/26)

Under the Supervision of

Mr. Jeebananda Panda

(Associate Professor)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**DELHI TECHNOLOGICAL UNIVERSITY**

SHAHBAD DAULATPUR, DELHI -110042, INDIA

JULY, 2016

# CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation, entitled "**Local prediction based difference expansion reversible watermarking**", in partial fulfillment of the requirements for the award of the degree of Master of Technology in **Signal Processing & Digital Design** and submitted to the **Department of Electronics and Communication Engineering** of **Delhi Technological University,** New Delhi is an authentic record of my own work carried out during the period from January 2016 to June 2016 under the supervision of **Mr. Jeebananda Panda** , Associate Professor, ECE Department.

Vinod Kumar Sisodia
2K13/SPD/26

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Mr. Jeebananda Panda
Associate Professor
ECE Department
Delhi Technological University

# CERTIFICATE

This is to certify that the dissertation title "**Local prediction based difference expansion reversible watermarking**" submitted by **Vinod Kumar Sisodia**, Roll. No. **2K13/SPD/26**, in partial fulfillment for the award of degree of Master of Technology in Signal Processing & Digital Design at **Delhi Technological University, Delhi**, is a bonafide record of student's own work carried out by him under my supervision and guidance in the academic session 2015-16. To the best of my belief and knowledge the matter embodied in dissertation has not been submitted for the award of any other degree or certificate in this or any other university or institute.

| | |
|---|---|
| **Dr. S Indu** | **Mr. Jeebananda Panda** |
| Head of Department | Associate Professor |
| ECE Department | ECE Department |
| Delhi Technological University | Delhi Technological University |

# ACKNOWLEDGEMENT

I feel great pleasure in expressing my regards and gratitude to my mentor **Mr. Jeebananda Panda**, Associate Professor, **Electronics and Communication Department**, **Delhi Technological University**, New Delhi for giving me the guidance and opportunity to work on above mentioned thesis work.

I am also deeply grateful to his support, insightful guidance, understanding and encouragement.

I am grateful to them for their generous guidance at every stage of the project. Their advice, suggestions and encouragement has let me through this dissertation

I also offer my sincere thanks to **Dr. S Indu,** Head of Department (ECE), and Delhi Technological University for introducing me to the world of **Signal Processing and Digital Design**.

Vinod Kumar Sisodia

2K13/SPD/26)

M.Tech. (SPDD)

(PART TIME)

Department of Electronics & Communication Engineering,

Delhi Technological University, Delhi-110042

# ABSTRACT

The recent advent in the field of multimedia proposed a many facilities in transport, transmission and manipulation of data. Along with this advancement of facilities there are larger threats in authentication of data, its licensed use and protection against illegal use of data. A lot of digital image watermarking techniques have been designed and implemented to stop the illegal use of the digital multimedia images. This paper compares the different attacks on extracted watermark. The comparison (PSNR, MSE, NC) of original image v/s watermarked image and extracted watermarked image before and after various attacks of the watermarked images has been calculated and verified on the parameters of PSNR (Peak Signal to Noise Ratio), MSE (Mean Square Error) and NC (Normal Correlation)

In this paper the image is watermarked using native prediction in distinction enlargement reversible watermarking. For every pixel, a least square predictor is computed on a block focused on the pixel and also the corresponding prediction error is distended. An equivalent predictor is recovered at detection with none further data. Experimental results are provided

Terms – Reversible watermarking, adaptive prediction, least square predictors, Salt and Pepper Noise, Gaussian Noise, Wiener Filter

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

## 1 INTRODUCTION

The proposed local prediction based reversible watermarking has been implemented. For each pixel, the least square predictor in a square block centered on the pixel is computed. The scheme is designed to allow the recovery of the same predictor at detection, without any additional information. The Proposed approach is applied on various test sample images to get the PSNR, MSE and Normal correlation on the 8x8 block size of local predictor. The results obtained so far show that the local prediction based schemes outperform their counterparts global least square and fixed prediction. The results have been obtained by using the local prediction with a basic difference expansion scheme with simple threshold control, histogram shifting and flag bits.

Also the attacks on watermarked image are shown and comparison is calculated of (PSNR, MSE, NC) between Original Image V/S Watermark image (No attack) and also between Watermarked image v/s Extracted Watermark image (after attacks)

The recent advent in the field of multimedia proposed a many facilities in transport, transmission and manipulation of data. Along with this advancement of facilities there are larger threats in authentication of data, its licensed use and protection against illegal use of data. A lot of digital image watermarking techniques have been designed and implemented to stop the illegal use of the digital multimedia images. This paper compares the different attacks on extracted watermark. The comparison (PSNR, MSE, NC) of original image v/s watermarked image and extracted watermarked image before and after various attacks of the watermarked images has been calculated and verified on the parameters of PSNR (Peak Signal to Noise Ratio), MSE (Mean Square Error) and NC (Normal Correlation)

### 1.1Digital watermarking

A digital watermark is a kind of marker covertly embedded in a noise-tolerant signal such as an audio, video or image data. It is typically used to identify ownership of the copyright of such signal. "Watermarking" is the process of hiding digital information in a carrier signal; the hidden information should, but does not need to, contain a relation to the carrier signal. Digital watermarks may be used to verify the authenticity or integrity of the carrier signal or to show the identity of its owners. It is prominently used for tracing copyright infringements and for banknote authentication.

Like traditional watermarks, digital watermarks are only perceptible under certain conditions, i.e. after using some algorithm, and imperceptible otherwise. If a digital watermark distorts the carrier signal in a way that it becomes perceivable, it is of no use. Traditional Watermarks may be applied to visible media (like images or video), whereas in digital watermarking, the signal may be audio, pictures, video, texts or 3D models. A signal may carry several different watermarks at the same time. Unlike metadata that is added to the carrier signal, a digital watermark does not change the size of the carrier signal.

The needed properties of a digital watermark depend on the use case in which it is applied. For marking media files with copyright information, a digital watermark has to be rather robust

against modifications that can be applied to the carrier signal. Instead, if integrity has to be ensured, a fragile watermark would be applied.
Both steganography and digital watermarking employ steganographic techniques to embed data covertly in noisy signals. But whereas steganography aims for imperceptibility to human senses, digital watermarking tries to control the robustness as top priority.
Since a digital copy of data is the same as the original, digital watermarking is a passive protection tool. It just marks data, but does not degrade it or control access to the data.
One application of digital watermarking is source tracking. A watermark is embedded into a digital signal at each point of distribution. If a copy of the work is found later, then the watermark may be retrieved from the copy and the source of the distribution is known. This technique reportedly has been used to detect the source of illegally copied movies

### 1.1.1 History

The term "Digital Watermark" was coined by Andrew Tirkel and Charles Osborne in December 1992. The first successful embedding and extraction of a steganographic spread spectrum watermark was demonstrated in 1993 by Andrew Tirkel, Charles Osborne and Gerard Rankin.[3]
Watermarks are identification marks produced during the paper making process. The first watermarks appeared in Italy during the 13th century, but their use rapidly spread across Europe. They were used as a means to identify the papermaker or the trade guild that manufactured the paper. The marks often were created by a wire sewn onto the paper mold. Watermarks continue to be used today as manufacturer's marks and to prevent forgery.

### 1.1.2 Applications

Digital watermarking may be used for a wide range of applications, such as:

- Copyright protection
- Source tracking (different recipients get differently watermarked content)
- Broadcast monitoring (television news often contains watermarked video from international agencies)
- Video authentication
- Software crippling on screen casting programs, to encourage users to purchase the full version to remove it.

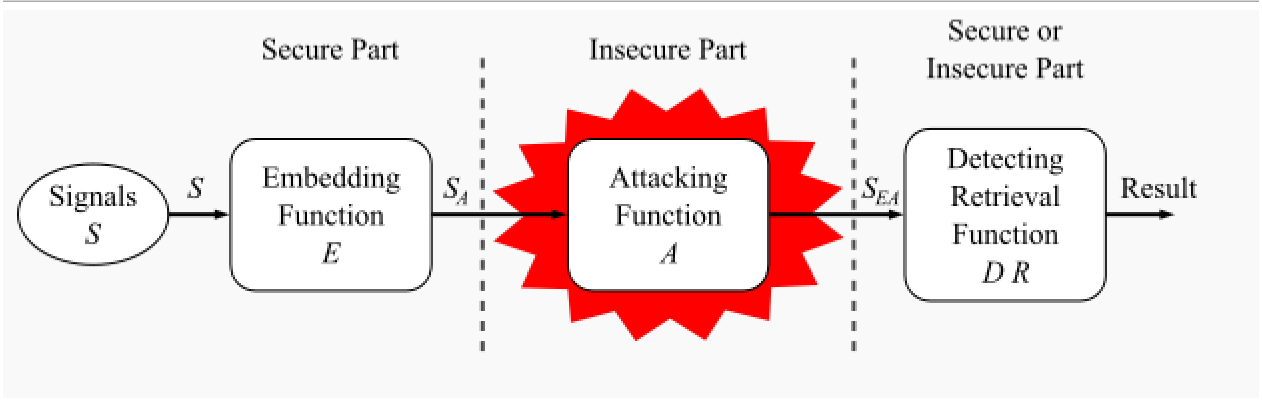


***Figure 1 Digital watermarking life-cycle phases***

General digital watermark life-cycle phases with embedding-, attacking-, and detection and retrieval functions

The information to be embedded in a signal is called a digital watermark, although in some contexts the phrase digital watermark means the difference between the watermarked signal and the cover signal. The signal where the watermark is to be embedded is called the host signal. A watermarking system is usually divided into three distinct steps, embedding, attack, and detection. In embedding, an algorithm accepts the host and the data to be embedded, and produces a watermarked signal.

Then the watermarked digital signal is transmitted or stored, usually transmitted to another person. If this person makes a modification, this is called an attack. While the modification may not be malicious, the term attack arises from copyright protection application, where third parties may attempt to remove the digital watermark through modification. There are many possible modifications, for example, lossy compression of the data (in which resolution is diminished), cropping an image or video or intentionally adding noise.

Detection (often called extraction) is an algorithm which is applied to the attacked signal to attempt to extract the watermark from it. If the signal was unmodified during transmission, then the watermark still is present and it may be extracted. In robust digital watermarking applications, the extraction algorithm should be able to produce the watermark correctly, even if the modifications were strong. In fragile digital watermarking, the extraction algorithm should fail if any change is made to the signal.

## 1.1.3 Classification

A digital watermark is called robust with respect to transformations if the embedded information may be detected reliably from the marked signal, even if degraded by any number of transformations. Typical image degradations are JPEG compression, rotation, cropping, additive noise, and quantization. For video content, temporal modifications and MPEG compression often are added to this list. A digital watermark is called imperceptible if the watermarked content is perceptually equivalent to the original, unwatermarked content. In general, it is easy to create either robust watermarks—or—imperceptible watermarks, but the creation of both robust—and—imperceptible watermarks has proven to be quite challenging. Robust imperceptible watermarks have been proposed as a tool for the protection of digital content, for example as an embedded no-copy-allowed flag in professional video content.

**Digital watermarking techniques may be classified in several ways**

### 1.1.3.1 Robustness

A digital watermark is called "fragile" if it fails to be detectable after the slightest modification. Fragile watermarks are commonly used for tamper detection (integrity proof). Modifications to an original work that clearly are noticeable commonly are not referred to as watermarks, but as generalized barcodes.

A digital watermark is called semi-fragile if it resists benign transformations, but fails detection after malignant transformations. Semi-fragile watermarks commonly are used to detect malignant transformations.

A digital watermark is called robust if it resists a designated class of transformations. Robust watermarks may be used in copy protection applications to carry copy and no access control information.

#### ***1.1.3.2 Perceptibility***

A digital watermark is called imperceptible if the original cover signal and the marked signal are perceptually indistinguishable.
A digital watermark is called perceptible if its presence in the marked signal is noticeable (e.g. Digital On-screen Graphics like a Network Logo, Content Bug, Codes, Opaque images). On videos and images, some are made transparent/translucent for convenience for people due to the fact that they block portion of the view.
This should not be confused with perceptual, that is, watermarking which uses the limitations of human perception to be imperceptible.

#### ***1.1.3.3 Capacity***

The length of the embedded message determines two different main classes of digital watermarking schemes:

- The message is conceptually zero-bit long and the system is designed in order to detect the presence or the absence of the watermark in the marked object. This kind of watermarking scheme is usually referred to as zero-bit or presence watermarking schemes. Sometimes, this type of watermarking scheme is called 1-bit watermark, because a 1 denotes the presence (and a 0 the absence) of a watermark.
- The message is an n-bit-long stream, with or and is modulated in the watermark. These kinds of schemes usually are referred to as multiple-bit watermarking or non-zero-bit watermarking schemes.

### **1.1.4 Embedding method**

A digital watermarking method is referred to as spread-spectrum if the marked signal is obtained by an additive modification. Spread-spectrum watermarks are known to be modestly robust, but also to have a low information capacity due to host interference.
A digital watermarking method is said to be of quantization type if the marked signal is obtained by quantization. Quantization watermarks suffer from low robustness, but have a high information capacity due to rejection of host interference.
A digital watermarking method is referred to as amplitude modulation if the marked signal is embedded by additive modification which is similar to spread spectrum method, but is particularly embedded in the spatial domain.

### **1.1.5 Evaluation and benchmarking**

The evaluation of digital watermarking schemes may provide detailed information for a watermark designer or for end-users, therefore, different evaluation strategies exist. Often used by a watermark designer is the evaluation of single properties to show, for example, an improvement. Mostly, end-users are not interested in detailed information. They want to know if a given digital watermarking algorithm may be used for their application scenario, and if so, which parameter sets seems to be the best.

