Stegcryption: Graphical User Interface (GUI) Based File Protection System Utilizing Cryptography and Steganography

Khabeellan A/L Sugumar
Forensics & Cybersecurity Research Center (FSEC)
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp062939@mail.apu.edu.my

Julia Juremi
Forensics & Cybersecurity Research Center (FSEC)
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
julia.juremi@staffemail.apu.edu.my

***Abstract*—Cyphers, or secret codes, have played an essential role in history as a means of communication and a tool for espionage and intelligence gathering. Throughout the centuries, various civilisations have developed cyphers to protect their messages from being intercepted by unauthorised parties. Encryption has evolved from the substitution and transposition cyphers used in history to complicated algorithms and protocols with excellent security and anonymity, as stated above. Encryption which comes under cryptography has become essential for protecting sensitive data in banking, healthcare, and national security. Its development and ongoing progress will likely continue to ensure digital communication and data security. Combining steganography and cryptography adds security, conceals, and verifies information. Several approaches combined provide an extra layer of security. Steganography may disguise information, making it harder for an enemy even to realise there is anything to decode. Whereas cryptography ensures data integrity and authentication, guaranteeing that data has not been tampered with or altered.**

Keywords—ciphers, cryptography, encryption, steganography

I. Introduction

Since ancient times, messages have been coded using cryptography, which is still utilised today in various applications such as e-commerce, bank cards, and computer passwords (Fortinet, 2023). Cryptography vastly uses encryption techniques; however, it is not limited to this, other techniques such as digital signatures, hashing, and key exchange protocols that safeguard data and communication come under the term cryptography as well (GeeksForGeeks, 2021). Steganography, much like cryptography, have been around for centuries. The name itself has Greek origins since it combines the words "steganos" which translates to concealed, and the word “graphein” which translates to “writing” (Fiscutean, 2021). Unlike encryption which keeps the information confidential by rendering it unreadable, steganography focuses on hiding the existence of the information entirely (Simplilearn, 2023). This is a clear advantage of steganography as by disguising the fact that a message even exists, it makes it far more challenging for unauthorised individuals to detect and intercept the sensitive information (Simplilearn, 2023). Therefore, steganography is a good option if multi-layer security wants to be employed on top of cryptography. Steganography can be used to hide information in various types of digital media, including images, audio, video, and text files. Steganography also enables individuals to communicate with others covertly in scenarios where open communication could be dangerous.

II. SIMILAR SYSTEMS IN THE MARKET

A. SilentEye

The SilentEye application is a free, open-source software used for steganography, the process of concealing private messages within another file, like a picture or audio file. It enables users to hide data from plain view by encrypting and embedding text or files within digital media (SilentEye, n.d.). Advanced algorithms are used by the programme to ensure that the hidden data is secure and undetectable (SilentEye, n.d.). The hidden data can then be extracted and decrypted by the recipient using the same application (SilentEye, n.d.). SilentEye is an effective tool for those who need to securely communicate with one another covertly.

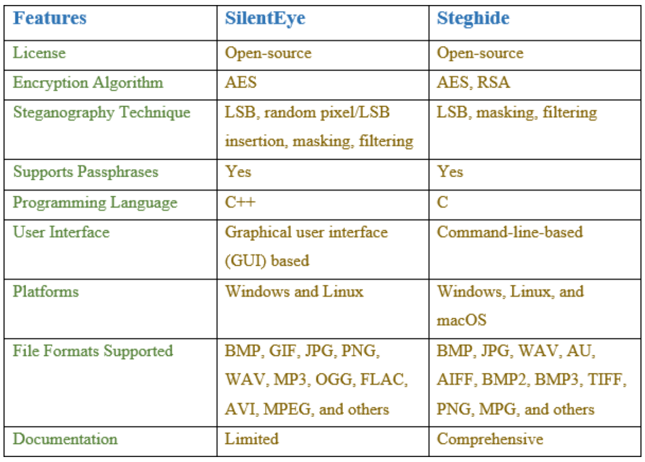
Figure 1 SilentEye logo (SilentEye, n.d.).

B. Steghide

The Steghide application is a free and open-source application used for steganography, which is the practice of hiding secret data within other non-secret files. It enables users to embed and extract secret messages or files into and from digital media. Steghide secures the hidden data using powerful encryption algorithms like AES and RSA and supports several steganography techniques like LSB, mask, and filter (Steghide, n.d.). Steghide additionally encourages the use of digital signatures and passphrases for secure communication and to guarantee the authenticity of the hidden data (Steghide, n.d.). Overall, Steghide is a strong tool for those who need to safely communicate while keeping it hidden from unauthorized parties.

