National College of Ireland

Configuration Manual

MSc Research Project
MSc Cyber Security

Aishwarya Tidke
Student ID: 22100377

School of Computing
National College of Ireland

Supervisor: Dr. Vanessa Ayala-Rivera

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aishwarya Tidke
Student ID:	22100377
Programme:	MSc Cyber Security
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Vanessa Ayala-Rivera
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	728
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aishwarya Tidke
22100377

1 Introduction

The security of digital data, especially photos, is a major problem in the current era of digital communication. Unauthorized sharing and use of photographs might result in problems including abuse and copyright violation. In order to preserve the ownership of digital images, this research intends to address these problems by designing a strong watermarking methodology that integrates cryptographic techniques, reversible discrete wavelet transform (RDWT), and homomorphic transformation. This project's main objective is to create a secure and effective technique for adding watermarks to digital photos. The hybrid approach ensures the integrity and confidentiality of the embedded information while leveraging the strengths of several methods to achieve robustness against common image processing attacks.

2 Requirements

To execute and test the project, make sure the following requirements are met:

2.1 Hardware requirements

- **Processor (CPU):** A multi-core processor with at least a quad-core configuration
- **Memory (RAM):** Minimum 8 GB of RAM
- **Storage:** 256 GB SSD or higher.
- **Operating System:** Windows, Linux, or macOS.

2.2 Software requirements

- Python 3.x
- OpenCV
- NumPy
- qrcode
- SciPy
- Cryptography

```python
import cv2
import numpy as np
import qrcode
from scipy.linalg import svd
import zlib
import os
from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305
import heapq
import pywt
from skimage.metrics import peak_signal_noise_ratio, mean_squared_error, structural_similarity
import matplotlib.pyplot as plt
```

Figure 1: Imported python libraries

```python
qr.add_data('/Users/aishwaryatidke/Desktop/Thesis/code/TEST/new code/QR_image.png')
qr.make(fit=True)

# Create an image from the QR Code instance
```

Figure 2: Add QR code user authentication data

- PyWavelets
- Scikit-image
- Matplotlib

3 Installations

- Install Python: Visit the official Python website, download, and install the recent version of Python.
- Set up a virtual environment (optional)
- Install the required packages: The required Python packages as shown in figure 1 can be installed by using the following command:

pip install opencv-python numpy qrcode[pil] scipy cryptography pywavelets scikit-image matplotlib

- Test images can be downloaded from these websites [1] [2]

4 Main Script Execution

- Code includes test.py inclusive of all algorithms.
- As shown in Figure 2, enter the user data in the code as needed.
- Also, add a watermarked image data path where the watermarked image needs to be saved as per Figure 3.

```
# Save the watermarked image
cv2.imwrite('/Users/aishwaryatidke/Desktop/Thesis/code/TEST/new code/code/output/watermarked_image.png', watermarked_image)
```

Figure 3: Add watermarked image path

- Go to the directory where script is placed and execute the following command

python test.py

- After running, it will prompt "Please enter the path to the image:" where the cover image path must be typed.

5 Project details

This section will go through project details including functions and code snapshots.

5.1 Homomorphic Transformation

The project starts by transforming the original image homomorphically. This adjustment improves the image's contrast in a way that is resistant to variations in lighting. It uses numpy libraries to perform mathematical operations like logarithmic transformation, Fourier transformation, Inverse Fourier transform, and Exponential transformation as shown in Figure 4

5.2 Redundant Discrete Wavelet Transform (RDWT)

The Redundant Discrete Wavelet Transform (RDWT) is applied to the homomorphically transformed image using this function. NumPy is used for numerical calculations, and PyWavelets (pywt) is used for wavelet transformation. The image is padded, then decomposed with RDWT and the 'haar' wavelet and recreated as shown in Figure 5

5.3 QR Code Generation

QR image is generated by taking input data from user by using qrcode library as given in Figure 6

5.4 Encryption

The QR code data is encrypted using this function. It makes use of NumPy to handle arrays, and ChaCha20Poly1305 for encryption. Encryption is performed using the ChaCha20Poly1305 algorithm as shown in Figure 7 .

5.5 Chaotic Logistic Map

Using the logistic map as given in Figure 8, this function generates a chaotic sequence. NumPy is used to handle arrays. The logistic map is used iteratively to construct a chaotic sequence of the given length.

