Configuration Manual

MSc Research Project

Master of Science in Cyber Security

Sameer Surendra Ambilpure

Student ID: 22211586

School of Computing

National College of Ireland

Supervisor: Michael Prior

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Sameer Surendra Ambilpure		
Student ID:	22211586		
Programme:	Master of Science in Cyber Security	**Year:**	2023-2024
Module:	Practicum Part 2		
Supervisor:	Michael Prior		
Submission Due Date:	12th August 2024		
Project Title:	Configuration Manual		
Word Count:	771	**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: Sameer Surendra Ambilpure

Date: 11th August 2024

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

Sameer Surendra Ambilpure

Student ID: 22211586

Table of Contents

1 Introduction

The configuration manual includes all the details about the tools and technologies used throughout the research implementation. The experimental setup is covered in section 2. Section 3 discusses the technologies and software tools used. Section 4 provides a step-by-step guide for setting up the software tool and explains the implementation process, from importing libraries and preprocessing to steganography, compression, encryption, and their reverse processes. Finally, section 5 lists the references for the software guide.

2 Experimental Setup

The experiment was carried out on a personal computer, which was set up specifically for this purpose.

- **Hardware Specifications:** i5 Fifth generation processor, 8GB RAM, 252GB SSD
- **Operating System:** Windows 10
- **Experimental Setup:** Windows 10, Anaconda 3.1 with Python 3.9.13, and VS Code

3 Technologies and Software used for Implementation

Software Used: Anaconda 3.1 with Python 3.9.13, VS Code.

Anaconda is a free software that offers a toolkit designed for research and science. Installing Anaconda provides access to various environments where you can code in Python or R. These environments, called integrated development environments (IDEs), make coding much easier. They are similar to text processors like Microsoft Word or Google Docs for writing text, but they offer much more functionality for coding and development (Rolon-Mérette et al., 2020).

4 Implementation

Step 1:Importing necessary libraries

```
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import pickle
from PIL import Image
import numpy as np
```

Figure (1):Importing libraries

Step 2:Performing data preprocessing

```
alltext=alltext.replace("\n", "")
key=key.replace("\n", "")
print(alltext)
print(key)
```

Figure (2):Data cleaning

Step 3:Embed the secret message into the cover image by modifying the least significant bits of the image pixels.

```
def encode_steganography(image_path, data):
    img = Image.open(image_path)
    img = img.convert("RGB")
    pixels = np.array(img)
    flat_pixels = pixels.flatten()

    binary_data = ''.join(format(ord(i), '08b') for i in data)
    binary_data += '1111111111111110'  # Delimiter to indicate end of data

    if len(binary_data) > len(flat_pixels):
        raise ValueError("Data is too large to hide in this image.")

    for i in range(len(binary_data)):
        flat_pixels[i] = (flat_pixels[i] & ~1) | int(binary_data[i])

    new_pixels = flat_pixels.reshape(pixels.shape)
    new_img = Image.fromarray(new_pixels)
    new_img.save('encoded_image.png')
```

Figure (3): Data hiding using LSB

Step 4:Apply the Lempel-Ziv-Welch (LZW) algorithm to compress the image with the embedded message to reduce its size.

```
def lzw_compress(data):
    dictionary = {chr(i): i for i in range(256)}
    p = ""
    result = []
    code = 256
    for c in data:
        pc = p + c
        if pc in dictionary:
            p = pc
        else:
            result.append(dictionary[p])
            dictionary[pc] = code
            code += 1
            p = c
    if p:
        result.append(dictionary[p])
    return result
```

Figure (4): Data compression using LZW

Step 5:Encrypt the compressed data using the Advanced Encryption Standard (AES) algorithm to secure the content.

```
def aes_encrypt(key, data):
    backend = default_backend()
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=backend)
    encryptor = cipher.encryptor()
    ct = encryptor.update(data) + encryptor.finalize()
    return iv + ct
```

Figure (5):Data encryption using AES

Step 6:Decrypt the AES-encrypted data back to its compressed form using the same key used for encryption.

```
def aes_decrypt(key, data):
    backend = default_backend()
    iv = data[:16]
    ct = data[16:]
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=backend)
    decryptor = cipher.decryptor()
    return decryptor.update(ct) + decryptor.finalize()
```

