

Configuration Manual

MSc Research Project
Cyber security

Bansie Vasudevan Iyengar
Student ID: X21121591

School of Computing
National College of Ireland

Supervisor: Mr. Jawad Salahuddin

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Bansie Vasudevan Iyengar
Student ID: X21121591
Programme: Cyber Security **Year:** 2022
Module: MSc Research project
Lecturer: Mr. Jawad salahuddin
Submission Due Date: 15/12/2022
Project Title: A study on Image, Audio, and Video Steganography
Word Count:827 **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: Bansie Vasudevan Iyengar
Date: 11/12/22

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Bansie Vasudevan Iyengar

Student ID: X21121591

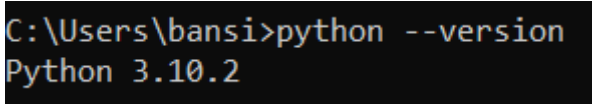
Introduction and Overview:

The proposed system is designed for the users to perform Image, Audio, Video steganography with the help of a web application. The web application is implemented with a clean and simple UI that is very easy to understand and adapt. This configuration manual consists of a comprehensive guide on how the research project is implemented, executed, which involves the system requirements and coding requirements. The frontend includes HTML, CSS, JS and bootstrap. For the backend we are completely using python as they support a lot of web app libraries, frameworks built in and also some cryptographic libraries that can be used for simple cryptographic operations.

Initial Setup:

- Operating system: Microsoft Windows 10 21H2
- Ram: 16GB
- Disk space required: 1GB
- Software used: Visual studio code
- The prototype is tested on local host

Python:



```
C:\Users\bansi>python --version
Python 3.10.2
```

Fig. 1 Python version

The python version that we have installed in 3.10.2 from python's official website

Python libraries and justification:

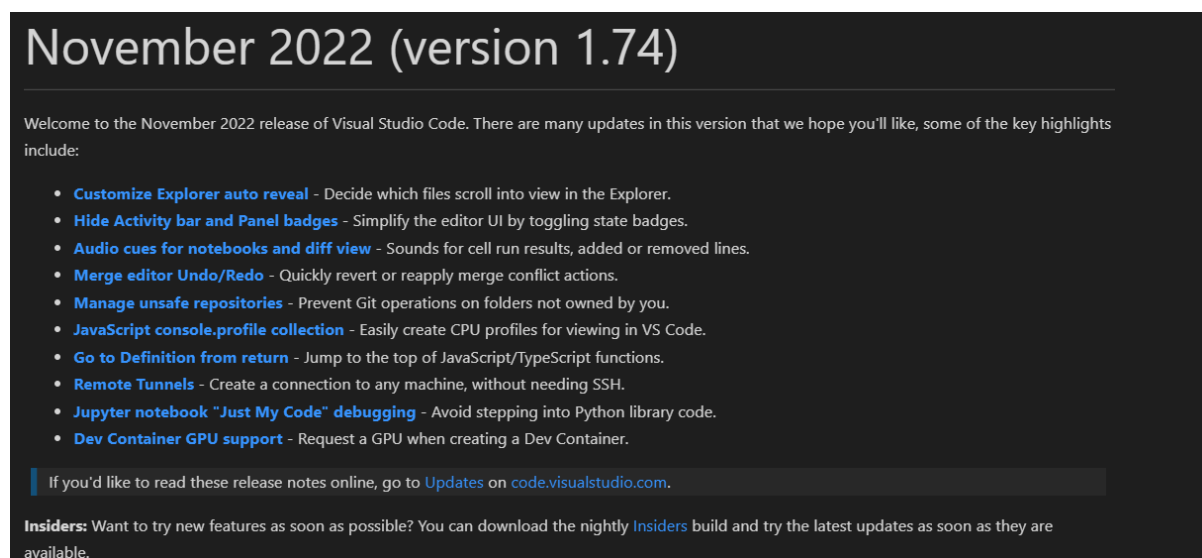
- APScheduler version 3.9.1
 - This library is used for checking the user's account if it is not expired on a periodic basis.
- Cryptography version 37.0.2
- Flask version 2.0.3
 - We are using this framework to make a container to connect frontend (HTML, CSS, JS) to backend (Python)

