E.P.C.S

Encrypted Personal Cloud Storage

Encrypted Personal Cloud Storage (EPCS)
Dominik Danek
Center number: 17407 - Hitchin Boys School
Candidate number: 6212

Notes:

Sources:

Source num	Source description		Link
1	A dive into encryption		https://2594413632-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2F-MHfP44c-IQ_Syc_GT2Z%2Fuploads%2Fw5EKHN4VXIbR4g6PiVCC%2FWaynflete.pdf?alt=media&token=01afb7a5-1711-40bf-9515-d65c79781410
2	RSA encryption explanation		https://www.youtube.com/watch?v=Pq8gNbvfaoM
3	Asymmetric encryption explained		https://www.youtube.com/watch?v=AQDCe585Lnc
4	Tutorial video that showcases the code that makes an encrypted copy of a file (in C)		https://www.youtube.com/watch?v=GNzeSTZjziI
5	Setting up the pi		https://www.raspberrypi.com/tutorials/nas-box-raspberry-pi-tutorial/
6	Flashing the OS onto the sd card		https://www.raspberrypi.com/documentation/computers/getting-started.html#installing-the-operating-system

7	Partition a harddrive		https://www.linkedin.com/pulse/automating-lvm-partition-using-python-script-aashish-vivek-bhat
8	Two pi's communicating over the same network using python		https://www.youtube.com/watch?v=T3XOac7QmAI
9	How to remotely access a pi		https://raspberrypi.stackexchange.com/questions/105080/how-to-remote-to-raspberry-pi-from-outside-local-network
10	DMZ explanations		https://www.reddit.com/r/HomeNetworking/comments/15bsy64/dmz_what_is_it_used_for/?rdt=63134
11	3x3 HILL CIPHER EXAMPLE		https://www.youtube.com/watch?v=qU1kcpuJXaw&list=WL&index=3
12	Rfc 959, FTP full description		https://datatracker.ietf.org/doc/html/rfc959

Table of contents:

A-level technical skills list:

- ☑ Uses a relational database management system (RDBMS)
- ☑ Uses matrix multiplication to multiply two matrices together
- ☑ Create inverse matrices that implement a mod function

- ☑ Is able to read file contents, apply the encryption key and generate an encrypted file
- ☑ Use the random library to generate new encryption keys
- ☑ Is able to encrypt numbers and special characters
- ☑ Upload and store encryption keys and the files in separate databases
- ☑ Implement port forwarding in my router using a DMZ
- ☑ Implement paramiko library and establish a connection between server and user DTP
- ☑ Is able to send files over the SFTP protocol
- ☑ Is able to download files from the NAS and send them over the SFTP protocol
- ☑ Build a Web server hosted on my own machine using a virtual environment
- ☑ Use Flask to implement the data flow diagrams
- ☑ Validate file type to only specific formats
- ☑ Be able to find the newest uploaded file to a file folder
- ☑ read/write files to binary so that they can be encrypted
- ☑ Convert file from binary to hexadecimal
- ☑ Convert hexadecimal files back to their original file format
- ☑ Set up a SFTP server on my NAS
- ☑ Use XOR gates for vernam encryption

Analysis:

For my NEA, I aim to create a Raspberry pi NAS (Network Attached Storage). It is a standalone storage drive that any device on the local area network can use to share and store files. One of its limitations is that you need to be situated within a close proximity to be able to upload files. However, what if you wanted to access and upload files remotely? Currently, not many suitable solutions exist. The best option is to use a 3rd party cloud service. This forces you to give your private data to large corporations which are regularly prone to attacks and fail at keeping your data safe. The solution would be a compact encrypted server, allowing you to automatically backup important documents, files and photos from remote locations.

What I aim to achieve with my project and the problems I want to solve:

