National College of Ireland

Configuration Manual

MSc Research Project
MSc Research Practicum/Internship part 2

ABHINANDAN DIGRAJE
Student ID: X22220526

School of Computing
National College of Ireland

Supervisor: Eugene McLaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abhinandan Digraje		
Student ID:	X22220526		
Programme:	Master of Science in Cyber Security	**Year:**	2023-2024
Module:	Msc Research Practicum/Internship part 2		
Lecturer:	Eugene McLaughlin		
Submission Due Date:	12-08-2024		
Project Title:	Configuration Manual		
Word Count:	775	**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: Abhinandan Shrenik Digraje

Date: 11-08-2024

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

ABHINANDAN DIGRAJE
Student ID: x22220526

1 Introduction

The main purpose of the project is to improve the file security by implementing/adding a extra layer of security over the encryption using the magic numbers. By modifying the magic number of the files before the encryption process to prevent the malicious data insertion to make the file which is encrypted more secure. The project shows combining the magic number modification with the traditional encryption process adds an extra layer of security. This configuration manual covers the entire experimental setups, platforms, used libraries and main functions.

2 Experimental Setup

Hardware required

- OS(Operating system): Windows
- Ram : 2GB or more.
- Processor : Dual core or more.
- Memory : 100 MB or more.

3 Implementation Platform

Platform Details

- Anaconda Navigator (Version : 2.5.0) (Anaconda, n.d.)
- Jupyter Notebook (Version : 6.5.4)
- Implementation Language : python

4 Implementation

1. **Importing the main libraries**.

```
In [ ]: import os
        import hashlib
        import hmac
        import tkinter as tk
        from tkinter import filedialog, messagebox
        from tkinter.font import Font
        from tkinter import ttk, Menu
        from PIL import Image, ImageTk
        from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
        from cryptography.hazmat.primitives import padding
        from cryptography.hazmat.backends import default_backend
        import time
        import matplotlib.pyplot as plt
```

- OS : Used to generate the random IVs and to handle the file operations
- hashlib : Used for HMAC generation using hashing functions(SHA-256 & SHA-4)
- hmac : Configuring hmac for the data integrity

- tkinter : Used to implement graphical user interface(GUI)
- PIL (Pillow) : Used to operate the image processing and to display the images.
- Cryptography : Used for encryption(AES, DES), decryption , cipher operational features and padding
- time : Used to count the time taken for the process of encryption and decryption.
- matplotlib : Used to create Charts.

2. Configuring magic number dictionary.

```
# Define a dictionary of file extensions and their magic numbers
magic_numbers = {
    b'\x89PNG\r\n\x1a\n': 'PNG image',
    b'%PDF-': 'PDF document',
    b'\xff\xd8\xff\xe0': 'JPEG image',
    b'GIF87a': 'GIF image',
    b'GIF89a': 'GIF image',
    b'RIFF....AVI ': 'AVI video',
    b'OggS': 'Ogg multimedia',
    b'MThd': 'MIDI audio',
    b'\x00\x00\x01\x00': 'ICO image',
    b'\x42\x4D': 'BMP image',
    b'\x49\x49\x2A\x00': 'TIFF image',
    b'\x4D\x4D\x00\x2A': 'TIFF image',
    b'\x1F\x8B\x08': 'GZIP compressed',
    b'\x50\x4B\x03\x04': 'ZIP archive',
    b'\x50\x4B\x05\x06': 'ZIP archive',
    b'\x50\x4B\x07\x08': 'ZIP archive',
    b'\x52\x61\x72\x21': 'RAR archive',
    b'\x7F\x45\x4C\x46': 'ELF executable',
    b'\x25\x21\x50\x53': 'PostScript',
    b'\xD0\xCF\x11\xE0\xA1\xB1\x1A\xE1': 'MS Office document',
    b'\x50\x4B\x03\x04\x14\x00\x06\x00': 'DOCX',
    b'\x50\x4B\x03\x04\x14\x00\x08\x00': 'DOCX',
    b'\x50\x4B\x03\x04\x14\x00\x08\x00': 'XLSX',
    b'\x50\x4B\x03\x04\x14\x00\x08\x00': 'PPTX',
    b'\x50\x4B\x03\x04\x14\x00\x06\x00': 'ODT',
    b'\x50\x4B\x03\x04\x14\x00\x06\x00': 'ODS',
    b'\x50\x4B\x03\x04\x14\x00\x06\x00': 'ODP',
    b'\x4D\x5A': 'EXE executable',
    b'\x23\x21': 'Script or text file',
    b'\xCA\xFE\xBA\xBE': 'Java class file',
    b'\xEF\xBB\xBF': 'UTF-8 BOM',
    b'\xFF\xFE': 'UTF-16LE BOM',
    b'\xFE\xFF': 'UTF-16BE BOM',
    b'\x00\x61\x73\x6D': 'WebAssembly binary',
    b'\x1A\x45\xDF\xA3': 'Matroska media',
    b'\x66\x4C\x61\x43': 'FLAC audio',
    b'\x4F\x67\x67\x53': 'Ogg Vorbis audio',
    b'\x52\x49\x46\x46': 'WAV audio',
    b'\x49\x44\x33': 'MP3 audio',
    b'\x77\x4F\x46\x46': 'WOFF font',
    b'\x77\x4F\x46\x32': 'WOFF2 font',
    b'\x00\x00\x01\xBA': 'MPEG video',
    b'\x00\x00\x01\xB3': 'MPEG video',
}
```

