Papertrail (Encrypted File Storage using Paper Application)

Team #7: Benjamin Catts, Mohammad Qasimi, Hooman Moridzadeh, Sohal Sudheer, and Connor Wadlin

Table of Contents

Preface: List of Figures

1: Introduction

For our project, we developed a secure way to take digital information and transform it into an encrypted physical medium and back again. Regardless of the application, this program allows users to take information from their electronic device and convert it into a QR code containing their data in an encrypted form. This QR code can be run through the program later to be unencrypted, so long as the password given is the same as during the encryption.

We set out to create a secure, simple, and smart way of taking information from a computer into the real world and that is what we've done! Now you do not have to worry about your personal information being stolen from your computer as you can keep all of your most important information in a physically secure place, like a lockbox in a bank.

1.1: Problem Statement

Some people have a desperate need to back up small amounts of sensitive data long-term and wish to do so securely. In the modern age, encrypting documents with important personal information such as bank information, passwords, and such has become a critical aspect of keeping your information safe, yet there are not many ways to do this. This is where we come in, as we offer a way to keep all of your most important information safe in an easy-to-use and simple-to-store manner. All you have to do is take your document, run it through the program, download the generated QR code, and print it out!

2: Use Cases

As our application is designed with ease of use in mind, users are only expected to be familiar with locating a file within their computer, printing files from a computer, and scanning QR codes. No need to spend years studying computers, this application does all of the hard work for you!

This application works in both Windows and Linux environments, so as long as the user has one of those types of devices it should work perfectly.

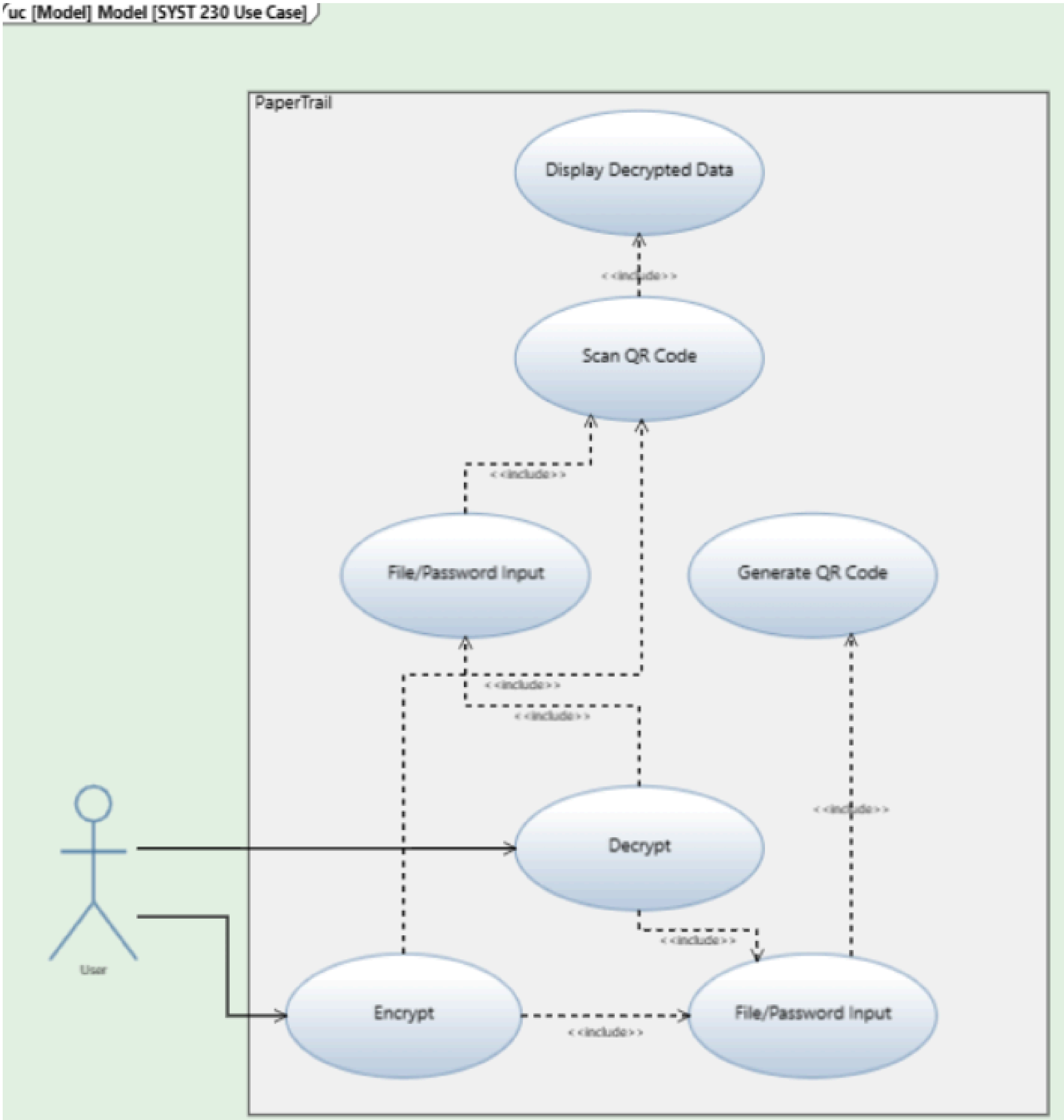
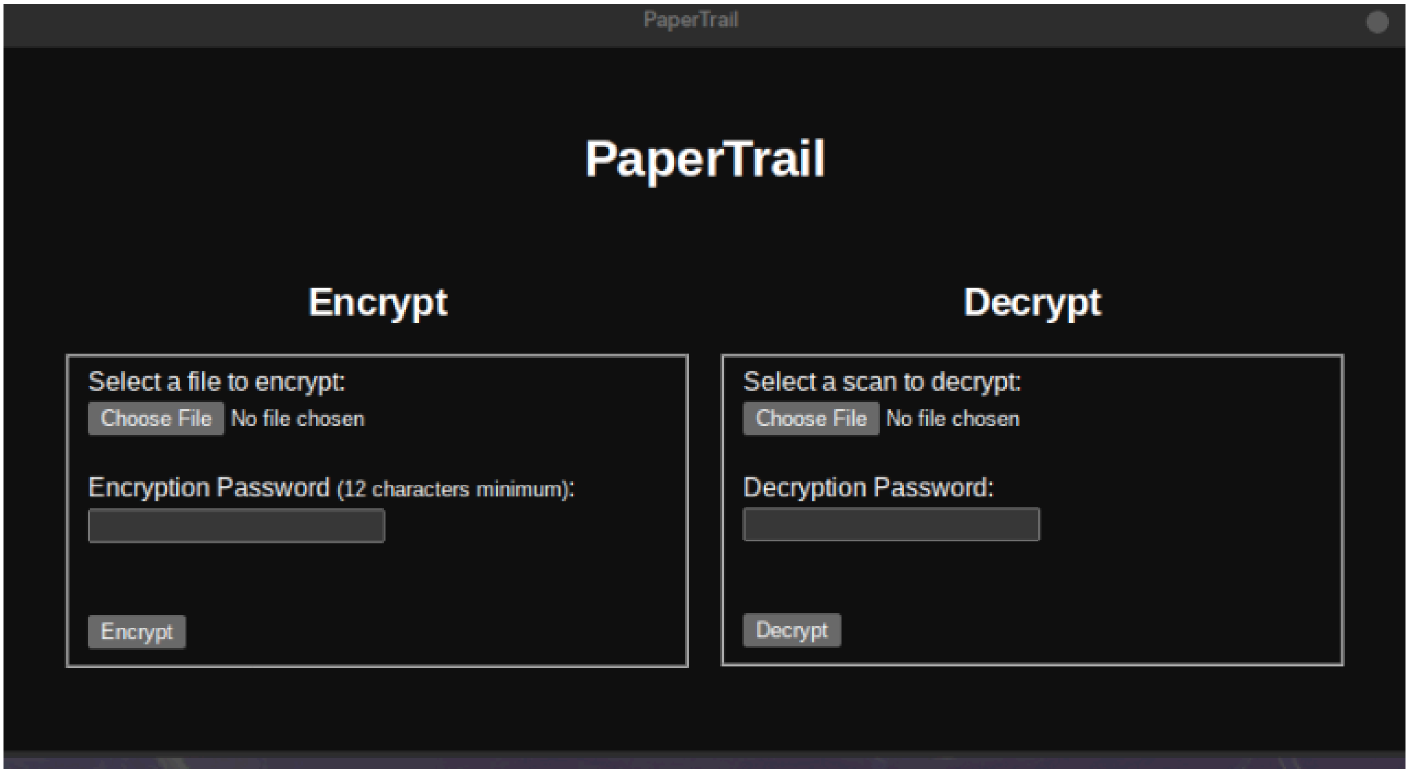


