

Steganography Based Data Hiding

Pravin Kumar Mahato 22MCA0142

Chirag Bolakani 22MCA0164

Guided By : Dr. Navaneethan C

**Course Title : Network and Information
Security**

Course Code : ITA6007



Abstract

Information security plays a major role in any data transfer. One way to restrict direct access and ensure more security can be obtained by information hiding that focuses on hiding the existence of secret messages. Steganography is a data-hiding technique that allows users to conceal messages, images, and other types of data within other forms of data. It is a way of communicating securely by hiding data in plain sight. By concealing data within other types of data, it can be protected from outside detection. This makes it much more difficult to detect and exploit than traditional cryptography. In the following study, we combine steganography, a data hiding technique along with a data encryption algorithm in order to achieve more efficiency in securing access to confidential data.

Introduction

The project carries great importance as it deals with the aspect of security in ensuring the integrity and authenticity of the data. Encrypting data has been a part of security/communication systems as a traditional layer of security. In this study we add an extra layer to this traditional approach by combining cryptography and steganography techniques to leverage the benefits of the algorithms and make our message communication more secure.

Approaching the data security to preserve its integrity by cryptography techniques is a way to deal attacks. Although in recent years it has been possible to break through these encryption techniques as computation power has been increased significantly in the previous decade. Adding an extra layer of steganography makes the data transmission more secure as it hides the data from the user. Only if the sender and the receiver of the file that contains the hidden data both know the mechanism used for data hiding and the encryption algorithm, they are able to access the data.

Literature Survery

- [2] talks about the recent advances in the steganography domains. It gives an overview of various machine learning, deep learning and statistical approaches being introduced to the field of steganography.
- The currently existing techniques for data encryption convert the plain text to cipher text. In this study we focus on hiding the cipher text in the image.
- The traditional steganography techniques include hiding the data using Least Significant Bit Substitution method. The substitution method has to be performed cautiously as overloading the cover image may lead to visible changes leaking the presence of the secret information [4] and [5].
- A combination of the Huffman encoding and LSB substitution methods is used in [6] on videos.
- In addition to the LSB approaches, [7] has proposed a combination of Discrete Cosine Transformation (DCT) and Discrete Wavelet Transformation (DWT) for concealing a hidden message within a cover video.
- To conceal the secret information, the pixel density of the photos is calculated and a pixel density histogram is created. The Poisson distribution is used in [8] and [9] to calculate the burst error.
- The RSA Encryption became vulnerable to many cyber attacks as studied from [11]. It talks about the various vulnerabilities in the algorithm and various attacks that have been found to be affecting its efficiency.
- We take inspiration from [12] where the authors took a similar approach in performing the data encryption using Advanced Encryption Standard (AES) Algorithm and Steganography using Least Significant Bit (LSB) Approach for data hiding.
- [13] provides details regarding the random number generator of the Linux kernel. It talks about the entropy collection and how random bits are extracted from the three pools.
- Linux random number generator is subjected to change from time to time due to new updates to the source code. [14] revisits the study of [13] in order to study the new changes done to it.
- [15] combines the use of SHA256 along with AES and Steganography in order to check the integrity and ensure the confidentiality of the data. In our current study we tend to take a similar approach.

Proposed Work

The proposed work consists of an application through which the user is able to encrypt a message and then hide it in a image. The same desktop application can be used to extract the encrypted message hidden in the image and then decrypt it using the key.

Overview

Flowchart for Data Hiding and Encryption :

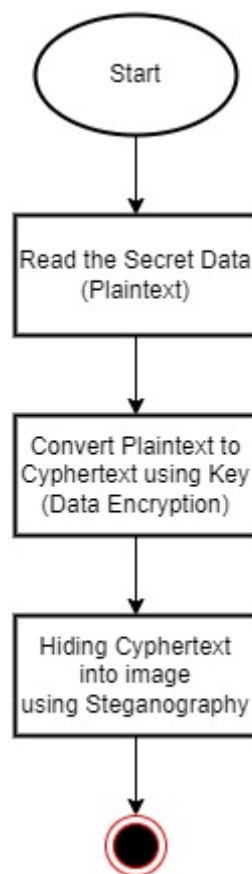


Figure 1

The preceding diagram (Figure 1) depicts the flow of the data hiding and encryption procedure. The user inputs the secret text that is used to encrypt the message before the system encrypts the message. Utilising an encryption algorithm, the plaintext is converted to ciphertext. The generated ciphertext is subsequently concealed within an image, from which the plaintext is extracted during the decryption and data extraction processes.

Data Decryption and Extraction :

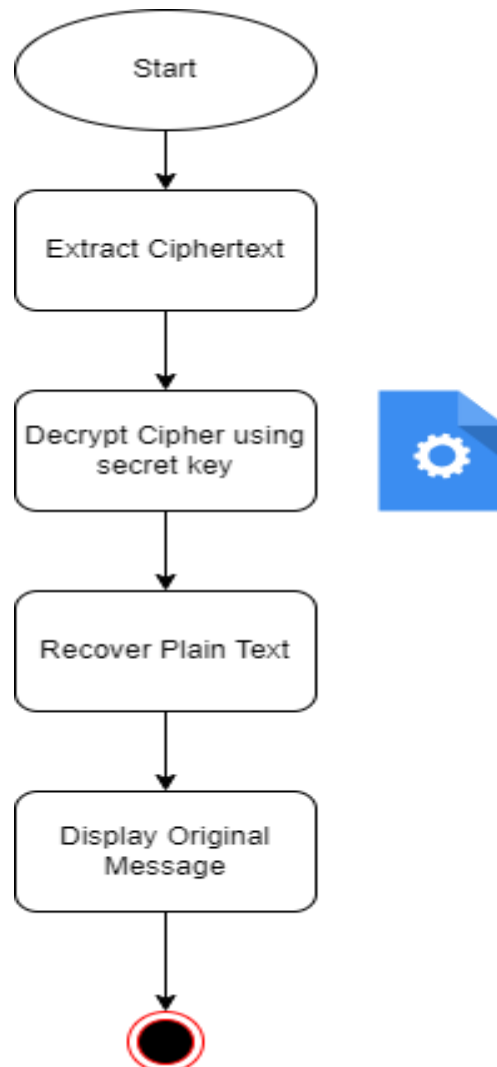


Figure 2

Figure 2 describes the data Decryption and Data Extraction process in the system. After the Encryption and Hiding of the message the data is hidden in an image. The cipher text is then extracted from this image. The Extracted cipher text is then converted to plain text. The user provides the key that is used initially in the encryption process for the algorithm.

At the decryption and data recovering step we focus on recovering the hidden message in the image. It is need to be ensured that there are no changes in the original hidden message during extraction of it. If the key does not match with the key that was used to encrypt the plain text then it indicates that there is a suspicious user might be trying to access the data hidden inside the image.

Encryption :

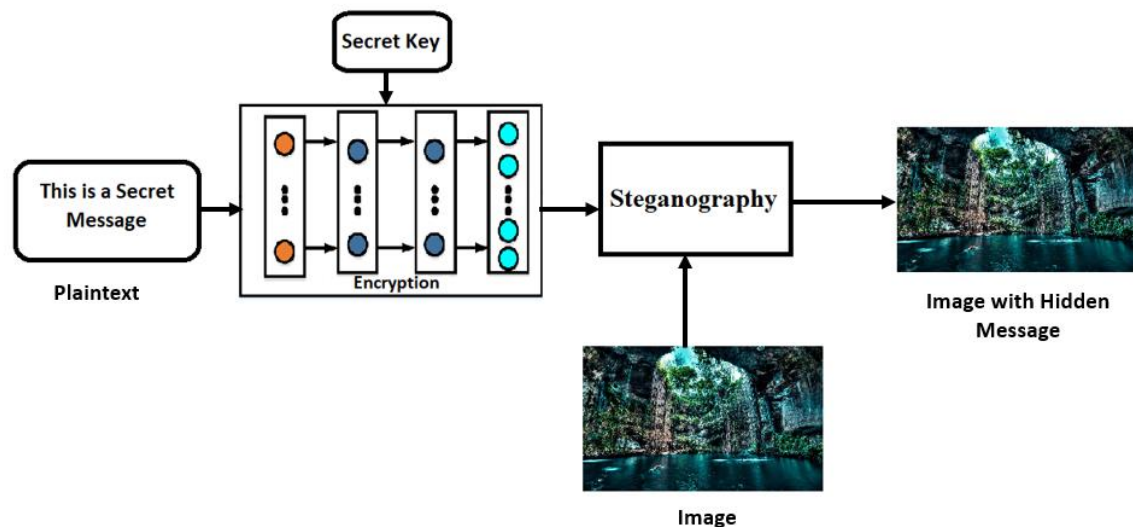


Figure 3

The User enters the plain text in the input field. The plain text is then encrypted using Advanced Encryption Standard Algorithm.

AES Algorithm

In 2001, the U.S. National Institute of Standards and Technology created the Advanced Encryption Standard, often known as Rijndael. A specification for the encryption of digital information.

AES Cipher :

AES is a symmetric block cipher technique. It uses the same key for encryption as well as for decryption. The algorithm handles multiple key and block sizes not limited to 64 or 56 compared to traditional algorithms like RSA. The block and key size of the algorithm can be anyone from 128, 192, or 256 bits and need not be the same.

AES Key Generation

All round key values from a key are calculated using the Key Schedule algorithm. Therefore, many different round keys will be created using the initial key, which will be used for the corresponding round of encryption.