Figure 2 Steghide logo (Steghide, n.d.)

Comparison between SilentEye and Steghide:



Features	SilentEye	Steghide
License	Open-source	Open-source
Encryption Algorithm	AES	AES, RSA
Steganography Technique	LSB, random pixel/LSB insertion, masking, filtering	LSB, masking, filtering
Supports Passphrases	Yes	Yes
Programming Language	C++	C
User Interface	Graphical user interface (GUI) based	Command-line-based
Platforms	Windows and Linux	Windows, Linux, and macOS
File Formats Supported	BMP, GIF, JPG, PNG, WAV, MP3, OGG, FLAC, AVI, MPEG, and others	BMP, JPG, WAV, AU, AIFF, BMP2, BMP3, TIFF, PNG, MPG, and others
Documentation	Limited	Comprehensive

Figure 3 Comparison between SilentEye and Steghide

III. Stegcryption Development

A. *Programming Language*

The chosen language to develop the proposed system is the Python programming language and PyCharm has been selected as the chosen Integrated Development Environment (IDE). One of the reasons for this choice is due to its ease of use, Python's easy to read syntax simplifies the programming process in comparison to Java. On top of this, based on research, it could be seen that Python has a vast library of built-in modules and third-party libraries that make it easier to implement encryption and steganography features, the Java has this too but not as vast as it is for Python.

B. *Libraries*

i. PyCryptodome Library

Library Name: PyCryptodome
Pip Install Command: pip install pycryptodome
Source: (https://pycryptodome.readthedocs.io/en/latest/src/introduction.html)
Description: A Python module that provides several encryption algorithms which include AES.

ii. Stegano Library

Library Name: Stegano
Pip Install Command: pip install stegano Source: (https://pypi.org/project/stegano/)
Description: A Python module that provides LSB algorithm for steganography implementation.

iii. Tkinter Library

Library Name: Tkinter
Pip Install Command: pip install tk
Source: (https://www.tutorialspoint.com/how-to-install-tkinter-in-python)
Description: A Python module that enables users to create a Graphical User Interface (GUI).

C. *Operating System*

For a computer to work, a functionally running operating system is essential. Without an operating system installed, personal computers are essentially worthless. The functionalities and the software running on computers are managed and watched over by the operating system to ensure that all software obtains what is required. In the market today, the most common operating systems include Windows, MacOS, and Linux. The proposed system will be developed using the Windows 10 operating system. Windows 10 was released on 29 of July 2015 by Microsoft (Fisher, 2022). The selected programming language and IDE can run smoothly on the selected operating system acting as a suitable choice for the development.

IV. STEGCRYPTION

Once the system initially starts up, the user will see the interface shown in Figure 4 below. It serves as both the home page and the page for file encryption. This window intends to give users a quick and easy method for password-protecting files using the AES encryption technology. The "Menu" portion on the left and the "File Encryption" part on the right make up the interface's two primary sections. Quick access to the system's various features, including file decryption, file hiding, and file revealing, is provided by the "Menu" section. The "File Encryption" part is solely for encrypting files. The user needs to enter both the file location of the file that they want to encrypt as well as the password in this window's "File Encryption" section. The user has two options for choosing the file: either typing in the location of the file or choosing the browse option to choose the specific file from the File Explorer. The user must click the "Encrypt File" button after entering the file location and the password to encrypt that file.

Users may effortlessly encrypt files due to the File Encryption interface's simple and straightforward design.

