Final Project

Group Members:

Christopher Erhard (cje5298@psu.edu)

James Zukowski (jjz5192@psu.edu)

Arib Hyder (aah5469@psu.edu)

Anoop Soodini (ars6375@psu.edu)

Professor:

Mark Mahon

Course:

CMPEN 462 - Wireless Security

Spring 2021

Abstract

For our final project, we built off what we have learned in class regarding cryptography and modulation. Our goal was to simulate an encryption/decryption program in Python which would also modulate and demodulate the encrypted binary data. For the encryption function, the program prompts the user to input a text file, converts this text file from characters to binary, encrypts the binary data with a key, prompts the user to choose between 3 modulation schemes, modulates this data by converting it into OFDM symbols, and finally stores the OFDM symbols along with the encryption key and modulation scheme into a csv file. For the decryption portion, the program prompts the user to input a csv file that was outputted by the encryption function of our project, converts the OFDM symbols back into encrypted binary data using the given modulation scheme, decrypts this data using the given key, converts the decrypted binary data to characters, and stores the output text result in a text file. To build this, we used Python 3 along with several libraries.

Flow Chart

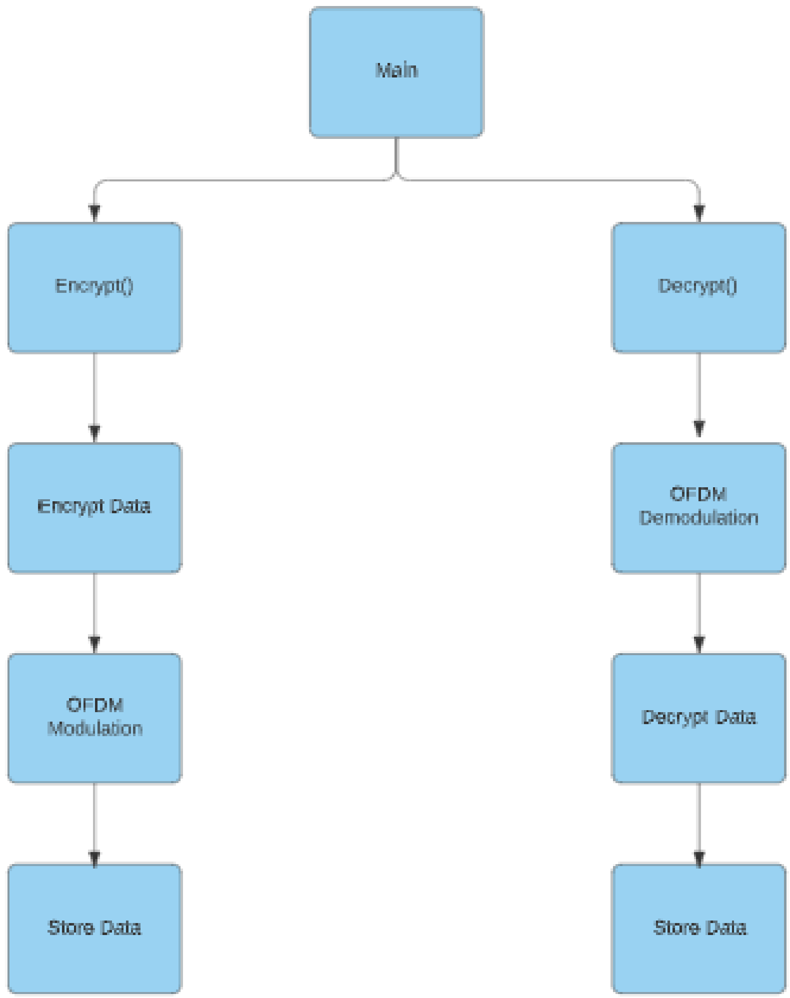


Figure 1: Application Flowchart

Introduction

Our final project's main goal was to build a user-friendly application that takes an input from the user, encrypts it, and then decrypts it whenever the user wants. In order to do this we began by looking over the two mini-projects that we built this semester. We knew that based on the two mini-projects, we would be able to adapt the code to work together in order to encrypt and decrypt data entered by the user. The program starts by asking if the user would like to encrypt or decrypt a file. Next, the user can upload a file for the application to encrypt or decrypt. The user will then be asked what modulation or demodulation technique they want to be used. Finally, the program will store the data into a separate file that can be accessed by the user.

Functional Blocks

Main

The main function controls the overall flow of our application. The function is encompassed in a while loop to continuously run until the user no longer wishes to encrypt or decrypt a file. The application first asks the user if they would like to encrypt or decrypt a file. The application then asks for the name of the file and searches for it in the current working directory. If the user wishes to decrypt a file, the modulation type and key are both taken from it and passed into the decrypt function. After every iteration of encryption or decryption, the application will ask the user if they wish to encrypt or decrypt another file. If they do not wish to, the program ends successfully.