Each key generation comprises of three sub-processes

- **RotWord** of Last Column

In this process, we take the last column of the key and perform Rotword.

- **SubByte of RotWord**

In this process, the first hexa decimal character become row and second become column and intersection point become new byte.

- **XOR with RCON** and first column of key and sub byte.

Now the Result of this will become the first column of Round key one.

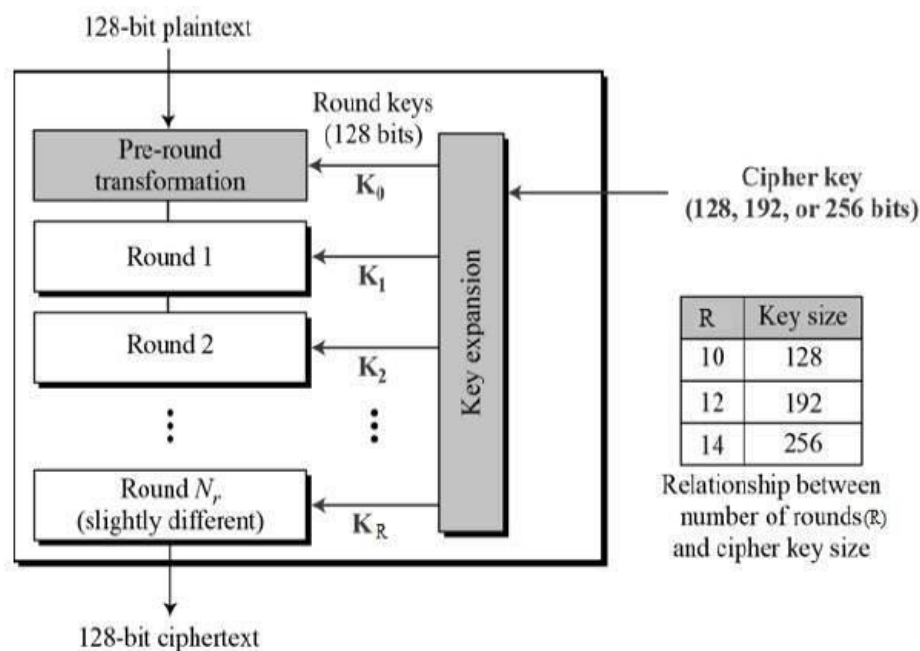


Figure 4

AES Encryption

AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Each of these rounds uses a unique 128-bit round key, which is derived from the original AES key.

Each round comprises of four sub-processes.

- **SubBytes**
- **ShiftRows**
- **MixColumns**
- **Add Round Key**

In **Byte Substitution (SubBytes)**, The 16 input bytes are substituted by looking into a fixed table (S-box). The result is in a matrix of four rows and four columns.

In **Shiftrows**, each of the four rows of the matrix is moved to the left as followd :

- First row-no change.

- Second row : one position towards left.
- Third row : two positions towards left.
- Fourth row : three positions towards left.
- The output is the same 16 bytes in a new matrix with a different order.

Now, In the **MixColumns** above matrix transforms each four-byte column using a special mathematical function. The four bytes of a column are passed into this function, which then outputs four completely new bytes that replace the original column. This results in the creation of a new 16-byte matrix. Notably, this procedure is not used in the final round.

In **Addroundkey**, the 16 bytes of the matrix have been regarded as 128 bits and XORed with the 128 bits of the round key. The output is the ciphertext if the current round is the last one. If not, the 128 bits that are produced are translated into 16 bytes, and the process is repeated.

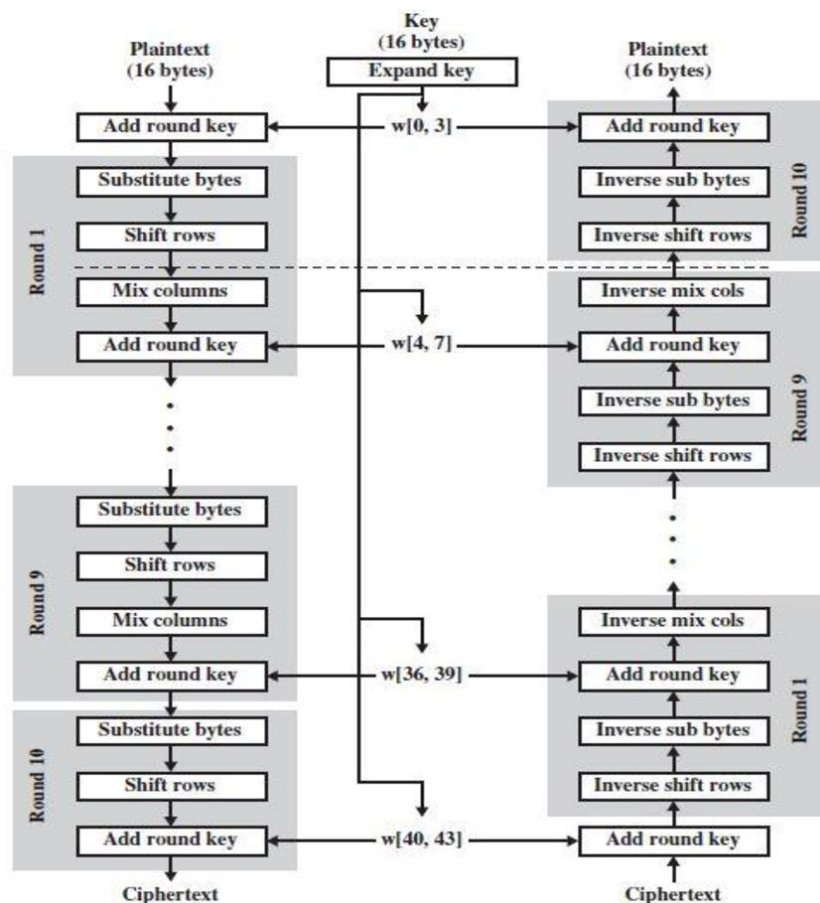


Figure 5

Decryption :

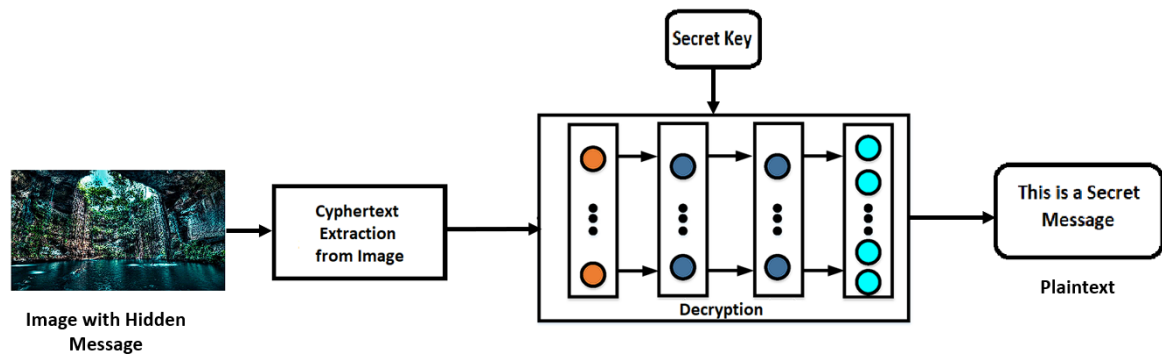


Figure 6

AES Decryption

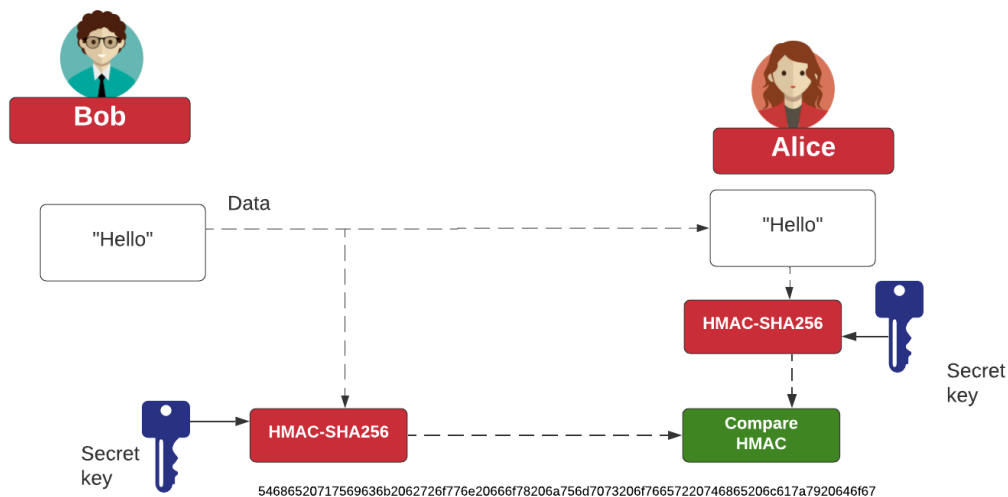
The decryption of an AES ciphertext is similar to the encryption process in reverse. Each round consists of four procedures carried out in reverse order:

- **Add round key**
- **Mix columns**
- **Shift rows**
- **Byte substitution**

Authentication

We make use of HMAC for checking the integrity of the data being received. It is a message authentication code using a cryptographic key, secret-key and a hash function. It is considered to be more secure than other authentication techniques. The hash function used in the current study is SHA256.