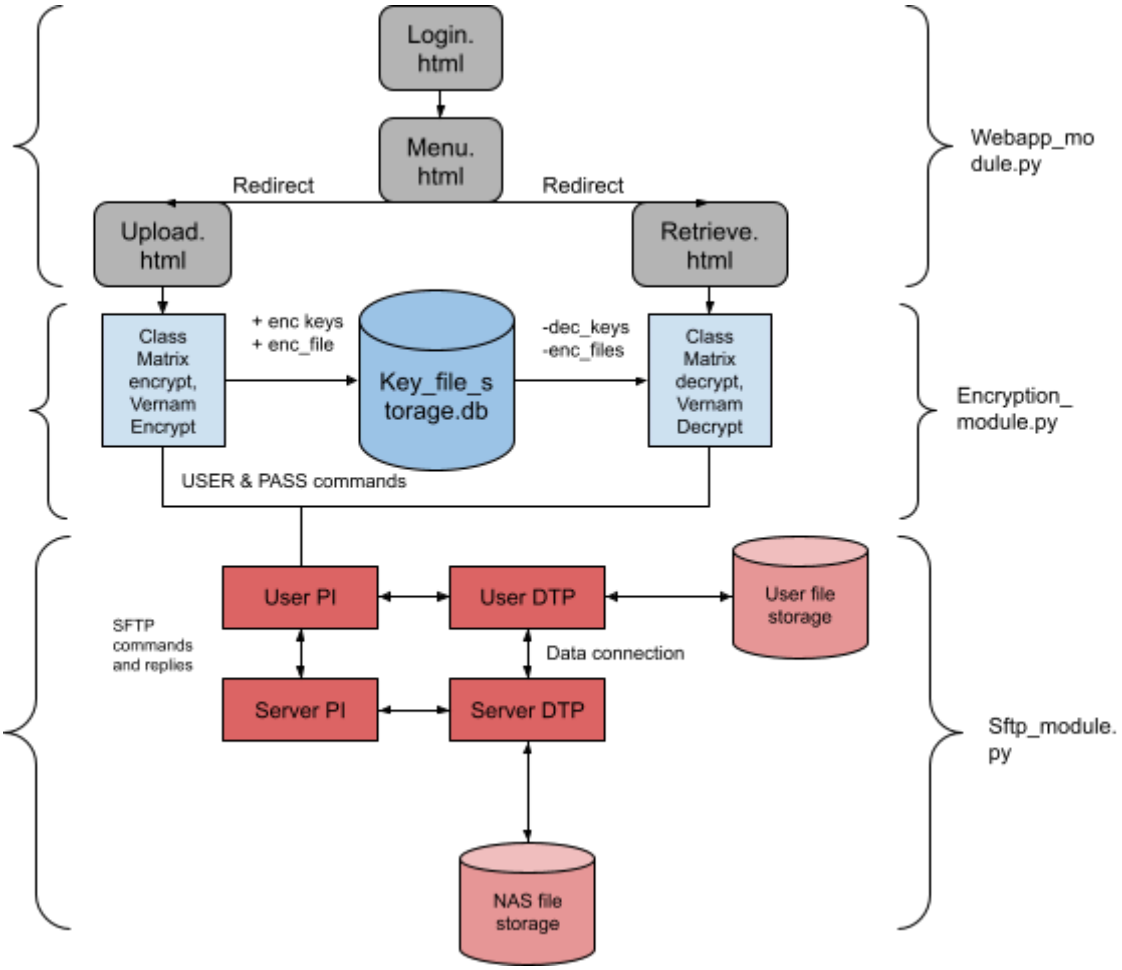
My project is aimed to be an alternative to 3rd party cloud services. The end-user would be someone that is seeking a more private, secure, and overall cheaper solution that can be accessed only by them. The security aspect should mainly be supported by a unique form of encryption that has not been created yet . Moreover, the Cloud service should also be accessible by the user from anywhere with an internet connection, therefore a website will act as the connecting piece. This will provide an easy form of access to the files, allowing the user to upload files with ease. My project is intended for a single user such as myself hence I will be criticising my own project. However throughout my analysis and design I will mention means of modifying certain aspects of my projects to accommodate for multiple users.

Summary of the key features of my project:

1. Has multiple encryption types that each provide unique benefits. The files should be stored in an encrypted state.
2. The user should be able to upload and retrieve files using SFTP. Be able to track the progress using a progress bar
3. A private web application serves as a graphical user interface between the user and the NAS. It should verify the users' identity through a login procedure and have separate pages for uploads and retrievals.
4. A database stores the keys and metadata of the file.

Here is a visual representation of how I want to layout my NEA to be structured:

Simplified Structure:



Remote access of the Pi:

To achieve remote access to a Raspberry Pi using port forwarding, I need to configure my network router to allow external connections to reach the pi. First, I need to connect my pi to my local network and assign it a local IP address , either by setting it manually on the Pi or reserving it in my router's DHCP settings. This step is usually done automatically when any device connects to that network anyway. Next, enable the desired service on the Pi, such as SSH for remote command-line access or a web server for hosting a website. Once the service is running, I will login to my router's admin interface (usually accessible via a web browser) and locate the port forwarding settings. Create a new port forwarding rule that maps an external port (e.g., 2222) to the Raspberry Pi's internal IP address and the port of the service (e.g., port 22 for

SSH). This means any incoming traffic to the router on the external port will be forwarded to the Pi. For security, avoid using default ports like 22 for SSH to reduce the risk of attacks.
While all my devices attached to my Router are in the same subnet and will be able to directly communicate to each other, they are not directly reachable from the Internet. From a security standpoint, that is desirable. This is however exactly what I want. (To clarify, there will be security measures in place such as a user login to determine if a user has access).