### 1.1.6 Reversible data hiding

Reversible data hiding is a technique which enables images to be authenticated and then restored to their original form by removing the digital watermark and replacing the image data that had been overwritten. This would make the images acceptable for legal purposes. The US Army also is interested in this technique for authentication of reconnaissance images.

### 1.1.7 Watermarking for relational databases

Digital watermarking for relational databases has emerged as a candidate solution to provide copyright protection, tamper detection, traitor tracing, and maintaining integrity of relational data. Many watermarking techniques have been proposed in the literature to address these purposes. A survey of the current state-of-the-art and a classification of the different techniques according to their intent, the way they express the watermark, the cover type, granularity level, and verifiability were published in 2010 by Halder et al. in the Journal of Universal Computer Science.

### 1.1.8 Reversible watermarking

A reversible watermarking algorithm with very high data-hiding capacity has been developed for color images. The algorithm allows the watermarking process to be reversed, which restores the exact original image. The algorithm hides several bits in the difference expansion of vectors of adjacent pixels. The required general reversible integer transform and the necessary conditions to avoid underflow and overflow are derived for any vector of arbitrary length. Also, the potential payload size that can be embedded into a host image is discussed, and a feedback system for controlling this size is developed. In addition, to maximize the amount of data that can be hidden into an image, the embedding algorithm can be applied recursively across the color components. Simulation results using spatial triplets, spatial quads, cross-color triplets, and cross-color quads are presented and compared with the existing reversible watermarking algorithms. These results indicate that the spatial, quad-based algorithm allows for hiding the largest payload at the highest signal-to-noise ratio.

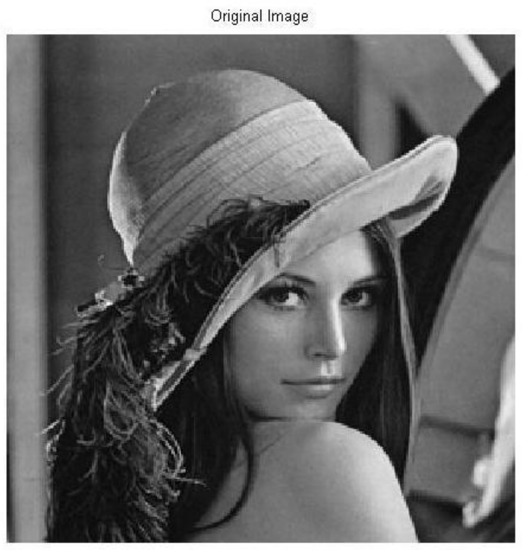
# CHAPTER 2

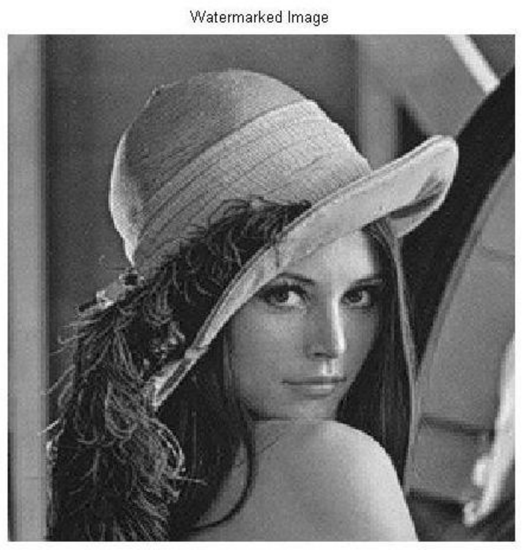
# (Implementation)

## 2.1 Attacks on Watermarked Image of lenna

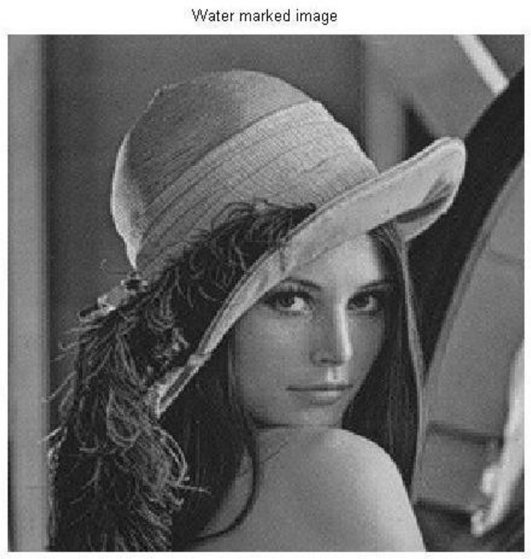
1. No attack
2. Blurr attack
3. Contrast attack
4. Sharpness attack
5. Cropping attack
6. Rotation attack
7. Low Pass Filter attack
8. High Pass Filter attack
9. Salt and Pepper Noise attack
10. Gaussian Noise attack
11. Median Filtering attack
12. Average Filter attack
13. Wiener filter attack
14. Jpeg Compression attack
15. Histogram Equalization attack

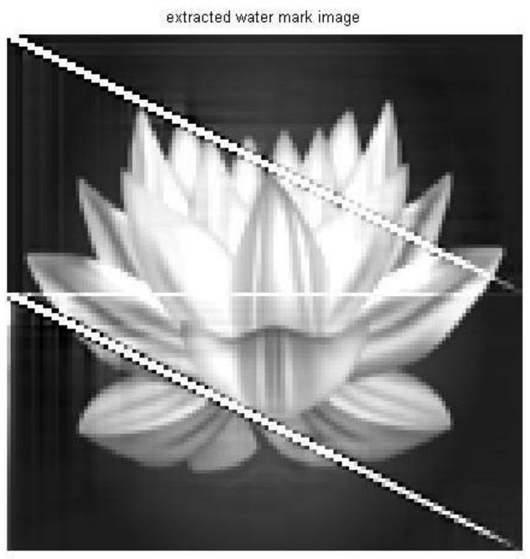
## 2.1.1 No attack





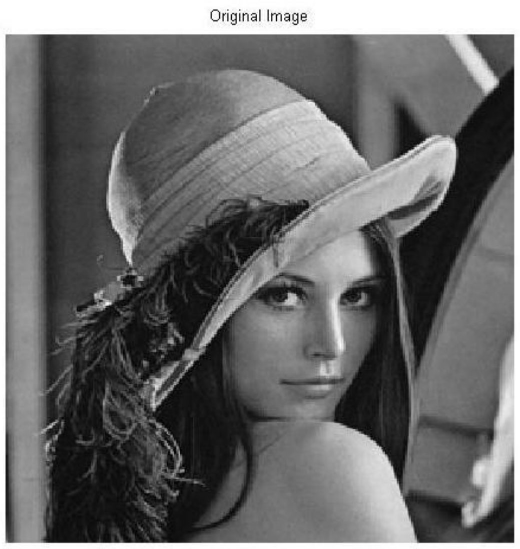
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

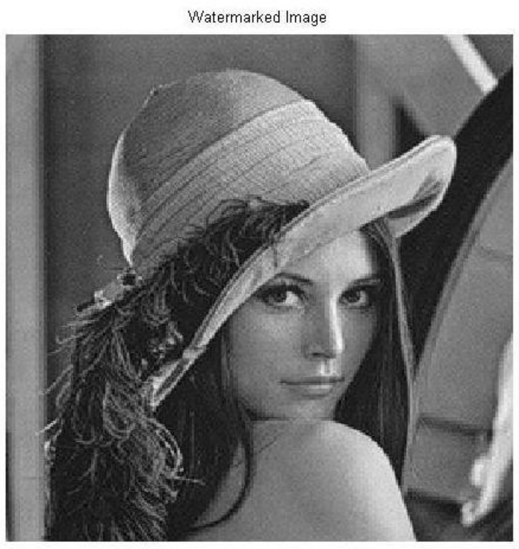




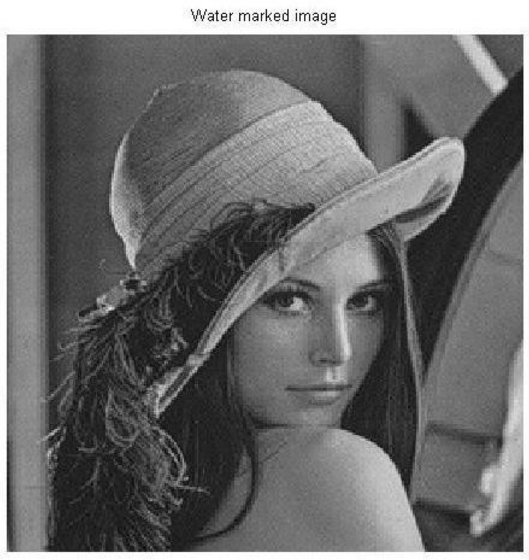
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 40814.375306<br><br>Normal Correlation = 0.990348<br><br>PSNR = 32.715052 |
|---|---|

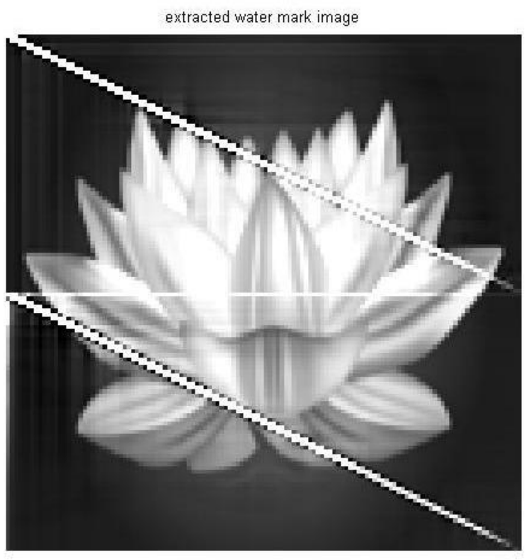
## 2.1.2 Blurr attack





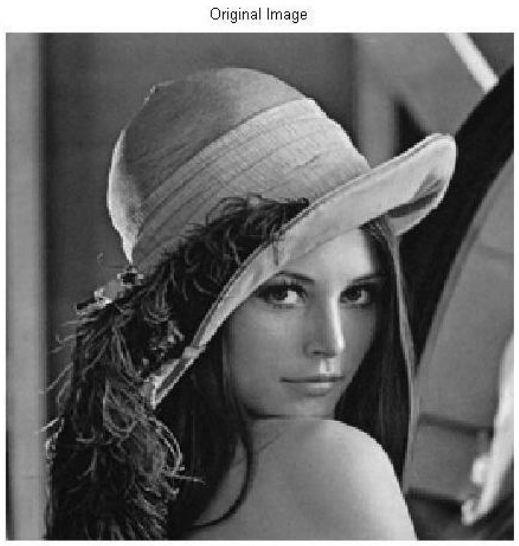
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |

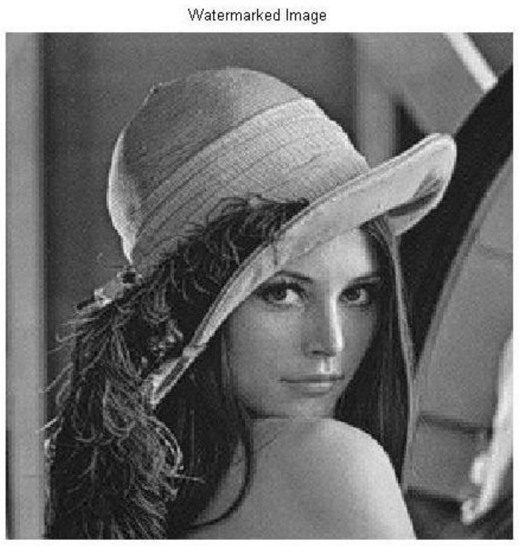




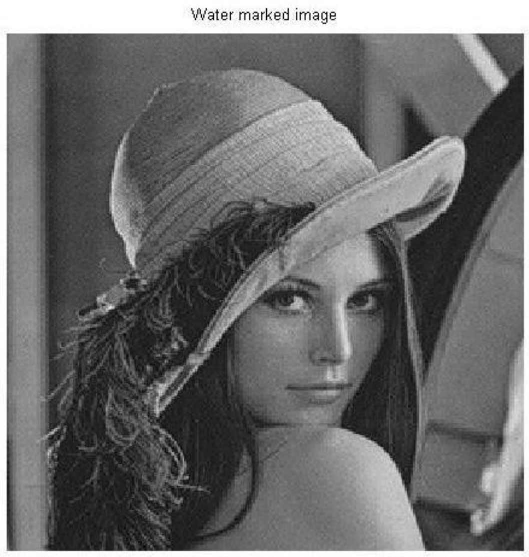
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 166525.658897<br><br>Normal Correlation = 0.950716<br><br>PSNR = 32.715052 |