- Flask_Mail version 0.9.1
 - This library is used to send the verification code to the user's registered email id.
- Flask_ReCaptcha version 0.4.2
 - This library is used for the captcha implementation along with flask WTF forms.
- Flask_WTF version 1.0.1
 - This library is used along with Flask_ReCaptcha library to implement the entirety of captcha process.
- Numpy version 1.22.3
 - This library is used to manipulate images and frames from a video during encoding and decoding process.
- opencv_python version 4.5.5.64
 - This library is used to open image files for encoding and decoding secret data.
- virustotal_api version 1.1.11
 - This library is used to scan the files using virus total that is uploaded by the user.

All the above mentioned libraries are listed in requirements.txt, which can be installed by using the following command.

```
pip install -r requirements.txt
```

VS Code:



November 2022 (version 1.74)

Welcome to the November 2022 release of Visual Studio Code. There are many updates in this version that we hope you'll like, some of the key highlights include:

- **Customize Explorer auto reveal** - Decide which files scroll into view in the Explorer.
- **Hide Activity bar and Panel badges** - Simplify the editor UI by toggling state badges.
- **Audio cues for notebooks and diff view** - Sounds for cell run results, added or removed lines.
- **Merge editor Undo/Redo** - Quickly revert or reapply merge conflict actions.
- **Manage unsafe repositories** - Prevent Git operations on folders not owned by you.
- **JavaScript console.profile collection** - Easily create CPU profiles for viewing in VS Code.
- **Go to Definition from return** - Jump to the top of JavaScript/TypeScript functions.
- **Remote Tunnels** - Create a connection to any machine, without needing SSH.
- **Jupyter notebook "Just My Code" debugging** - Avoid stepping into Python library code.
- **Dev Container GPU support** - Request a GPU when creating a Dev Container.

If you'd like to read these release notes online, go to [Updates on code.visualstudio.com](#).

Insiders: Want to try new features as soon as possible? You can download the nightly [Insiders](#) build and try the latest updates as soon as they are available.

Fig. 2 VS Code version

We are making use of the latest version of VS code that is available as on date.

Database:

We are using SQLite3 for our database needs. SQLite3 is built in to the flask environment that we are using. SQLite3 can be installed by using the following command.

pip install db-sqlite3

HeidiSQL:

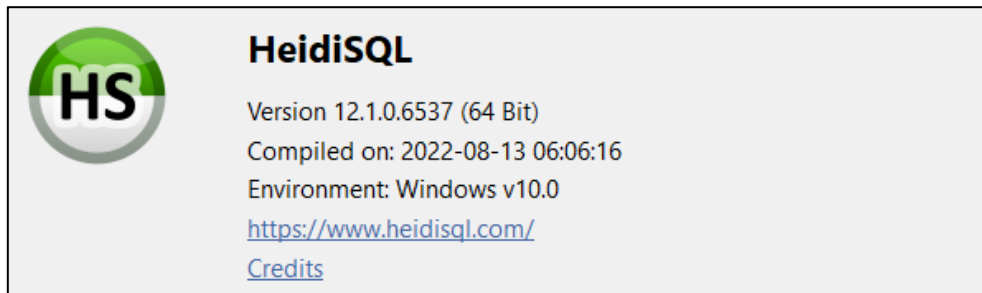


Fig. 3 HeidiSQL version

We are making use of heidiSQL to visualize SQL tables and data that we are manipulating.