Figure (6):Data decryption using AES

Step 7:Decompress the decrypted data using the LZW algorithm to retrieve the image with the embedded message.

```
def lzw_decompress(data):
    dictionary = {i: chr(i) for i in range(256)}
    result = []
    code = 256
    p = chr(data.pop(0))
    result.append(p)
    for k in data:
        if k in dictionary:
            entry = dictionary[k]
        elif k == code:
            entry = p + p[0]
        result.append(entry)
        dictionary[code] = p + entry[0]
        code += 1
        p = entry
    return ''.join(result)
```

Figure (7):Decompression using LZW

Step 8:Extract the hidden message from the image by reading the least significant bits of the image pixels.

```python
def decode_steganography(image_path):
    img = Image.open(image_path)
    pixels = np.array(img)
    flat_pixels = pixels.flatten()

    binary_data = ""
    for value in flat_pixels:
        binary_data += str(value & 1)
        if binary_data[-16:] == '1111111111111110':
            break

    binary_data = binary_data[:-16]
    data = ''.join(chr(int(binary_data[i:i+8], 2)) for i in range(0, len(binary_data), 8))
    return data
```

Figure (8):Performing reverse steganography using LSB

Step 9:Evaluate the performance metrics, including data hiding rate, encryption strength, and compression rate, to assess the efficiency and robustness of the implemented process.

```python
# Calculate Compression Rate
def calculate_compression_rate(original_size, compressed_size):
    return original_size / compressed_size

# Calculate Encryption Strength
def calculate_encryption_strength(original_data, encrypted_data):
    return len(original_data) / len(encrypted_data)

# Calculate Data Dimming Rate
def calculate_data_dimming_rate(original_data, steganographed_data):
    return (1 - (len(steganographed_data) / len(original_data))) * 100
```

Figure (9):Performance evaluation

Step 10: Testing the strength of AES-256 encryption by using Hashcat to brute-force the password and then attempting to decrypt the file with OpenSSL. Despite these efforts, the decryption was unsuccessful, indicating that your encryption is robust and the password was not easily cracked.

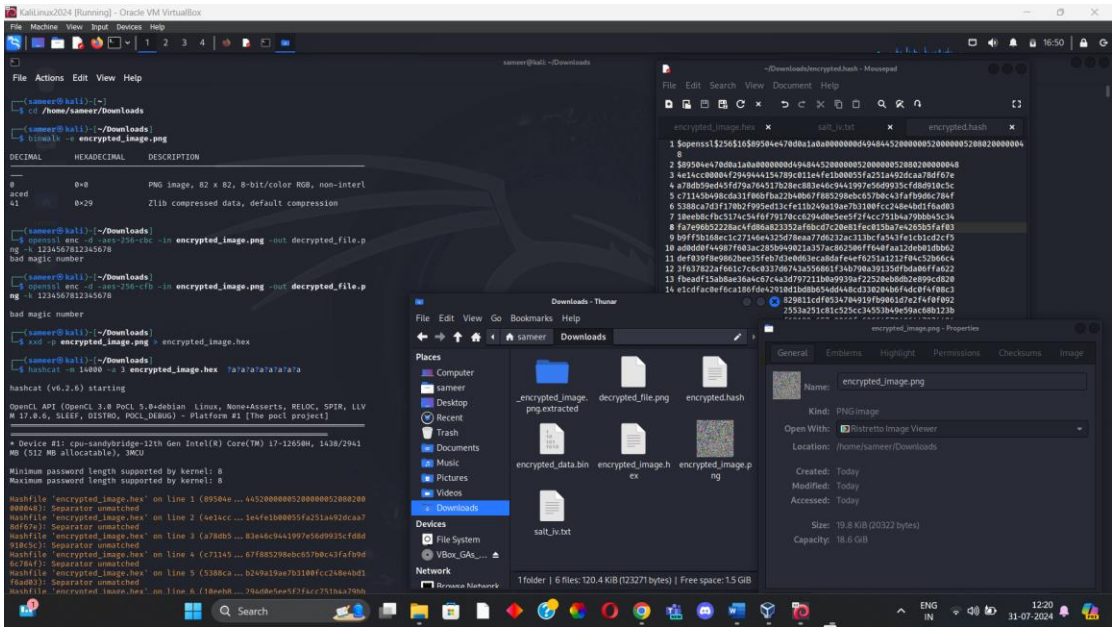

Figure (10):Testing using kali

References

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T. and Church, K. (2020). Introduction to Anaconda and Python: Installation and setup. The Quantitative Methods for Psychology, [online] 16(5), pp.S3–S11. doi:https://doi.org/10.20982/tqmp.16.5.s003.