HMAC is used to determine the message received has been altered or not. The sender and the receiver exchange the secret key. While sending the data a hash value is generated and the file is sent along with this hash. On the receiver side the file is checked for any changes. The hash is calculated again from the received file. If the hash matches the HMAC then the integrity of the data is maintained.



Secret Key Generation

The secret key that is used for key expansion process is generated using `os.urandom()` function available in python's `os` module. It returns a random-sized byte string appropriate for use in cryptography. From a source of randomness native to the OS, this function returns random bytes. Although the exact quality of the returned data depends on the OS implementation, it should be sufficiently unexpected for cryptographic applications.

According to the Linux documentation[10], the random number generator compiles entropy from various sources, including device drivers and the surrounding environment. Moreover, the generator makes an educated guess as to the number of noise bits contributing to the total entropy. Random numbers are drawn from this entropy pool. 'When read, the `/dev/random` device will only return random bytes within the estimated number of bits of noise in the entropy pool. One-time pad and key creation are two examples of applications that require high-quality randomness, the kind provided by `/dev/random`.

Using a strong randomly generated secret key ensures better encryption and reduces the vulnerability to cracking the encryption using a brute force attack.

Code :

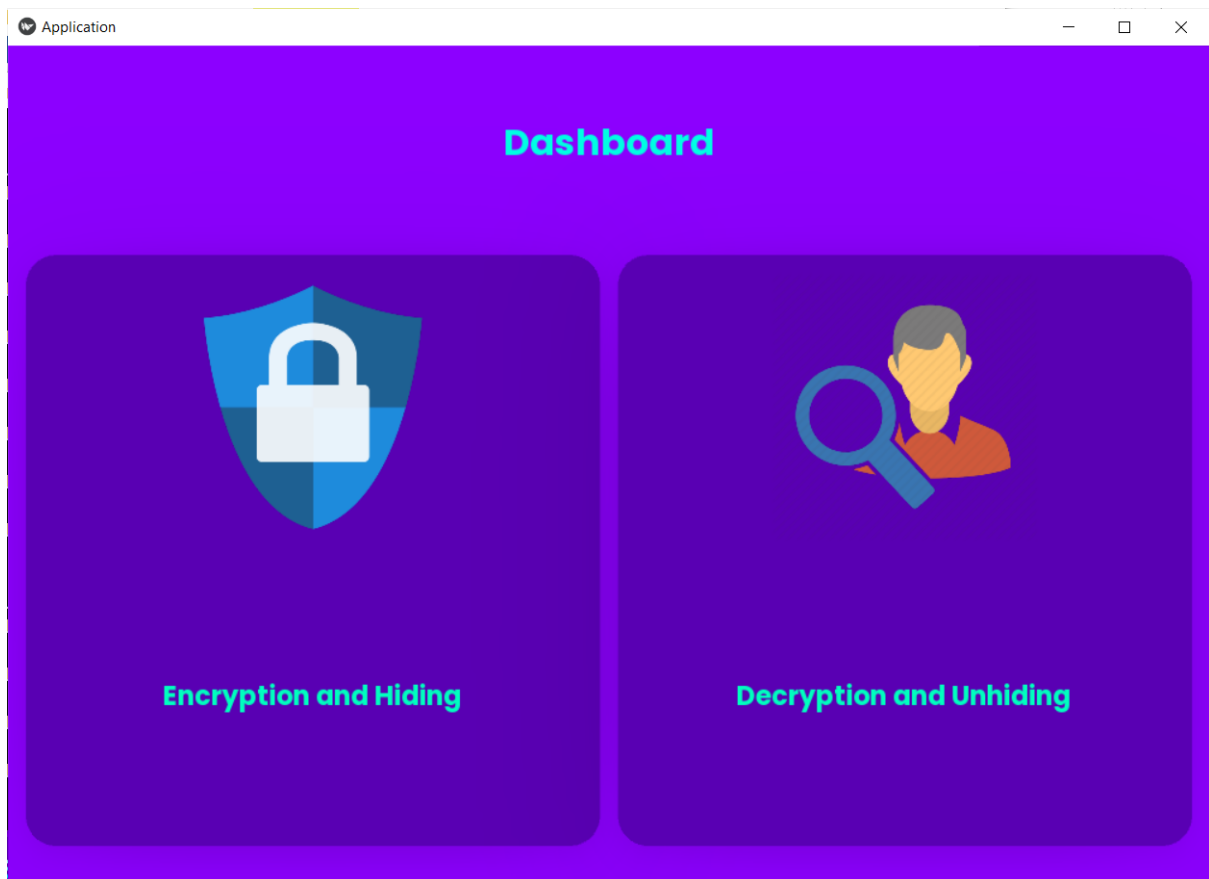
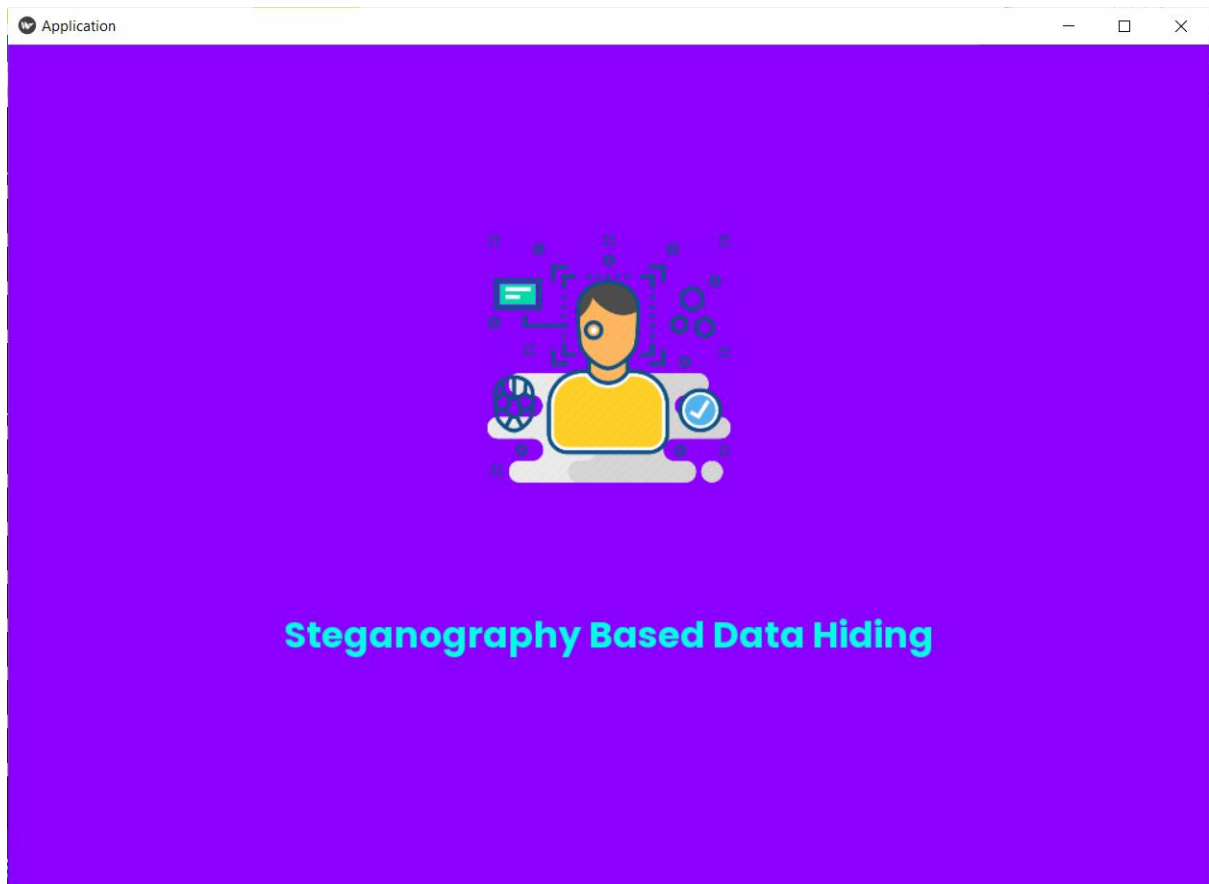
```
main.py > Application
1  from kivy import *
2  from kivymd.app import MDApp
3  from kivy.clock import Clock
4  from kivy.lang import Builder
5  from kivy.core.window import Window
6  from kivy.uix.screenmanager import ScreenManager
7  from plyer import filechooser
8  from cryptography.fernet import Fernet
9  from stegano import lsb
10 Clock.max_iteration = 2
11
12 Window.size = (1000, 700)
13
14
15 class Application(MDApp):
16
17     def build(self):
18         global screen_manager
19         screen_manager = ScreenManager()
20         screen_manager.add_widget(Builder.load_file("kivyFile/pre-splash.kv"))
21         screen_manager.add_widget(Builder.load_file("kivyFile/dashboard.kv"))
22         screen_manager.add_widget(Builder.load_file("kivyFile/hiding.kv"))
23         screen_manager.add_widget(Builder.load_file("kivyFile/unhiding.kv"))
24
25         return screen_manager
26
27     def on_start(self):
28         Clock.schedule_once(self.dashboard, 4)
29         # self.dashboard()
30
31     def dashboard(self, *args):
32         screen_manager.current = "dashboard"
33
34     def hiding(self, *args):
35         global h
36         h = screen_manager.get_screen("hiding")
37         screen_manager.current = "hiding"
38
```

