CS 6348 - Data Application and Security

Final Project Report

Sai Rohith Koritala (sxk210141)
Venkatatarun Madhavarapu (VXM180032)
Tejas Bogguram Vasudev(Txb1870006)
Jay Challangi (JXC180095)

Major Steps of this Implementation

1. **Implementing encryption of metadata into a frame**

   Each encrypted file video begins with a metadata frame that contains two pieces of information: the title of the file and the padding length used during video creation. The title and padding length are separated by a delimiter, typically a question mark, and the data is terminated with the same delimiter. The remaining part of the metadata frame consists of padding. This entire frame is encrypted using AES encryption. The metadata frame is important because it contains the necessary information to reconstruct the file, including the file type and the specific data that needs to be decrypted to reveal the file contents, as well as the padding data that can be ignored. The padding length refers to the additional data appended to the encrypted file to ensure a proper frame is displayed.

2. **Implementing encryption of data**

   Our code defines a class myAES that can encrypt and decrypt files using the AES encryption algorithm in Galois/Counter Mode (GCM). This class uses the Python Crypto library for generating a random salt, deriving a key from a password using the script key derivation function, and creating an AES cipher object for encryption/decryption.

   The AES class takes three arguments: the name of the input file, the name of the output file, and a password used to derive the encryption key. The encrypt method reads in the input file, generates a random salt, derives an encryption key from the password and the salt using a script, writes the salt and the nonce (generated by the AES cipher) to the output file, encrypts the data from the input file using the AES cipher in GCM mode, and writes the encrypted data to the output file. Finally, it appends an authentication tag (also generated by the AES cipher) to the output file for integrity verification.

3. **Implementing uploading the frames as a video onto Youtube**

   Once the method gets the file path, password, Youtube Video Title, and description. The method starts off by encrypting the file using the AES encryption described above. Then the method takes the encrypted file and converts it into a binary string, and then generates the video by converting the binary string into a series of frames. Once the video is saved successfully the program performs a Google authentication with the user and then uploads the video onto YouTube with the title and description provided by the user.

   The passwordEncryptor is an initialization of the PublicPrivate class. This class has multiple methods

   1. The generateKeys() function generates a private-public key pair using an RSA algorithm with a key size of 2048 bits. The private key is then saved in a file private_key.pem in the

PEM format using PKCS8 private key encoding. The public key is also saved in a file public_key.pem in the PEM format using SubjectPublicKeyInfo public key encoding.

2. The encrypt() function takes the path to the public key file and a password string as arguments. It loads the public key from the file, encodes the password string in UTF-8 format, and encrypts it using the public key with the OAEP padding scheme, which uses the SHA-256 hash function for both the mask generation function and the hash function. The encrypted password is saved in a file encrypted_password.key.
3. The decrypt() function takes the path to the private key file and the path to the encrypted password file as arguments. It loads the encrypted password from the file, loads the private key from the file, and decrypts the encrypted password using the private key with the same OAEP padding scheme. The decrypted password is printed to the console in UTF-8 format.

Once the password is secured. The program moves onto the file. Where it encrypts it using the AES encryption method described above. The encrypted file is then converted to a binary string using the file_to_binary function, and the length of the resulting binary string is printed. Padding is then added to the binary string so that it is a multiple of 320*180 bits as each pixel is represented by a 4X4 block, only 302*180 bits of data can be stored in each frame. Metadata is also added to the binary string, which contains the name of the original file and the length of the padding. , and the number of images required to generate the video is calculated. A video writer is then initialized using the cv2 library with a resolution of 1280x720 and a frame rate of 25. The binary string is then looped through, and images are generated for each block of 320*180 bits. Each image is created with a resolution of 1280x720 pixels and all white pixels. The binary string is then parsed, and the pixel values are updated based on the bits in the string. 16 Black pixels are added for a value of 1 and 16 white pixels for a value of 0. The final image is saved and written to the video file.
The ytHandler.upload function is then called to upload the video to a YouTube channel with a title and description, and a message is returned stating that the video was created.

4. Implementing decryption of the video that can be downloaded from Youtube

The decrypt method reads in the encrypted file, extracts the salt and nonce from the file, derives the encryption key from the password and salt using the script, creates an AES cipher object using the derived key and the nonce, reads the encrypted data from the file, decrypts the data using the AES cipher, and writes the decrypted data to the output file. Finally, it verifies the authentication tag appended to the file for integrity verification.

If any loved ones wish to decrypt the data containing the video that was uploaded by the user, they would have to follow these steps:

- They must manually download the video from the platform where it was uploaded.
- Next, they need to contact the user to obtain the password that was used to encrypt the video.
- Then, once they have received the password, they can use their private key to decrypt it. and access the data.