Since I am creating a Cloud storage device I will want to connect a Foreign public IP with the Local and static IP of my NAS. This will be done using port forwarding. I will need to configure my router to be able to receive information at a given port. The program I plan on creating will send the file to that specified port of the public IP address of my router and then to the private IP of my NAS. I can obtain its private Ip using the ping command in SSH. If I want to make this port available to the public I need to open my router's web administration at https://192.168.0.1 and create a port mapping or NAT (Network Address Translation) rule for the file sharing service of my NAS. This will map external requests from the internet at my specified port to the private ip address of my local router (192.168.A.B). This now makes it possible to connect the FTP server with an external IP (192.168.X.Y)

The Protocol I will be using to do this will be SFTP (secure file transfer protocol). This allows me to filter incoming traffic to only be files. It also makes it more efficient and faster in terms of data transmission times. I will also need to account for the fact that the user's device IP will change, while the IP address of the NAS will remain static. Therefore I will need a way to track the devices' changing address.

Additionally, I can check if I was successful in making the file sharing service public by running the command `sudo netstat -tulpen` . This will show me all the services that are active on that network and their source Ip's.

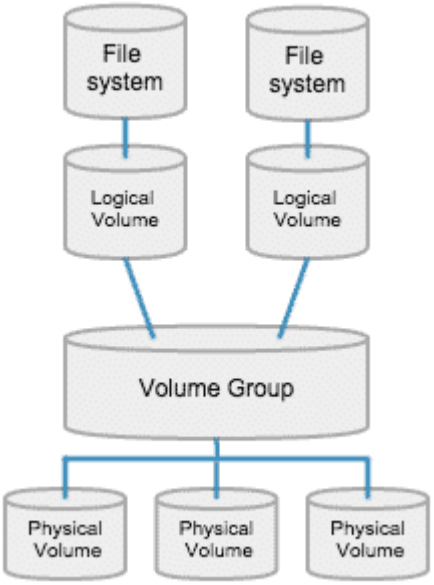
Website:

The website I intend to build will act as an interface between the intended user and the storage device. This will enable users to remove and add new files to a specific folder/section of the HardDrive. The user will only be able to read sections that correspond directly to their access level. This access level will be predetermined in the full user database. Which will restrict which sections they can modify and see. There will also need to be a file upload system on the website which will serve as a gateway to the NAS.

The user must also be able to organise the state of the file storage on the Hard Drive. This means being able to add, delete, name partitions. This would not be done through the website, instead It should be done through means such as SSH when the user is on the same network as the raspberry pi NAS. Additionally they must be able to decrypt the file to see what is contained in it. I have a few options for organising the structure of the drive if there were to be multiple users. One option is LVM (Logical Volume Management) which provides some benefits to traditional partitioning. Such as improved flexibility, meaning you can resize the partitions.

The goal of LVM is to facilitate managing the conflicting storage needs of multiple end users. The most common use case for LVM is to mark physical devices as PVs (Physical Volume) and then use these devices to create a VG pool from which LVs (logical Volumes) can be allocated to increase disk space. To put it simply it allows multiple users to put files on the disk at the same time

Using the volume management approach, the administrator is not required to allocate all disk storage space at initial setup. Some can be held in reserve for later allocation. The system admin can use LVM to segment logically similar data or combine partitions, increasing throughput and making it simpler to resize and move storage volumes as needed.



Additionally I will need a digital space where the encrypting and decrypting will take place. There are a few options for that, one of which is encrypting the files straight on the raspberry pi. This makes it convenient since I don't need to create a new infrastructure to handle the instructions, such as an additional server; separate for handling encryption requests. However this does mean that the files will not be encrypted during transmission phase, leaving them vulnerable. To handle this I could use an SFTP server that can securely transfer files over the internet. On the other hand the users' machine would be more than suitable for this application so I will opt for that.

What is an SFTP server?:

A secure file transfer protocol (SFTP) is a transmission protocol that uses secure shell encryption to be able to safely transfer files over the internet. It works by opening two channels, one connection is designated for the commands and replies between the two clients, while the second is used for the data transfer. The initial connection is supported and initiated by a FTP client software which is what I will be attempting to program. The program should resemble something like Filezilla which is a popular File transfer program. It enables the exchange of data between a server and a device. The server in this case will be my NAS which will be accessible through the website that a user

can access as a client. The SFTP server will be located on the website hosting servers which will allow the person to be able to interact with it through the web application.
Unlike traditional FTP, which transmits data in plaintext, SFTP encrypts all communications, ensuring confidentiality and integrity. It operates over port 22, the same as SSH, and requires authentication through passwords, SSH keys, or both. Once authenticated, the client gains access based on predefined permissions, allowing secure file operations such as uploading, downloading, deleting, and modifying files. The encryption ensures data is not intercepted or tampered with during transmission, making SFTP a preferred choice for handling sensitive data. Another advantage is its ability to support resume capabilities, allowing interrupted transfers to be continued from where they left off. Additionally, SFTP servers enable fine-grained access control, allowing administrators to restrict users to specific directories and limit their permissions. However, there are some drawbacks. SFTP can be more complex to set up than standard FTP, requiring SSH key management and proper firewall configurations. It also consumes more processing power due to encryption, which may impact performance on lower-end systems. Additionally, while SFTP is secure, incorrect permissions or misconfigurations can still lead to unauthorized access.