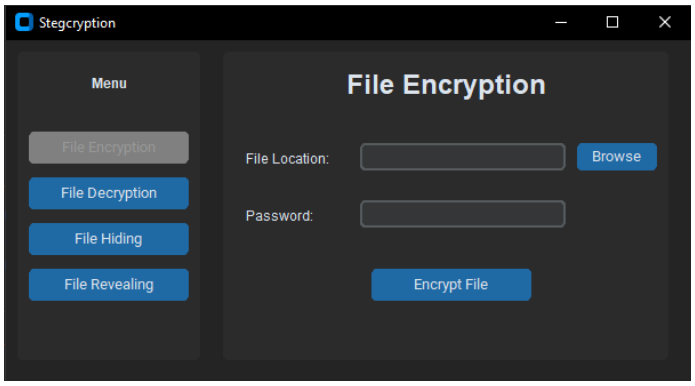


Figure 1 Stegcryption File Encryption Window

The GUI File Decryption interface serves as the file decryption page and gives users an easy-to-use method to decrypt their password-protected AES-encrypted files. The interface as shown in Figure 5 is separated into two main areas, like the previous page: the "Menu" portion on the left and the "File Decryption" section, which is responsible for the decryption process, on the right. Users are prompted to provide the file location of the encrypted file and the correct decryption password. The file location can be entered manually, or the browse button can be used to choose the file from File Explorer. Once both the file location and password are entered, users can click the "Decrypt File" button to decrypt the specific file. The function seeks to ensure user-friendliness and effectiveness for a seamless decryption process.

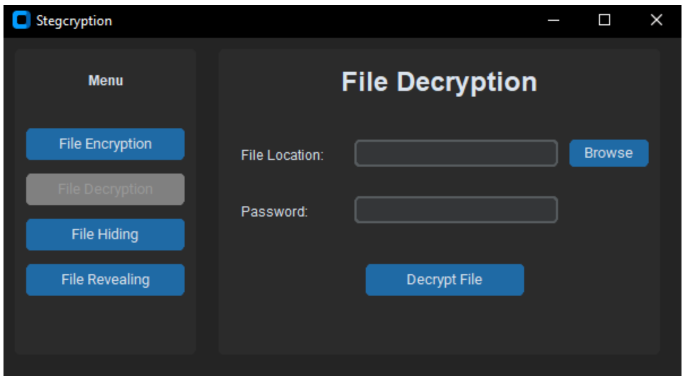


Figure 5 Stegcryption File Decryption Window

The GUI File Hiding interface acts as the file hiding page, allowing users to hide files within image files (png) using steganography and a password. The interface as shown in Figure 6 is separated into two main sections: on the left the "Menu" section and on the right the "File Hiding" section, which is dedicated to the hiding process. Users are prompted to provide the file path of the desired file to be hidden, the image location (png) in which the file should be hidden, and a password to safeguard the file. They can either type the file and image locations or use the browse buttons to use File Explorer to select the files. Furthermore, to ensure proper hiding, the selected image must be at least 8 times larger than the selected file. Once all fields have been filled, users can perform the hiding procedure by clicking the "Hide File" button. The goal of this function is to offer a user-friendly and straightforward experience for hiding sensitive files within images.

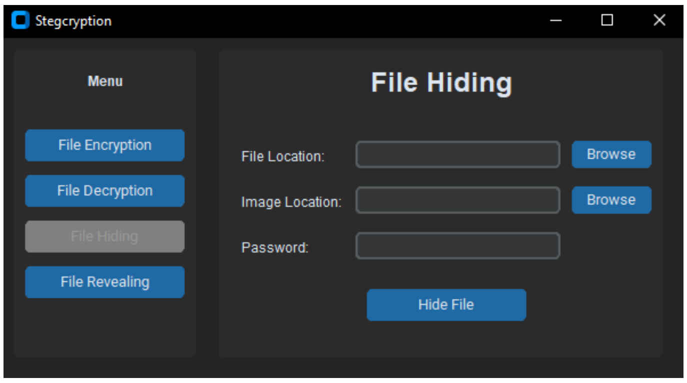


Figure 6 Stegcryption File Hiding Window

The GUI File Revealing interface acts as the file revealing page, allowing users to easily reveal hidden files from images (png) utilizing steganography and the correct password. The interface as shown in Figure 7 is separated into two main sections: the "Menu" component on the left, which provides quick access to additional capabilities, and the "File Revealing" component on the right, which is dedicated to the revealing process. Users are required to enter the image location (png format) containing the hidden file and the correct password used during hiding for the revealing process. They can either manually type in the image location path or use the browse button to select the image from File Explorer. It's essential that the selected image contains a hidden file for successful revealing. Once all fields have been filled, users can begin the revealing process by clicking the "Reveal File" button. This function aims to create an accessible and user-friendly experience for recovering hidden files from images with the correct password.