[1] https://www.hlevkin.com/hlevkin/06testimages.htm
[2] https://sipi.usc.edu/database/database.php?volume=misc

```python
# Function to apply Homomorphic Transform to the image
def homomorphic_transform(image, gamma=1.0, cutoff_frequency=30):
    # Step 1: Convert to logarithmic scale
    log_image = np.log1p(np.float32(image))

    # Step 2: Apply Fourier transform
    spectrum = np.fft.fft2(log_image)

    # Step 3: Create a high-pass filter
    rows, cols = image.shape
    center_row, center_col = rows // 2, cols // 2
    mask = np.ones((rows, cols), np.uint8)
    radius = cutoff_frequency
    center = (center_row, center_col)
    cv2.circle(mask, center, radius, 0, -1)

    # Apply the high-pass filter
    spectrum_filtered = spectrum * mask

    # Step 4: Inverse Fourier transform
    inverse_transform = np.fft.ifft2(spectrum_filtered)

    # Step 5: Exponential transformation
    enhanced_image = np.expm1(np.real(inverse_transform))

    return enhanced_image
```

Figure 4: Homomorphic transformation

```python
# Function to apply RDWT to the image
def rdwt_transform(image):
    rows, cols = image.shape
    # Pad the image to make the dimensions even
    if rows % 2 == 1:
        image = np.vstack([image, np.zeros((1, cols))])
    if cols % 2 == 1:
        image = np.hstack([image, np.zeros((rows + (rows % 2), 1))])

    # Decompose the image using RDWT
    coeffs = pywt.swt2(image, 'haar', level=1)

    # Reconstruction
    reconstructed_image = pywt.iswt2(coeffs, 'haar')

    # Remove padding if added
    if rows % 2 == 1 or cols % 2 == 1:
        reconstructed_image = reconstructed_image[:rows, :cols]

    return np.uint8(reconstructed_image)
```

Figure 5: Redundant Discrete Wavelet Transform (RDWT)

```python
# Generate QR code
qr = qrcode.QRCode(
    version=1,
    error_correction=qrcode.constants.ERROR_CORRECT_L,
    box_size=10,
    border=4,
)

qr.add_data('/Users/aishwaryatidke/Desktop/Thesis/code/TEST/new code
qr.make(fit=True)

# Create an image from the QR Code instance
qr_image = qr.make_image(fill_color="black", back_color="white")
```

Figure 6: QR image generation

```python
# Function to encrypt and compress the QR code
def encrypt_and_compress_qr(qr_data):
    # Encrypt QR data
    key = ChaCha20Poly1305.generate_key()
    chacha = ChaCha20Poly1305(key)
    nonce = os.urandom(12)
    encrypted_qr = chacha.encrypt(nonce, qr_data, None)
```

Figure 7: XChaCha20-Poly1305 encryption algorithm

```python
# New function to apply Chaotic Logistic Map
def chaotic_logistic_map(length, r=4, x0=0.5):
    # r: the bifurcation parameter
    # x0: the initial value
    # length: the length of the chaotic sequence to generate
    chaotic_sequence = np.empty(length, dtype=np.uint8)
    x = x0
    for i in range(length):
        x = r * x * (1 - x)
        chaotic_sequence[i] = int(x * 256) % 256
    return chaotic_sequence
```

Figure 8: Chaotic Logistic Map

```python
# New function to perform Huffman coding
def huffman_encode(data):
    # Convert the numpy array data to a list of bytes if it is not already
    if isinstance(data, np.ndarray):
        data = data.tolist()

    # Build the frequency table
    frequency = {byte: data.count(byte) for byte in set(data)}
    heap = [[freq, [sym, ""]] for sym, freq in frequency.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    huffman_tree = sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))
    huffman_dict = {sym: code for sym, code in huffman_tree}
    # Encode the data
    encoded_data = ''.join(huffman_dict[byte] for byte in data)
    return encoded_data, huffman_dict
```

Figure 9: Huffman Coding

```python
# Apply SVD to the LL subband
  U, S, Vt = np.linalg.svd(ll_subband, full_matrices=False)

  # Ensure watermark has the same shape as S
  watermark_resized = cv2.resize(watermark_image, (S.shape[0], 1), interpolation=cv2.INTER_LINEAR)

  # Embed watermark into SVD components
  S += alpha * watermark_resized.flatten()

  # Reconstruct the LL subband
  modified_ll_subband = U @ np.diag(S) @ Vt
```

Figure 10: Singular Value Decomposition (SVD)

5.6 Huffman Coding

This function encodes input data using Huffman coding as per Figure 9. It employs NumPy for array manipulation, heapq for heap queue operations, and collections for collection manipulation. The function creates a frequency table, then a Huffman tree, and finally encodes the data.

5.7 Watermark Embedding - SVD

NumPy is used for numerical operations, cv2 is used for image processing, and PyWavelets (pywt) is used for wavelet transform to embed watermark in the cover image as per Figure 10. The function converts the watermark data to an image before embedding it in the image's Low-Low (LL) subband's Singular Value Decomposition (SVD) components.

```
# Calculate metrics
psnr_value = peak_signal_noise_ratio(original_image, watermarked_image)
mse_value = mean_squared_error(original_image, watermarked_image)
ssim_value, _ = structural_similarity(original_image, watermarked_image, full=True)
```

Figure 11: Evaluation metrics

5.8 Performance Evaluation

Performance is evaluated using metrics functions available in skimage.metrics library which is shown in Figure 11