Figure 1: Use Case Diagram

2.1: User Interface



Select a file to encrypt:

No file chosen

Encryption Password (12 characters minimum):

Select a scan to decrypt:

No file chosen

Decryption Password:

Figure 2: User Interface

2.2: Use Case Scenarios

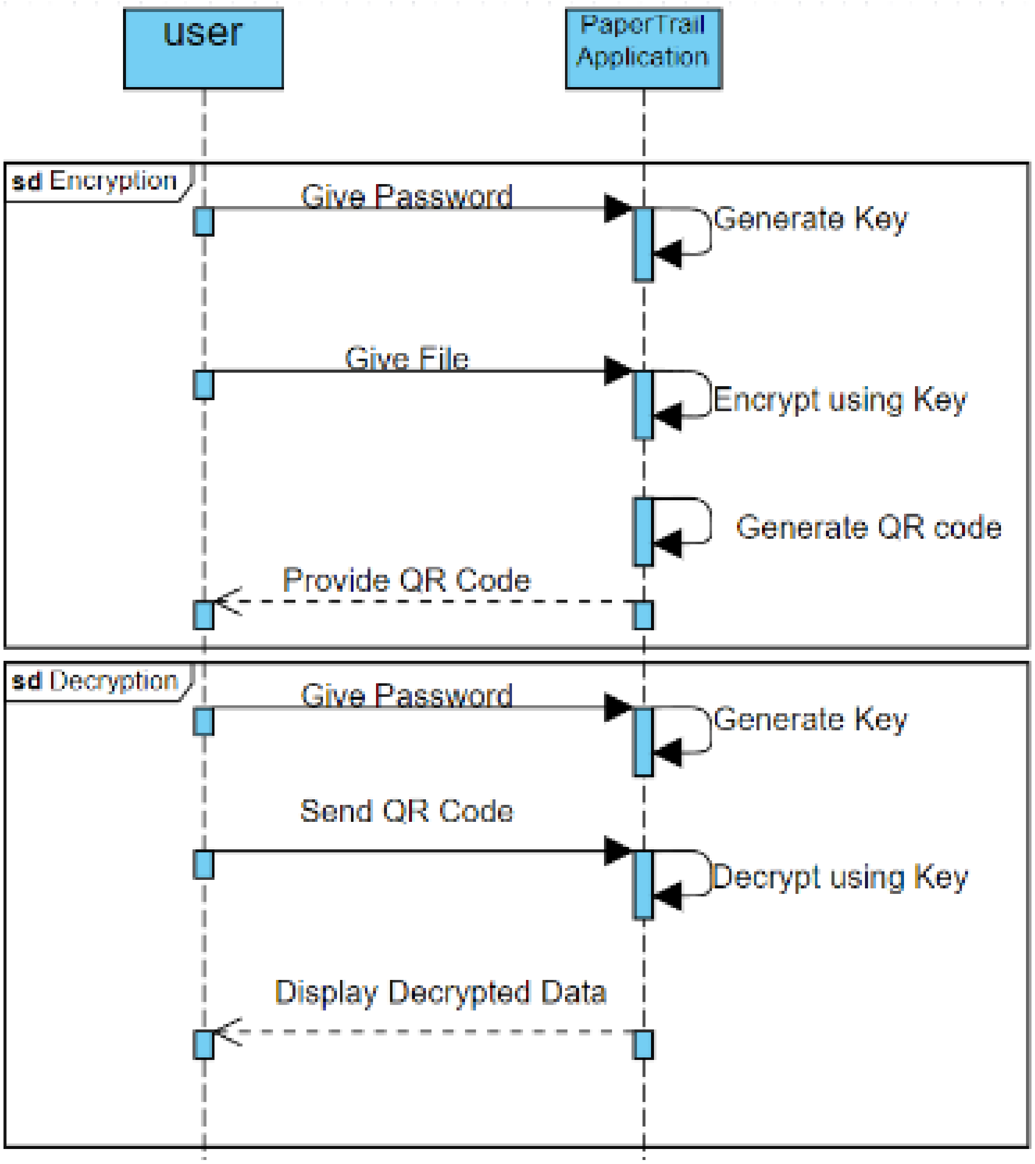


Figure 3: Use Case Scenario

3: Requirements

We have broken down our requirements into multiple categories to make everything as clear as possible regarding what this application does.