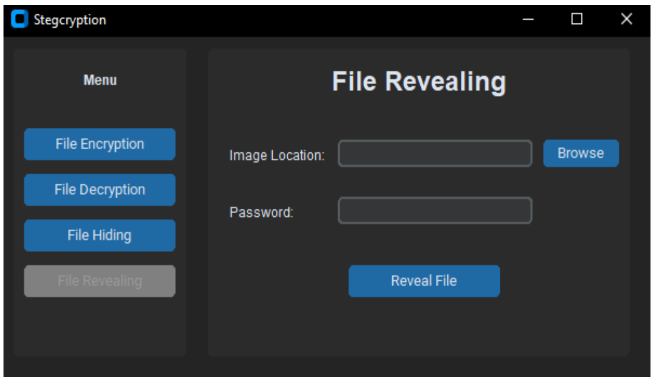
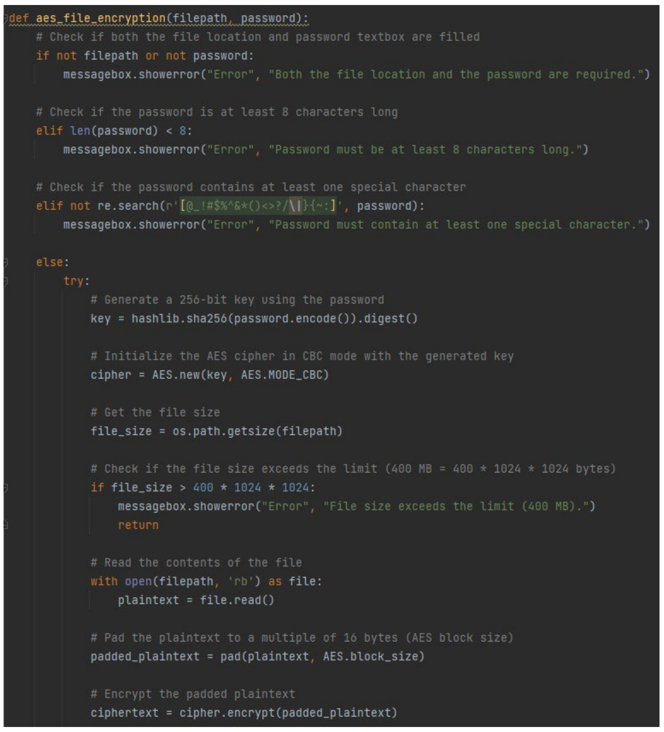


Figure 7 Stegcryption File Revealing Window

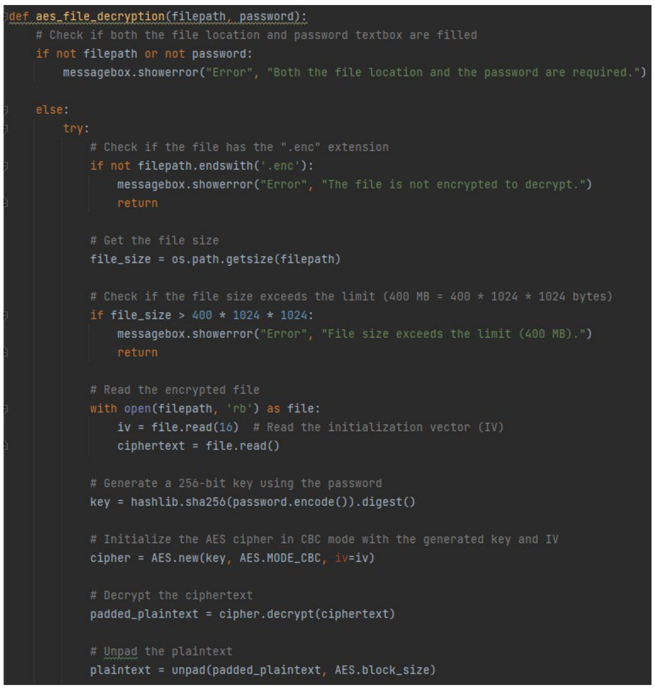
Figure 8 shows the function called aes_file_encryption which encrypts files using the Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining (CBC) mode. The function takes the file path to be encrypted as well as a password entered by the user. It verifies the length of the password, ensuring that it is at least 8 characters long and contains at least one special character. If the validation is successful, the code generates a 256-bit encryption key from the password, reads the specified file's content, pads it to a multiple of the AES block size, and encrypts it with AES-CBC. The encrypted data is then written to a new file within a "e files" folder created in the same place as the input file, along with the initialization vector (IV). The function also handles various error conditions and displays suitable error or success messages.



```python
def aes_file_encryption(filepath, password):
    # Check if both the file location and password textbox are filled
    if not filepath or not password:
        messagebox.showerror("Error", "Both the file location and the password are required.")

    # Check if the password is at least 8 characters long
    elif len(password) < 8:
        messagebox.showerror("Error", "Password must be at least 8 characters long.")

    # Check if the password contains at least one special character
    elif not re.search(r'[@_!#$%^&*()<>?/\|}{~:]', password):
        messagebox.showerror("Error", "Password must contain at least one special character.")

    else:
        try:
            # Generate a 256-bit key using the password
            key = hashlib.sha256(password.encode()).digest()

            # Initialize the AES cipher in CBC mode with the generated key
            cipher = AES.new(key, AES.MODE_CBC)

            # Get the file size
            file_size = os.path.getsize(filepath)

            # Check if the file size exceeds the limit (400 MB = 400 * 1024 * 1024 bytes)
            if file_size > 400 * 1024 * 1024:
                messagebox.showerror("Error", "File size exceeds the limit (400 MB).")
                return

            # Read the contents of the file
            with open(filepath, 'rb') as file:
                plaintext = file.read()

            # Pad the plaintext to a multiple of 16 bytes (AES block size)
            padded_plaintext = pad(plaintext, AES.block_size)

            # Encrypt the padded plaintext
            ciphertext = cipher.encrypt(padded_plaintext)
```