- Magic number dictionary used to correctly identify the file type.

3. Modification of magic number.

- Modifying Manually : Used to alter the magic number manually by entering the number.

```
def manual_modify(magic_number, xor_value):
    return bytes([(b + xor_value) % 256 for b in magic_number])
```

- Restoration : This function restores the original magic number by reversing the XOR function which is implemented in manually modification.

```
def manual_restore(modified_magic_number, xor_value):
    return bytes([(b - xor_value) % 256 for b in modified_magic_number])
```

- Modifying Dynamically : This function is used to alter the magic number dynamically(automatically) using standard XOR operation (predefined value).

```
# Modify the magic number
modified_magic_number = manual_modify(magic_number, 1)  # Dynamic modification
print(f"Modified Magic Number: {modified_magic_number}")
```

4. **Implenting HMAC.**
   - Configured HMAC for data integrity check using cryptographic has functions(SHA-256, SHA3)

```python
def generate_hmac(data, key, algorithm):
    if algorithm == 'SHA-3':
        return hmac.new(key, data, hashlib.sha3_256).digest()
    elif algorithm == 'SHA-256':
        return hmac.new(key, data, hashlib.sha256).digest()
    else:
        raise ValueError("Unsupported HMAC algorithm.")
```

5. **Length correction using Key padding for ecnryption.**
   - Used to make sure the length of the encryption key.

```python
def pad_key(key, length):
    if len(key) > length:
        return key[:length]
    return key.ljust(length, b'\0')
```

6. **Encrytion Process.**
   - IVs : Used to generate a random initialization vector for the AES and DES.
   - Cipher : Used to initialize cipher feature in the encryption algorithm process.
   - Padding : Used to align the data length with the block size of algorithms

```python
def encrypt_file(data, key, hmac_key, algorithm, key_length, hmac_algorithm, filename):
    iv = os.urandom(16) if algorithm == 'AES' else os.urandom(8)
    if algorithm == 'AES':
        if key_length == 128:
            key = pad_key(key, 16)  # AES-128 uses 16-byte key
        elif key_length == 256:
            key = pad_key(key, 32)  # AES-256 uses 32-byte key
        else:
            raise ValueError("Unsupported key length for AES.")
        cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    elif algorithm == 'DES':
        key = pad_key(key, 8)  # DES key is 8 bytes
        cipher = Cipher(algorithms.TripleDES(key), modes.CBC(iv), backend=default_backend())
    else:
        raise ValueError("Unsupported encryption algorithm.")

    encryptor = cipher.encryptor()
    padder = padding.PKCS7(cipher.algorithm.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    # Generate HMAC
    hmac_value = generate_hmac(encrypted_data, hmac_key, hmac_algorithm)

    # Include HMAC, IV, and file extension in the encrypted file
    file_extension = os.path.splitext(filename)[1].encode()
    return len(file_extension).to_bytes(1, 'big') + file_extension + hmac_value + iv + encrypted_data
```