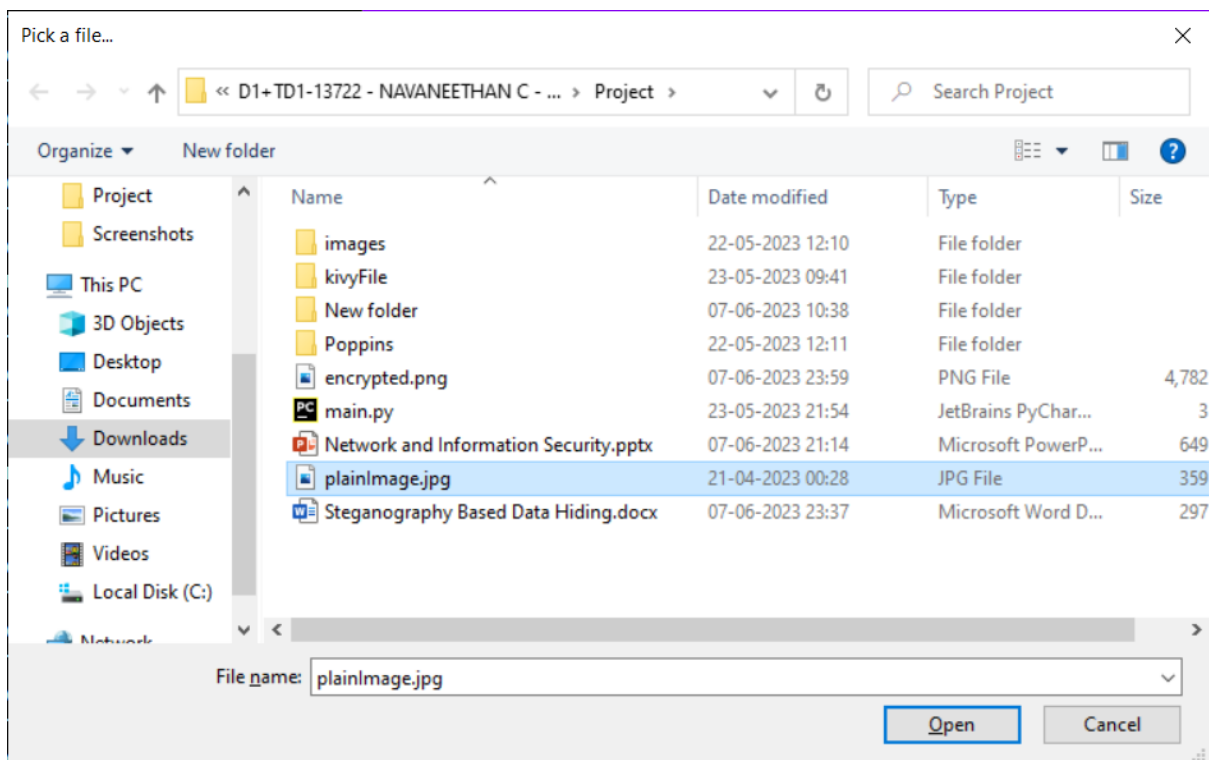
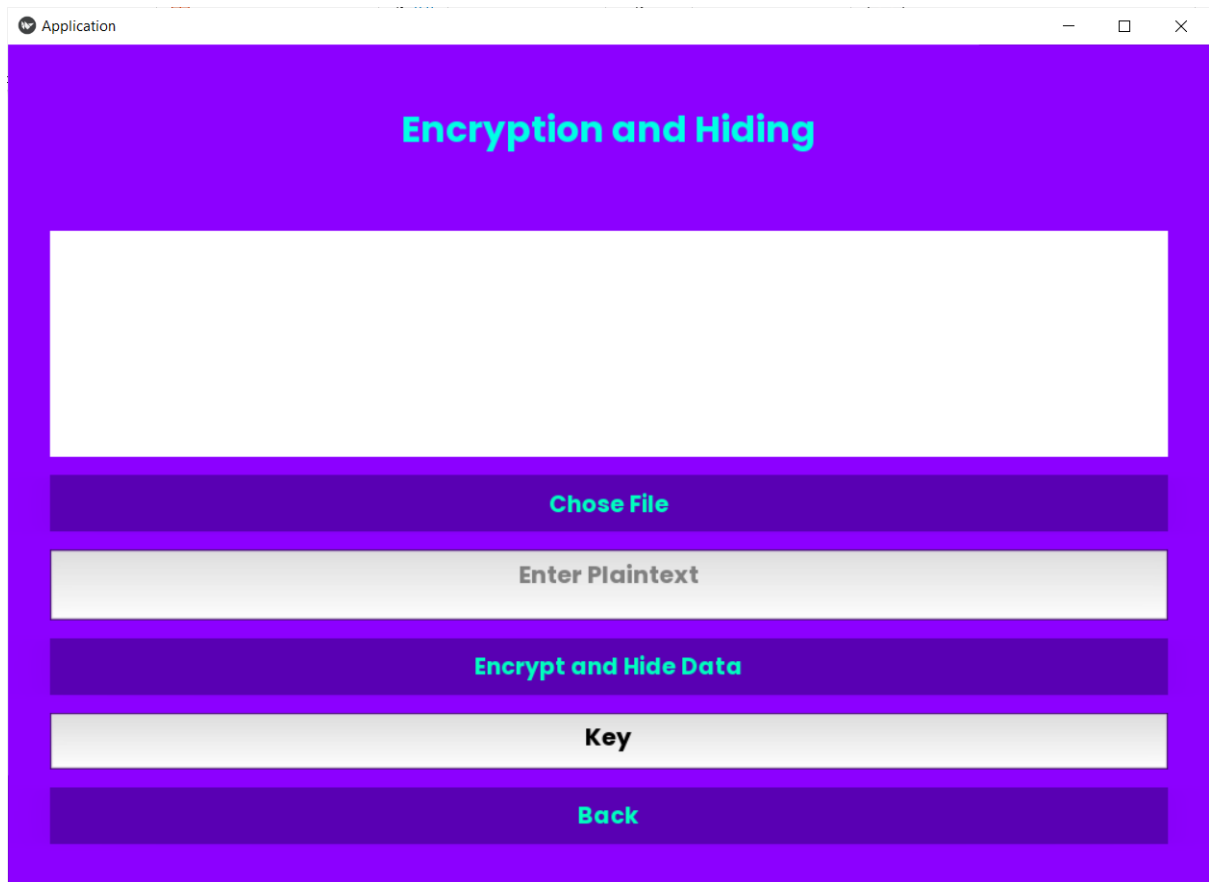
```
main.py > Application
34 def hiding(self, *args):
35     global h
36     h = screen_manager.get_screen("hiding")
37     screen_manager.current = "hiding"
38
39 def hselect_file(self):
40     filechooser.open_file(on_selection = self.hselected)
41
42 def hselected(self, filename):
43     global plainimage
44     # print ("selected: %s" % filename[0])
45     plainimage = filename[0]
46     h.ids.imgid.source = filename[0]
47
48 def encryption(self, ptext):
49     key = Fernet.generate_key()
50     # print(key)
51     cypher = Fernet(key)
52     plaintext = ptext
53     cyphertext = cypher.encrypt(bytes(plaintext,'utf8'))
54     # print(str(cyphertext,'utf8'))
55     secret = lsb.hide(plainimage, str(cyphertext,'utf8'))
56     secret.save("./encrypted.png")
57     h.ids.privatekey.text = str(key,'utf8')
58
59
60 def unhiding(self, *args):
61     global u
62     u = screen_manager.get_screen("unhiding")
63     screen_manager.current = "unhiding"
64
65 def uselect_file(self):
66     filechooser.open_file(on_selection = self.uselected)
67
68 def uselected(self, filename):
69     global cypherimage
70     # print ("selected: %s" % filename[0])
71     cypherimage = filename[0]
72     u.ids.imgId.source = filename[0]
```

main.py > Application > un hiding

```
59
60  def unhiding(self, *args):
61      global u
62      u = screen_manager.get_screen("unhiding")
63      screen_manager.current = "unhiding"
64
65  def uselect_file(self):
66      filechooser.open_file(on_selection = self.uselected)
67
68  def uselected(self, filename):
69      global cypherimage
70      # print ("selected: %s" % filename[0])
71      cypherimage = filename[0]
72      u.ids.imgId.source = filename[0]
73
74  def decryption(self, ktext):
75      key = ktext
76      # print(key)
77      try:
78          cypher = Fernet(bytes(key, 'utf8'))
79          cyphertext = lsb.reveal(cypherimage)
80          # print(str(cyphertext, 'utf8'))
81          plaintext = cypher.decrypt(bytes(cyphertext, 'utf8'))
82          # print(str(plaintext, 'utf8'))
83          u.ids.plainText.text = str(plaintext, 'utf8')
84      except Exception as e:
85          print(str(e))
86          u.ids.plainText.text = "{HMAC SHA256 Authentication Failed}"
87
88  if __name__ == "__main__":
89      Application().run()
90
```


```
new.py > Application > comperision
27
28     def oselect_file(self):
29         |     filechooser.open_file(on_selection = self.oselected)
30
31     def oselected(self, filename):
32         |     global originalimage
33         |     originalimage = filename[0]
34         |     print(originalimage)
35
36     def eselect_file(self):
37         |     filechooser.open_file(on_selection = self.eselected)
38
39     def eselected(self, filename):
40         |     global cypherimage
41         |     cypherimage = filename[0]
42         |     print(cypherimage)
43
44     def comperision(self):
45         |     img1 = Image.open(originalimage)
46         |     img2 = Image.open(cypherimage)
47
48         |     img1_list = list(img1.getdata())
49         |     img2_list = list(img2.getdata())
50
51         |     lst = []
52
53         |     for i in range(0,len(img1_list)):
54         |         |     if(img1_list[i]!=img2_list[i]):
55         |         |         |     lst.append((57,255,20))
56         |         |     else:
57         |         |         |     lst.append(img1_list[i])
58
59         |     image_out = Image.new(mode = "RGB", size = (img1.size))
60         |     image_out.putdata(lst)
61         |     image_out.save("out.png")
62         |     k.ids.gimgid.source = "./out.png"
63         |     os.startfile(os.getcwd()+"\out.png")
64
```