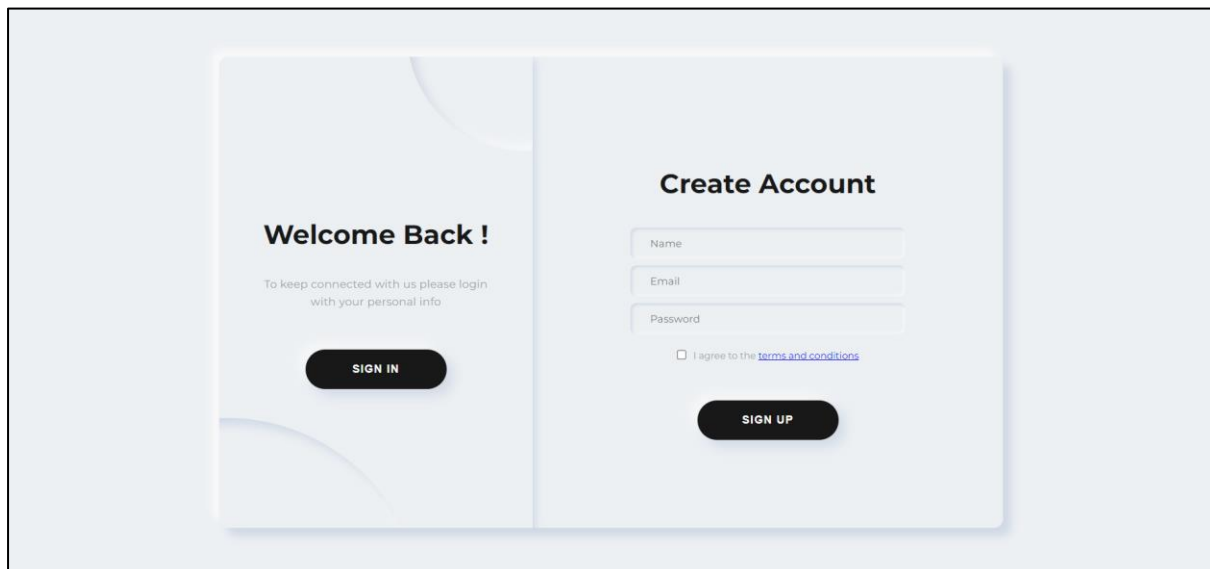


Fig. 4 sign in / sign up page

Fig. 4 shows the landing page of the web application that greets the user at the time of sign in / sign up.