Encryption:

History:

One of the earliest known forms of cryptography is the Caesar Cipher, supposedly used by Julius Caesar in his private messages. The aim of the cipher is simple - to be unreadable to anyone except the intended recipient. First, place two lists of all alphabet characters below one another. Now we shift the bottom list by a certain factor. This means we move the letters to the left, with any characters moved past the right end being looped back to the right. Let's say we pick a shift of 7 (which we can call the key):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z H I J K L M N O P
Q R S T U V W X Y Z A B C D E F G

Now we can choose a message, let's say ATTACK. What we would do is replace each character in ATTACK with the corresponding letter in the bottom list. For example, A would be replaced by H. If we continue through the word ATTACK, the encrypted version of the text is HAAHJR. But the Caesar Cipher has several weaknesses. The first is, logically, that there are only a total of 25 unique ciphertexts (encrypted versions of the plaintext, the original message) we can possibly generate as, after we pass a shift of 25, we essentially just start again. This makes it susceptible to a brute-force attack, where we compute all possible shifts from 0 to 25 and analyse them to find the original message.

Development of my own form of Encryption.

Encryption will be a paramount feature in the design of my NAS. This is because personal cloud storage devices do already exist however they do not have a way of encrypting the data while it is being stored. Therefore by Encrypting it using my own algorithm, I will improve security significantly as the data is no longer vulnerable.

As stated previously, one of the simplest forms of encryption is the caesar cipher and works as a good example to show how you can convert from plaintext to cipher text using a key. It does this using a shift in the alphabet and mapping each letter to a new one, creating ciphertext. However, this form of encryption is easily brute forced as there are only 25 possible shifts you can do.

On the other end of the spectrum, something like a Vernam cipher is considered to be a perfect form of encryption. This is because it satisfies the Shannon Theorem which states that: (1). the key length must at least be the length of the entire message, (2) That the key is disposed of after it is used , and (3) the key is truly random. These rules ensure that there are no patterns in the ciphertext and that cryptanalysis techniques such as frequency analysis prove to be useless.

In order to make the data on my NAS as safe as possible I hope to create my own form of encryption. This makes it significantly safer, because it will not be an existing cipher which has recognisable patterns. This ensures that even if an attacker had the key and the ciphertext it would still be unbreakable, since they do not know the algorithm that is required for the conversion.

For my own Algorithm I aim to follow the Shannon Theorem and I am planning to combine asymmetric encryption with the laws used by the Vernam cipher. The Vernam cipher uses XOR gates which I do not plan to implement. I am however intrigued by the concept of using matrix multiplication to encrypt messages. This would mean I need to construct two matrices of equal size, one containing the key and one containing the plaintext.

$$Plaintext_{Matrix}\,(\alpha) * Cipherkey_{Matrix}\;\;(\beta) = CipherText_{Matrix}(\alpha\beta)$$

It would work in a way that each letter would be replaced by its respective numbered position in the alphabet. Here is a visual representation of the letter mapping.(E.g letter P would correspond to 15)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	…
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	…

However in order for this to work both the Cipherkey and the Plaintext would have to be split into smaller chunks such as 2 x 2 matrices ,hence into groups of 4 letters. This would make it feasible for encryption.
Decrypting the CipherText would require making an inverse matrix of each 2 x 2 CipherKey matrix and multiplying it by the CipherText to get the Plaintext. The proof would look like this:

$$\alpha * \beta = \alpha\beta \qquad \Leftarrow Encryption$$

$$\alpha\beta \qquad \Leftarrow CipherText\ Matrix$$

$$\alpha\beta^{-1}\beta \qquad \Leftarrow$$

Multiply by the inverse matrix of the CipherKey

$$\alpha I \qquad \Leftarrow$$

This would give the Plaintext matrix as the remainder

Unfortunately I am limited to the matrix arrangements that I can use. For example something like a (2 x 3) matrix would not be mathematically possible because it cannot have an inverse matrix. This restricts the matrices to only have the same row and column count e.g (n x n). I will most likely settle for a 3x3 matrix as it is significantly more complex to get the inverse of the matrix hence harder to break.

Security:

The system security of my public website as well as the physical security of my storage will need to be very resistant to exploits or attacks. This is because I am going to be opening ports in my firewall in order to enable the transmission of data. This could make my entire network and all its connected devices vulnerable to attacks. As a result I need to ensure the validity of incoming requests or packets that come through both the NAS router and the LAN network router of my device.