Application

Encryption and Hiding



Chose File

Enter Plaintext


Encrypt and Hide Data

Key

Back

Application

Encryption and Hiding



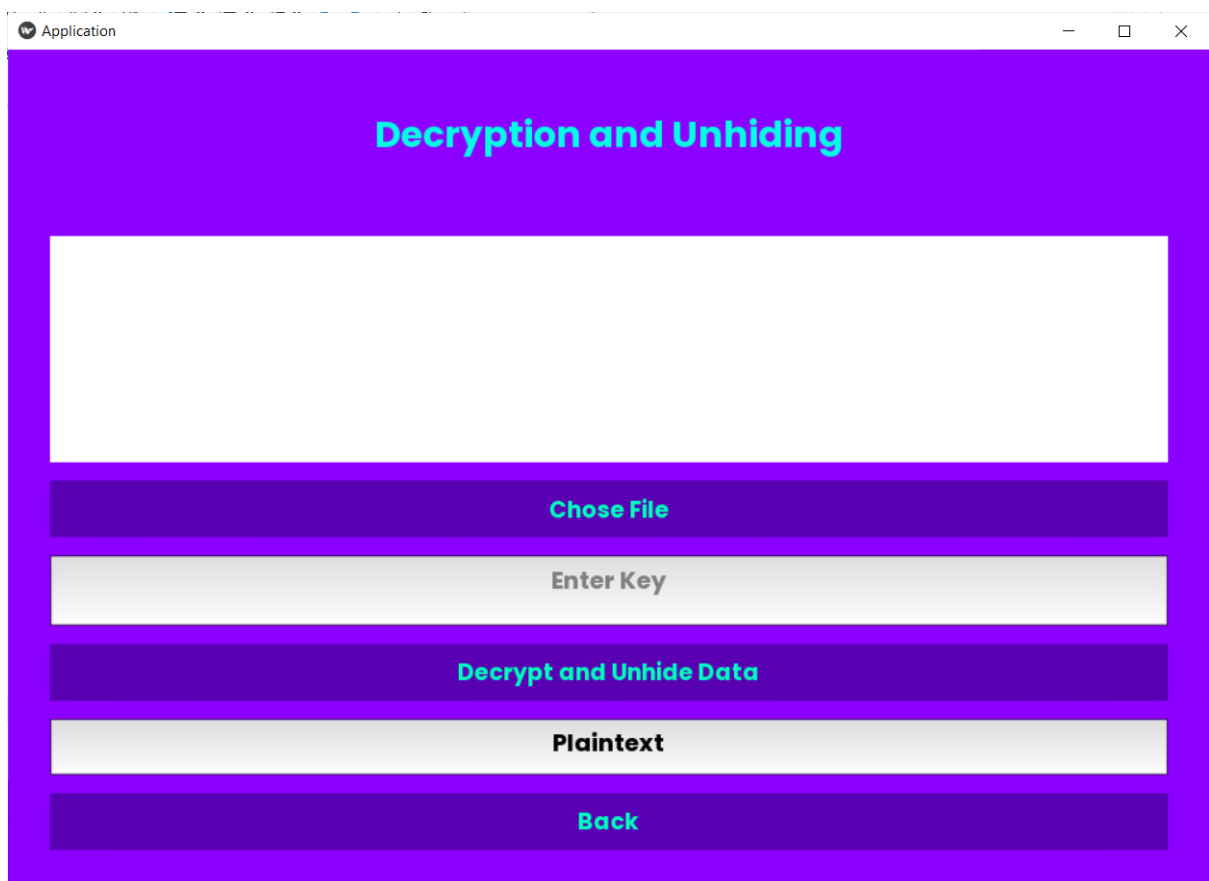
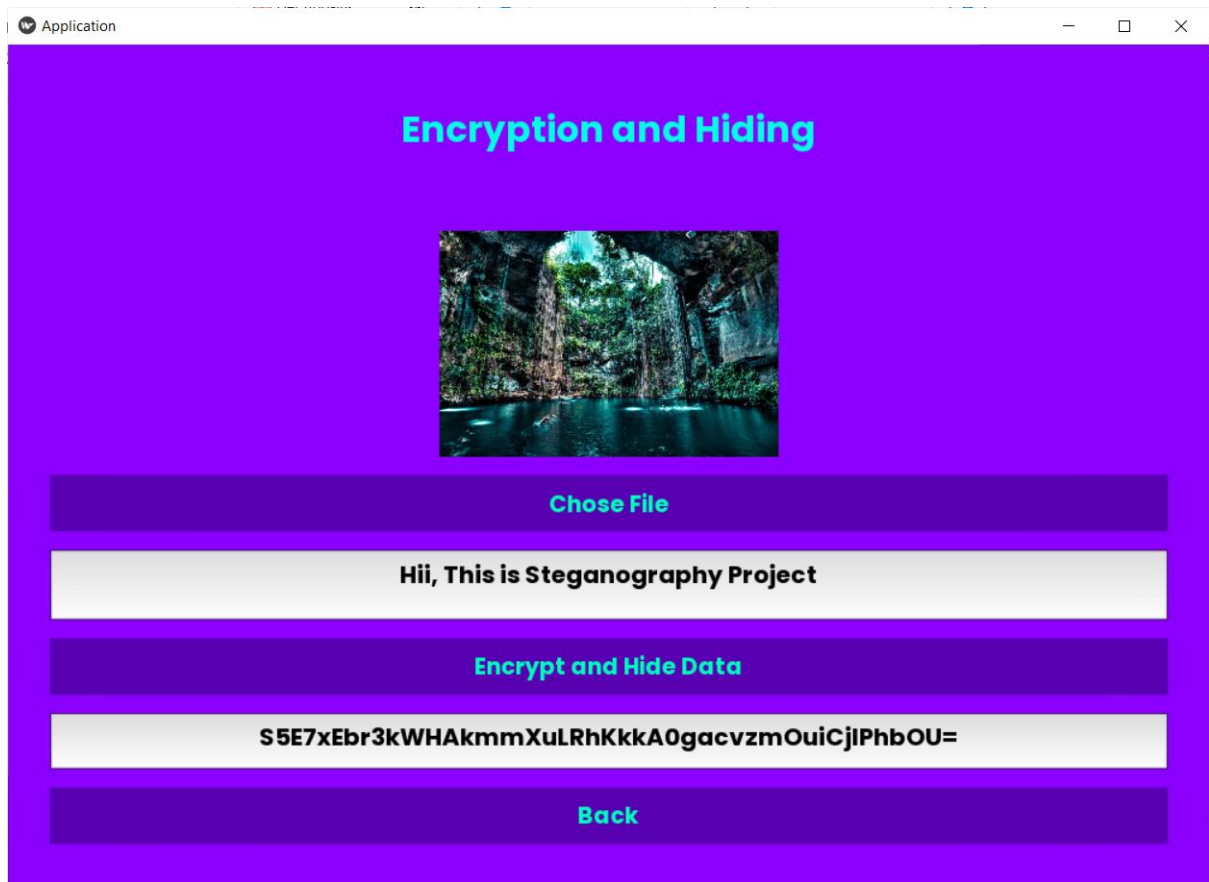
Chose File