Figure 8: Sample Code for File Encryption

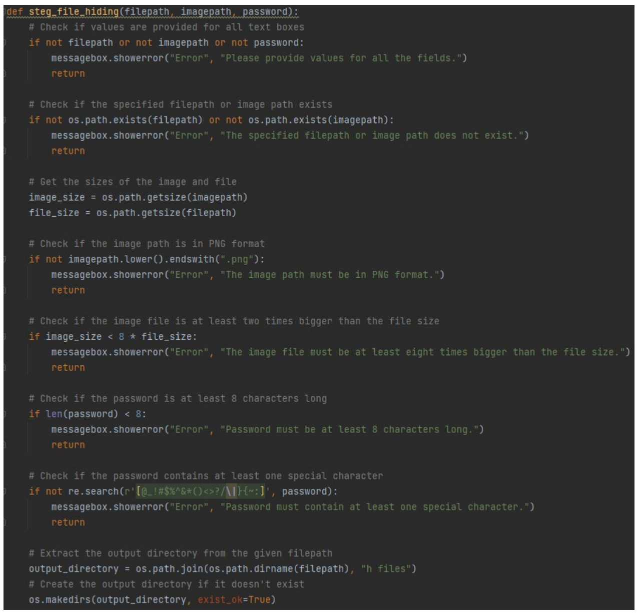
Figure 9 shows the aes_file_decryption function that performs the decryption on the encrypted files. It starts by checking if the required file path and password are provided. If so, it validates whether the file is encrypted, size, and prompts the user for a password. The code then reads the initialization vector and ciphertext from the encrypted file, generates a decryption key from the password, and uses AES in CBC mode to decrypt the ciphertext. After decryption, it removes padding and saves the decrypted data to a new file in a folder named "d files". The code also handles potential errors, including file not found, incorrect password, and other exceptions, displaying appropriate messages to the user.



```python
def aes_file_decryption(filepath, password):
    # Check if both the file location and password textbox are filled
    if not filepath or not password:
        messagebox.showerror("Error", "Both the file location and the password are required.")

    else:
        try:
            # Check if the file has the ".enc" extension
            if not filepath.endswith('.enc'):
                messagebox.showerror("Error", "The file is not encrypted to decrypt.")
                return

            # Get the file size
            file_size = os.path.getsize(filepath)

            # Check if the file size exceeds the limit (400 MB = 400 * 1024 * 1024 bytes)
            if file_size > 400 * 1024 * 1024:
                messagebox.showerror("Error", "File size exceeds the limit (400 MB).")
                return

            # Read the encrypted file
            with open(filepath, 'rb') as file:
                iv = file.read(16)  # Read the initialization vector (IV)
                ciphertext = file.read()

            # Generate a 256-bit key using the password
            key = hashlib.sha256(password.encode()).digest()

            # Initialize the AES cipher in CBC mode with the generated key and IV
            cipher = AES.new(key, AES.MODE_CBC, iv=iv)

            # Decrypt the ciphertext
            padded_plaintext = cipher.decrypt(ciphertext)

            # Unpad the plaintext
            plaintext = unpad(padded_plaintext, AES.block_size)
```