Encrypt

For our encrypt function, we are passed a text file and the name of that file. From here, we extract the name of the file without the extension for naming output files (this is stored in a variable is called "base") and create variables to store our data for later in the function. Next we loop through our input file to store all of the text in a string. From here, we convert this string of ASCII characters to its equivalent string of binary characters for encryption and store this in base_toBinary.txt. Next, using Fernet encryption from the Python cryptography library, we generate a key, encode our data using the encode() function (this is required for using the Fernet encrypt function), and encrypt our data using the fernet.encrypt() function. After this, we convert our encoded encrypted data back from encoded characters and back into encrypted binary bits for modulation. We store our encrypted binary data in a text file called base_encryptedBinary_in.txt. After this, we have our encrypted binary data so all that is left to do is modulate this data. The user is prompted to choose between BPSK, QPSK, and 16QAM modulation schemes. Once one is chosen, we loop through the bits of our encrypted binary data to convert the bits to OFDM symbols. Finally, when this is finished we store the list of OFDM symbols, the chosen modulation scheme, and the encryption key (for use later on with the decryption function) and a csv file called base_MODULATION.csv (Where MODULATION is the modulation scheme chosen for that encryption) for output. This output will be the input for our decrypt function.

Decrypt

For our decrypt function, we pass in the name of the file (this is stored in a variable called “base”), a list of OFDM symbols, the modulation type, and the encryption key (all generated by the encrypt function and stored in its output). The encryption key is then UTF-8 encoded for use with the Python cryptography functions later. The function next checks the modulation type and demodulates the input list of OFDM symbols by converting them into binary bits based on the modulation type. After this step, our demodulated binary data is still encrypted but no longer in the form of OFDM symbols, and this encrypted binary data is now stored in a file called base_encryptedBinary_out.txt. Next, we convert our provided encoded key into a Fernet key for decryption using the Fernet function. We next convert our binary data from binary bits to ASCII characters for use with the Python cryptography decryption function. We then UTF-8 encode these ASCII characters using the encode() function and decrypt this data using the fernet.decrypt() function. Our decrypted binary data is then stored in a file called base_decryptedBinary.txt. Finally, we loop through our decrypted binary data and convert the bits to 8-bit ASCII characters to receive our final desired decrypted text. This final decrypted text is then stored in an output text file called base_decryptedText.txt.

Problems

Our initial plan was to implement this application on a Raspberry Pi. However, due to time constraints and shipping issues, we were unable to receive the Raspberry Pi in a timely fashion. Due to this, we decided to implement the application in a fashion that would be similar to how it would behave on the device. To test the application, we used the Windows and Mac command lines and ran the program using the following command: python3 FinalProject.py. We kept our input files in the same working directory as the Python file.

Other problems encountered in this project had to do mostly with going from text to bits and back. Converting our original encryption text input from characters to binary bits was not too difficult, but once the fernet.encrypt() and fernet.decrypt() functions were introduced more problems soon arose. When trying to use them initially, we did not realize that we would not be able to just pass our string of binary characters to the fernet.encrypt() function for encryption. Upon learning more about these functions, we learned that we needed to encode the input to these functions to use them. This also introduced the problem of converting the encrypted data back to binary for modulation. Once this was solved, a similar process was done in the decryption function that was essentially the reverse of the solution found to fix this problem in the encryption function.

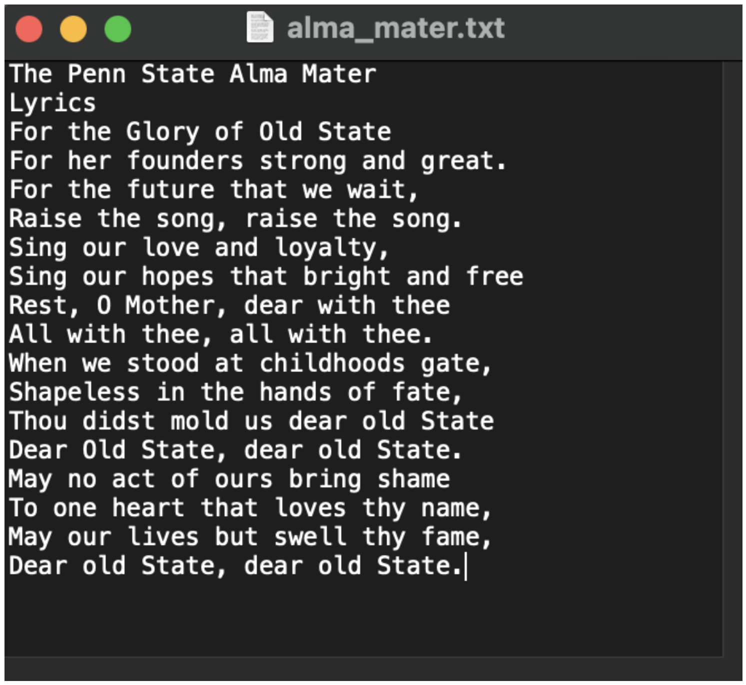
One last small problem we encountered was in installing the appropriate Python libraries needed for our project. We found that using the Python pip tool was the best way to solve this.