Functional requirements are the processes that the program must be able to do. Functions such as encryption, decryption, key generation, and more are included in this section.

Usability requirements are the aspects of the application with which the user can interact with. Being able to input a file, type in a password, and more are included in this section.

Implementation requirements are the technical backends required for the application to run properly. Having the proper version of Python, being able to run HTML, and such are included in this section.

Dependencies are parts of the code that are required for the application to run properly. There are a variety of libraries brought in to perform specific tasks that are included in this section.

3.1: Functional Requirements

- Able to turn files into password-encrypted data encoded in a series of QR codes exported in an easily printable format
- Able to, using a scan of the series of QR codes in the easily printable format mentioned in the previous requirement, be able to decode the QR codes back into data that can then be decrypted with the same password
- The program can take in any standard binary or ASCII file data
- If a file is encrypted and turned into a printable format by the program and then that printable-format file is run through the program, scanned, and decrypted, the data returned must be binary equivalent to the original file

3.2: Usability Requirements

- Has both encryption and decryption options available to the user
- Has clear instructions such that a user can easily tell the functionality of the program
    - This means that a user can visually tell what actions are required, not an actual instructions manual
- User can select a file from their file explorer using the provided button without having to trace the whole path
- An encrypted file with the QR code will be provided to the user and it will be clear when this has been done through a visual representation via a new window pop-up
- User can select a QR code and have the data scanned such that decryption can occur using the provided password
- A downloadable file will be provided to the user once decryption is finished using the correct password through a pop-up message.
- Any errors throughout the program will be made known to the user through a visual representation

3.3: Implementation Requirements

- Platforms: Windows [Windows 11 x86_64], Linux [Fedora Silverblue x86_64 (uses GNU libc)]
- Language(s):
    - Python 3.9
    - HTML/CSS
    - Javascript

3.4: Dependencies

- Eel (PyPI) (Docs) - UI framework, allows UI to be written in HTML/CSS/JS while still operating offline
    - qrcode (PyPI) (Docs) - Allows us to generate QR codes
    - pillow (PyPI) (Docs) - Adds more functionality to qrcode, lets us work with image data to combine codes onto one nicely formatted page, and allows for export as PDF
    - cryptography (PyPI) (Docs) - A high-level library for encrypting and decrypting data, can be used to encrypt data using Fernet symmetric encryption, also contains PBKDF2HMAC for turning user-provided password into Fernet keys
    - base64 (builtin) - A way to encode binary data into ASCII characters, a necessary dependency for key derivation from the encryption password
    - pyinstaller (PyPI) (Docs) - Allows the application to be packaged as a standalone executable so that end users don't need Python installed or to run the program from the command line

4: Application Design

During the creation of the application, design was something we wanted to ensure was as easy for people to use as possible while still accomplishing all of our tasks. As different aspects of the overall design were handled in different ways, we have chosen to break up the overall design into multiple components.

The first section regards the technical backend, in which we detail the dependencies, files, and functions of the application. This can be seen in Figures 4 and 5, in which we illustrate what makes up the application and what each part does.

The second section regards the UI and what the user should expect to see during the use of the application. This is illustrated in many ways, including Figure 6 where we show off the UI and its features, Figures 7 and 8 with how the application works based on user input, and our extensive table of tests regarding the functionality of the program.

4.1: Application Technical Structure

Our GUI imports three different objects, the OS library, the Tkinter library, and our PaperTrail driver file. Our driver file imports many other libraries and our PaperTrail document file to encrypt and decrypt based on our user's request. Our document file imports some other libraries, but this time to generate the PDFs and other background work required to allow the user to access their files. The GUI keeps everything simple so that the user can just get what they want done without having to deal with the extra work.