7. **Decryption Process.**
   - Verification of HMAC : Used for verification of the data and to ensure the data has not been altered or tempered by comparing with the stored HMAC.
   - Cipher : Used to initialize cipher feature in the decryption algorithm process.
   - Un-padding : Used to remove the padding which was implemented during the encryption.
   - Restoration of Magic number : Used to restore the magic number to its original.

```python
def decrypt_file(encrypted_data, key, hmac_key, algorithm, key_length, hmac_algorithm):
    ext_len = encrypted_data[0]
    file_extension = encrypted_data[1:1+ext_len]
    hmac_len = hashlib.new('sha3_256' if hmac_algorithm == 'SHA-3' else hmac_algorithm.lower()).digest_size
    hmac_value = encrypted_data[1+ext_len:1+ext_len+hmac_len]

    if algorithm == 'AES':
        if key_length == 128:
            key = pad_key(key, 16)  # AES-128 uses 16-byte key
        elif key_length == 256:
            key = pad_key(key, 32)  # AES-256 uses 32-byte key
        else:
            raise ValueError("Unsupported key length for AES.")
        iv = encrypted_data[1+ext_len+hmac_len:1+ext_len+hmac_len+16]
        encrypted_data = encrypted_data[1+ext_len+hmac_len+16:]
    elif algorithm == 'DES':
        key = pad_key(key, 8)  # DES key is 8 bytes
        iv = encrypted_data[1+ext_len+hmac_len:1+ext_len+hmac_len+8]
        encrypted_data = encrypted_data[1+ext_len+hmac_len+8:]
    else:
        raise ValueError("Unsupported encryption algorithm.")

    # Verify HMAC
    calculated_hmac = generate_hmac(encrypted_data, hmac_key, hmac_algorithm)
    if not hmac.compare_digest(hmac_value, calculated_hmac):
        raise ValueError("Data integrity check failed. The file has been tampered with.")

    cipher = Cipher(algorithms.AES(key) if algorithm == 'AES' else algorithms.TripleDES(key), modes.CBC(iv), backend=default_ba
    decryptor = cipher.decryptor()
    decrypted_padded_data = decryptor.update(encrypted_data) + decryptor.finalize()
    unpadder = padding.PKCS7(cipher.algorithm.block_size).unpadder()
    decrypted_data = unpadder.update(decrypted_padded_data) + unpadder.finalize()
    return decrypted_data, file_extension
```

8. File handling and Integration of GUI.

- File Reading and saving the encrypted and decrypted file in system.

```python
def read_file(filename, magic_number_length):
    if not os.path.isfile(filename):
        raise FileNotFoundError(f"File does not exist: {filename}")
    with open(filename, 'rb') as f:
        magic_number = f.read(magic_number_length)
        file_data = f.read()
    return file_data, magic_number
```

```python
def download_file(filename):
    try:
        # Prompt user for save location
        save_filename = filedialog.asksaveasfilename(defaultextension=".encrypted")
        if save_filename:
            # Copy file to save location
            with open(filename, 'rb') as f_src, open(save_filename, 'wb') as f_dest:
                f_dest.write(f_src.read())
            messagebox.showinfo("Download Complete", "File downloaded successfully.")
    except Exception as e:
        messagebox.showerror("Error", str(e))

def save_decrypted_file(data, file_extension):
    try:
        # Prompt user for save location with original extension
        save_filename = filedialog.asksaveasfilename(defaultextension=file_extension)
        if save_filename:
            # Write decrypted data to file
            with open(save_filename, 'wb') as f:
                f.write(data)
            messagebox.showinfo("Save Complete", "Decrypted file saved successfully.")
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

- GUI Implementation:

```
# GUI Setup
root = tk.Tk()
root.title("File Encryption and Decryption")