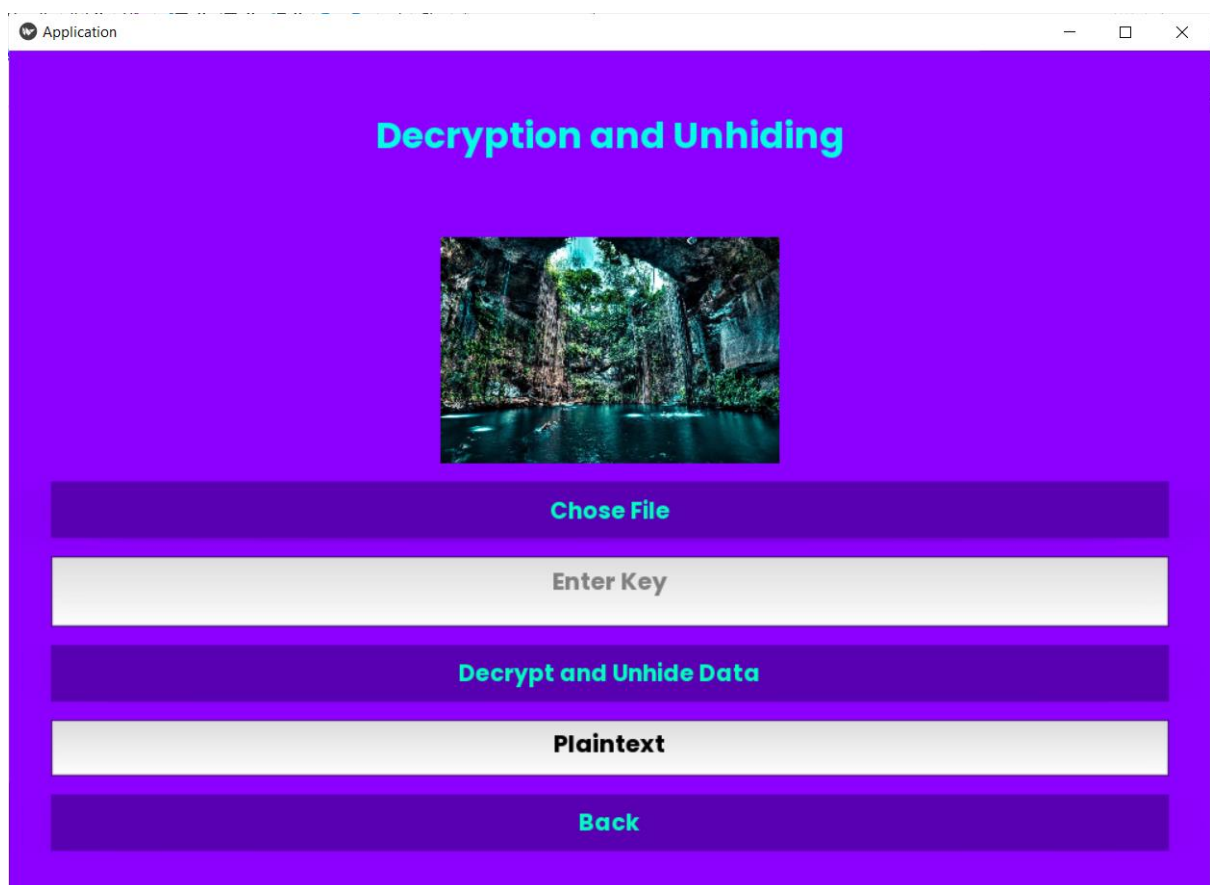
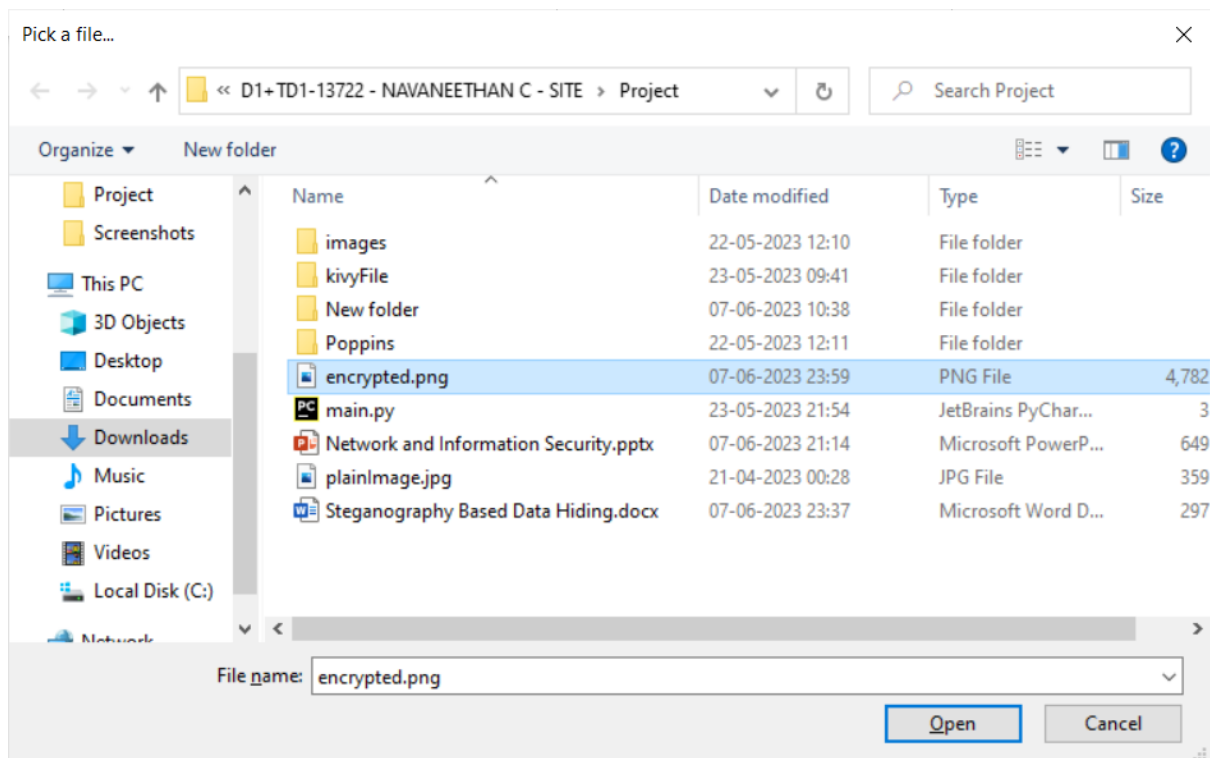
Hii, This is Steganography Project

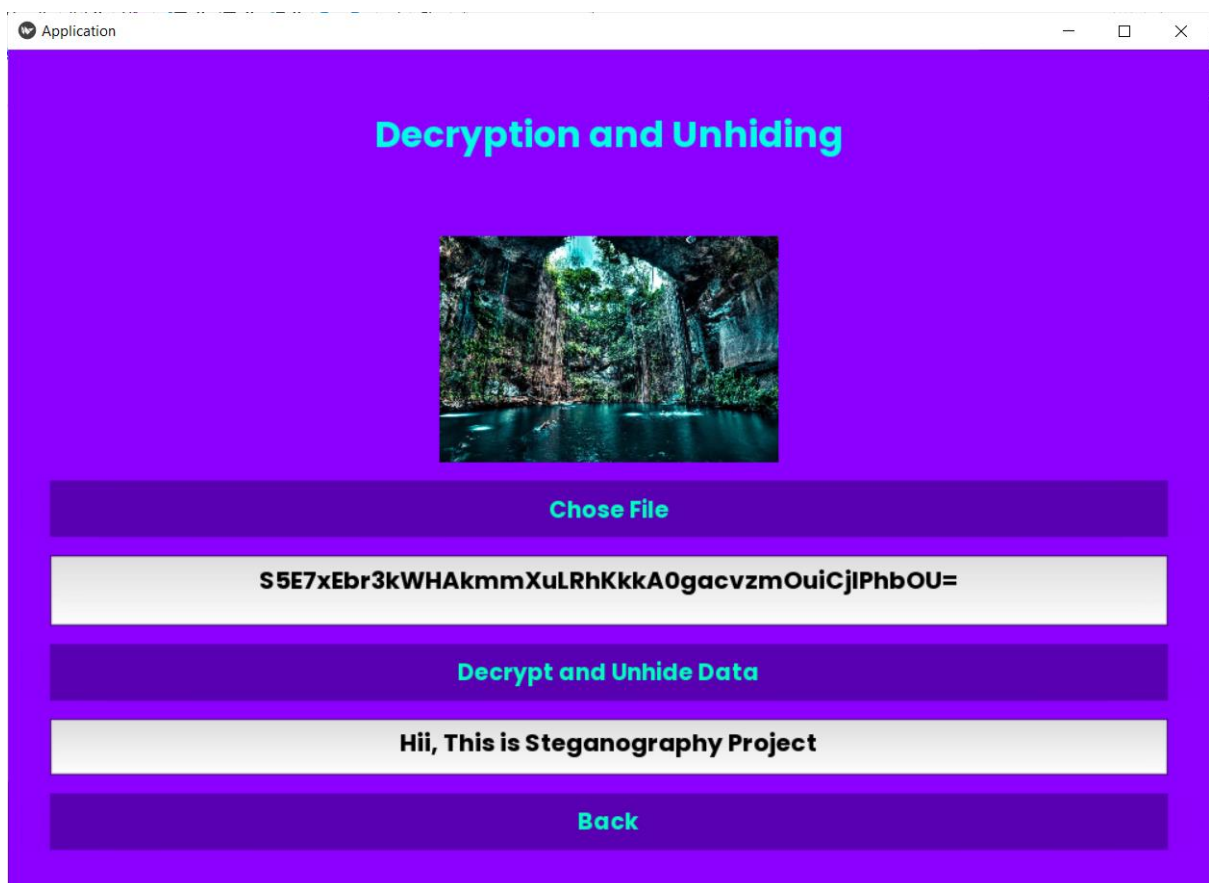
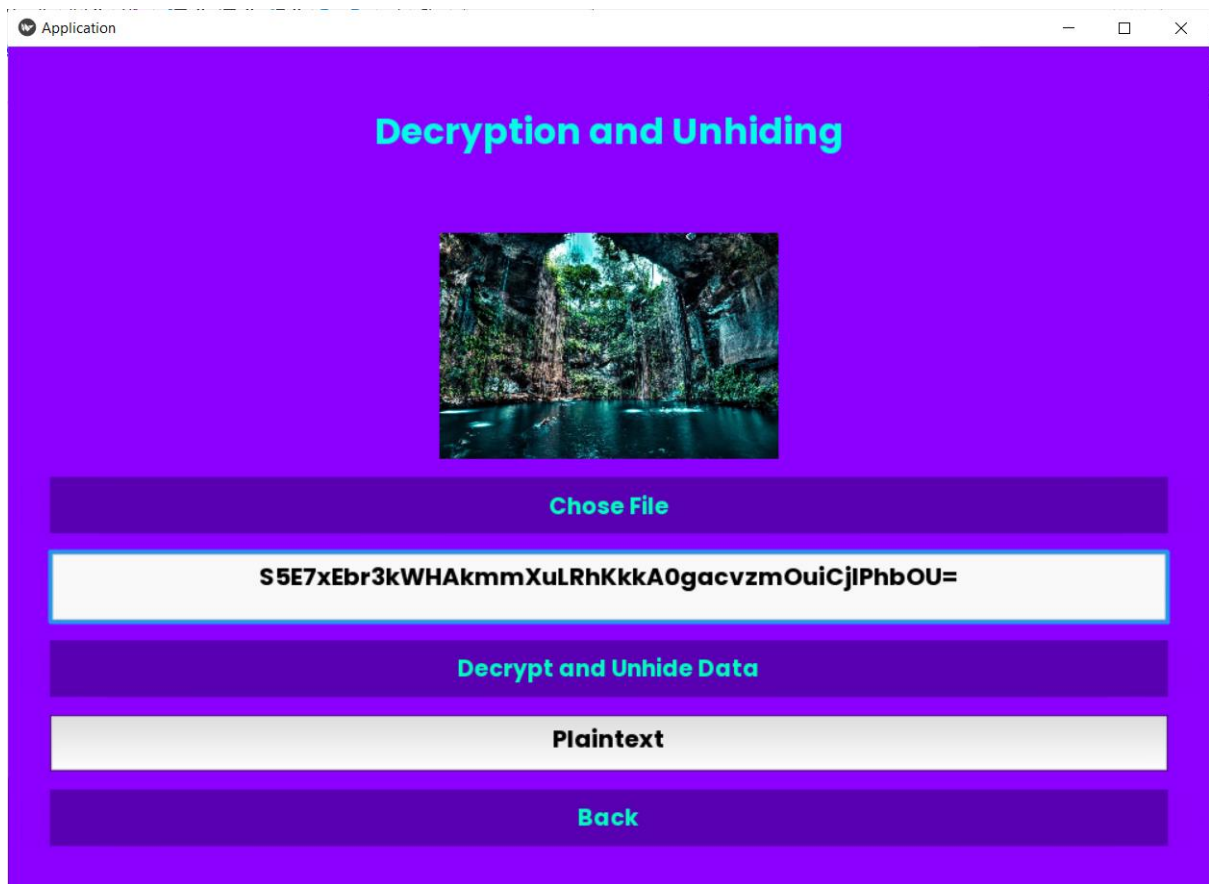
Encrypt and Hide Data