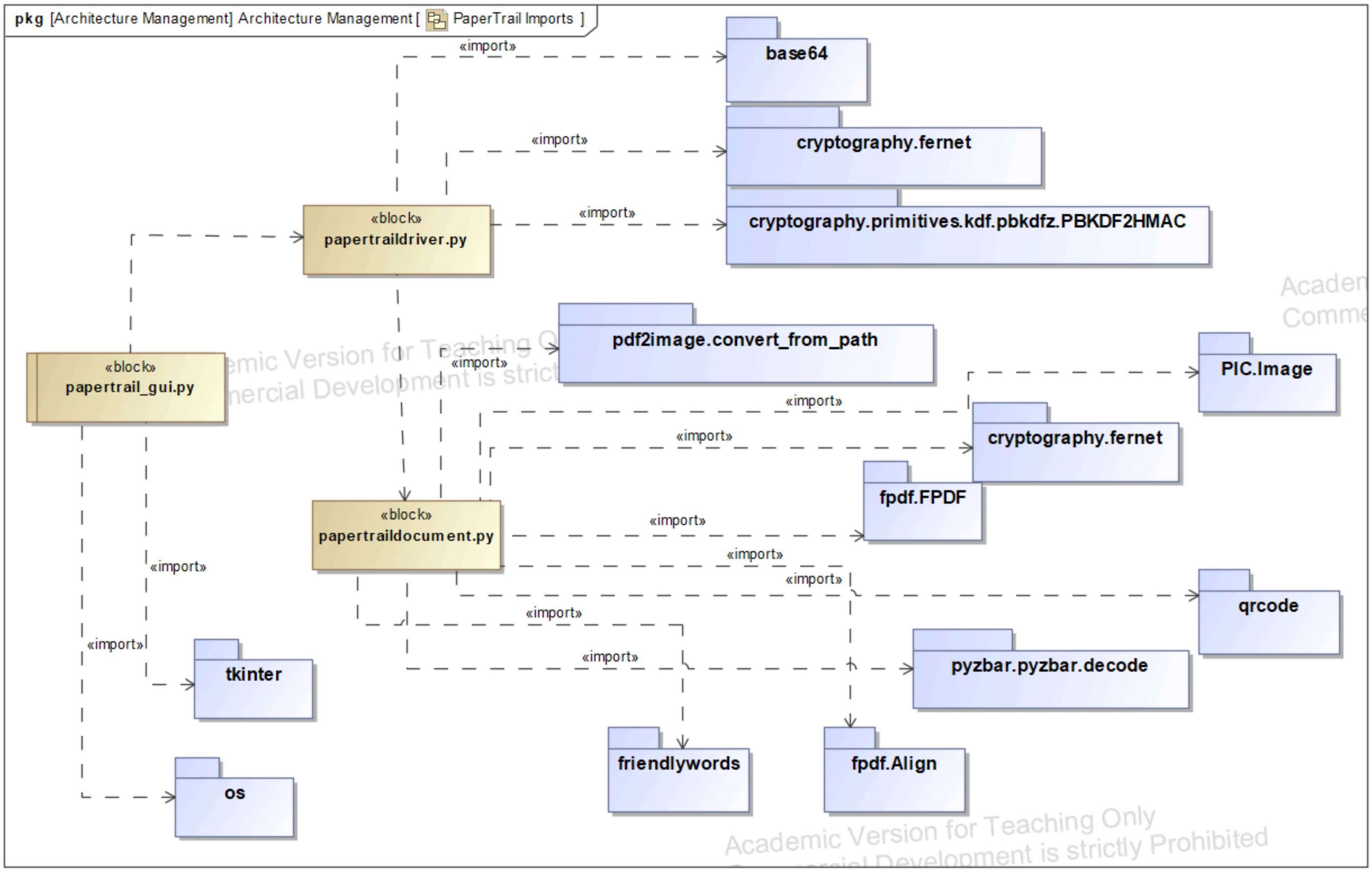


Figure 4: Package Diagram

The body of our application is broken into three main blocks based on the three files we use to perform the required tasks. As this application was designed with ease of use in mind, each of the blocks simply flow from one to another to process all of the different aspects required to provide the user with the safety they desire. Within each block is a list of the important functions that occur within, which helps to further illustrate what each block does and how they come together to create the finished file.