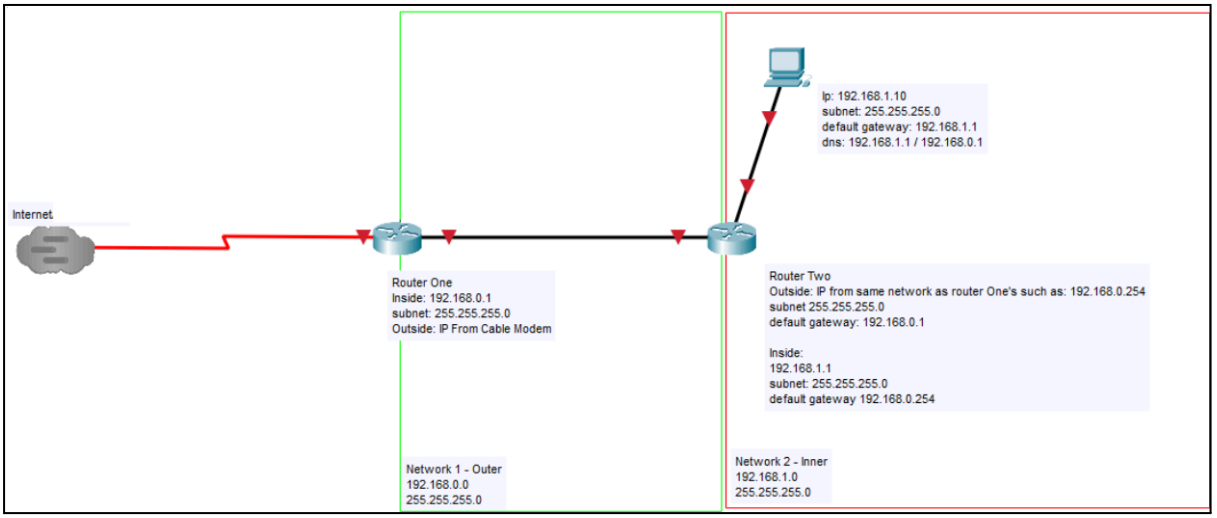
One idea that I had to improve the security was to use an obscure domain name. This in theory would make it hard to find by accident on the internet. It may actually be suitable for home networks that are generally not the main targets of attacks. However something like a large corporation would need more than just a hidden domain name as their main security method against attacks. This is because exploitation methods have been developed to search for obscure domain names because they tend to be vulnerable.

This is why the website will need a user login which will have a database of registered users. This will verify their identity and filter out any unwanted users from gaining access to the files. Since I do not want to put my network at

risk I will be using Google's OAuth Authentication server which grants an access token to the valid users.
Additionally I could incorporate a Vpn server on the raspberry pi which would prevent me from having to open more ports for any new services I will be adding. Therefore only the connections coming into the vpn server have to be scanned. This is useful because it narrows down the surface of attacks to just be on the vpn server, which will make it easier to identify intrusions.

Some routers also contain a DMZ (demilitarised zone) which creates a secondary LAN inside of a LAN. It is predominantly used for public facing machines, which is perfect for my application. It will allow for just my NAS device to bypass the firewall rules set and open the desired ports. This means I can isolate the NAS from the rest of the devices which still follow the firewall rules.

Alternatively Making a second Network on my router which is predominantly used by the NAS could protect the devices connected on the other network. This should make an isolated environment to test the file sharing capabilities without compromising the security of connected devices. If that fails to work I could connect another router that is part of the same subnet essentially creating a two router system.
An example image:



Changing the port numbers for HTTPS, FTP, and SSH to something uncommon would also make my system more secure. This is because attackers usually scan the default port numbers for vulnerabilities. Hence making it very difficult for an attacker to find the port number of interest since there are a total of 65,535 possible port numbers they would need to scan. This is referred to as security through obscurity.

In order to ensure that the required data packets are coming from their intended origin, a key based exchange will need to be taking place. This will verify the data after the transmission process over the internet. This is commonly done through the use of digital signatures.

Aims and Objectives:

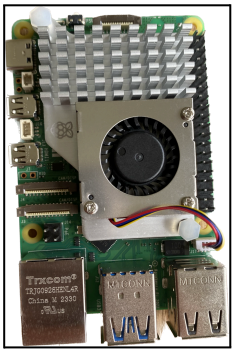
1. Physical NAS
    - 1.1. Has a HardDrive
        - 1.1.1. Must be of sufficient size (minimum 50GB)
        - 1.1.2. Contains a SATA port
        - 1.1.3. Has a mechanical read/write system
    - 1.2. Has a capable microprocessor (raspberry pi)
        - 1.2.1. Must be capable of receiving data via ethernet and wireless
        - 1.2.2. Must have a usb port
    - 1.3. Needs to be given sufficient power
        - 1.3.1. The HardDrive must be able to run at sufficient rpm
        - 1.3.2. The raspberry pi must be able to (by default) communicate with a device on the same network through ssh
    - 1.4. Have a functional on/off switch
    - 1.5. A protective casing should be able to hold all components and be damage resistant
    - 1.6. Optional power supply for usage without the availability of an outlet
2. Encryption