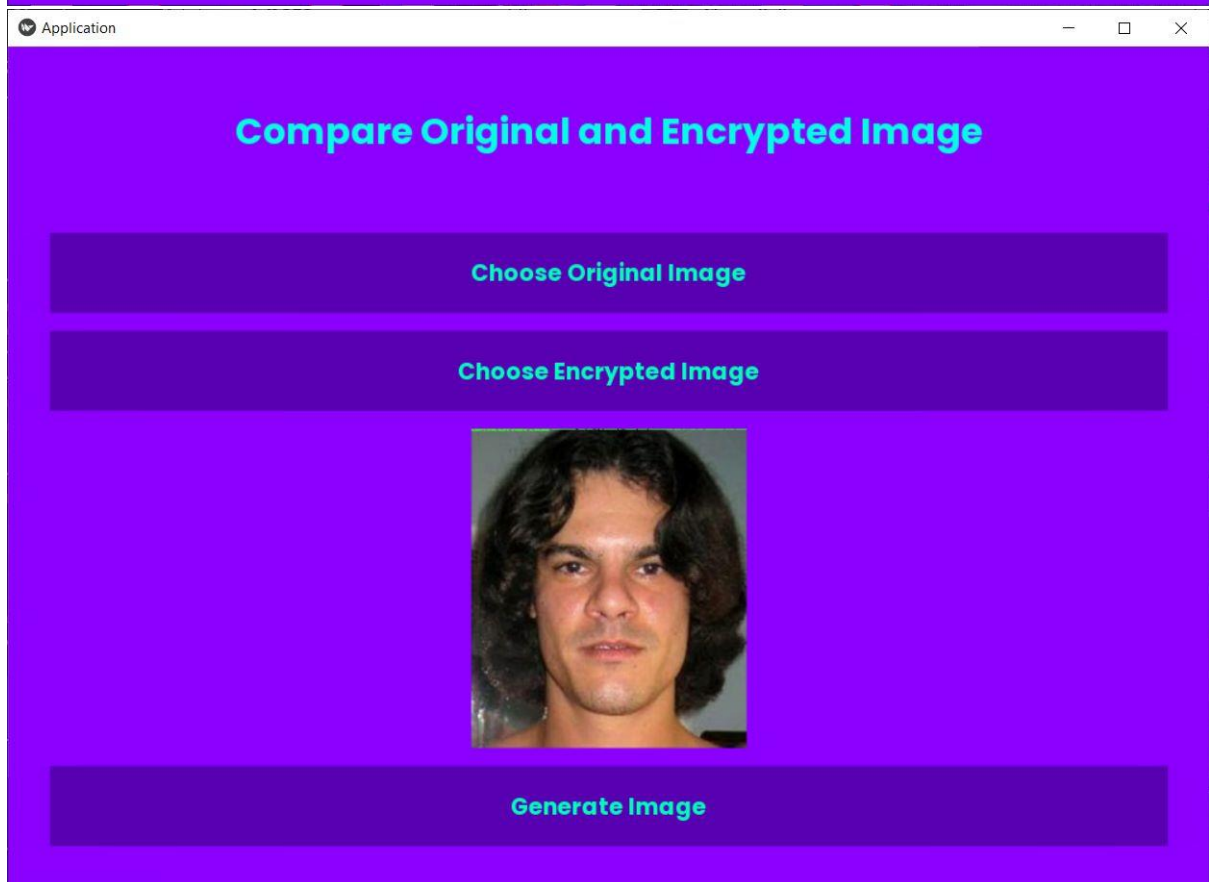
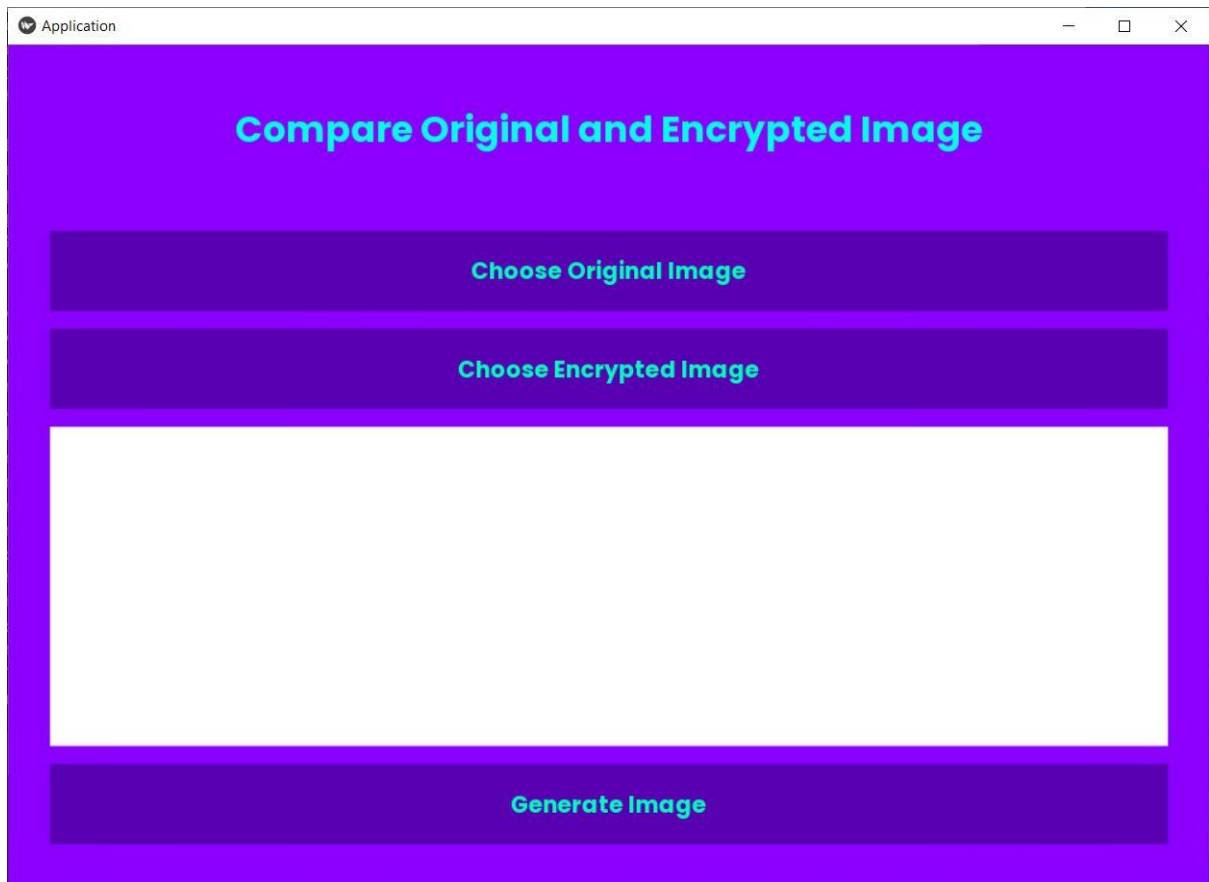
Key