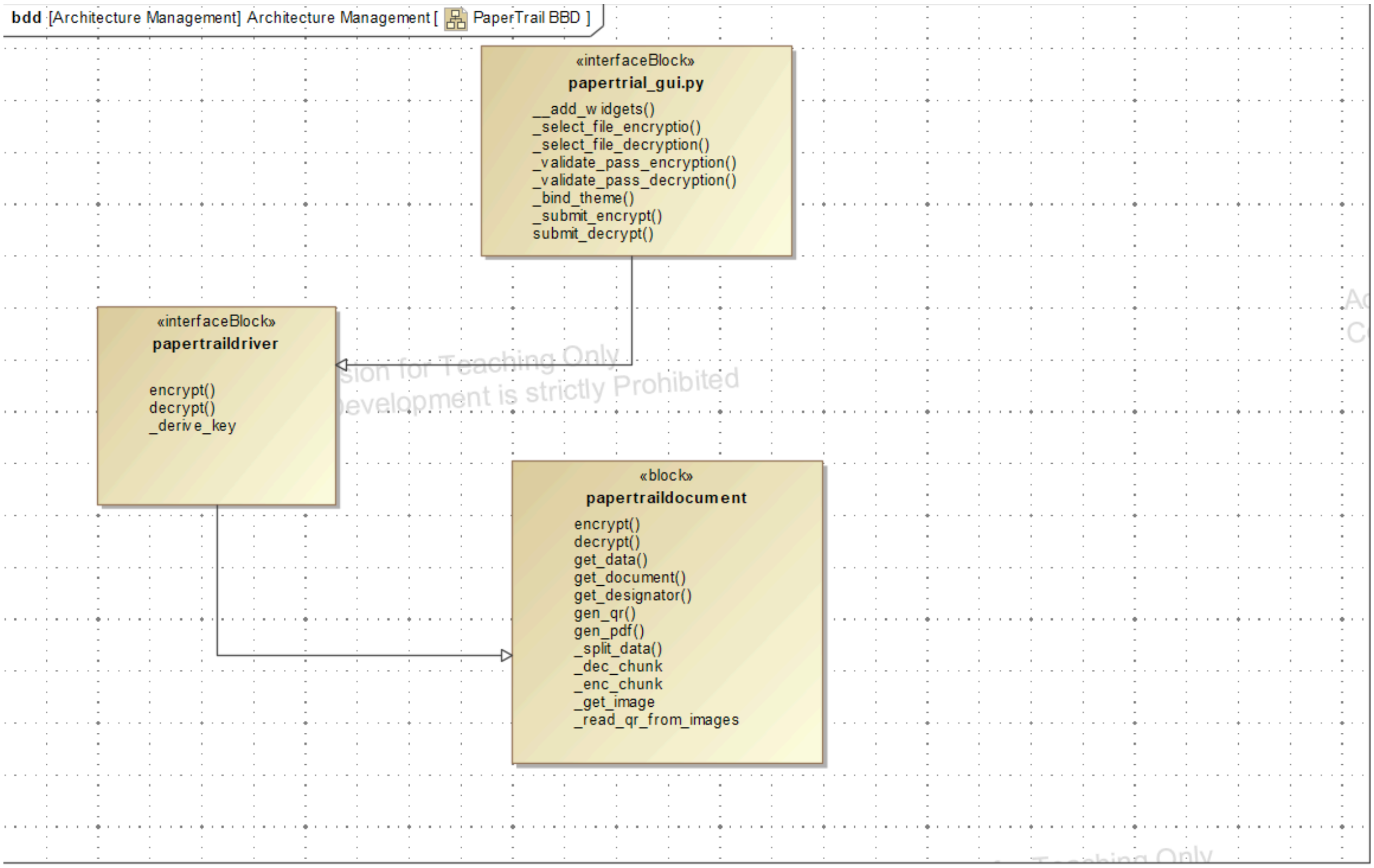
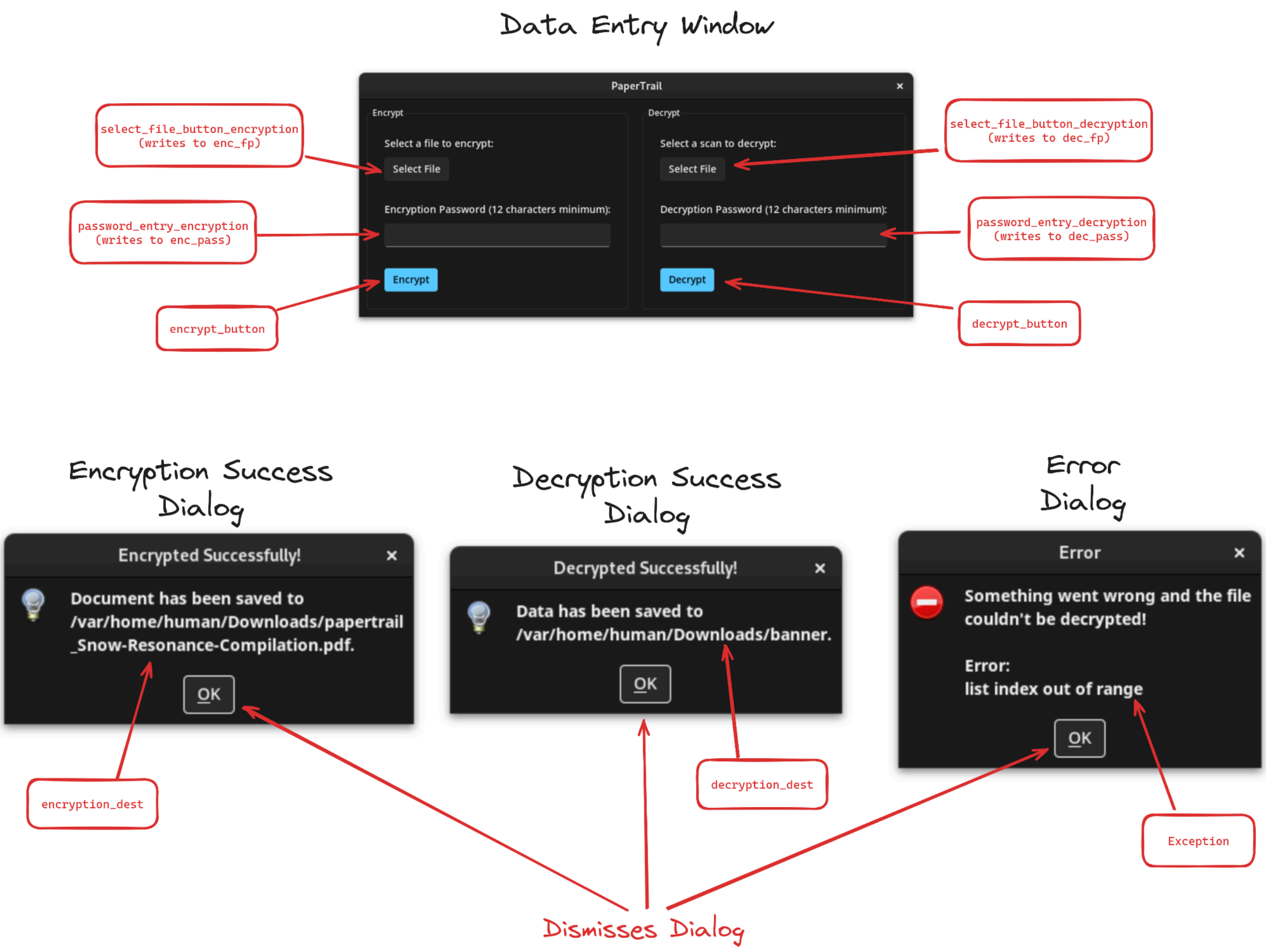


Figure 5: Block Definition Diagram

4.2: User Interface Behavior

We have created a labeled version of our UI that shows the main screen as well as some of the smaller pop-up windows the user may encounter. On the main screen, we have buttons that allow the user to select a file, either for encryption or decryption. Once the user has selected a file using one of the provided buttons, there is a text box below used to type in the password that will be used for encryption or decryption respectively. Once a file has been chosen and a password has been entered, there are buttons that the user can click to simply encrypt or decrypt the provided file using the provided password.

Below the main window are three different pop-up windows. The first two on the left illustrate successful encryption and decryption respectively, while the rightmost one shows the error screen. The error screen is the same for encryption and decryption, with the only difference being the text regarding whether the file could not be encrypted or decrypted. All three of these windows have an “OK” button used to close the window as well as an “X” in the top right corner.



Data Entry Window
PaperTrail
Encrypt
Select a file to encrypt:
Select File
Encryption Password (12 characters minimum):
Encrypt
Decrypt
Select a scan to decrypt:
Select File
Decryption Password (12 characters minimum):
Decrypt
select_file_button_encryption (writes to enc_fp)
password_entry_encryption (writes to enc_pass)
encrypt_button
select_file_button_decryption (writes to dec_fp)
password_entry_decryption (writes to dec_pass)
decrypt_button
Encryption Success Dialog
Decryption Success Dialog
Error Dialog
Encrypted Successfully!
OK
Decrypted Successfully!
OK
Error
OK
encryption_dest
decryption_dest
Exception
Dismisses Dialog

Document has been saved to /var/home/human/Downloads/papertrail_Snow-Resonance-Compilation.pdf.

Data has been saved to /var/home/human/Downloads/banner.

Something went wrong and the file couldn't be decrypted!