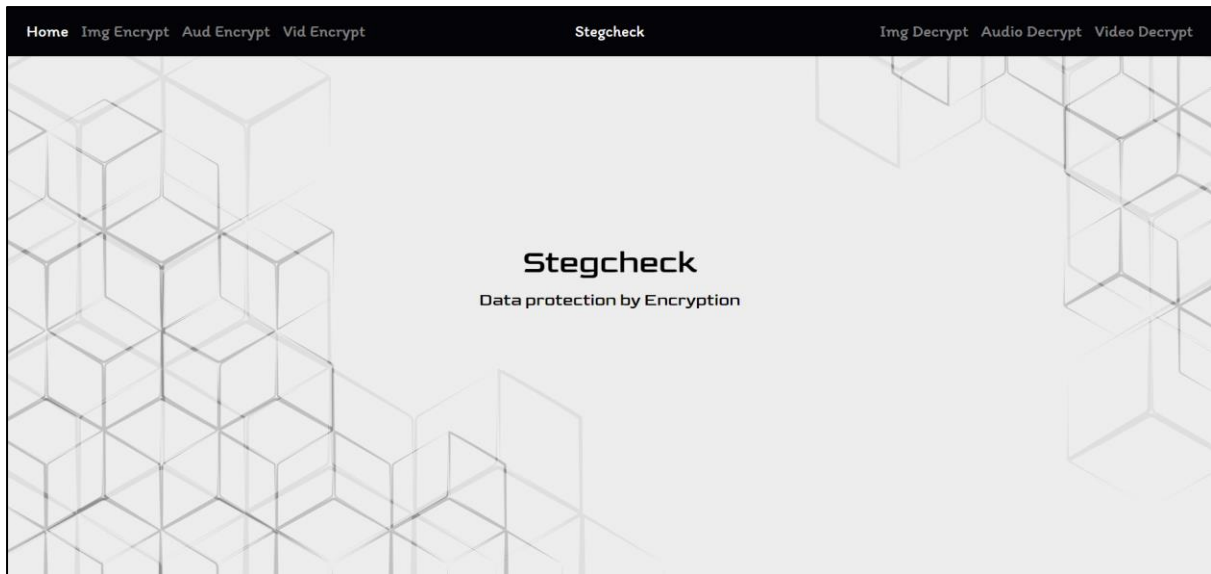


Fig. 5 landing page

Fig 5 shows the landing page after a successful sign in

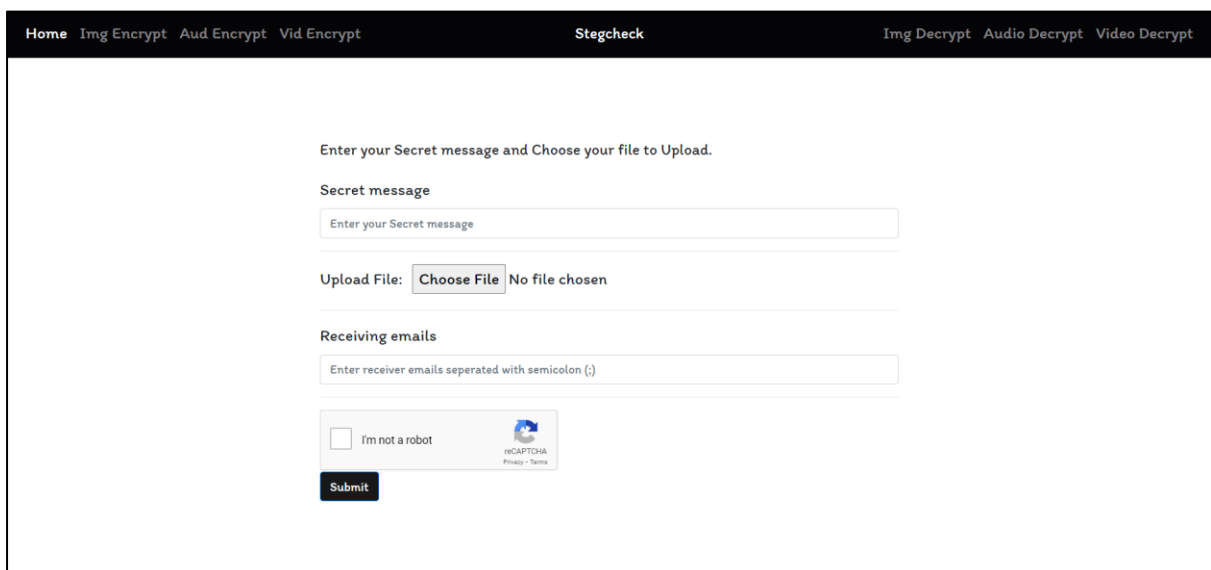


Fig. 6 Image steganography encryption page

Fig. 6 shows the page that the user will use for performing image steganography.