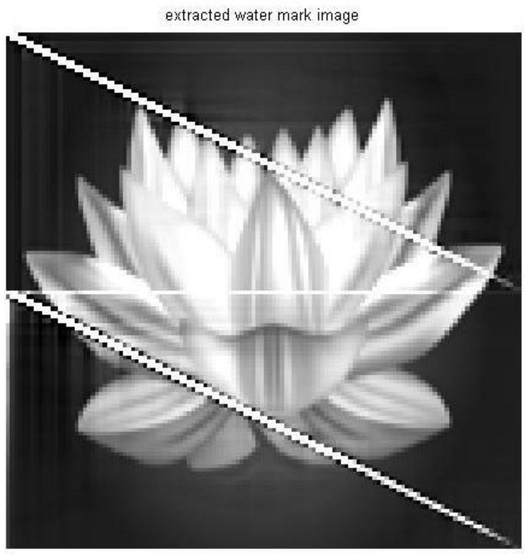
## 2.1.3 Contrast Adjustment  attack





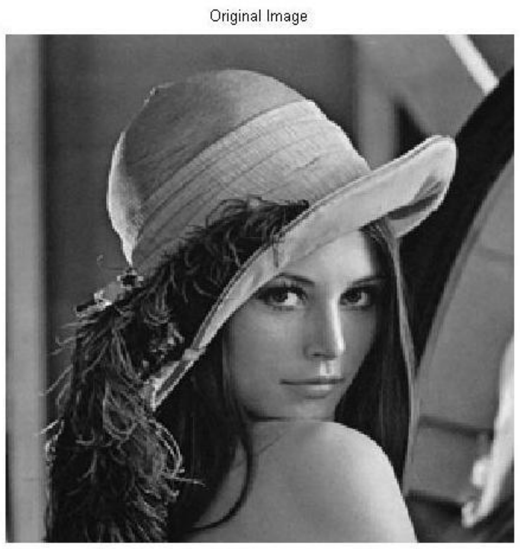
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

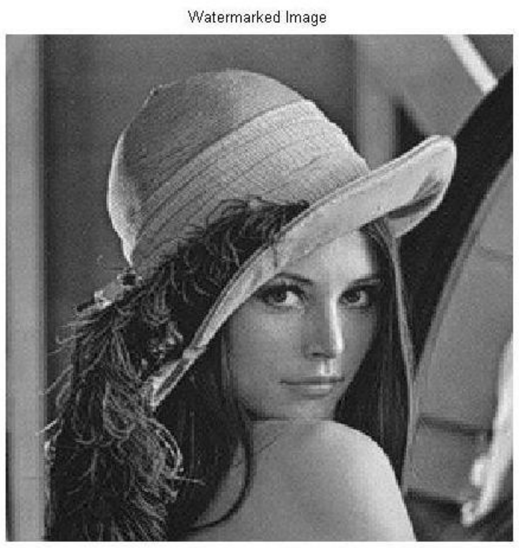




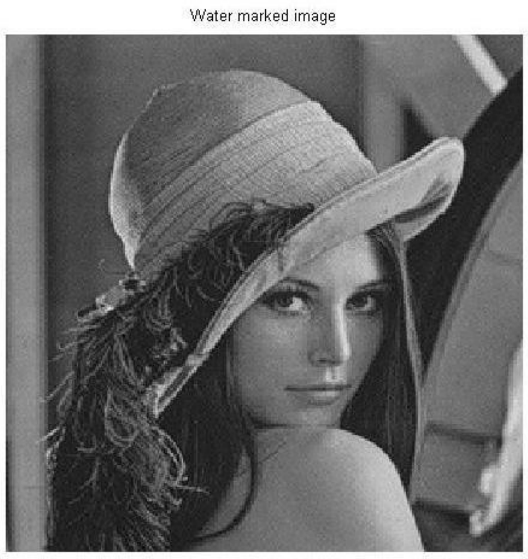
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 518807.516512<br><br>Normal Correlation = 0.029910<br><br>PSNR = 32.715052 |
|---|---|

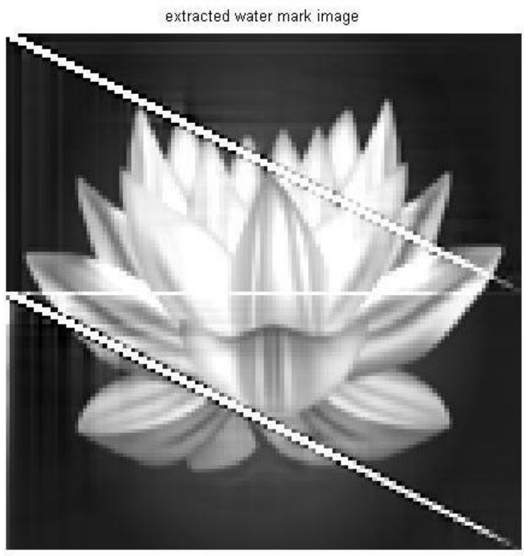
### 2.1.4 Sharpness attack





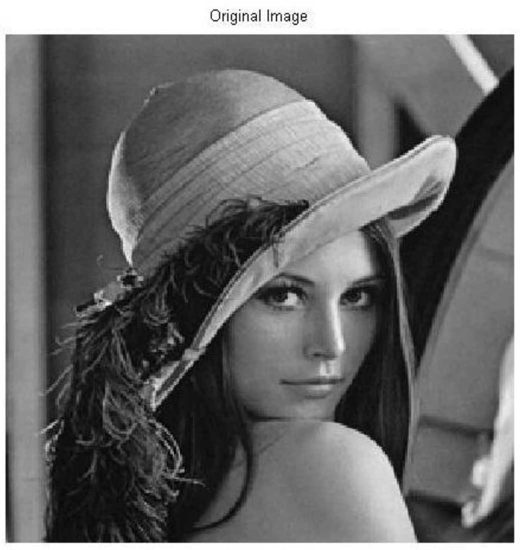
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

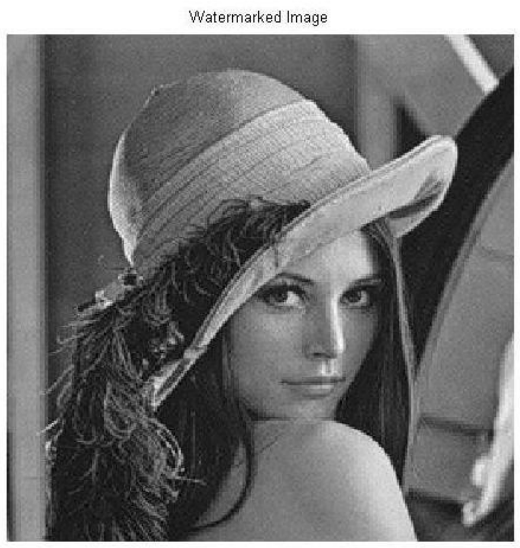




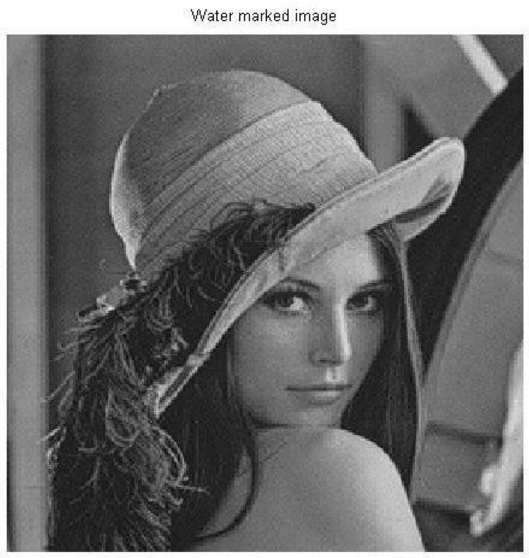
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 5428372.871539<br><br>Normal Correlation = 0.970145<br><br>PSNR = 32.715052 |
|---|---|

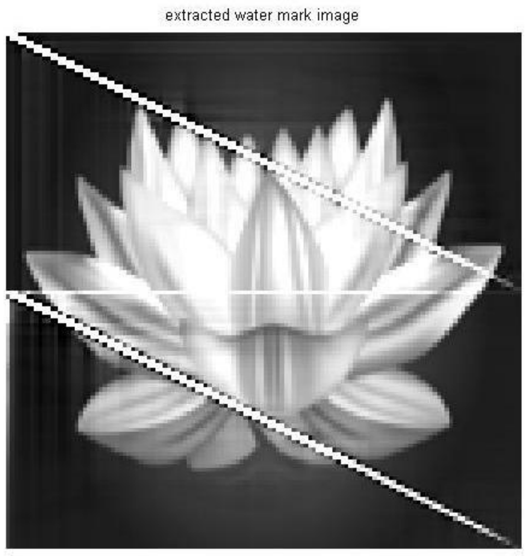
## 2.1.5 Cropping attack





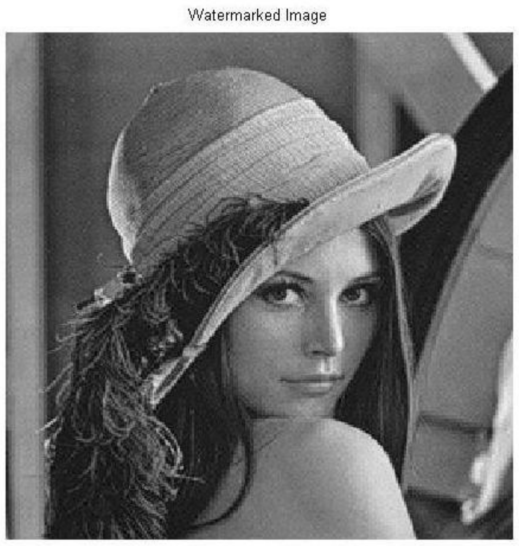
| **Calculating (PSNR, MSE, NC)**<br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br>Normal Correlation = 1.000125<br>PSNR = 32.715052 |
|---|---|



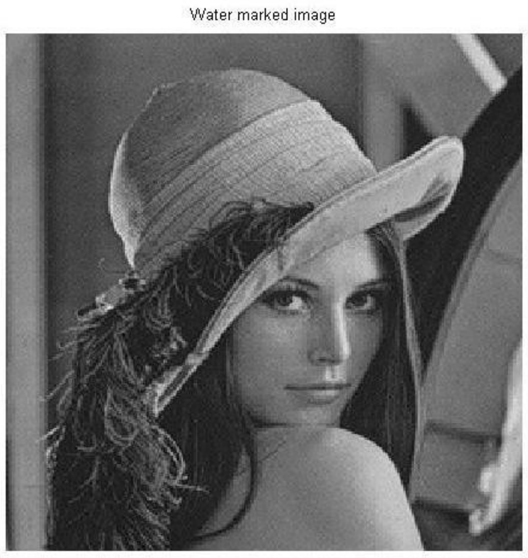


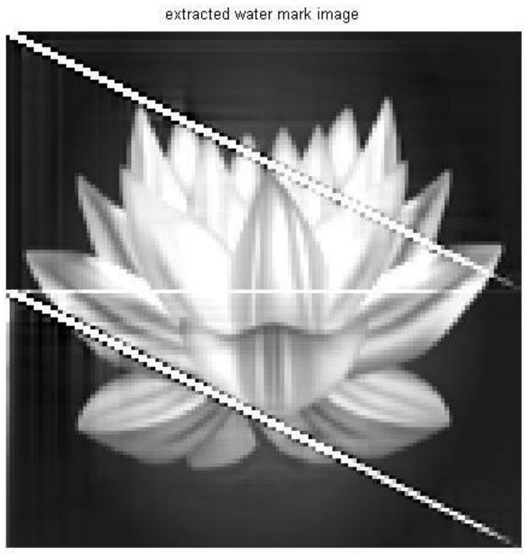
| **Calculating (PSNR, MSE, NC)**<br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 2806.179273<br>Normal Correlation = 0.977702<br>PSNR = 32.715052 |
|---|---|

## 2.1.6 Rotation attack



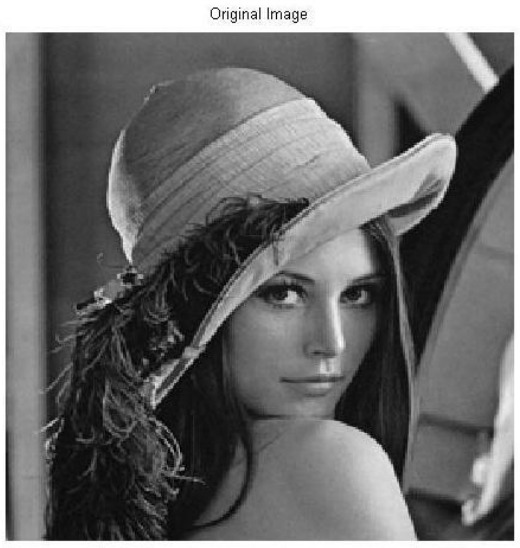
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

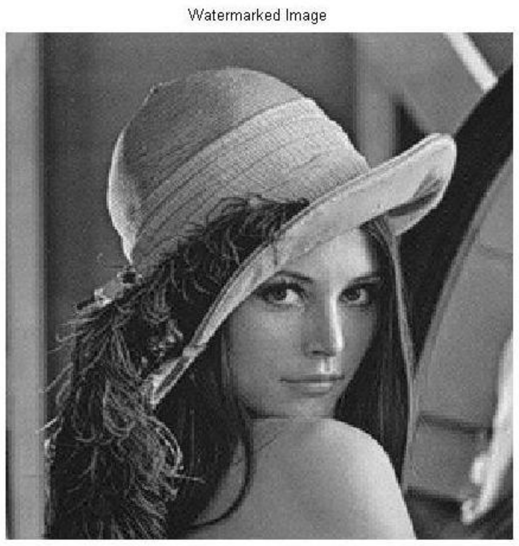




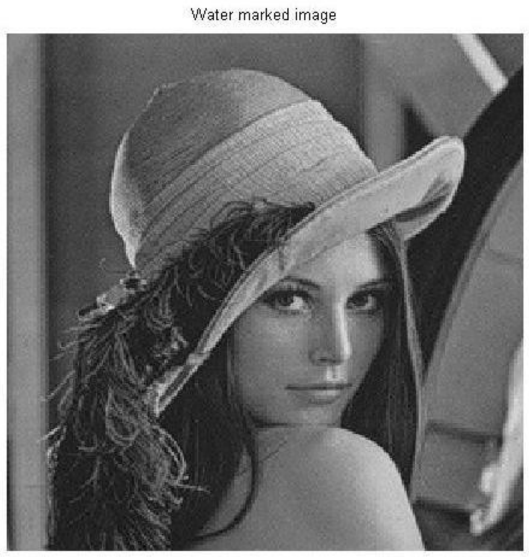
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 7508.493995<br><br>Normal Correlation = 0.972933<br><br>PSNR = 32.715052 |
|---|---|