# Maximize the window
root.state('zoomed')

# Set Times New Roman font
times_font = Font(family="Times New Roman", size=12)

# Create Notebook for tabs
notebook = ttk.Notebook(root)
notebook.place(relx=0.3, rely=0.5, anchor="center")

# Manual tab
manual_tab = tk.Frame(notebook, padx=10, pady=10, bg='lightgrey')
notebook.add(manual_tab, text="Manual")

label_manual_magic_number_length = tk.Label(manual_tab, text="Magic Number Length:", bg='lightgrey', font=times_font)
label_manual_magic_number_length.grid(row=0, column=0, pady=5)
entry_manual_magic_number_length = tk.Entry(manual_tab, font=times_font)
entry_manual_magic_number_length.grid(row=0, column=1, pady=5)

label_manual_encryption_key = tk.Label(manual_tab, text="Encryption Key:", bg='lightgrey', font=times_font)
label_manual_encryption_key.grid(row=1, column=0, pady=5)
entry_manual_encryption_key = tk.Entry(manual_tab, show="*", font=times_font)
entry_manual_encryption_key.grid(row=1, column=1, pady=5)

manual_encryption_key_toggle = tk.Button(manual_tab, text='Show', command=lambda: toggle_password(entry_manual_encryption_key,
manual_encryption_key_toggle.grid(row=1, column=2, padx=5)

label_manual_hmac_key = tk.Label(manual_tab, text="HMAC Key:", bg='lightgrey', font=times_font)
label_manual_hmac_key.grid(row=2, column=0, pady=5)
entry_manual_hmac_key = tk.Entry(manual_tab, show="*", font=times_font)
entry_manual_hmac_key.grid(row=2, column=1, pady=5)

manual_hmac_key_toggle = tk.Button(manual_tab, text='Show', command=lambda: toggle_password(entry_manual_hmac_key, manual_hmac_
manual_hmac_key_toggle.grid(row=2, column=2, padx=5)

label_manual_encryption_method = tk.Label(manual_tab, text="Encryption Method:", bg='lightgrey', font=times_font)
label_manual_encryption_method.grid(row=3, column=0, pady=5)
manual_encryption_method_var = tk.StringVar()
manual_encryption_method_options = tk.OptionMenu(manual_tab, manual_encryption_method_var, "AES", "DES")
manual_encryption_method_var.set("AES")
manual_encryption_method_options.grid(row=3, column=1, pady=5)

label_manual_key_length = tk.Label(manual_tab, text="Key Length:", bg='lightgrey', font=times_font)
label_manual_key_length.grid(row=4, column=0, pady=5)
manual_key_length_var = tk.IntVar()
manual_key_length_options = tk.OptionMenu(manual_tab, manual_key_length_var, 128, 256)
manual_key_length_var.set(128)
manual_key_length_options.grid(row=4, column=1, pady=5)
```

```
label_manual_hmac_algorithm = tk.Label(manual_tab, text="HMAC Algorithm:", bg='lightgrey', font=times_font)
label_manual_hmac_algorithm.grid(row=5, column=0, pady=5)
manual_hmac_algorithm_var = tk.StringVar()
manual_hmac_algorithm_options = tk.OptionMenu(manual_tab, manual_hmac_algorithm_var, "SHA-256", "SHA-3")
manual_hmac_algorithm_var.set("SHA-256")
manual_hmac_algorithm_options.grid(row=5, column=1, pady=5)

# Add manual magic number option
manual_method_var = tk.StringVar()
manual_method_options = tk.OptionMenu(manual_tab, manual_method_var, "Manual XOR")
manual_method_var.set("Manual XOR")
manual_method_options.grid(row=6, column=0, pady=5)

label_manual_magic_number = tk.Label(manual_tab, text="Manual Number:", bg='lightgrey', font=times_font)
label_manual_magic_number.grid(row=7, column=0, pady=5)
entry_manual_magic_number = tk.Entry(manual_tab, font=times_font)
entry_manual_magic_number.grid(row=7, column=1, pady=5)

button_manual_encrypt = tk.Button(manual_tab, text="Encrypt Files", command=process_manual_files, font=times_font)
button_manual_encrypt.grid(row=8, column=0, columnspan=2, pady=10, sticky="we")

button_manual_decrypt = tk.Button(manual_tab, text="Decrypt Files", command=decrypt_and_check_manual, font=times_font)
button_manual_decrypt.grid(row=9, column=0, columnspan=2, pady=10, sticky="we")