- 2.1. Has a unique encryption algorithm using 3x3 matrices that has not been used before.
  - 2.1.1. Can efficiently encrypt a file of varying magnitude in an appropriate amount of time
- 2.2. Has appropriate exception handling
  - 2.2.1. Has a method for dealing with a message that cannot be divisible by the size of the matrix. Specifically the remaining end of a message.
  - 2.2.2. Has a method for dealing with Capital Letters, Numbers , and Special characters
- 2.3. The Cipherkey is a truly random assortment of Letters
  - 2.3.1. A frequency analysis of the Ciphertext will not resemble patterns of normal english (e.g high frequency of the letter 'e')
- 2.4. The Cipherkey and Inverse key will be stored in a database on my computer
  - 2.4.1. Each file is encrypted with a new randomly generated cipherkey
  - 2.4.2. There should be separate databases for the files and the keys with a relational table connecting them
    - 2.4.2.1. All databases to be in 1st normal form or greater
    - 2.4.2.2. User should be able to retrieve the contents of the database
- 2.5. There should be a working decryption algorithm
  - 2.5.1. Matrix inverses must be possible, therefore the matrix must have a determinant value that is greater than zero.
  - 2.5.2. Multiplying the Ciphertext by the inverse matrix of the Cipherkey would result in the plaintext
  - 2.5.3. There should be an algorithm to convert the hexadecimal back to its original file format version.
- 2.6. Implement a Vernam cipher
  - 2.6.1. Uses XOR gates. Should also be possible to decrypt the ciphertext

3. Web application and Security
    - 3.1. Make a rule in the web administration of my router to receive packets from the website
        - 3.1.1. Open port 443 for a secure connection to the website (if public)
        - 3.1.2. Open port 22 for SSH capabilities (key exchange), also the port for SFTP
    - 3.2. I have enabled a DMZ in my router which will forward the requests of Ip addresses coming from the internet to my NAS
    - 3.3. Have remote access of the raspberry pi
        - 3.3.1. Is able to upload an encrypted text file over the internet using the Secure File Transfer Protocol
            - 3.3.1.1. Can specify which sector of the HardDrive the file goes into
        - 3.3.2. Is able to retrieve a file via the same SFTP protocol
            - 3.3.2.1. Has a progress bar to show how much of the file has been downloaded
        - 3.3.3. Is able to remove a file
    - 3.4. Has a user file management system
        - 3.4.1. Has different directories for different users
    - 3.5. Validate the access of a user
        - 3.5.1. Contains a user login page with an email and a password
            - 3.5.1.1. Is able to hide the password when being typed

Design:

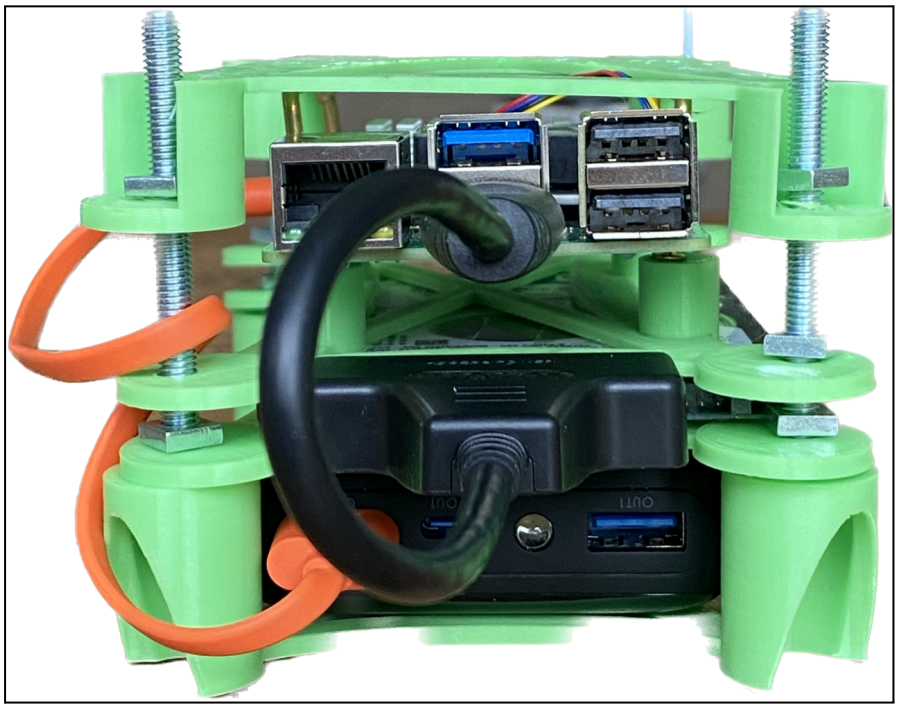
Parts list:

Part	Image	Description
PowerBank		Used to supply power to the raspberry pi. It makes the NAS portable and means it can work even if there are no available power sockets. Also means that I dont need multiple power connections going to each component.
SATA cables		They connect the HadDrive with the raspberry pi. Are used to only transfer data and not power.
Raspberry pi 5		Microprocessor which manages all of the instructions and is accessible via wirelessly.
HardDrive		A Magnetic HardDrive is a high capacity mode of storage. It is also relatively cheap, however its access speeds are not optimal.