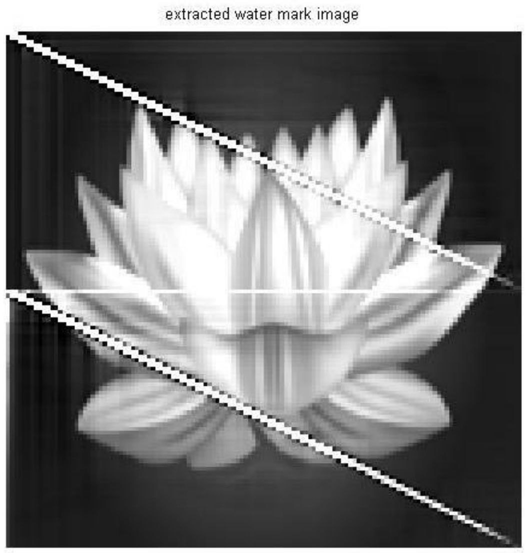
### **2.1.7 Low pass filter attack**





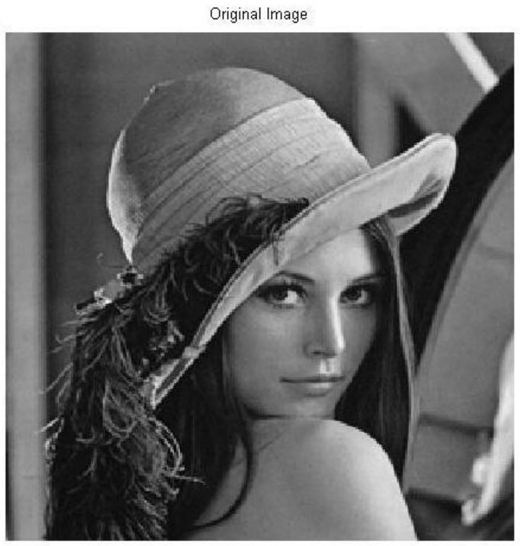
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

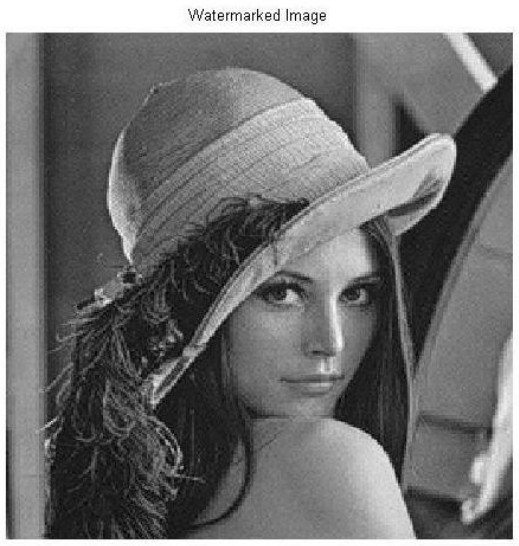




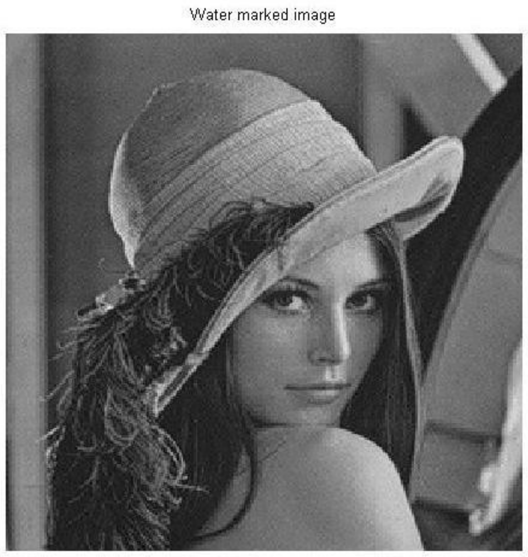
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 166525.658897<br><br>Normal Correlation = 0.950716<br><br>PSNR = 32.715052 |
|---|---|

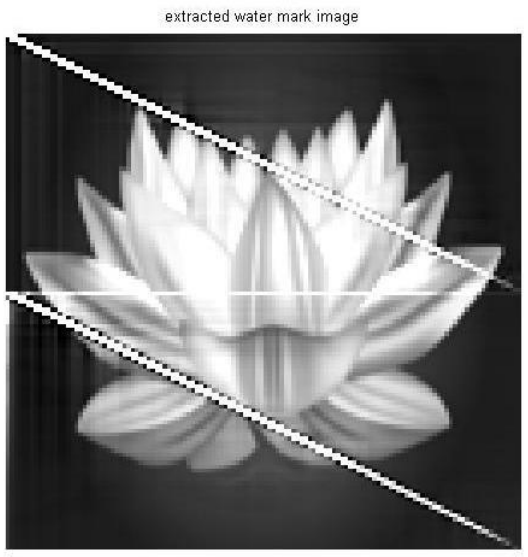
### 2.1.8 High Pass Filter attack





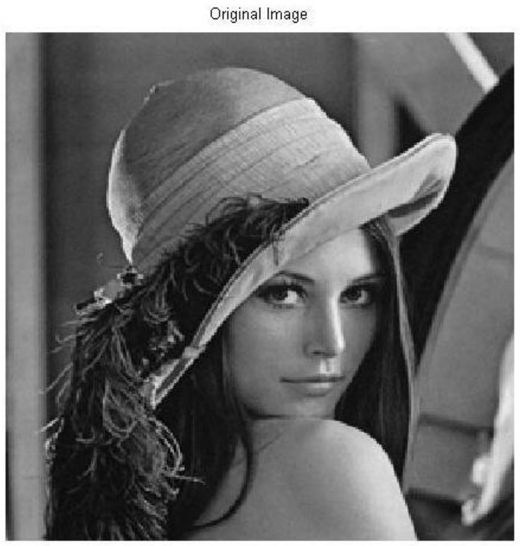
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |

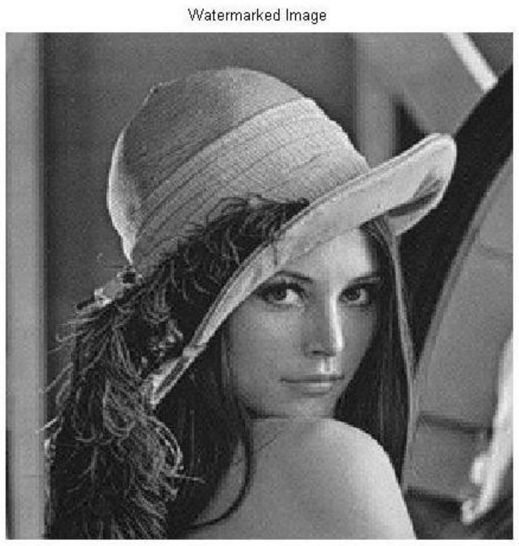




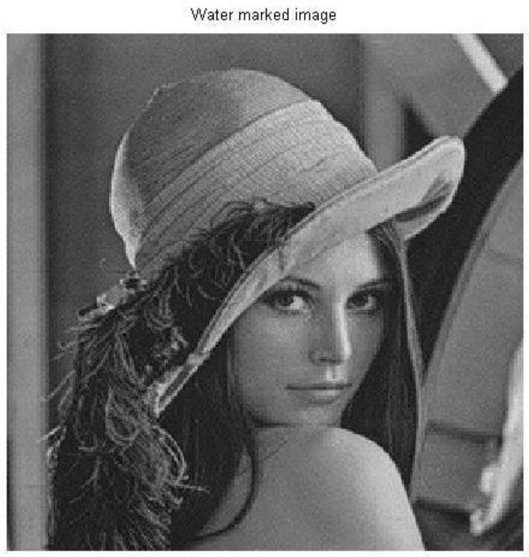
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 214497.696785<br><br>Normal Correlation = 0.949264<br><br>PSNR = 32.715052 |

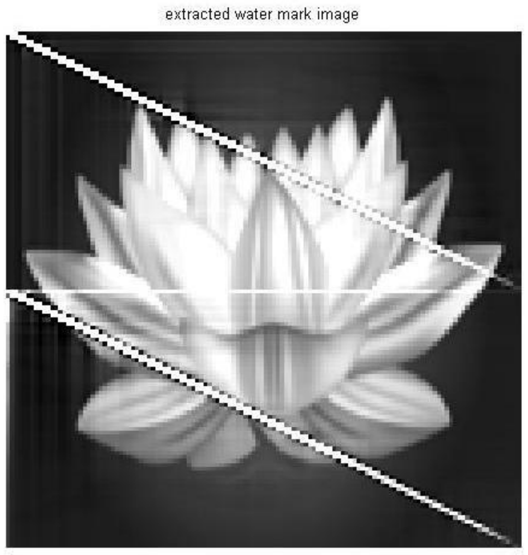
## 2.1.9 Salt and Pepper Noise attack





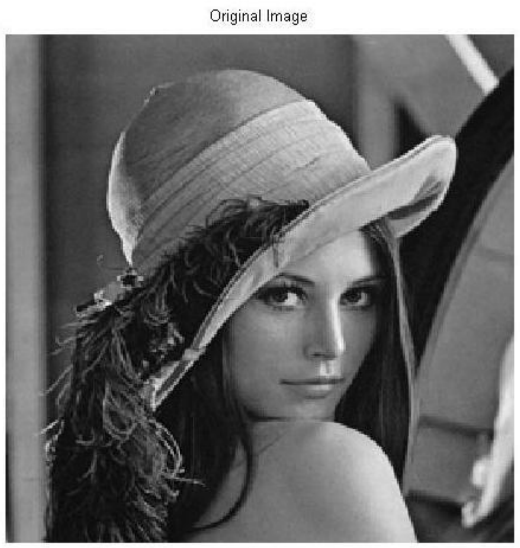
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |

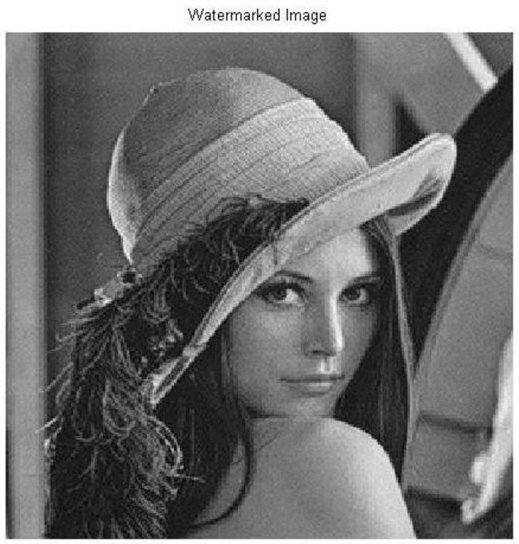




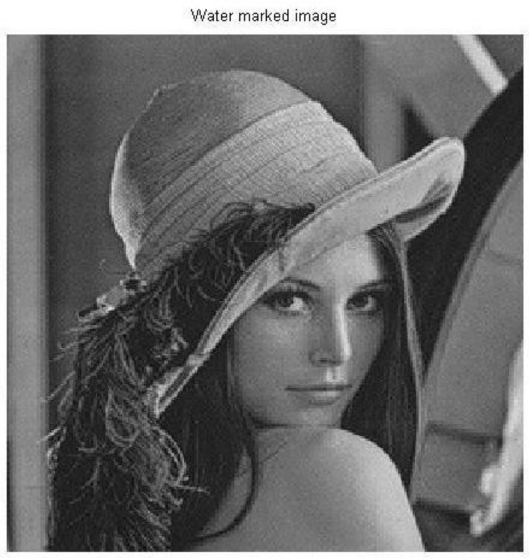
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 509759.609768<br><br>Normal Correlation = 0.314284<br><br>PSNR = 32.715052 |

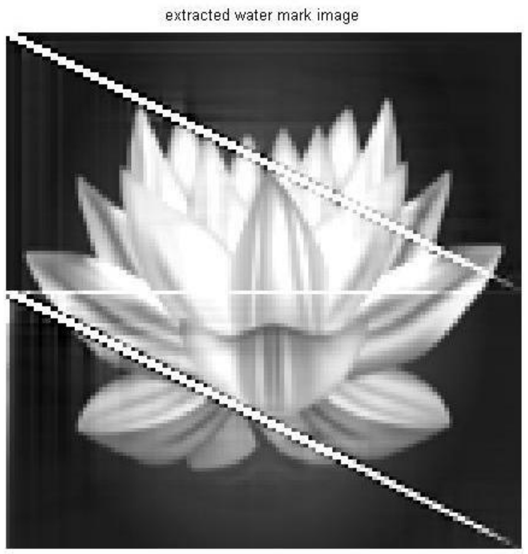
### 2.1.10 Gaussian Noise attack





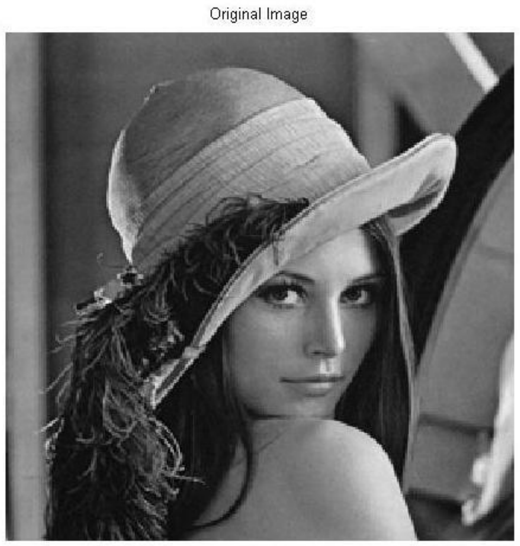
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |

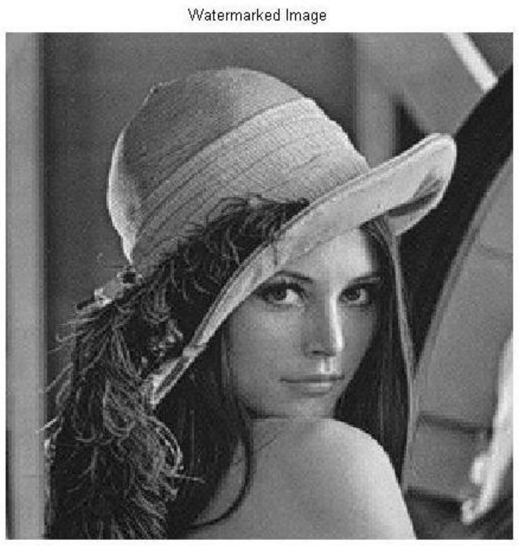




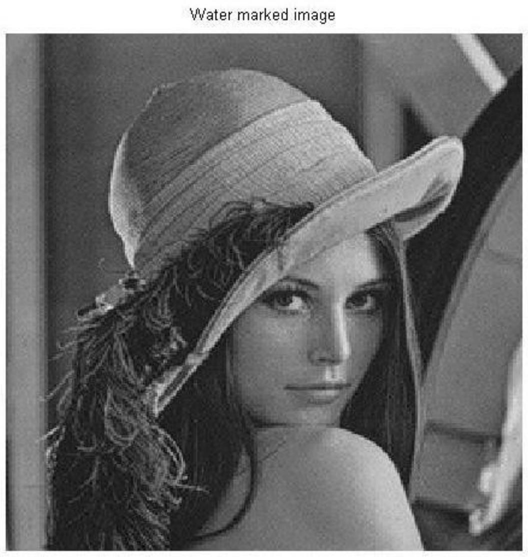
| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 513524.212367<br><br>Normal Correlation = 0.234332<br><br>PSNR = 32.715052 |

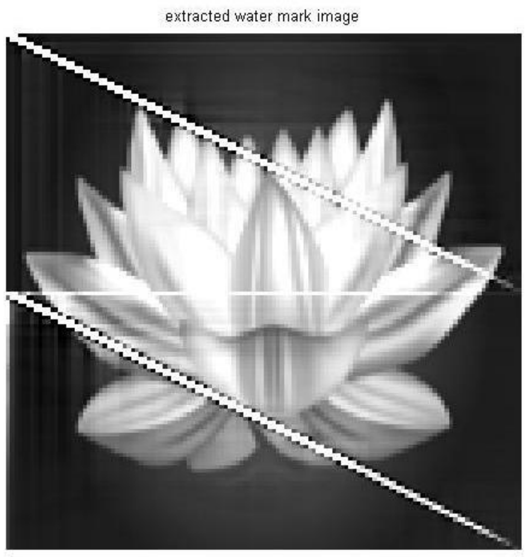
## **2.1.11 Median Filtering  attack**





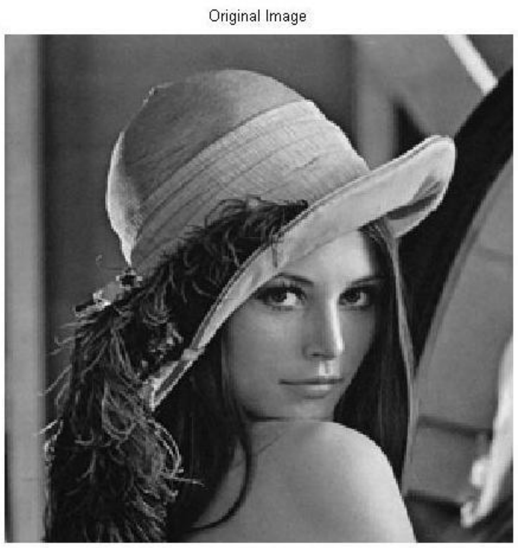
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

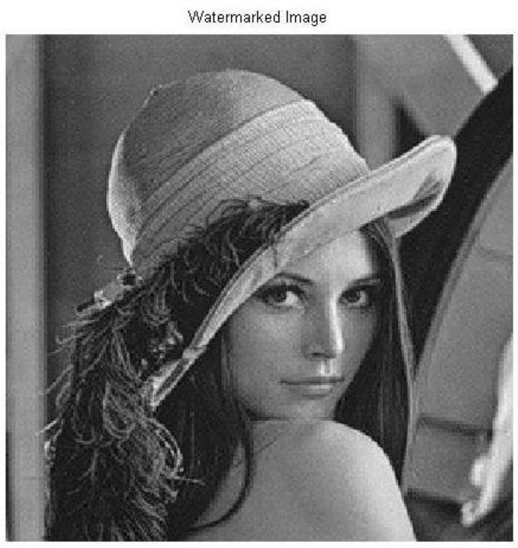




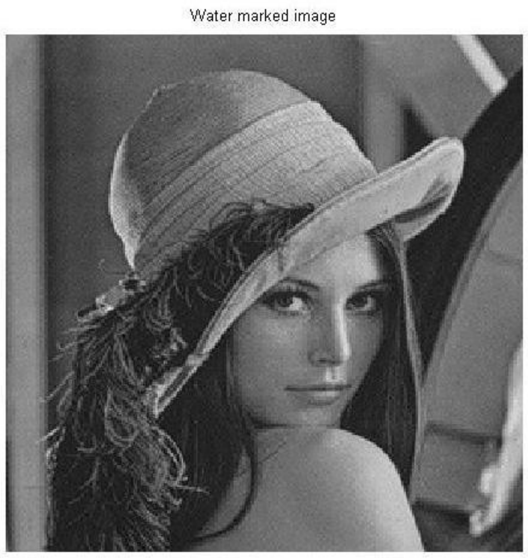
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 24998.595358<br><br>Normal Correlation = 0.972950<br><br>PSNR = 32.715052 |
|---|---|

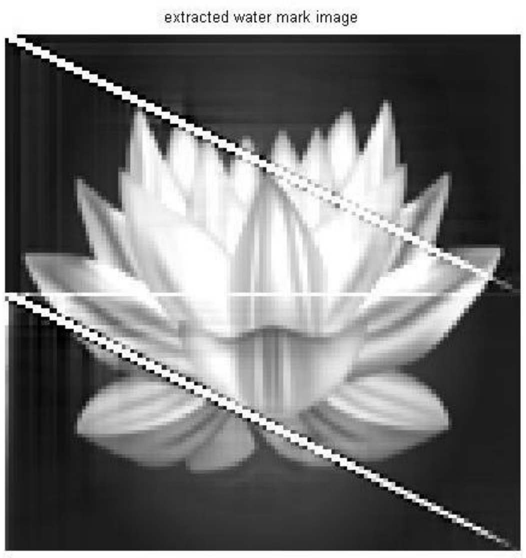
## 2.1.12 Average Filtering attack





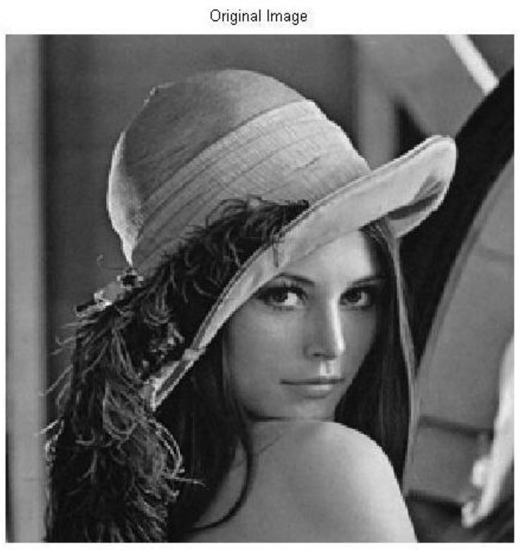
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

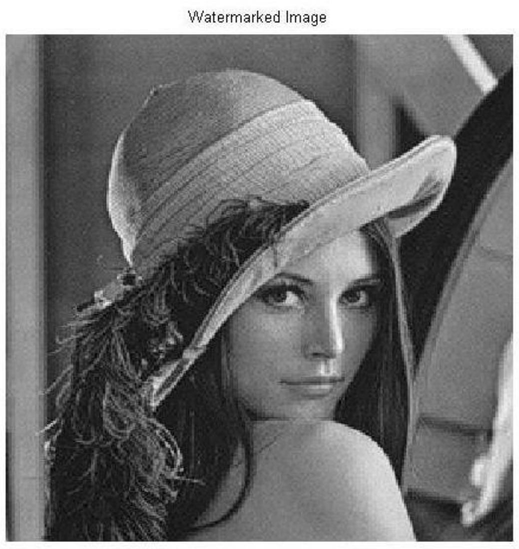




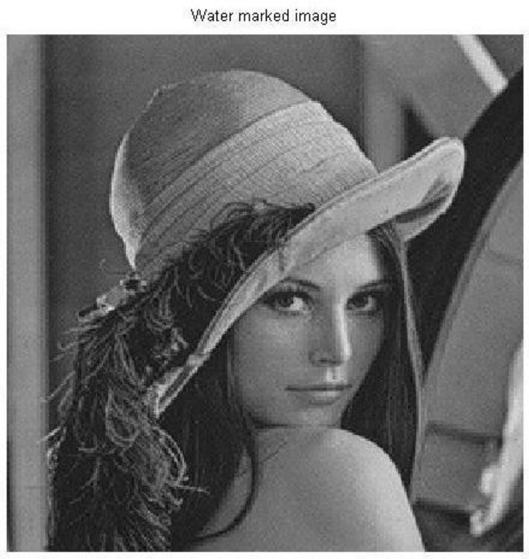
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 109816.308165<br><br>Normal Correlation = 0.968298<br><br>PSNR = 32.715052 |
|---|---|

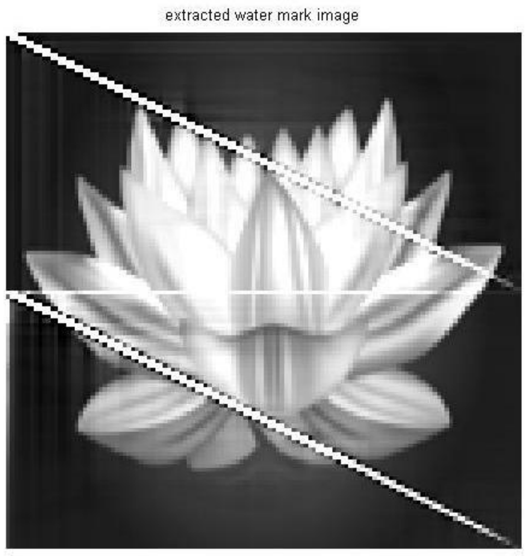
### 2.1.13 Wiener Filter attack





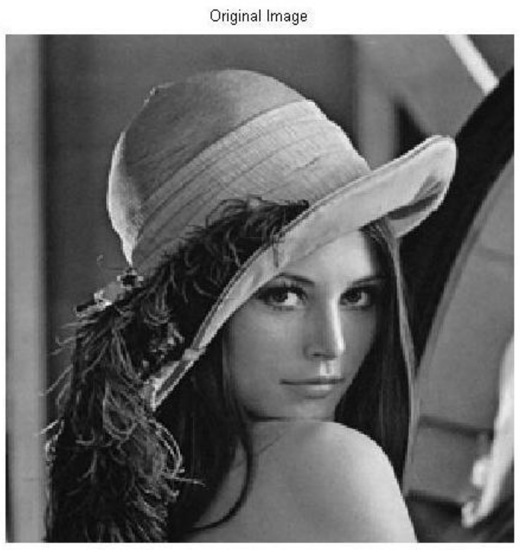
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|

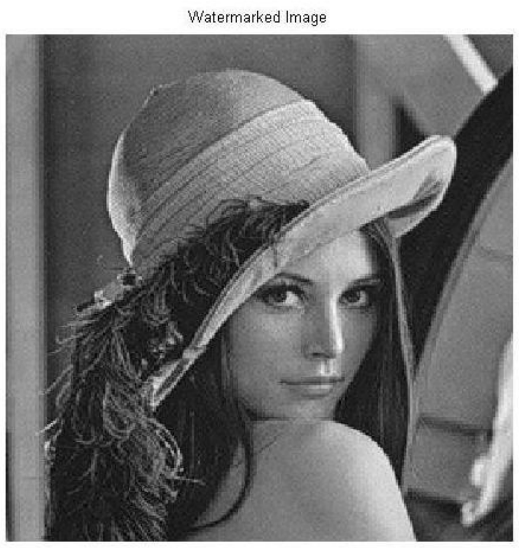




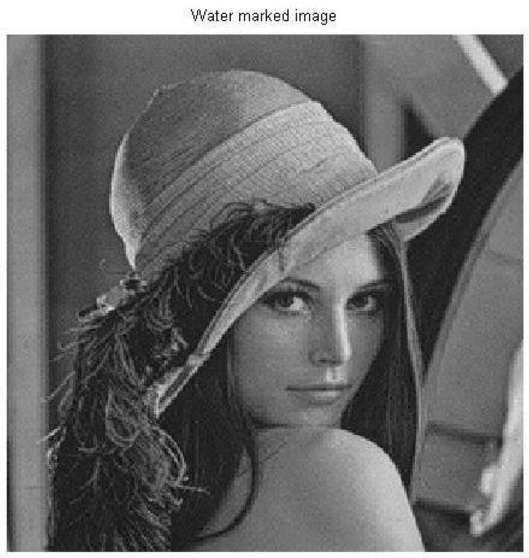
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 9625.208648<br><br>Normal Correlation = 0.972853<br><br>PSNR = 32.715052 |
|---|---|