Results and Functionality

Testing

In order to extensively test the functionality of our program, we began by using a couple of different text files. The first test we ran was using the Penn State Alma Mater. We had to check to make sure that the result of our decrypt function is equivalent to the input text file. We also stored our results at several points during our functions. For example, we stored the binary output after encryption and before modulation in our encrypt function and also our encrypted binary data after demodulation in our decrypt function to compare similarity. We did this to make sure that we were getting the same demodulated data from our decrypt function as we were getting before modulation in our encrypt function. This helped us to narrow down where things were going wrong if we didn't receive the desired output (i.e. if these two were the same, our modulation/demodulation were correct and the error lies elsewhere). We also did something similar, storing the initial binary data received from converting our input text to binary in encrypt to compare to the binary in our decrypt right before converting from binary to characters for our final output. This would show us whether errors were occurring in the step where we went from binary to ASCII characters or not. Once we were sure our application was functioning as expected, we tried with a much larger input file, a text file of the Declaration of Independence. This file also was encrypted and decrypted as expected. Both of these input files are included in our submission. We omitted screenshots of the test of the Declaration of Independence due to their large size, but our results can be confirmed by running our application using this input as provided.

Result Images



alma_mater.txt

```
The Penn State Alma Mater
Lyrics
For the Glory of Old State
For her founders strong and great.
For the future that we wait,
Raise the song, raise the song.
Sing our love and loyalty,
Sing our hopes that bright and free
Rest, O Mother, dear with thee
All with thee, all with thee.
When we stood at childhoods gate,
Shapeless in the hands of fate,
Thou didst mold us dear old State
Dear Old State, dear old State.
May no act of ours bring shame
To one heart that loves thy name,
May our lives but swell thy fame,
Dear old State, dear old State.
```