5. Implementing UI to automate decryption of video after downloading from Youtube

The Python GUI is made using PySimpleGUI. Users of the GUI have two options: save new files or access files that have already been saved.

The main_window() function creates the main window of the GUI, which displays the logo of the application and two buttons: "Save a new file" and "Retrieve an already saved file". The function also creates a "terminal" window that displays messages to the user. The function enters a loop that waits for the user to click on a button or close the window. If the user clicks on the "Save a new file" button, the function calls the save_file_window() function. If the user clicks on the "Retrieve an already saved file" button, the function calls the retrieve_file_window() function. If the user closes the window, the loop terminates and the GUI is closed.

An additional window with a form for the user to fill out is created by the save_file_window() function. The following inputs are requested by the form from the user:

1. When a user selects the "Select file to upload" button, a file browser displays, allowing them to upload a file.
2. This is entered into a password field.
3. A file browser that opens after the user hits the "Select public Key file" button is used to access the public key file.
4. Title of a video This was entered into a text box.
5. a description of a video This was entered into a text box.

The function gets the input field values once the user presses the "Submit" button, prints some of them to the console, updates the GUI's "terminal" window, then calls the Handler.save() method to start the processing of the file.

The user can fill out a form in a new window that is created by the retrieve_file_window() function. The following inputs are requested by the form from the user:

1. The path to the YouTube video A file browser that opens after the user selects the "Select Youtube video to upload" button.
2. Then the user provides their password in plain text.

After the user clicks on the "Submit" button, the function retrieves the values of the input fields, prints some of them to the console, updates the "terminal" window of the GUI, and invokes the Handler.retrieve() method to retrieve the file. Finally, the function displays a message indicating the file name that was retrieved.

The Difficulties Faced During Implementation

Initially, Pytube was the preferred tool for downloading videos from YouTube. However, due to a recent update by YouTube regarding the way metadata is handled, this method became obsolete. Because of this, we needed to redo our implementation to allow users to directly download videos from YouTube and decrypt them using the program. As a result, the user now has to input the file path, and the program's user interface had to be updated to accommodate downloaded videos. Our initial implementation enabled users to interact with the program without accessing YouTube directly. However, due to the update, we had to redo our implementation and cause the user to have more involvement in this process. The user needs to download the video (using a third-party application) and provide the file path to our program.

When uploading videos to YouTube, a common issue that arises is the loss of data due to compression. The video compression process can lead to a significant reduction in quality, particularly in terms of resolution and sharpness. As a result, videos can appear pixelated or blurry, making it challenging to distinguish small details or specific elements. In our case when we would download the video for decryption the contents would be altered before and after uploading it to Youtube due tot he nature of Youtube's compression algorithm. To address this issue, we sought to find a way to prevent data loss during the video compression process. After much experimentation, we came up with a novel approach - representing each bit in the video using a 16 by 16 square of pixels. By doing so, we ensured that the video retained all of the encrypted file data without being significantly affected by the compression process. Additionally, we ensured that the uploaded video had a resolution of 720p, which provided sufficient clarity and sharpness to display the 16 by 16 square of pixels accurately. The use of this approach ensured that the video quality was preserved and lessened the effect of Youtube's compression algorithm on our uploaded video, in comparison to 1080p.

Optimizing the program's performance was a significant challenge that we encountered during the development process. Initially, the program ran very slowly, which impacted its overall efficiency and user experience. As a result, we had to make a concerted effort to improve its performance and ensure that it could execute tasks quickly and seamlessly. To achieve this, we employed a range of optimization techniques, including refactoring the codebase to improve its organization and structure. We also focused on minimizing the program's memory usage, reducing the number of unnecessary computations, and eliminating redundant code. We reviewed the algorithms used in the program, seeking to optimize them for performance while retaining their accuracy and functionality.

At the beginning of the project, one of the challenges we faced was identifying a suitable Python library that could effectively encode and decode videos. However, after much research and experimentation, we eventually settled on using OpenCV (cv2) as our primary library for video encoding and decoding. This decision was based on several factors, including its broad support for multiple video formats and codecs, ease of use, and extensive documentation. OpenCV provided us with access to a range of advanced video processing capabilities, such as frame manipulation, video filtering, and object detection. This allowed us to add more advanced features to our program and enhance its overall functionality.