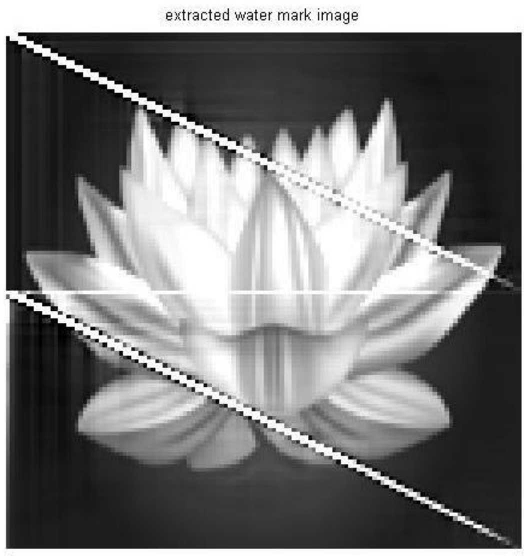
## 2.1.14 Jpeg Compression attack





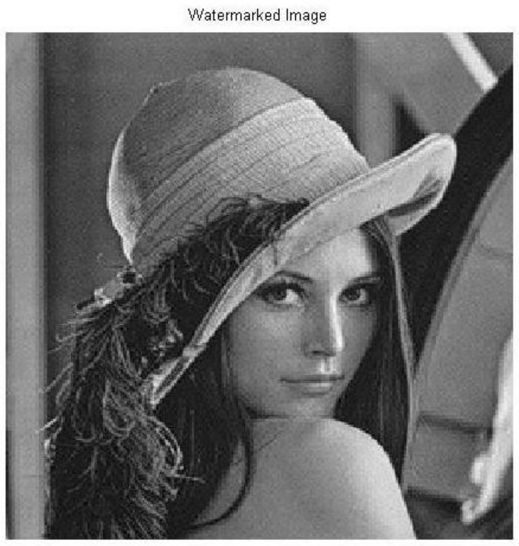
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |
|---|---|



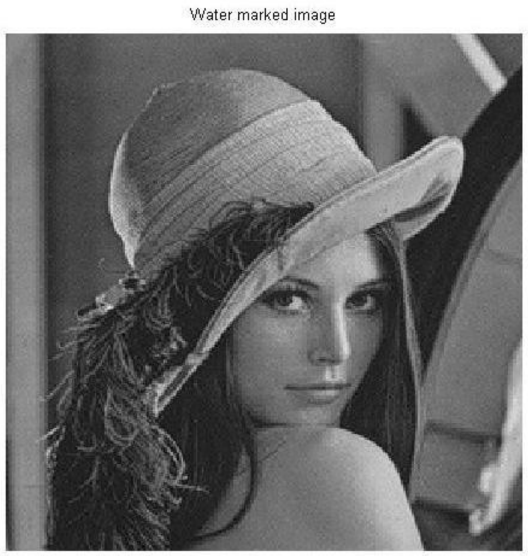


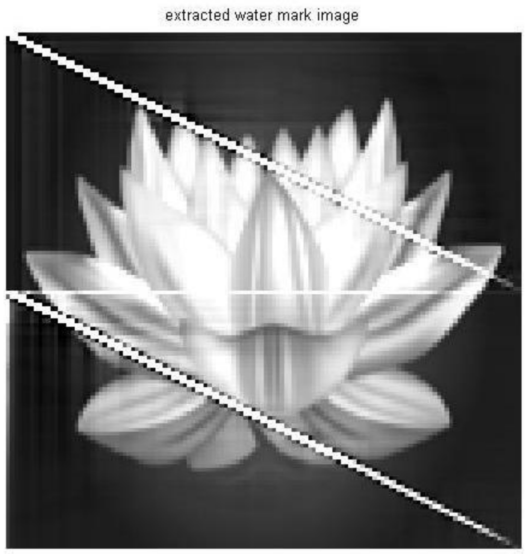
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 497522.214721<br><br>Normal Correlation = 0.461665<br><br>PSNR = 32.715052 |
|---|---|

### 2.1.15 Histogram Equalization attack



| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Original Image (Lenna) **V/S** Watermark image (Lotus) | MSE = 28.557404<br><br>Normal Correlation = 1.000125<br><br>PSNR = 32.715052 |



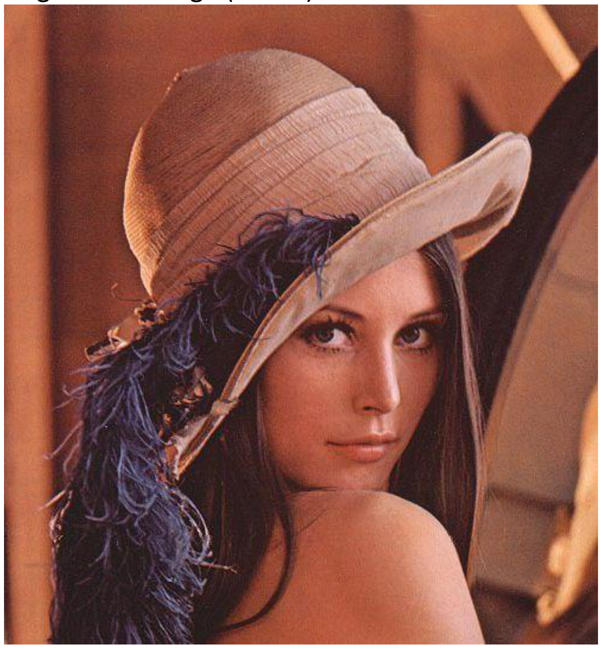


| | |
|---|---|
| **Calculating (PSNR, MSE, NC)**<br><br>Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks | MSE = 518807.516512<br><br>Normal Correlation = 0.029910<br><br>PSNR = 32.715052 |

# CHAPTER 3

## Results

Original test mage (Lenna)



Watermark Image used (Lotus)

## 3.1 Calculating (PSNR, MSE, NC)

## Between Original Image (Lenna) V/S Watermark image (Lotus)

***Table 1***

| **PSNR** | **MSE** | **NC** |
|---|---|---|
| 32.715052 | 28.557404 | 1.000125 |

PSNR – Peak Signal to Noise Ratio

MSE – Mean Square Error

NC – Normal Correlation

**Attacks on Watermarked Image of lenna**

1. No attack
2. Blurr attack
3. Contrast attack
4. Sharpness attack
5. Cropping attack
6. Rotation attack
7. Low Pass Filter attack
8. High Pass Filter attack
9. Salt and Pepper Noise attack
10. Gaussian Noise attack
11. Median Filtering attack
12. Average Filter attack
13. Wiener filter attack
14. Jpeg Compression attack
15. Histogram Equalization attack

## 3.2 Extraction of watermark after attack

## Calculating (PSNR, MSE, NC) Between Watermarked image of Lenna v/s Extracted Watermark image of Lotus after attacks

***Table 2***

| S.No. | Attacks | PSNR | MSE | NC |
|---|---|---|---|---|
| 1 | No attack | 32.715052 | 40814.375306 | 0.990348 |
| 2 | Blurr attack | 32.715052 | 166525.658897 | 0.950716 |
| 3 | Contrast Adjustment attack | 32.715052 | 518807.516512 | 0.029910 |
| 4 | Sharpness attack | 32.715052 | 5428372.871539 | 0.970145 |
| 5 | Cropping attack | 32.715052 | 2806.179273 | 0.977702 |
| 6 | Rotation attack | 32.715052 | 7508.493995 | 0.972933 |
| 7 | Low Pass Filter attack | 32.715052 | 166525.658897 | 0.950716 |
| 8 | High Pass Filter attack | 32.715052 | 214497.696785 | 0.949264 |
| 9 | Salt and Pepper Noise attack | 32.715052 | 509759.609768 | 0.314284 |
| 10 | Gaussian Noise attack | 32.715052 | 513524.212367 | 0.234332 |
| 11 | Median Filtering attack | 32.715052 | 24998.595358 | 0.972950 |
| 12 | Average Filtering attack | 32.715052 | 109816.308165 | 0.968298 |
| 13 | Wiener Filter attack | 32.715052 | 9625.208648 | 0.972853 |
| 14 | Jpeg Compression attack | 32.715052 | 497522.214721 | 0.461665 |
| 15 | Histogram Equalization attack | 32.715052 | 518807.516512 | 0.029910 |

PSNR – Peak Signal to Noise Ratio

MSE – Mean Square Error

NC – Normal Correlation

## 3.3 Conclusion

The proposed local prediction based reversible watermarking has been implemented. For each pixel, the least square predictor in a square block centered on the pixel is computed. The scheme is designed to allow the recovery of the same predictor at detection, without any additional information. The Proposed approach is applied on various test sample images to get the PSNR, MSE and Normal correlation on the 8x8 block size of local predictor. The results obtained so far shows that the local prediction based schemes outperform their counterparts global least square and fixed prediction. The results have been obtained by using the local prediction with a basic difference expansion scheme with simple threshold control, histogram shifting and flag bits.

Also the attacks on watermarked image are shown and comparison is calculated of (PSNR, MSE, NC) between Original Image V/S Watermark image (No attack) and also between Watermarked image v/s Extracted Watermark image (after attacks)

# APPENDIX

## Appendix A

### Implemented Results

```
%-- 19-Dec-16 11:17 AM --%

final_code_No_attack

final_code_2_Blurr_attack

final_code_3_Contrast_Adjustment_attack

final_code_4_Sharpness_attack

final_code_5_Cropping_attack

final_code_6_Rotation_attack

final_code_7_Low_Pass_Filter_attack

final_code_8_High_Pass_Filter_attack

final_code_9_Salt_and_Pepper_Noise_attack

final_code_10_Gaussian_Noise_attack

final_code_11_Median_Filtering_attack

final_code_12_Average_Filtering_attack

final_code_13_Wiener_Filter_attack

final_code_14_Jpeg_Compression_attack

final_code_15_Histogram_Equalization_attack


>> final_code_No_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125
```

```
Extraction parameters

MSE = 40814.375306

Normal Correlation = 0.990348

PSNR = 32.715052


>> final_code_2_Blurr_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 166525.658897

Normal Correlation = 0.950716

PSNR = 32.715052


>> final_code_3_Contrast_Adjustment_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 518807.516512

Normal Correlation = 0.029910
```

```
PSNR = 32.715052


>> final_code_4_Sharpness_attack
Iteration # 1
Iteration # 2
MSE = 28.557404
PSNR = 32.715052
Normal Correlation = 1.000125

Extraction parameters
MSE = 5428372.871539
Normal Correlation = 0.970145
PSNR = 32.715052

>> final_code_5_Cropping_attack
Iteration # 1
Iteration # 2
MSE = 28.557404
PSNR = 32.715052
Normal Correlation = 1.000125

Extraction parameters
MSE = 2806.179273
Normal Correlation = 0.977702
PSNR = 32.715052

>> final_code_6_Rotation_attack
```

```
Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 7508.493995

Normal Correlation = 0.972933

PSNR = 32.715052


>> final_code_7_Low_Pass_Filter_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 166525.658897

Normal Correlation = 0.950716

PSNR = 32.715052


>> final_code_8_High_Pass_Filter_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052
```

```
Normal Correlation = 1.000125


Extraction parameters

MSE = 214497.696785

Normal Correlation = 0.949264

PSNR = 32.715052


>> final_code_9_Salt_and_Pepper_Noise_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 509759.609768

Normal Correlation = 0.314284

PSNR = 32.715052


>> final_code_10_Gaussian_Noise_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 513524.212367
```

```
Normal Correlation = 0.234332

PSNR = 32.715052



>> final_code_11_Median_Filtering_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 24998.595358

Normal Correlation = 0.972950

PSNR = 32.715052


>> final_code_12_Average_Filtering_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 109816.308165

Normal Correlation = 0.968298

PSNR = 32.715052
```

```
>> final_code_13_Wiener_Filter_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 9625.208648

Normal Correlation = 0.972853

PSNR = 32.715052


>> final_code_14_Jpeg_Compression_attack

Iteration # 1

Iteration # 2

MSE = 28.557404

PSNR = 32.715052

Normal Correlation = 1.000125


Extraction parameters

MSE = 497522.214721

Normal Correlation = 0.461665

PSNR = 32.715052


>> final_code_15_Histogram_Equalization_attack

Iteration # 1

Iteration # 2

MSE = 28.557404
```

PSNR = 32.715052

Normal Correlation = 1.000125

Extraction parameters

MSE = 518807.516512

Normal Correlation = 0.029910

PSNR = 32.715052

## <u>Appendix B</u>

### <u>(Algorithm)</u>

The proceedings for each pixel $x_{i,j}$ are as follows

1. Following or using the local LS prediction scheme or the fixed predictor , we are computing the pixel $x_{i,j}$ as:

- The pixel $x_{i,j}$ which is centered for the Block B × B is extracted :
- Having the block which is extracted is replaced with the pixel $\tilde{x}_{i,j}$
- Scanning the block by creating $x_{i,j}$ and $y_{i,j}$
- By solving $y_{i,j} = x_{i,j}\, v_{i,j}$ the local predictor $v_{i,j}$ is computed
- Compute$\hat{x}_{I,j}$.