# Dynamic tab
dynamic_tab = tk.Frame(notebook, padx=10, pady=10, bg='lightgrey')
notebook.add(dynamic_tab, text="Dynamic")

label_dynamic_magic_number_length = tk.Label(dynamic_tab, text="Magic Number Length:", bg='lightgrey', font=times_font)
label_dynamic_magic_number_length.grid(row=0, column=0, pady=5)
entry_dynamic_magic_number_length = tk.Entry(dynamic_tab, font=times_font)
entry_dynamic_magic_number_length.grid(row=0, column=1, pady=5)

label_dynamic_encryption_key = tk.Label(dynamic_tab, text="Encryption Key:", bg='lightgrey', font=times_font)
label_dynamic_encryption_key.grid(row=1, column=0, pady=5)
entry_dynamic_encryption_key = tk.Entry(dynamic_tab, show="*", font=times_font)
entry_dynamic_encryption_key.grid(row=1, column=1, pady=5)

dynamic_encryption_key_toggle = tk.Button(dynamic_tab, text='Show', command=lambda: toggle_password(entry_dynamic_encryption_ke
dynamic_encryption_key_toggle.grid(row=1, column=2, padx=5)

label_dynamic_hmac_key = tk.Label(dynamic_tab, text="HMAC Key:", bg='lightgrey', font=times_font)
label_dynamic_hmac_key.grid(row=2, column=0, pady=5)
entry_dynamic_hmac_key = tk.Entry(dynamic_tab, show="*", font=times_font)
entry_dynamic_hmac_key.grid(row=2, column=1, pady=5)

dynamic_hmac_key_toggle = tk.Button(dynamic_tab, text='Show', command=lambda: toggle_password(entry_dynamic_hmac_key, dynamic_h
dynamic_hmac_key_toggle.grid(row=2, column=2, padx=5)

label_dynamic_encryption_method = tk.Label(dynamic_tab, text="Encryption Method:", bg='lightgrey', font=times_font)
label_dynamic_encryption_method.grid(row=3, column=0, pady=5)
dynamic_encryption_method_var = tk.StringVar()
dynamic_encryption_method_options = tk.OptionMenu(dynamic_tab, dynamic_encryption_method_var, "AES", "DES")
dynamic_encryption_method_var.set("AES")
dynamic_encryption_method_options.grid(row=3, column=1, pady=5)
```

```
label_dynamic_key_length = tk.Label(dynamic_tab, text="Key Length:", bg='lightgrey', font=times_font)
label_dynamic_key_length.grid(row=4, column=0, pady=5)
dynamic_key_length_var = tk.IntVar()
dynamic_key_length_options = tk.OptionMenu(dynamic_tab, dynamic_key_length_var, 128, 256)
dynamic_key_length_var.set(128)
dynamic_key_length_options.grid(row=4, column=1, pady=5)

label_dynamic_hmac_algorithm = tk.Label(dynamic_tab, text="HMAC Algorithm:", bg='lightgrey', font=times_font)
label_dynamic_hmac_algorithm.grid(row=5, column=0, pady=5)
dynamic_hmac_algorithm_var = tk.StringVar()
dynamic_hmac_algorithm_options = tk.OptionMenu(dynamic_tab, dynamic_hmac_algorithm_var, "SHA-256", "SHA-3")
dynamic_hmac_algorithm_var.set("SHA-256")
dynamic_hmac_algorithm_options.grid(row=5, column=1, pady=5)

button_dynamic_encrypt = tk.Button(dynamic_tab, text="Encrypt Files", command=process_dynamic_files, font=times_font)
button_dynamic_encrypt.grid(row=6, column=0, columnspan=2, pady=10, sticky="we")

button_dynamic_decrypt = tk.Button(dynamic_tab, text="Decrypt Files", command=decrypt_and_check_dynamic, font=times_font)
button_dynamic_decrypt.grid(row=7, column=0, columnspan=2, pady=10, sticky="we")
```

5 References

Anaconda, n.d. *Anaconda Navigator.* [Online]
Available at: https://docs.anaconda.com/navigator/
[Accessed 2024].