Physical Assembly:

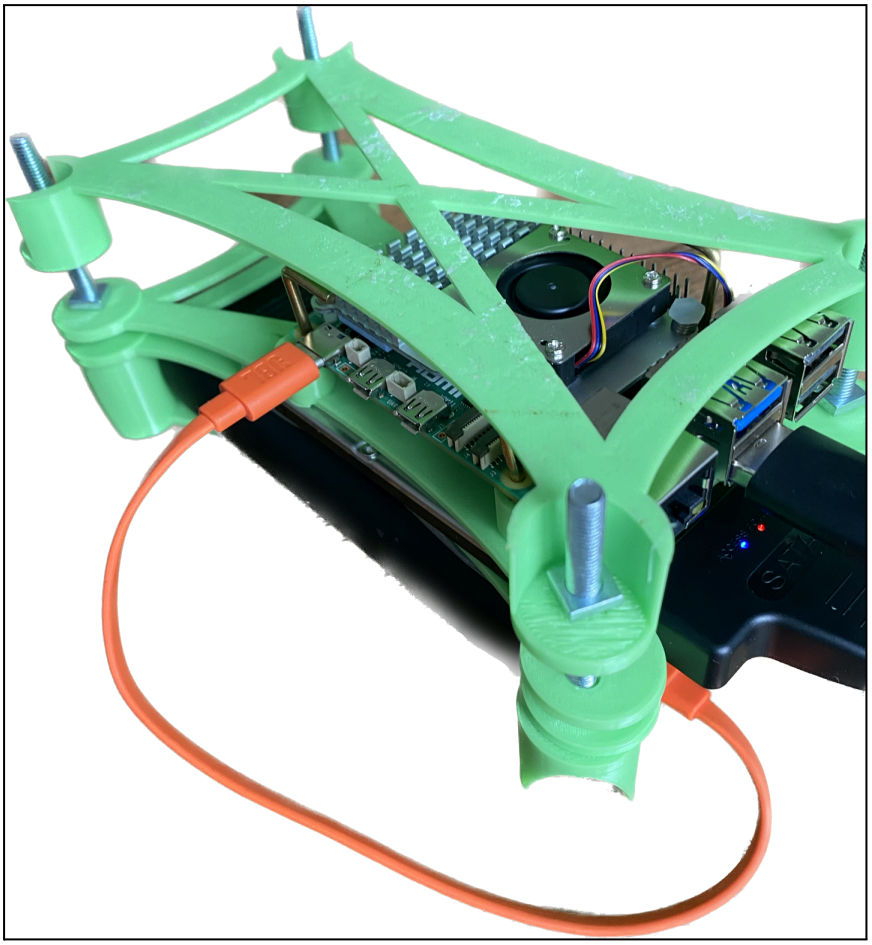
To protect and store all the components I need a supporting structure to hold it all together. A cheap solution to this is a 3d printed case made from PLA.

- The raspberry pi is bolted on the top of the structure, the harddrive is sandwiched in the middle layers, and the power bank is located on the very bottom due to its size and weight.

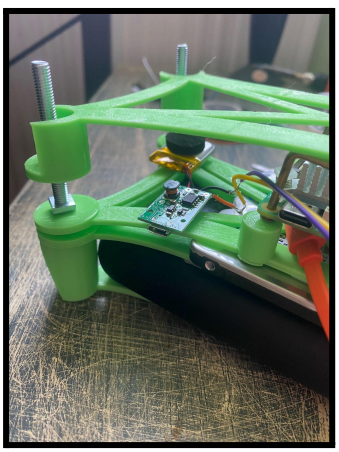


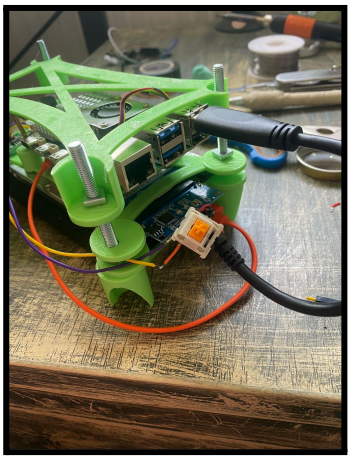
After flashing the SD card successfully with the Raspberry pi 32 bit operating system ~(Raspbian), I needed to configure it. This included adding the SSID of my network, naming my raspberry pi, so that I don't need to refer to its ip

address if I need to ssh into it, or refer to it in my program.



I was having an issue with seeing the harddrive that is connected to the pi board via the SATA cable. The reason why this may be is that the HDD isn't receiving enough power from the power supply. This is because when I connect the HDD to the board it spins at insufficient rpm. This tells me that the power is divided amongst the HDD and Raspberry pi and the HDD isn't getting enough of it . To overcome this I will need to take apart the SATA cable and solder on an external source of power just for the Hard Drive.