2. Compute the prediction error $e_{i,j}$
3. If $|e_{i,j}| < T$ compute $\grave{x}_{i,j}$ with eq (2) or otherwise with eq(3).
4. If $\grave{x}_{i,j} \in [0, T-1]$ □$[255 - T, 255]$, replace $x_{i,j}$ by $\grave{□}_{i,j}$ if $\grave{□}_{i,j} \in [0,255]$.
5. Do not replace $x_{i,j}$ if $\grave{□}_{i,j} \nexists [0,255]$ and if the flag bit b=0 insert it to the next embeddable pixel

$$x'_{i,j} = x_{i,j} + e_{i,j} + b \qquad (2)$$

$$x'_{i,j} = \begin{cases} x_{i,j} + T, & \text{if } e_{i,j} \geq T \\ x_{i,j} - (T-1), & \text{if } e_{i,j} \leq -T \end{cases} \qquad (3)$$

Local prediction reversible watermarking

In order to illustrate the reduction of the prediction error provided by using a distinct predictor for each pixel, a simple example is presented. Let us consider the case of the rhombus context and let us evaluate the mean squared prediction error for local LS prediction computed on a B × B sliding window. The improvement depends on the image content, namely it is more significant for images with a high content of texture or fine details than for the ones with large uniform areas.

Let the pixels be embedded in a raster-scan order, pixel by pixel and row by row, from the upper left corner to the lower right one. Obviously, the decoding proceeds in reverse order, from bottom to top, starting with the last embedded pixel. If the prediction context has *k* pixels, the central pixel takes part in *k* other prediction

1) The vector corresponding to the central pixel, $x_{i,j}$ , as well as the ones that contain the central pixel ($x_{l,m}$, with $x_{i,j} \in x_{l,m}$ ) are eliminated from $x_{i,j}$ and the central pixel as well as the pixels $x_{l,j}$mare eliminated from $y_{i,j}$;
2) Before the construction of $x_{i,j}$ and $y_{i,j}$, the central pixel of the block $x_{i,j}$ is replaced by an estimate $\tilde{\square}_{i,j}$ computed by using a fixed predictor as the one of equation (4).

$$\tilde{x}_{i,j} = \frac{x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}}{4} \qquad (4)$$

## APPENDIX C

### Watermark algorithm (matlab)

```
function [watermarkedImage, len_water] = Algorithm_1(im_orig,T,Dir_em)

[sx sy] = size(im_orig);

if Dir_em == 1

   DIREC = [1 2];

else

   DIREC = [2 1];
```

```
end


differenceFunc=inline('x(1)-x(2)');

averageFunc=inline('floor((x(1)+x(2))/2)');

d = blkproc(double(im_orig),DIREC,differenceFunc);

g = blkproc(double(im_orig),DIREC,averageFunc);

[m n]=size(d);

M = zeros(m,n);

SETID = zeros(m,n); % identifies later as to which set a particulat pair belongs


s1Index = 1;

s2Index = 1;

s21Index = 1;

s22Index = 1;

s3Index = 1;

s4Index = 1;

s5Index = 1;

S1=[]; S2=[]; S21=[]; S22=[]; S3=[]; S4=[]; S5=[];

for i=1:size(d,1)

    for j=1:size(d,2)

        if (abs(2*d(i,j)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1) ...

                | abs(2*d(i,j)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)) & (d(i,j) == 0 | d(i,j) == -1)

            % set S1 contains all pixel pairs that satisfy the difference value condition
    S1(s1Index) = d(i,j);

    s1Index = s1Index + 1;

    M(i,j) = 1;

    SETID(i,j) = 1;

elseif d(i,j) == 0 | d(i,j) == -1

    % not expandable zeros

    S5(s5Index) = d(i,j);

    s5Index = s5Index + 1;

    M(i,j) = 0;

    SETID(i,j) = 6;

elseif abs(2*d(i,j)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1) ...

        | abs(2*d(i,j)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)

    % set S2 contains all expandable pixel pairs that are not in S1

    S2(s2Index) = d(i,j);

    s2Index = s2Index + 1;

    if abs(d(i,j)) <= T

        % set S21 contains all pixel pairs whose difference value is less than threshold

        S21(s21Index) = d(i,j);

        s21Index = s21Index + 1;

        M(i,j) = 1;

        SETID(i,j) = 2;

    else

        % set S22 contains all pixel pairs whose difference value is greater than threshold
```

```
            % embedding is performed in set S22
            S22(s22Index) = d(i,j);
            s22Index = s22Index + 1;
            M(i,j) = 0;
            SETID(i,j) = 3;
        end
    elseif abs(2*floor(d(i,j)/2)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1)...
            | abs(2*floor(d(i,j)/2)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)
        % set S3 contains all changeable pixel pairs
        % embedding is performed in set S3
        S3(s3Index) = d(i,j);
        s3Index = s3Index + 1;
        M(i,j) = 0;
        SETID(i,j) = 4;
    else
        % set s4 contains all non changable pixel pairs
        S4(s4Index) = d(i,j);
        s4Index = s4Index + 1;
        M(i,j) = 0;
        SETID(i,j) = 5;
    end
  end
end
```

```
%figure,imshow(M,[]),title('Location Map');
B1 = reshape(M,1,size(M,1)*size(M,2));


Q1=[];
for i=1:length(S22)
   Q1(i) = bitget(abs(S22(i)),1);
end
b = length(Q1);


% Extract LSB's of coeffs in set S3
Q2=[];
for i=1:length(S3)
   Q2(i) = bitget(abs(S3(i)),1);
end
c = length(Q2);


B2=[Q1 Q2];


%% STEP 5A: Compress the Location map using Arithmetic coding
% convert to 1-2 vector from 0-1
for i=1:length(B1)
   if B1(i) == 0
```

```
        seq1(i) = 1;
    else
        seq1(i) = 2;
    end
end
% find how many times 1 occurs in the binary sequence
[xx,yy,val] = find(seq1 == 1);
totalVals = length(seq1);

% code the sequence using arithmetic coding
count1 = [round((size(xx,2)/totalVals)*100) round(((totalVals - size(xx,2))/totalVals) * 100)];
encodedLM = arithenco(seq1,count1);
% str = sprintf('LMap Length = %d ------ Compressed LMap Length =
%d',size(seq1,2),size(encodedLM,2));
% disp(str);

%% STEP 5B: Compress the Original LSB's using Arithmetic coding
% convert to 1-2 vector from 0-1
for i=1:length(B2)
    if B2(i) == 0
        seq2(i) = 1;
    else
        seq2(i) = 2;
    end
end
% find how many times 1 occurs in the binary sequence
[xx,yy,val] = find(seq2 == 1);
totalVals = length(seq2);

% code the sequence using arithmetic coding
count2 = [round((size(xx,2)/totalVals)*100) round(((totalVals - size(xx,2))/totalVals) * 100)];
encodedS22_S23 = arithenco(seq2,count2);
% str = sprintf('Original Bits Length = %d ------ Compressed Bits Length =
%d',size(seq2,2),size(encodedS22_S23,2));
% disp(str);

%% STEP 6: Calculate Available embedding capacity
% Calculate total Available Embedding Capacity
totalCapacity = length(S1) + length(S21) + length(S22) + length(S3);

% Calculate payload size
HEADERLEN = 112;
totalPayload = length(encodedLM) + length(encodedS22_S23);
len_water = totalCapacity - totalPayload - HEADERLEN;
totalPayload = length(encodedLM) + length(encodedS22_S23) + len_water;
```

```
% str = sprintf('Total Capacity = %d ------ Watermark Length = %d',totalCapacity,len_water);

% disp(str);


% Convert header info into bit stream

hbinSeq = dec2bin(count1(1,1),8);

hbinSeq = strcat(hbinSeq,dec2bin(count1(1,2),8));

hbinSeq = strcat(hbinSeq,dec2bin(count2(1,1),8));

hbinSeq = strcat(hbinSeq,dec2bin(count2(1,2),8));

hbinSeq = strcat(hbinSeq,dec2bin(length(seq1),16));

hbinSeq = strcat(hbinSeq,dec2bin(length(encodedLM),16));

hbinSeq = strcat(hbinSeq,dec2bin(length(seq2),16));

hbinSeq = strcat(hbinSeq,dec2bin(length(encodedS22_S23),16));

hbinSeq = strcat(hbinSeq,dec2bin(len_water,16));


BH = zeros(1,length(hbinSeq));

for i=1:length(hbinSeq)

    BH(i) = str2num(hbinSeq(i));

end

%% STEP 7: Generate Random Watermark (Payload)

B3=randint(1,len_water,[0 1],100);


% concatenate all bitstream to obtain the embedding stream

%B = [BH B1 B2 B3];
```

```
B = [BH encodedLM encodedS22_S23 B3];


%% STEP 8: Embedd Watermark in image
index = 1;
wIndex = 1;
exitLoop = 0;
watermarkedImage = zeros(sx,sy);
for i=1:size(d,1)
    if Dir_em == 1
        index = 1;
    end
    for j=1:size(d,2)
        if SETID(i,j) ~= 5 | SETID(i,j) ~= 6
            neg = 0;
            if d(i,j) < 0
                neg = 1;
            end

            if SETID(i,j) == 1 | SETID(i,j) == 2
                % Embedd watermark by shifting
                val = bitshift(abs(d(i,j)),1);
            elseif SETID(i,j) == 3 | SETID(i,j) == 4
                % Embedd watermark by replacement
                val = abs(d(i,j));
            end

            binSeq = dec2bin(val,8);
            if B(1,wIndex) == 1
                binSeq(8) = '1';
            else
                binSeq(8) = '0';
            end

            val = bin2dec(binSeq);

            if neg == 1
                d(i,j) = -val;
            else
                d(i,j) = val;
            end

            if Dir_em == 1
                watermarkedImage(i,index) = g(i,j) + floor((d(i,j) + 1)/2);
                watermarkedImage(i,index + 1) = g(i,j) - floor(d(i,j)/2);
                index = index + 2;
            else
```

```
            watermarkedImage(index,j) = g(i,j) + floor((d(i,j) + 1)/2);
            watermarkedImage(index + 1,j) = g(i,j) - floor(d(i,j)/2);
        end
        if wIndex < totalCapacity
            wIndex = wIndex + 1;
        else
            exitLoop = 1;
            break;
        end
    else
        if Dir_em == 1
            watermarkedImage(i,index) = im_orig(i,index);
            watermarkedImage(i,index + 1) = im_orig(i,index + 1);
            index = index + 2;
        else
            watermarkedImage(index,j) = im_orig(index,j);
            watermarkedImage(index + 1,j) = im_orig(index + 1,j);
        end
    end
end
if exitLoop == 1
    break;
end
```

```
    if Dir_em ~= 1

        index = index + 2;

    end

end
```

## Appendix D

**MATLAB CODE (Watermarking)**

```
clear all;
close all;
warning off;
E_di = 1;
T = 20;
Emb_t = 2;
im=imread('12.jpg');
im=rgb2gray(im);
Image_im = imresize(im,[256 256],'bicubic');

im_n = Image_im;
len_t = 0;
for k=1:Emb_t
   str = sprintf('Itr :: %d',k);
   disp(str);

   [im_n,WM_LEN] = Algorithm_1(im_n,T,E_di);
   if E_di == 0
       E_di = 1;
   else
       E_di = 0;
   end
   len_t = len_t + WM_LEN;
end


I0 = double(Image_im);
I1 = (im_n);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
```

```
str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(Image_im,im_n);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(Image_im,im_n);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(2);
subplot(1,2,1);
imshow(Image_im,[]);
title('Original Image');
subplot(1,2,2);
imshow(im_n,[]);
title('Watermarked Image');
```