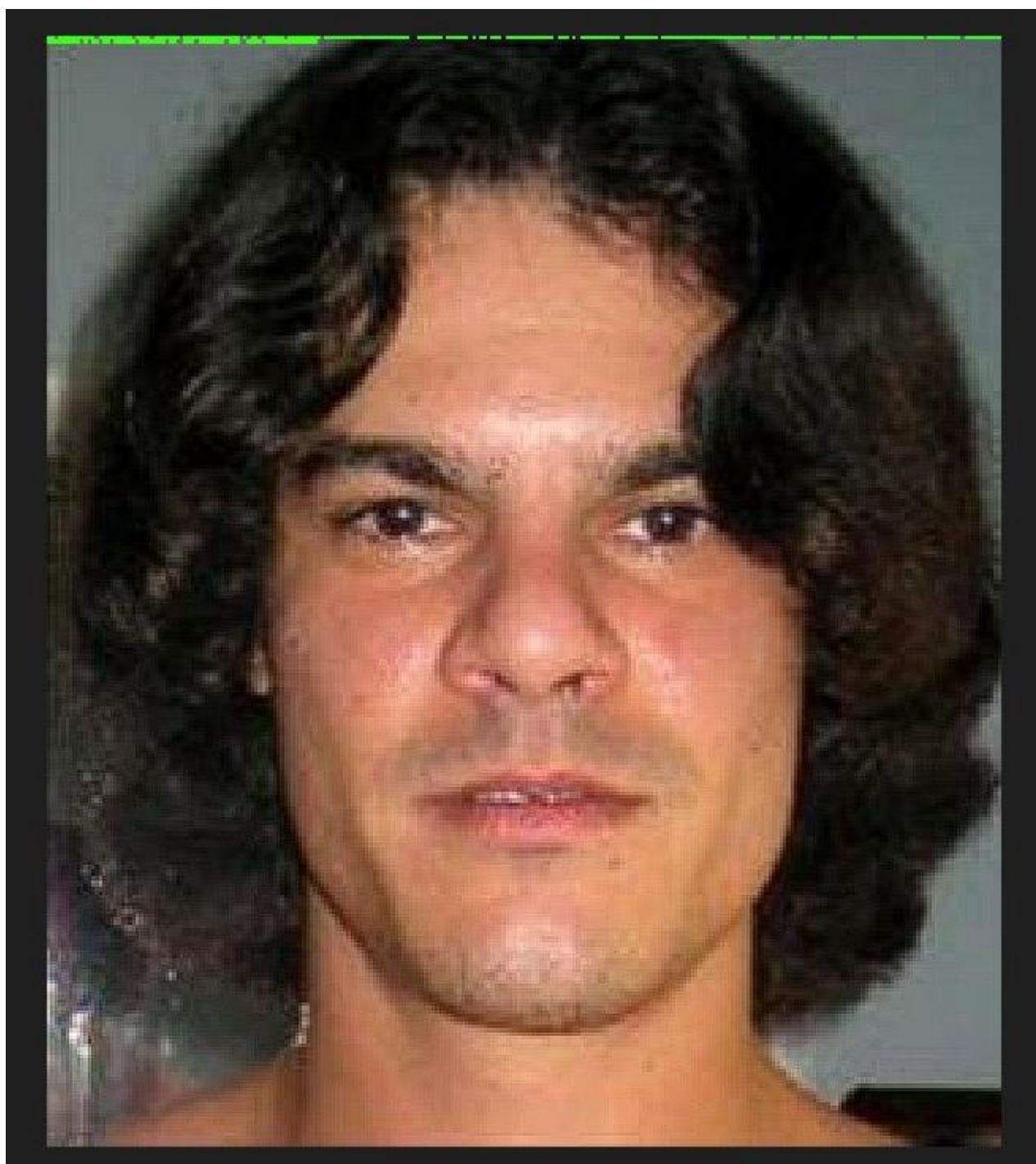
Back











Conclusion

In the presented work we successfully performed the art of data hiding in the files. The message to be hidden in the file was successfully stored in the bits of an image. In this study we combine AES algorithm along with HMAC SHA256 to ensure the data integrity of the message to be transmitted. We look forward to improvising the system more by implementing the future scope of the project.

Future Work

We look forward to include XXHash for even better secret key generation.

- xxHash is an extremely fast hashing algorithm that can process at the maximum speed of RAM. The code is highly portable, producing identical hashes on all platforms (little/big endian). The algorithmic components in the library are listed below:
- XXH32: uses 32-bit arithmetic to generate 32-bit hashes.
- XXH64: uses 64-bit arithmetic to generate 64-bit hashes.
- XXH3 (since v0.8.0): Using vectorized arithmetic, it generates 64 or 128-bit hashes. The 128-bit variation is known as XXH128.

A Benchmark of xxHash [11]:

Hash Name	Width	Bandwidth (GB/s)	Small Data Velocity	Quality	Comment
XXH3 (SSE2)	64	31.5 GB/s	133.1	10	
XXH128 (SSE2)	128	29.6 GB/s	118.1	10	
memcpy	N/A	28.0 GB/s	N/A	N/A	for reference
City64	64	22.0 GB/s	76.6	10	
T1ha2	64	22.0 GB/s	99.0	9	Slightly worse collisions
City128	128	21.7 GB/s	57.7	10	
XXH64	64	19.4 GB/s	71.0	10	
SpookyHash	64	19.3 GB/s	53.2	10	
Mum	64	18.0 GB/s	67.0	9	Slightly worse collisions
XXH32	32	9.7 GB/s	71.9	10	
City32	32	9.1 GB/s	66.0	10	
Murmur3	32	3.9 GB/s	56.1	10	
SipHash	64	3.0 GB/s	43.2	10	
FNV64	64	1.2 GB/s	62.7	5	Poor avalanche properties
Blake2	256	1.1 GB/s	5.1	10	Cryptographic
SHA1	160	0.8 GB/s	5.6	10	Cryptographic but broken
MD5	128	0.6 GB/s	7.8	10	Cryptographic but broken

References

1. L. Negi and L. Negi, "Image Steganography Using Steg with AES and LSB," 2021 IEEE 7th International Conference on Computing, Engineering and Design (ICCED), Sukabumi, Indonesia, 2021, pp. 1-6, doi: 10.1109/ICCED53389.2021.9664834.
2. Subramanian, Nandhini & Elharrouss, Omar & Al-ma'adeed, Somaya & Bouridane, Ahmed. (2021). Image Steganography: A Review of the Recent Advances. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3053998.
3. Dagadita, Monica & Slusanschi, Emil & Dobre, Razvan. (2013). Data Hiding Using Steganography. 159-166. 10.1109/ISPDC.2013.29.
4. N. F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. Computer, 31(2):26–34, 1998.
5. Shilpa Gupta, Geeta Gujral, and Neha Aggarwal. Enhanced least significant bit algorithm for image steganography. IJCEM International Journal of Computational Engineering & Management, 15(4):40–42, 2012.
6. Shwe Sin Myat Than. Secure data transmission in video format based on lsb and huffman coding. International Journal of Image, Graphics and Signal Processing, 12(1):10, 2020.
7. R. J. Mstafa, K. M. Elleithy, and E. Abdelfattah. A robust and secure video steganography method in dwt-dct domains based on multiple object tracking and ecc. IEEE Access, 5:5354–5365, 2017.
8. Wei Lu, Yingjie Xue, Yuileong Yeung, Hongmei Liu, Jiwu Huang, and Yun Shi. Secure halftone image steganography based on pixel density transition. IEEE Transactions on Dependable and Secure Computing, 2019.
9. Yi Zhang, Chuan Qin, Weiming Zhang, Fenlin Liu, and Xiangyang Luo. On the fault-tolerant performance for a class of robust image steganography. Signal Processing, 146:99–111, 2018.
10. Z. Gutterman, B. Pinkas and T. Reinman, "Analysis of the Linux random number generator," 2006 IEEE Symposium on Security and Privacy (S&P'06), Berkeley/Oakland, CA, USA, 2006, pp. 15 pp.-385, doi: 10.1109/SP.2006.5.
11. Boneh, Dan. (2002). Twenty Years of Attacks on the RSA Cryptosystem. NOTICES OF THE AMS. 46.

- 12.L. Negi and L. Negi, "Image Steganography Using Steg with AES and LSB," 2021 IEEE 7th International Conference on Computing, Engineering and Design (ICCED), Sukabumi, Indonesia, 2021, pp. 1-6, doi: 10.1109/ICCED53389.2021.9664834.
- 13.Z. Gutterman, B. Pinkas and T. Reinman, "Analysis of the Linux random number generator," 2006 IEEE Symposium on Security and Privacy (S&P'06), Berkeley/Oakland, CA, USA, 2006, pp. 15 pp.-385, doi: 10.1109/SP.2006.5.
- 14.P. Lacharme, A. Rock, V. Strubel, and M. Videau, "The Linux Pseudorandom Number Generator Revisited." HAL CCSD, 2012.
- 15.C. Chaitanya, C. N. Reddy, K. S. Vignesh, P. R. Krishna, A. Roshini and K. Swetha, "Enhanced Hash Based Image Steganography Technique to Increase Data Integrity and Confidentiality," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2021, pp. 1456-1461, doi: 10.1109/ICACCS51430.2021.9442036.