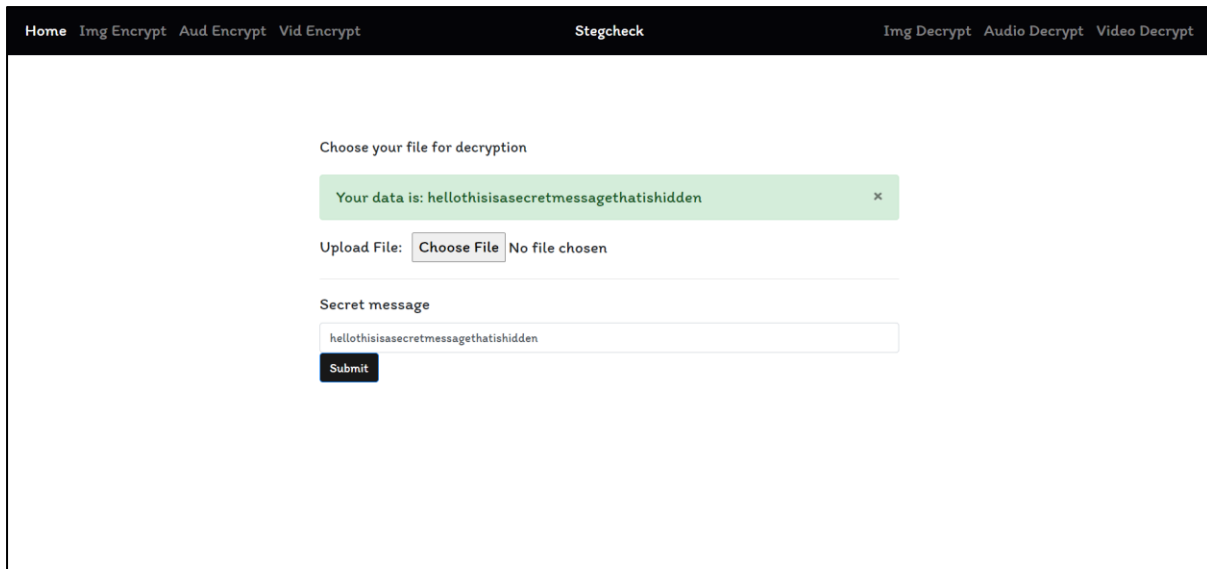


Fig. 7 Image decryption process

Fig. 7 shows how the user can decrypt and decode the image to get the plain text.

```

hash_file.py > hash_file
1  import hashlib
2
3  def hash_file(filename):
4      h = hashlib.sha1()
5      with open(filename,'rb') as file:
6          chunk = 0
7          while chunk ≠ b'':
8              chunk = file.read(1024)
9              h.update(chunk)
10     return h.hexdigest()

```

Fig. 8 hash_file.py

Fig. 8 shows us the method hash_file() which is used to get the hex hash of the input file which will be used by the virus total API.

```

virus_scan.py > virus_scan
1  from virus_total_api import PublicApi as VirusTotalPublicApi
2
3  def virus_scan(file_hash):
4      print("incoming filehash: ", file_hash)
5      API_KEY = '0640b09df7218cc31db9366a5244edfb71b2daeed4ad514e9db328c899b3311e'
6      vt = VirusTotalPublicApi(API_KEY)
7      response = vt.get_file_report(file_hash)
8      print("printing the response from virus total:", response['results'])
9      return [response['results']['positives']]

```

Fig. 9 virus_scan.py

Fig. 9 shows us the method `virus_scan()` that takes file hash as input and send it to virus total for scanning and traverse through the JSON response and returns the number of positives.

```
def encode_image(image, secret_message):
    n_bytes = image.shape[0] * image.shape[1] * 3 // 8
    if len(secret_message) > n_bytes:
        raise ValueError("this text could not be encoded in this image")
    secret_message += "!@#$$%)(*&^"
    data_index = 0
    binary_secret_msg = message_to_bin(secret_message)
    data_len = len(binary_secret_msg)
    for values in image:
        for pixel in values:
            # convert RGB values to binary format
            r, g, b = message_to_bin(pixel)
            # modify the least significant bit only if there is still data to store
            if data_index < data_len:
                # hide the data into least significant bit of red pixel
                pixel[0] = int(r[:-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            if data_index < data_len:
                # hide the data into least significant bit of green pixel
                pixel[1] = int(g[:-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            if data_index < data_len:
                # hide the data into least significant bit of blue pixel
                pixel[2] = int(b[:-1] + binary_secret_msg[data_index], 2)
                data_index += 1
            # if data is encoded, just break out of the loop
            if data_index >= data_len:
                break
    return image
```