Figure 2: Input text file

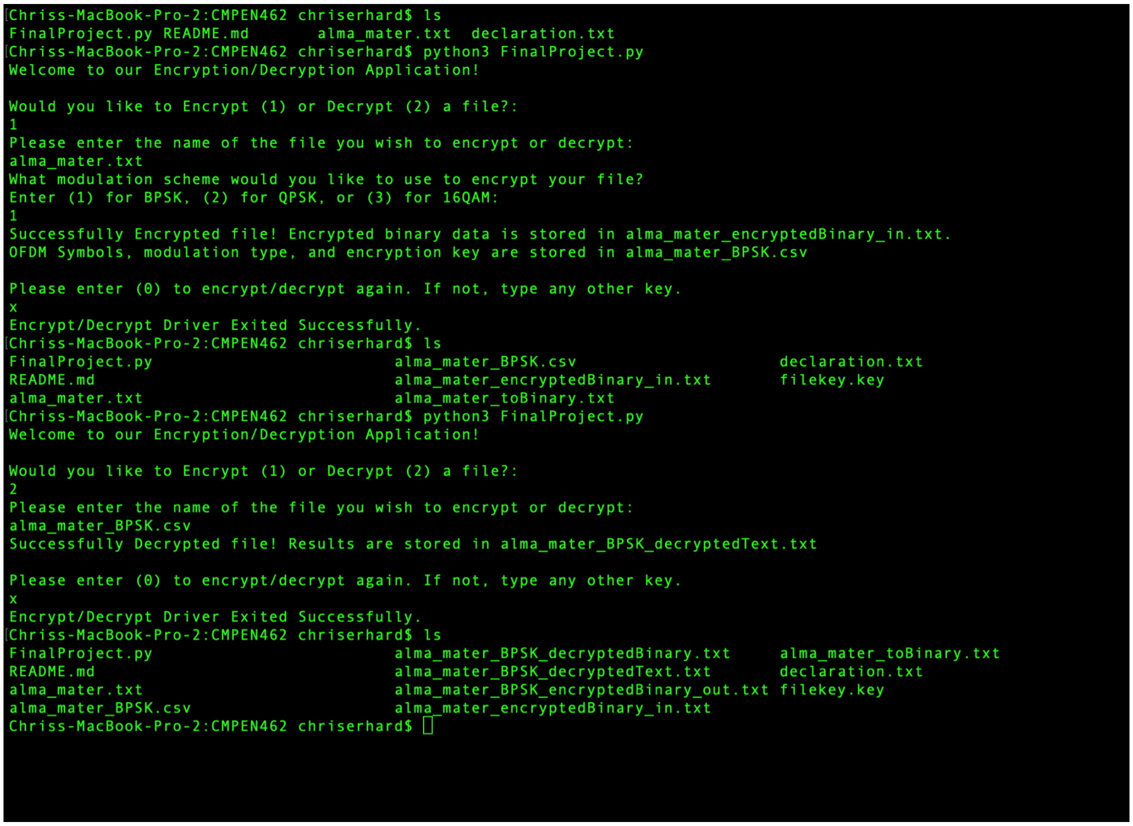


Figure 3: Running Application

Figure 4: Input converted to binary in encrypt (not yet encrypted)

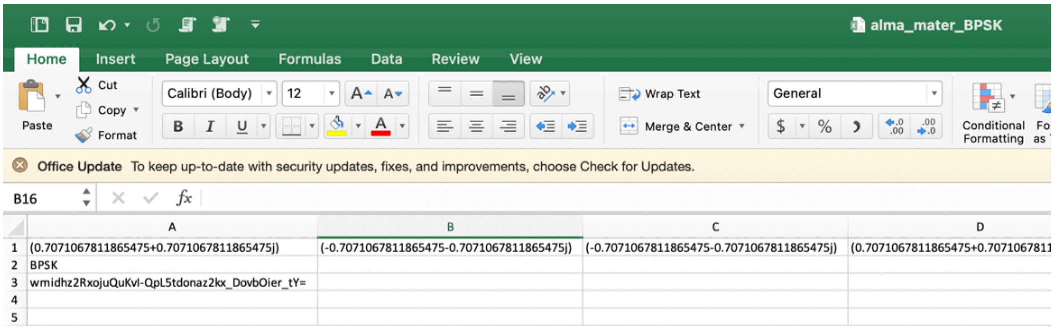


Figure 5: Output csv file produced by encrypt function (first row contains OFDM symbols, second row contains modulation scheme, third row contains encryption/decryption key)

Figure 6: Binary data after demodulation and decryption in decrypt function (final step before converting to ASCII)

alma_mater_BPSK_decryptedText.txt

```
The Penn State Alma Mater
Lyrics
For the Glory of Old State
For her founders strong and great.
For the future that we wait,
Raise the song, raise the song.
Sing our love and loyalty,
Sing our hopes that bright and free
Rest, O Mother, dear with thee
All with thee, all with thee.
When we stood at childhoods gate,
Shapeless in the hands of fate,
Thou didst mold us dear old State
Dear Old State, dear old State.
May no act of ours bring shame
To one heart that loves thy name,
May our lives but swell thy fame,
Dear old State, dear old State.
```

Figure 7: Final output after decryption

Work Performed and Roles

Task	Completed by
Main()	Anoop Soodini, James Zukowski, Chris Erhard
Encrypt()	Anoop Soodini, James Zukowski, Arib Hyder
Decrypt()	Chris Erhard, Arib Hyder
Encryption Testing	Chris Erhard, Anoop Soodini, James Zukowski
Decryption Testing	Chris Erhard, Anoop Soodini, James Zukowski
Functionality Testing	James Zukowski, Anoop Soodini, Chris Erhard
Writeup	Chris Erhard, Anoop Soodini, James Zukowski, Arib Hyder

Table 1: Worked Performed and Roles

While writing the script for the program, we began by first designing a flowchart of what needs to get done and then we split up the work as such. James and Anoop began writing the driver code in the main function, and Chris worked on improving the functionality of it. Encrypt and Decrypt were written by all members collectively. For testing, Chris took care of testing the encryption and decryption, while James and Anoop focused on testing the user interface. The write-up was done by all members of the group and work was divided evenly throughout.

While writing the program, we used Discord and text messages as our main form of communication. On Discord, we talked to each other and simultaneously shared our screens to work collaboratively. We created a Github repository to manage our code and keep track of commits and changes to our code. While working, we would help each other with individual sections and commit changes when finished so the next person could work on their portion. We used texting to decide when to meet and if anyone had any questions about any part of the project. To complete the report, we created a shared Google Doc and worked on together while on a Discord call.