Error:
list index out of range

Figure 6: Labelled User Interface

Illustrated below are the different states that the application goes through during normal operations. First, the application is launched and the main window is opened. This starts the Data Entry Mode, in which data can be input by a user. This state will remain until a call to encrypt or decrypt is made. If a call to encrypt a file is made, the application moves into Encryption Mode and performs the relevant tasks within. Depending on the outcome of the encryption, a message either stating the success or failure of the encryption will be provided as the application moves back into the Data Entry Mode. If a call to decrypt is made instead, a very similar process occurs as encryption, in which the Decryption Mode will start and all relevant tasks within are done. Again, depending on the outcome of the decryption, a message either stating the success or failure of the decryption will be given as the application moves back into the Data Entry Mode. This can occur however many times the user wants until the application is closed, in which the application no longer can do anything and will shut down.

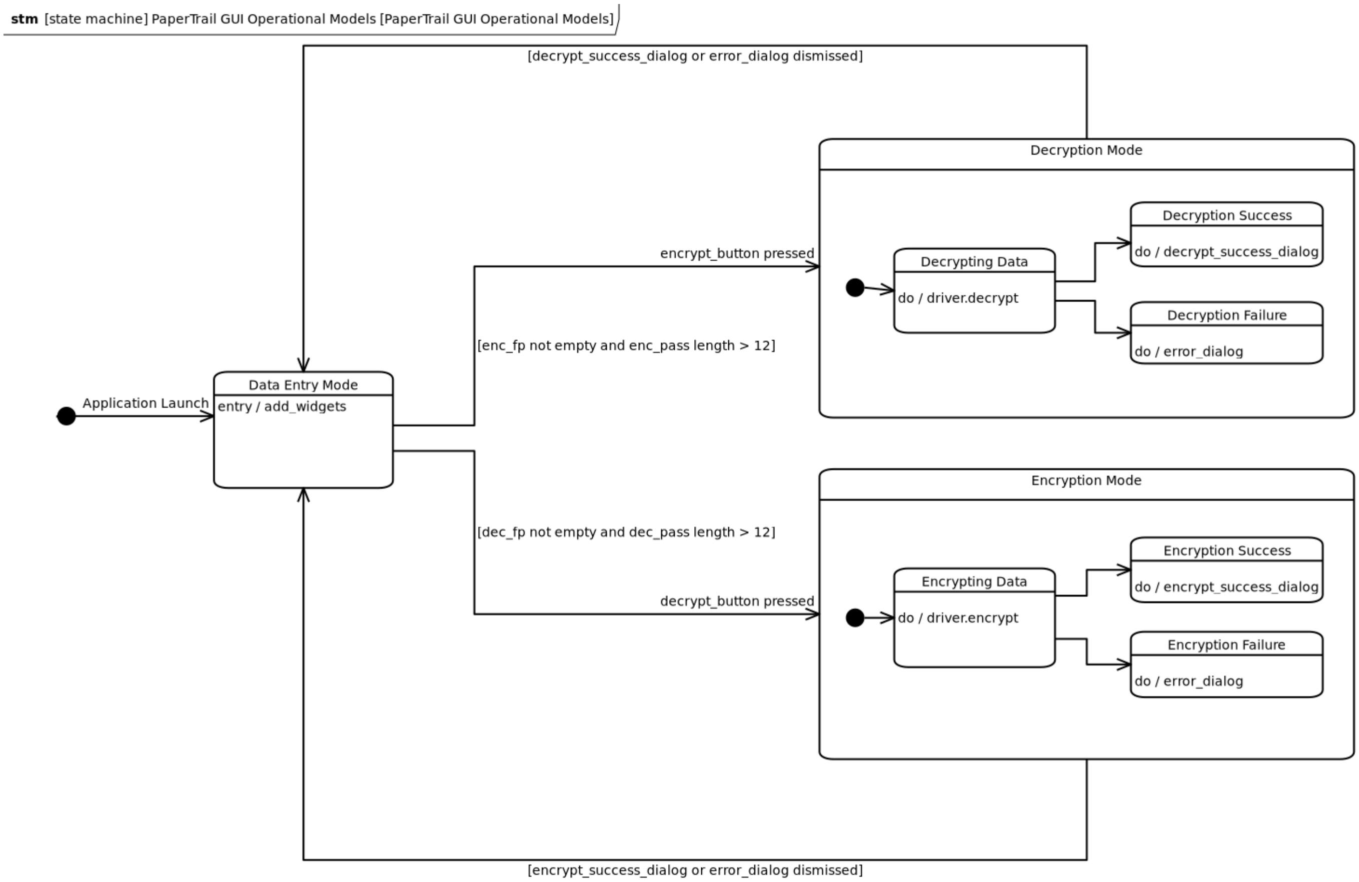
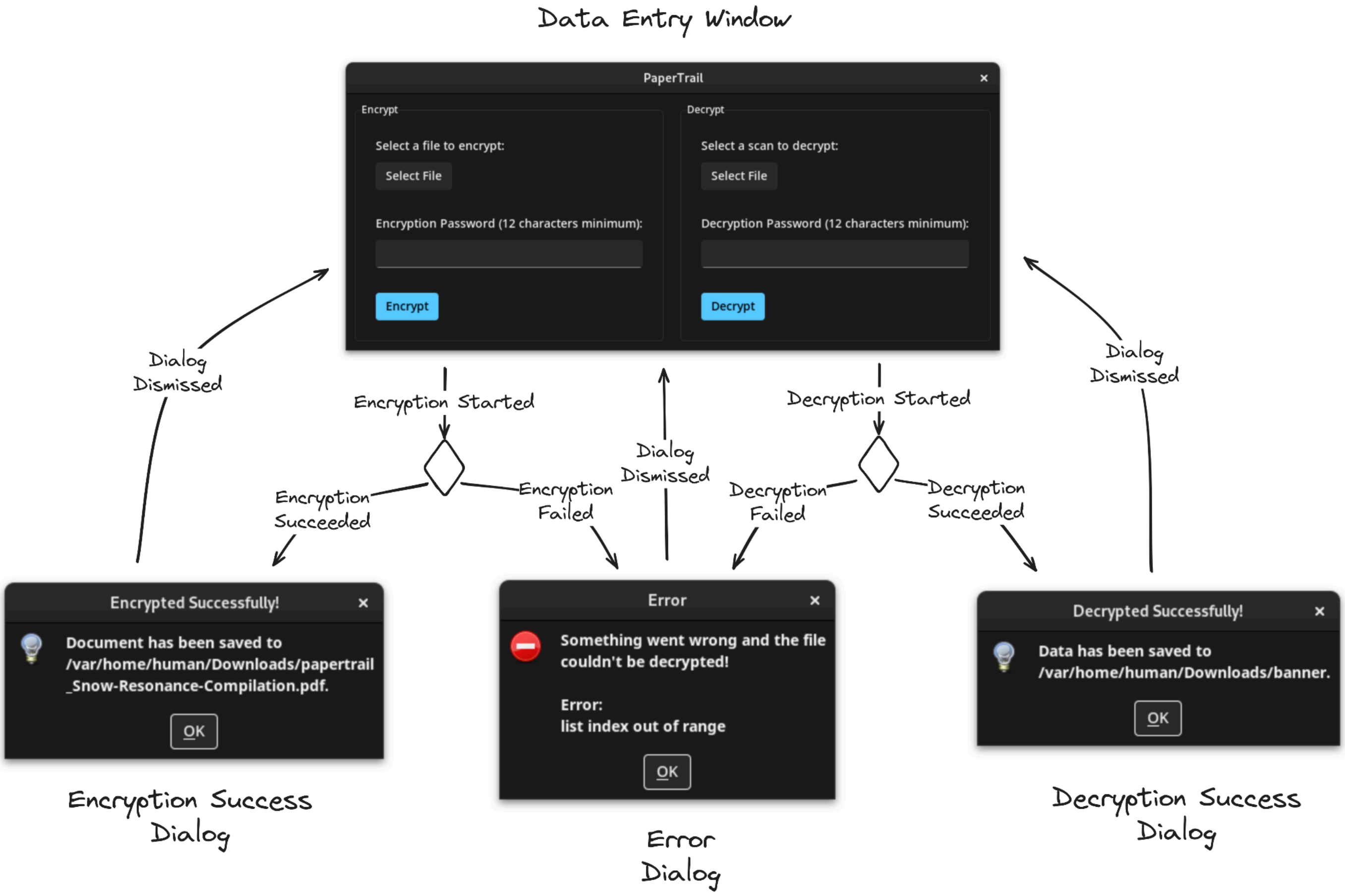