*get_nc function*

```
function out = get_nc(orig_w_img,extrac_w_img,j)

if (nargin < 3)
        j=0;
end
extrac_w_img=(uint8(extrac_w_img));
orig_w_img=uint8(orig_w_img);
orig=imresize(orig_w_img,[size(extrac_w_img,1),size(extrac_w_img,1)]);

up=extrac_w_img.*orig;
upper=sum(sum(up));

lo=extrac_w_img.*extrac_w_img;
lower=sum(sum(lo));
if(j==0)
nc=(upper/lower)-0.16;
else
    nc=(upper/lower);
end
```

```
out=nc;




function [watermarkedImage, len_water] = LP_water_mark_code(im_orig,T,Dir_em)

[sx sy] = size(im_orig);

if Dir_em == 1
   DIREC = [1 2];
else
   DIREC = [2 1];
end

diff_fucn=inline('x(1)-x(2)');
averageFunc=inline('floor((x(1)+x(2))/2)');
d = blkproc(double(im_orig),DIREC,diff_fucn);
g = blkproc(double(im_orig),DIREC,averageFunc);
[m n]=size(d);
M = zeros(m,n);
SETID = zeros(m,n); % identifies later as to which set a particulat pair belongs

s1Index = 1;
s2Index = 1;
s21Index = 1;
s22Index = 1;
s3Index = 1;
s4Index = 1;
s5Index = 1;
S1=[]; S2=[]; S21=[]; S22=[]; S3=[]; S4=[]; S5=[];
for i=1:size(d,1)
   for j=1:size(d,2)
      if (abs(2*d(i,j)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1) ...
            | abs(2*d(i,j)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)) & (d(i,j) == 0 | d(i,j) == -1)

         S1(s1Index) = d(i,j);
         s1Index = s1Index + 1;
         M(i,j) = 1;
         SETID(i,j) = 1;
      elseif d(i,j) == 0 | d(i,j) == -1

         S5(s5Index) = d(i,j);
         s5Index = s5Index + 1;
         M(i,j) = 0;
         SETID(i,j) = 6;
```

```
        elseif abs(2*d(i,j)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1) ...
                | abs(2*d(i,j)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)

            S2(s2Index) = d(i,j);
            s2Index = s2Index + 1;
            if abs(d(i,j)) <= T

                S21(s21Index) = d(i,j);
                s21Index = s21Index + 1;
                M(i,j) = 1;
                SETID(i,j) = 2;
            else

                S22(s22Index) = d(i,j);
                s22Index = s22Index + 1;
                M(i,j) = 0;
                SETID(i,j) = 3;
            end
        elseif abs(2*floor(d(i,j)/2)+0) <= min(2*(255-g(i,j)),2*g(i,j) + 1)...
                | abs(2*floor(d(i,j)/2)+1) <= min(2*(255-g(i,j)),2*g(i,j) + 1)

            S3(s3Index) = d(i,j);
            s3Index = s3Index + 1;
            M(i,j) = 0;
            SETID(i,j) = 4;
        else
            % set s4 contains all non changable pixel pairs
            S4(s4Index) = d(i,j);
            s4Index = s4Index + 1;
            M(i,j) = 0;
            SETID(i,j) = 5;
        end
    end
end

B1 = reshape(M,1,size(M,1)*size(M,2));

Q1=[];
for i=1:length(S22)
    Q1(i) = bitget(abs(S22(i)),1);
end
b = length(Q1);

Q2=[];
for i=1:length(S3)
    Q2(i) = bitget(abs(S3(i)),1);
```

```
end
c = length(Q2);

B2=[Q1 Q2];



for i=1:length(B1)
   if B1(i) == 0
      seq1(i) = 1;
   else
      seq1(i) = 2;
   end
end


[xx,yy,val] = find(seq1 == 1);
totalVals = length(seq1);

count1 = [round((size(xx,2)/totalVals)*100) round((((totalVals - size(xx,2))/totalVals) * 100)];
encodedLM = arithenco(seq1,count1);



for i=1:length(B2)
   if B2(i) == 0
      seq2(i) = 1;
   else
      seq2(i) = 2;
   end
end


[xx,yy,val] = find(seq2 == 1);
totalVals = length(seq2);

count2 = [round((size(xx,2)/totalVals)*100) round((((totalVals - size(xx,2))/totalVals) * 100)];
encodedS22_S23 = arithenco(seq2,count2);


totalCapacity = length(S1) + length(S21) + length(S22) + length(S3);

hd_rl = 112;
totalPayload = length(encodedLM) + length(encodedS22_S23);
len_water = totalCapacity - totalPayload - hd_rl;
totalPayload = length(encodedLM) + length(encodedS22_S23) + len_water;
```

```
hb_se = dec2bin(count1(1,1),8);
hb_se = strcat(hb_se,dec2bin(count1(1,2),8));
hb_se = strcat(hb_se,dec2bin(count2(1,1),8));
hb_se = strcat(hb_se,dec2bin(count2(1,2),8));
hb_se = strcat(hb_se,dec2bin(length(seq1),16));
hb_se = strcat(hb_se,dec2bin(length(encodedLM),16));
hb_se = strcat(hb_se,dec2bin(length(seq2),16));
hb_se = strcat(hb_se,dec2bin(length(encodedS22_S23),16));
hb_se = strcat(hb_se,dec2bin(len_water,16));

BH = zeros(1,length(hb_se));
for i=1:length(hb_se)
   BH(i) = str2num(hb_se(i));
end


B3=randint(1,len_water,[0 1],100);

B = [BH encodedLM encodedS22_S23 B3];

index = 1;
wIndex = 1;
exitLoop = 0;
watermarkedImage = zeros(sx,sy);
for i=1:size(d,1)
   if Dir_em == 1
      index = 1;
   end
   for j=1:size(d,2)
      if SETID(i,j) ~= 5 | SETID(i,j) ~= 6
         neg = 0;
         if d(i,j) < 0
            neg = 1;
         end

         if SETID(i,j) == 1 | SETID(i,j) == 2
            val = bitshift(abs(d(i,j)),1);
         elseif SETID(i,j) == 3 | SETID(i,j) == 4
            val = abs(d(i,j));
         end

         binSeq = dec2bin(val,8);
         if B(1,wIndex) == 1
            binSeq(8) = '1';
            else
                binSeq(8) = '0';
            end

            val = bin2dec(binSeq);

            if neg == 1
                d(i,j) = -val;
            else
                d(i,j) = val;
            end

            if Dir_em == 1
                watermarkedImage(i,index) = g(i,j) + floor((d(i,j) + 1)/2);
                watermarkedImage(i,index + 1) = g(i,j) - floor(d(i,j)/2);
                index = index + 2;
            else
                watermarkedImage(index,j) = g(i,j) + floor((d(i,j) + 1)/2);
                watermarkedImage(index + 1,j) = g(i,j) - floor(d(i,j)/2);
            end
            if wIndex < totalCapacity
                wIndex = wIndex + 1;
            else
                exitLoop = 1;
                break;
            end
        else
            if Dir_em == 1
                watermarkedImage(i,index) = im_orig(i,index);
                watermarkedImage(i,index + 1) = im_orig(i,index + 1);
                index = index + 2;
            else
                watermarkedImage(index,j) = im_orig(index,j);
                watermarkedImage(index + 1,j) = im_orig(index + 1,j);
            end
        end
    end
    if exitLoop == 1
        break;
    end
    if Dir_em ~= 1
        index = index + 2;
    end
end
```

# APPENDIX E
## (MATLAB CODE FOR ALL ATTACKS)

**Local Prediction Based difference expansion Reversible Watermarking**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('24.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');

markedImage = originalImage;
len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [markedImage,WM_LEN] = Algorithm_1(markedImage,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = (markedImage);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,markedImage);
str1 = sprintf('PSNR = %f',PSNR_OUT);
```

```
disp(str1);
Nc=get_nc(originalImage,markedImage);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(2);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(markedImage,[]);
title('Watermarked Image');
```

**final_code_1_No_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end
```

```
% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
%H = fspecial('motion');
wi=Aw1;
%wi = imfilter(Aw1,H,'replicate');
%wi=imnoise(Aw1,'gaussian',0,0.01); % noise gaussian
%wi=imrotate(Aw1,45); % rotation 45deg
% imshow(wi1);
%wi=medfilt2(Aw1); % median filtering
%wi=histeq(Aw1); % histogram equilier
%wi = imadjust(Aw1); %contrast adjustment
%wi=wiener2(Aw1); % winer filtering
%f = ones(6,6)/36;
%wi = filter2(f, Aw1); % high pass filter
```

```
%wi=jpeg_compression(Aw1);%jpeg compression
%wi=uint8(wi);


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_2_Blurr_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
```

```
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);
```

```
figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
H = fspecial('motion');
wi = imfilter(Aw1,H,'replicate');


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
```

```
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_3_Contrast_Adjustment_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
```

```
str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi = imadjust(Aw1); %contrast adjustment


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
```

```
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_4_Sharpness_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end
```

```
% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
H = fspecial('unsharp');
wi = imfilter(Aw1,H,'replicate');

% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
```

```
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_5_Cropping_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
```

```
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=imcrop(Aw1,[1,1,200 200]);
```

```
% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_6_Rotation_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker
```

```
markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
```

```
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=imrotate(Aw1,45); % rotation 45deg


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_7_Low_Pass_Filter_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);
```

```
[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
H = fspecial('motion');
%wi=Aw1;
wi = imfilter(Aw1,H,'replicate');
%wi=imnoise(Aw1,'gaussian',0,0.01); % noise gaussian
%wi=imrotate(Aw1,45); % rotation 45deg
% imshow(wi1);
%wi=medfilt2(Aw1); % median filtering
%wi=histeq(Aw1); % histogram equilier
%wi = imadjust(Aw1); %contrast adjustment
%wi=wiener2(Aw1); % winer filtering
%f = ones(6,6)/36;
%wi = filter2(f, Aw1); % high pass filter
%wi=jpeg_compression(Aw1);%jpeg compression
%wi=uint8(wi);


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
```

```
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_8_High_Pass_Filter_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
```

```
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
f = ones(6,6)/36;
```

```
wi = filter2(f, Aw1); % high pass filter


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_9_Salt_and_Pepper_Noise_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
```

```
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
```

```
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=imnoise(Aw1,'salt & pepper', 0.02);

% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_10_Gaussian_Noise_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);
```

```
[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=imnoise(Aw1,'gaussian',0,0.01); % noise gaussian


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
```

```
figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_11_Median_Filtering_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
%H = fspecial('motion');
%wi=Aw1;
%wi = imfilter(Aw1,H,'replicate');
%wi=imnoise(Aw1,'gaussian',0,0.01); % noise gaussian
%wi=imrotate(Aw1,45); % rotation 45deg
% imshow(wi1);
wi=medfilt2(Aw1); % median filtering
%wi=histeq(Aw1); % histogram equilier
%wi = imadjust(Aw1); %contrast adjustment
%wi=wiener2(Aw1); % winer filtering
%f = ones(6,6)/36;
%wi = filter2(f, Aw1); % high pass filter
%wi=jpeg_compression(Aw1);%jpeg compression
%wi=uint8(wi);
```

```
% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_12_Average_Filtering_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
```

```
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
  str = sprintf('Iteration # %d',k);
  disp(str);

  [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
  if emb_dir == 0
     emb_dir = 1;
  else
     emb_dir = 0;
  end
  len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
```

```
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=averagefilter(Aw1);

% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_13_Wiener_Filter_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);
```

```
[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=wiener2(Aw1); % winer filtering

% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);
```

```
subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_14_Jpeg_Compression_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);

   [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
   if emb_dir == 0
       emb_dir = 1;
   else
       emb_dir = 0;
   end
   len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')
```

```
I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=jpeg_compression(Aw1);%jpeg compression
wi=uint8(wi);


% extraction part
fprintf('Extraction parameters \n\n\n');
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);
```

```
Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

**final_code_15_Histogram_Equalization_attack**

```
clear all;
close all;
warning off;
emb_dir = 1;
T = 20;
NO_TIMES_EMBEDD = 2;
im=imread('lenna.jpg');
im=rgb2gray(im);
originalImage = imresize(im,[256 256],'bicubic');
n=rgb2gray(imread('lotus.jpg'));  % load the water marker

markedImage = originalImage;
c=0.01;% scaling value
m1=markedImage;   % converting image to grayscal
m2=imresize(m1,[256 256]); % resizing the image
wm=imresize(double(n),[128 128]); % resizing water marker
[Aw1,Uw1,Uw2]=function_1(m2,wm);

len_t = 0;
for k=1:NO_TIMES_EMBEDD
   str = sprintf('Iteration # %d',k);
   disp(str);
```

```
    [Aw1,WM_LEN] = Algorithm_1(Aw1,T,emb_dir);
    if emb_dir == 0
        emb_dir = 1;
    else
        emb_dir = 0;
    end
    len_t = len_t + WM_LEN;
end

% imwrite(uint8(markedImage),'WatermarkedImage.tif','tif');
% figure,imshow(markedImage,[]),title('Watermarked Image')

I0 = double(originalImage);
I1 = double(Aw1);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);

str = sprintf('MSE = %f',MSE);
disp(str);


[PSNR_OUT,Z] = psnr(originalImage,Aw1);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);
Nc=get_nc(originalImage,Aw1);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);

figure(200);
subplot(1,2,1);
imshow(originalImage,[]);
title('Original Image');
subplot(1,2,2);
imshow(Aw1,[]);
title('Watermarked Image');


%% attack part

%% attack part
wi=histeq(Aw1); % histogram equilier

% extraction part
fprintf('Extraction parameters \n\n\n');
```

```
[W]=function_extraction(wi,Uw1,Uw2);

I0 = double(W);
I1 = double(wm);
Id = (I0-I1);
signal = sum(sum(I0.^2));
noise = sum(sum(Id.^2));
MSE = noise/numel(I0);
str = sprintf('MSE = %f',MSE);
disp(str);

Nc=get_nc(W,wm);
str2 = sprintf('Normal Correlation = %f',Nc);
disp(str2);


[PSNR,zz] = psnr(wm,W);
str1 = sprintf('PSNR = %f',PSNR_OUT);
disp(str1);

figure(100);

subplot(1,2,2);
imshow(uint8(W));
title('extracted water mark image');


subplot(1,2,1);
imshow(uint8(Aw1));
title('Water marked image');
```

## Appendix-F

**<u>REFERENCES:</u>**

[1] https://en.wikipedia.org/wiki/Digital_watermarking

[2]https://www.digimarc.com/docs/default-source/technology-resources/published-technical-papers

[3] https://www.digimarc.com/docs/default-source/technology-resources/published-technical-papers/reversible-techniques/dmrc_generalized_integer.pdf

[4] http://aty.sdsu.edu/~aty/bibliog/latex/scan/blur.html

[5] http://www.dspguide.com/ch23/5.htm

[6]https://en.wikipedia.org/wiki/Acutance

[7] https://en.wikipedia.org/wiki/Cropping_ (image)

[8]https://www.wavemetrices.com/products/igorpro/imageprocessing/imagetransforms/imagerotation.htm

[9] https://en.wikipedia.org/wiki/High-pass_filter

[10] https://diffractionlimited.com/help/maximdl/Low-Pass_Filtering.htm

[11] https://en.wikipedia.org/wiki/Salt-and-pepper_noise

[12] https://en.wikipedia.org/wiki/Gaussian_noise

[13] https://en.wikipedia.org/wiki/Median_filter

[14] http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

[15] https://en.wikipedia.org/wiki/Wiener_filter

[16] https://en.wikipedia.org/w/index.php?title=JPEG&printable=yes

[17] https://en.wikipedia.org/wiki/Histogram_equalization