Figure 9: Sample Code for File Decryption

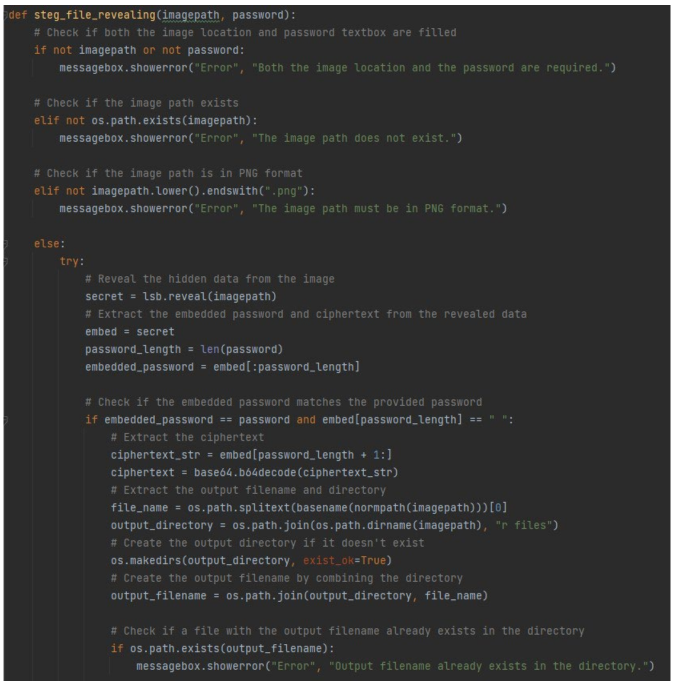
Figure 10 shows function named steg_file_hiding is used to implement a steganography- based file hiding process. It validates input data such as the source file path, picture filename, and password to ensure they fit predefined requirements. It computes sizes, formats, and conditions to assure data hiding compatibility. Following validation, it combines the ciphertext of the original file with a password, encodes it using Base64, and embeds it into a PNG picture using LSB steganography. The resulting stegano image is saved to a folder named "h files". Success and error messages are displayed, describing the outcome of the hiding procedure and any possible issues.



```python
def steg_file_hiding(filepath, imagepath, password):
    # Check if values are provided for all text boxes
    if not filepath or not imagepath or not password:
        messagebox.showerror("Error", "Please provide values for all the fields.")
        return

    # Check if the specified filepath or image path exists
    if not os.path.exists(filepath) or not os.path.exists(imagepath):
        messagebox.showerror("Error", "The specified filepath or image path does not exist.")
        return

    # Get the sizes of the image and file
    image_size = os.path.getsize(imagepath)
    file_size = os.path.getsize(filepath)

    # Check if the image path is in PNG format
    if not imagepath.lower().endswith(".png"):
        messagebox.showerror("Error", "The image path must be in PNG format.")
        return

    # Check if the image file is at least two times bigger than the file size
    if image_size < 8 * file_size:
        messagebox.showerror("Error", "The image file must be at least eight times bigger than the file size.")
        return

    # Check if the password is at least 8 characters long
    if len(password) < 8:
        messagebox.showerror("Error", "Password must be at least 8 characters long.")
        return

    # Check if the password contains at least one special character
    if not re.search(r'[@_!#$%^&*()<>?/\|}{~:]', password):
        messagebox.showerror("Error", "Password must contain at least one special character.")
        return

    # Extract the output directory from the given filepath
    output_directory = os.path.join(os.path.dirname(filepath), "h files")
    # Create the output directory if it doesn't exist
    os.makedirs(output_directory, exist_ok=True)
```