stm [state machine] PaperTrail GUI Operational Models [PaperTrail GUI Operational Models]
[decrypt_success_dialog or error_dialog dismissed]
Decryption Mode
Decryption Success
do / decrypt_success_dialog
encrypt_button pressed
Decrypting Data
do / driver.decrypt
Decryption Failure
do / error_dialog
[enc_fp not empty and enc_pass length > 12]
Application Launch
Data Entry Mode
entry / add_widgets
Encryption Mode
[dec_fp not empty and dec_pass length > 12]
Encryption Success
Encrypting Data
do / encrypt_success_dialog
decrypt_button pressed
do / driver.encrypt
Encryption Failure
do / error_dialog
[encrypt_success_dialog or error_dialog dismissed]

Figure 7: State Machine Diagram

Below is a flow chart of what a user may expect to see as they use the application. It is very similar in description to the previous diagrams, as the screens the user sees are the different states and messages displayed after returning to the Data Entry Mode.



Document has been saved to /var/home/human/Downloads/papertrail_Snow-Resonance-Compilation.pdf.

Something went wrong and the file couldn't be decrypted!

Error:
list index out of range

Data has been saved to /var/home/human/Downloads/banner.

Figure 8: Behavior Flow Chart

4.3: Testing Procedure and Results

We created a vigorous and extensive set of tests to ensure that our program did everything we expected it to do. While not all tests were successful, these tests were based on previous requirements that were no longer to be upheld due to technical limitations, such as the ability to decrypt a file using an incorrect password and have it generate a file. Tests were created through a thorough investigation of the code, diagrams, and previous documentation to ensure that all functions, requirements, and features were working as intended. The tests were carried out on multiple different devices to ensure all features worked across operating systems. Adjustments were made after the tests were completed to ensure accuracy in what we are providing as a service, as we did not want to mislead anyone with outdated claims.

State	**Function**	**Description of test**	**Expected Outcomes (before execution)**	**Actual Outcomes (after execution)**	**Comments (if applicable)**	**Test Results (pass or fail)**
Data Input Mode	Select File	User is able to select a file and have it be the one used	Can select a file and the selected file is used	Successfully able to select file and have it be used	Works fine	**PASS**
Data Input Mode	Select File	User can select any sort of standard binary or ASCII file	Any standard binary or ASCII file can be selected and used	Couldn't find a filetype that would cause any problems	Works fine	**PASS**
Data Input Mode	Type Password	User can type in the box any amount of characters equal to or above 12.	Allows for any combination of characters to be a password so long as they are equal to or above 12 characters total	Takes any characters in a string 12 or longer and uses it as a password	Works fine	**PASS**
Data Input Mode	Type Password	Generates an error if the password is less than 12 characters long	Gives an error and makes the user put in a longer password	Does give the error we were expecting	Works fine	**PASS**
Destination Selection	File location	User can select the location of where they want their file to go and the file actually goes there	User can locate a file location and the file will go there	File goes to the chosen file location	Works fine	**PASS**
Destination Selection	Name file	User can choose to change the generated name to any filename so long as it fits their OS's file syntax	File generated should be PDF after the user changes the name	It works fine, however. The generated name saved as a PDF file but when I changed the name of the file it saved as a plain file without any specific type doc,pdf..etc		**FAIL**