Fig. 10 image steganography `encode_image()`

Fig. 10 shows us the `encode_image()` method which takes the image and the encrypted secret message as input and initiates the encoding process.

```
def decode_image(image, filename, email):
    print("Calling decode_image, printing user's email:", email)
    binary_data = ""
    dec_img_hash = get_hash("./" + filename)
    con = sqlite3.connect('testdb.db')
    cur = con.cursor()
    cur.execute("SELECT * from media_info where multimedia_hash = ?", (dec_img_hash,))
    result = cur.fetchone()
    cur.execute("SELECT receiving_email from media_info where multimedia_hash = ?", (dec_img_hash,))
    receiving_emails = cur.fetchone()
    receiving_emails_list = (str(list(receiving_emails))[2:-2].split(";"))
    flag = 0
    for receiver_email in receiving_emails_list:
        if email == receiver_email:
            flag = 1
            break;
    if result and flag == 1:
        for values in image:
            for pixel in values:
                r, g, b = message_to_bin(pixel)
                binary_data += r[-1]
                binary_data += g[-1]
                binary_data += b[-1]
            all_bytes = [ binary_data[i: i+8] for i in range(0, len(binary_data), 8) ]
            decoded_data = ""
            for byte in all_bytes:
                decoded_data += chr(int(byte, 2))
                if decoded_data[-10:] == "!@#$$%)(*&^":
                    break
            # print(decoded_data[-10])
            return decoded_data[-10]
    else:
        print("this image cannot be decrypted")
    con.close()
```

Fig. 11 Image steganography `decode_image()`

Fig. 11 shows us the `decode_image()` method which takes the image filename and the receiver's email as input and checks if the receiver's email is listed in the table for this particular file's hash. If the email is present we proceed with the decoding process and the decoded encrypted text is sent to text decryption so that we get the plain text to present to the receiver.

```
def encode(audio, message, sender_email):
    print(" ... Encoding Starts ... ")
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
    message = message + int((len(frame_bytes) - (len(message)*8*8))/8) * '#'
    bits = list(map(int, ''.join([bin(ord(i)).lstrip('0b').rjust(8,'0') for i in message]))
    for i, bit in enumerate(bits):
        frame_bytes[i] = (frame_bytes[i] & 254) | bit
    frame_modified = bytes(frame_bytes)
    new_audio = wave.open("stegaudio.wav", "wb")
    new_audio.setparams(audio.getparams())
    new_audio.writeframes(frame_modified)
    new_audio.close()
    audio.close()
    aud_hash = get_hash("./stegaudio.wav")
    con = sqlite3.connect('testdb.db')
    cur = con.cursor()
    query = "update media_info set multimedia_hash = ? where sender_email = ? and type = ?"
    data = (aud_hash, sender_email, "audio")
    cur.execute(query, data)
    con.commit()
    con.close()
    print("steganography process complete ... ")
    print("new audio will be stored as stegaudio.wav")
```

Fig. 12 Audio steganography encode()

Fig. 12 shows us the `encode()` method in the audio steganography process where it takes the actual audio, the secret message and the `sender_email` as input, and starts the encoding process by converting the wav file into bits and enumerating the said bits along with the secret message.

```
def extract_frame(filename):
    vidcap = cv2.VideoCapture(filename)
    success, image = vidcap.read()

    temp_folder = "./temp1/"
    count=0
    while True:
        success, image = vidcap.read()
        if not success:
            break
        cv2.imwrite(os.path.join(temp_folder, "{:d}.png".format(count)), image)
        count += 1
```

Fig. 13 Video steganography extract_frame()

Fig. 13 shows us the `extract_frame()` method that is used for video steganography which just takes the filename as input, and we are using `opencv` to open and read the video file and extract each frame from the video and store it in a temp folder. Then we can make use of the methods that we used for image steganography to encode and decode message into each frame.

References:

[1] <https://code.visualstudio.com/download>

[2] <https://www.heidisql.com/>

[3] *Virustotal API V3 overview* (no date) *VirusTotal*. Available at:
<https://developers.virustotal.com/reference/overview> (Accessed: December 10, 2022).