The Division of Labor among Group Members
Tejas Vasudev:
Tejas handled the creation and reading of the video, and also handled the youtube authentication and upload

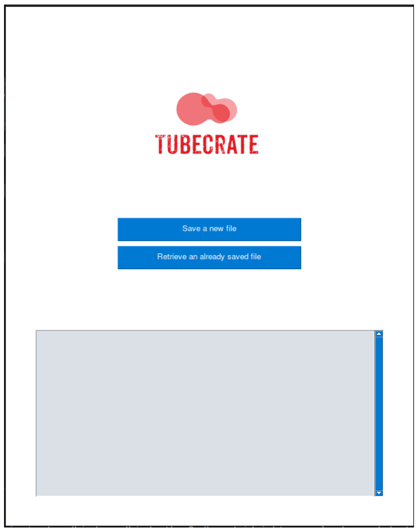
Jay Challangi:
Jay's responsibilities were to handle the RSA encryption and decryption of the password. And also the private key and public key encryption and decryption

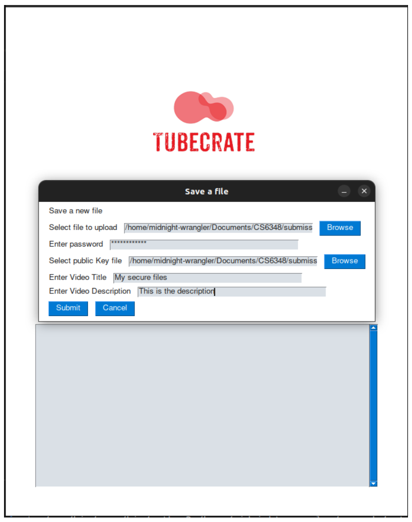
Tarun Madhavarapu:
Tarun's responsibilities were to create the UI, and the metadata

Sai Rohith Koritala:
Rohith was responsible for the creation of the UI and workflow

Usage of the Software with Screenshots

Images of our UI and video uploaded onto Youtube.





Save a new file

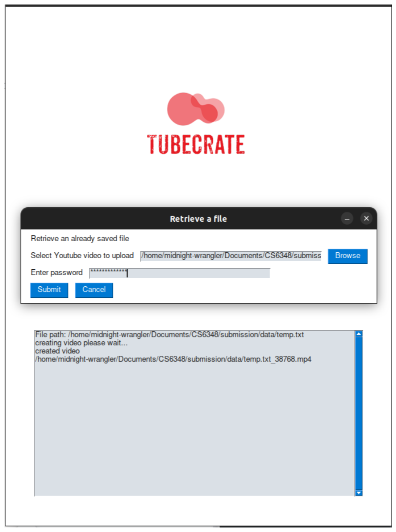
Select file to upload /home/midnight-wrangler/Documents/CS6348/submiss

Enter password ************

Select public Key file /home/midnight-wrangler/Documents/CS6348/submiss

Enter Video Title My secure files

Enter Video Description This is the description



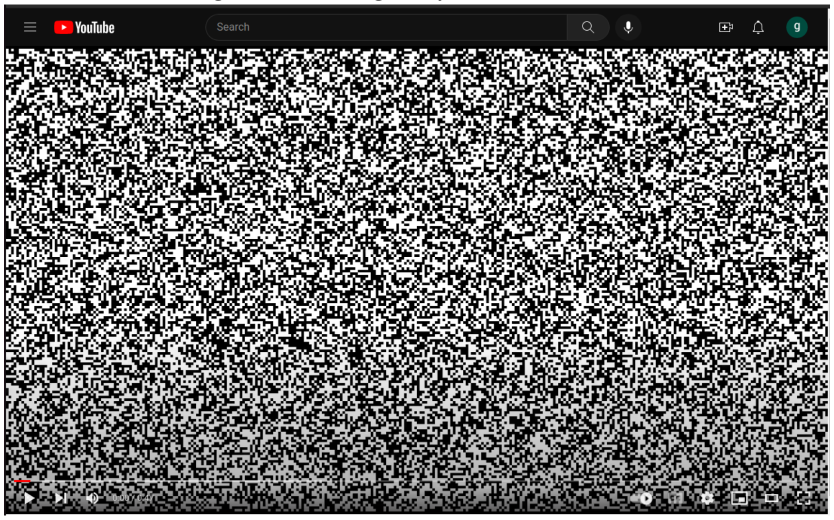
Retrieve an already saved file

Select Youtube video to upload /home/midnight-wrangler/Documents/CS6348/submiss

Enter password *************

```
File path: /home/midnight-wrangler/Documents/CS6348/submission/data/temp.txt
creating video please wait...
created video
/home/midnight-wrangler/Documents/CS6348/submission/data/temp.txt_38768.mp4
```

1. Images opening page with 2 buttons
2. Saving a new file onto youtube
3. Generating the file using the youtube video



Video on Youtube