Encryption / Decryption	Encryption	User chosen file is actually encrypted and unable to be converted into plaintext without use of decryption	Encrypted file when open using a text editor is not human-readable	Gives you a bunch of weird symbols that are not human-readable (did actually encrypt the file)	Great, works fine	**PASS**
Encryption / Decryption	Decryption	User chosen file can be decrypted **regardless of password** into human-readable text	Password given for decryption should decrypt the file into human-readable text	Decryption does not work without typing the same password as was used for encryption	Generates an error message	**FAIL**
Encryption / Decryption	Decryption	User-chosen file can be converted back into original file when given the correct password	When given the same password as encryption for decryption, the information should be the same after decryption as it was before encryption	Gives the same information after decryption as before encryption	Decrypted file is a plain file choosing to open with notepad the description looks different than the actual encrypted file However with Microsoft Word it works fine.	**PASS**
Encryption / Decryption	Decryption	User-chosen file converted into human-readable file but NOT the original when given incorrect password (generates text but not what was given by user)	Decryption should generate a new file that is human-readable but has information different from the original file before encryption	Does not let the decryption occur as the passwords used for encryption and decryption are NOT the same.	Gives an error that decryption could not happen	**FAIL**
Encryp	Decrypti	Error is	Should give an error	Gave an error when		**PASS**

tion / Decryption	on	generated when incorrect file type is chosen (non-PDF)	when a non-PDF file is chosen	a non-PDF file was chosen		
Product Display Mode	Display decrypted documents	Pops up screen for user that shows the decrypted document after decryption occurs	Should pop up a screen that shows the file location of the decrypted document	Gives a message saying that decryption occurred successfully and where the file is located	Message shows that decryption has occurred along with the file path	**PASS**
Product Display Mode	Display QR code	Pops up screen for user that shows the QR code generated after encryption occurs	Should pop up a screen with a QR code on it after encryption	Gives a message saying that encryption occurred successfully and where the file is located	Message shows that encryption was successful along with file path for pdf after it has been saved	**PASS**
General behaviors	Lack of crashes	Application does not crash during the course of testing	Application should not crash	Application does not crash	I experienced no crashes	**PASS**
General behaviors	Application installation	Users can download the Papertrail application without any errors caused by the program (only user-side errors like storage)	No errors should occur during the download	No errors occurred during download	Download was fast and easy with no errors	**PASS**
General behaviors	Application able to run	User can open and close the application without any errors (can double-click to open and X button in top	Should allow the user to close the application without any problems	Doesn’t cause any problems when closing the application	Functions as expected	**PASS**

		right closes application)				

5: Conclusion

In the end, our team set out to create a secure way for people to transform digital information into a physical medium to ensure its safety, and that is what we have done. Over the course of development, we have learned many things. Not only all of the technical knowledge required to implement such encryption and decryption schemes, but also how to polish and market such a product for practical use. This has been a great learning opportunity for all members involved, as many of us learned a variety of skills during the course of development.

We have succeeded in creating a proper encryption and decryption application that can be used to securely store information over a physical medium, such as a piece of paper. While not all of the features we wished for could be implemented, this is something that can be iterated on in future versions of the program as we will get to later.

5.1: Comparison to Competition

Our application Papertrail has succeeded in surpassing our competition Paperback in a variety of ways. First, due to the nature of information being encrypted using Papetrail, we do not require multiple key parts to be generated. This reduces the likelihood that vital information will be lost, as the user only has to worry about one document and password instead of many. Second, Papertrail uses a Fernet encryption and decryption scheme, which is known for being very strong. Paperback on the other hand does not, so if malicious actors assembled all key parts, your information would be compromised. Papertrail also has a very easy-to-use UI, which allows for many more people to utilize our application over Paperback.

5.2: Operational Challenges

During the development of this application, we did run into a few challenges. First, some of the features we wanted to implement couldn't be implemented. For example, due to the nature of Fernet encryption and decryption, there is no way to have any key be used for decryption. This is because Fernet uses symmetric key encryption, so the key has to be the same for encryption and decryption. Utilizing an asymmetric key scheme would allow for greater protection, as it would mitigate a vulnerability within the program that allows malicious actors to gain access to your information via combination testing (brute force, dictionary, etc). Second, the large degree in technical abilities among the group led to some confusion at times regarding the functionality and behavior of the application, which led to misunderstanding at times. Much of this was remedied through open communication both at meetings and over online communications, but still was a hurdle for the team as a whole. Third, our team utilized a variety of outside libraries that we were unfamiliar with, so there was a slower start to development. As we had to understand the complexities of our dependencies before we could properly implement them, this led to some timeline problems that were thankfully addressed.

5.3: Concepts for Future Improvements

We would like greater security and data density in future iterations of the application. On the security side, doing a rewrite that would allow any password to decrypt the file would remove the vulnerability previously mentioned. Of course, this incorrect decryption password would not give the original file's contents, but instead random garbage that would be meaningless to an attacker. On the data density side, allowing multiple QR codes to be generated would allow for greater data density. Currently, we can hold up to 2143 bytes per page due to the limitations of v40 QR codes, but allowing for multiple QR codes to be generated would allow for greater information storage capabilities. So long as you had enough paper, you could hypothetically store whatever you wanted securely and easily.

6: References

(include anything used in the writing of this report that was NOT made by me (connor))

6.1: Appendix (Code)

Files in ZIP folder:

- PaperTrail-v0.1.0-win_x64.exe: Windows EXE of papertrail-gui with all dependencies bundled in
- PaperTrail-v0.1.0-linux_x64: Linux binary of papertrail-gui, bundles all Python dependencies but requires poppler-utils and zbar/libzbar to be installed on the system
- papertrail-gui/: Package containing the frontend code of the project
  - src/papertrail-gui/papertrail-gui.py: papertrail-gui module, contains the frontend written in tkinter
  - resources/: Directory with resources that the application depends on, namely poppler and zbar dlls to be included in the pyinstaller generated exe
  - *.spec: PyInstaller spec files to build the binaries for the platform in the filename. The PyInstaller executables can be generated by installing the dependencies in requirements.txt and then running pyinstaller <platform>.spec
  - requirements.txt: Contains the dependencies for the GUI application, including importing the papertrail package
- papertrail/: Package containing the backend code of the project
  - src/papertrail/papertraildocument.py: papertraildocument module, contains the

PaperTrailDocument class that holds most program logic, including encryption, pdf generation, and decryption

- src/papertrail/papertraildriver.py: papertraildriver module, contains the PaperTrailDriver class with functions that abstract usage of the PaperTrailDocument class, allowing that class to be more modular and extensible, this is the module that papertrail-gui imports
- pyproject.toml: packaging configuration file that holds package information and dependencies, allows the package to be installed with pip from the GitHub repository

ZIP folder link