Figure 10: Sample Code for File Hiding

The function named steg_file_revealing shown in Figure 11 aids in the extraction of a hidden file from a steganography-encoded PNG image using the LSB steganography technique. The function initially examines the image path and password inputs to ensure they are provided and meet specified conditions. Following validation, it applies LSB steganography to reveal hidden data from the image. The embedded password and ciphertext are then extracted, confirming the accuracy of the specified password. If the extraction is successful, the ciphertext is decoded and saved as a new file in the "r files" folder and a success message is displayed. In the event of an invalid password, the absence of concealed data, or any other issues, relevant error messages are displayed.



```python
def steg_file_revealing(imagepath, password):
    # Check if both the image location and password textbox are filled
    if not imagepath or not password:
        messagebox.showerror("Error", "Both the image location and the password are required.")

    # Check if the image path exists
    elif not os.path.exists(imagepath):
        messagebox.showerror("Error", "The image path does not exist.")

    # Check if the image path is in PNG format
    elif not imagepath.lower().endswith(".png"):
        messagebox.showerror("Error", "The image path must be in PNG format.")

    else:
        try:
            # Reveal the hidden data from the image
            secret = lsb.reveal(imagepath)
            # Extract the embedded password and ciphertext from the revealed data
            embed = secret
            password_length = len(password)
            embedded_password = embed[:password_length]

            # Check if the embedded password matches the provided password
            if embedded_password == password and embed[password_length] == " ":
                # Extract the ciphertext
                ciphertext_str = embed[password_length + 1:]
                ciphertext = base64.b64decode(ciphertext_str)
                # Extract the output filename and directory
                file_name = os.path.splitext(basename(normpath(imagepath)))[0]
                output_directory = os.path.join(os.path.dirname(imagepath), "r files")
                # Create the output directory if it doesn't exist
                os.makedirs(output_directory, exist_ok=True)
                # Create the output filename by combining the directory
                output_filename = os.path.join(output_directory, file_name)

                # Check if a file with the output filename already exists in the directory
                if os.path.exists(output_filename):
                    messagebox.showerror("Error", "Output filename already exists in the directory.")
```

Figure 11: Sample Code for File Revealing

V. Conclusion

The development of the Stegcryption system is a significant step toward equipping users with the tools they need to protect their data in today's digital world, where the security of sensitive and personal data is of utmost significance. In its current form, the system includes essential features like file encryption and steganographic data hiding that are intended to provide users a way to protect their confidential information as well as file decryption and data revealing to bring back the data to its initial form. To fully understand the system's limitations and possibilities for enhancement, it is essential to examine it from a variety of perspectives.

The system certainly meets the critical need for data security, but it is not exempt from limitations that reduce its overall efficacy. Its file size restriction during the encryption process is one noticeable restriction. At this point, the system places a 400MB limitation on file sizes. In a time where huge files, such as high-definition videos and extensive datasets, are becoming more prevalent, this constraint is especially relevant. As a result, the system's inability to handle larger files may seriously limit its usefulness, especially in professional situations where thorough data protection is essential.

The system's approach to data recovery is another crucial factor that requires attention. In its current version, the system does not include a built-in mechanism for file recovery if a user forgets their encryption or hiding password. This oversight creates a significant danger because it suggests that any forgotten password can result in irreversible data loss. A system designed to prioritize user experience and data security should incorporate robust password recovery or reset features to mitigate such risks effectively.

Additionally, the system's limitation to processing one file at a time is a major barrier to efficiency, particularly for users handling numerous files at once. Since many files must be encrypted or hidden in real-world circumstances, the restriction on single-file processing can seriously impede workflow. It would be worthwhile to expand the system's capabilities to allow batch processing to increase its usefulness.

References

[1] Fortinet. (2023, March 14). What is Cryptography? Retrieved from Fortinet: https://www.fortinet.com/resources/cyberglossary/what-is-cryptography

[2] GeeksForGeeks. (2021, March 5). Difference between Encryption and Cryptography. Retrieved from GeeksForGeeks:https://www.geeksforgeeks.org/difference-between- encryption-and-cryptography/

[3] Fiscutean, A. (2021, September 15). Steganography explained and how to protect against it. Retrieved from CSO:https://www.csoonline.com/article/3632146/steganography- explained-and-how-to-protect-against-it.html

[4] Simplilearn. (2023, February 24). What is Steganography? Types, Techniques, Examples & Applications. Retrieved from Simplilearn: https://www.simplilearn.com/what-is- steganography-article

[5] SilentEye. (n.d.). SilentEye. Retrieved from SilentEye: https://achorein.github.io/silenteye/?i1s1

[6] Steghide. (n.d.). Steghide. Retrieved from Steghide: https://steghide.sourceforge.net/

[7] Fisher, T. (2022, August 5). What Is Windows 10? Retrieved from Lifewire: https://www.lifewire.com/windows-10-2626217