I use the command “lsblk” to see the list of available storage devices.

```
danek@malina:~ $ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
mmcblk0     179:0    0 14.8G  0 disk
├─mmcblk0p1 179:1    0  512M  0 part /boot/firmware
└─mmcblk0p2 179:2    0 14.3G  0 part /
danek@malina:~ $
```

Initially it was only recognising the SD card that has the Operating system installed on it. This is labelled as “mcblk”. If it sees the HardDrive then it should show up as “sda”.

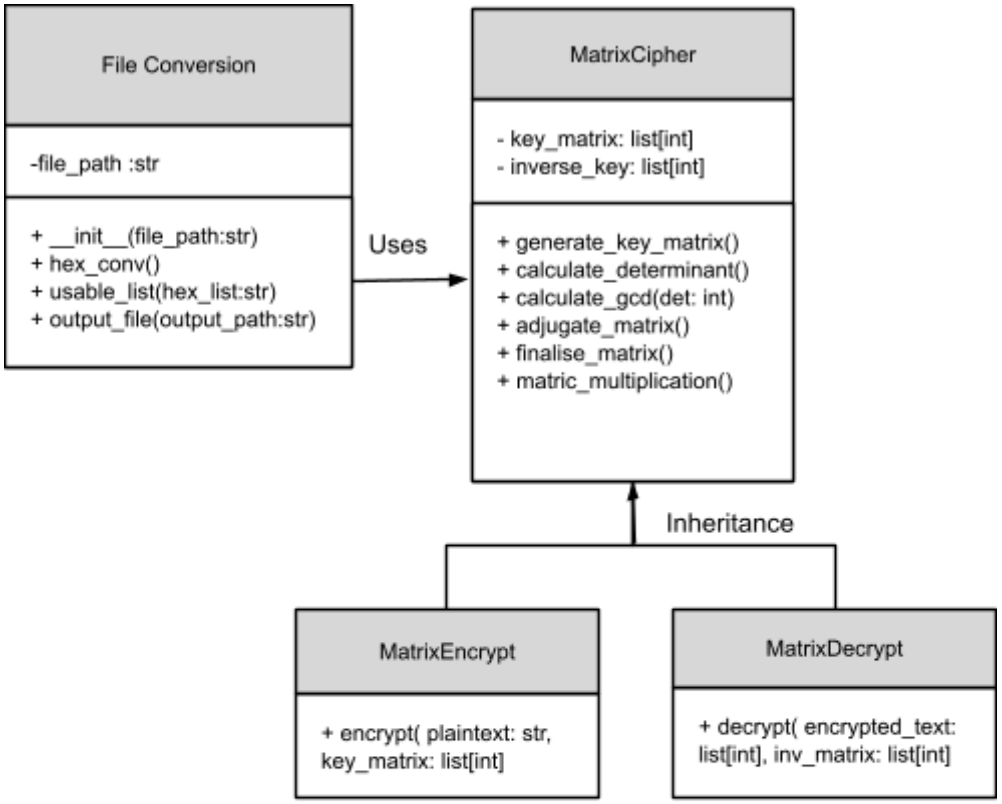
I found a micro usb circuit board from a pair of headphones. This ended up being quite effective in supplying power to the HardDrive. I followed some schematics from an article that is listed in my sources.

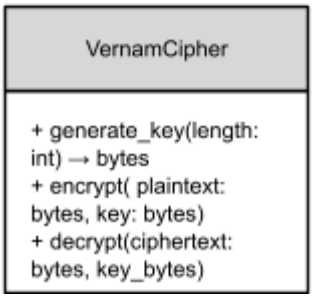
```
danek@malina:~ $ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda           8:0    0 298.1G  0 disk
mmcblk0     179:0    0  14.8G  0 disk
├─mmcblk0p1 179:1    0   512M  0 part /boot/firmware
└─mmcblk0p2 179:2    0  14.3G  0 part /
danek@malina:~ $
```

As shown on the command prompt, my HardDrive is now visible to my computer allowing me to access the full 320 GB of free space.
Initially I was utilising the wireless feature on the RaspberryPi to connect to my LAN but this limits my bandwidth and increases Latency. So I plan on connecting it to the Switch/Hub directly via ethernet.

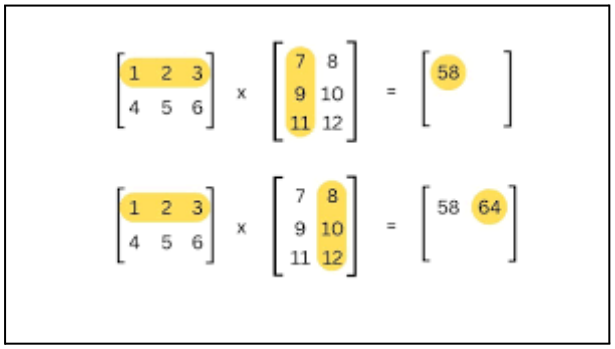
3x3 Matrix Cipher : Multiplications and Inverse Matrices:

Class diagram:





For my own form of encryption I will be using Matrix multiplication, which is a mathematical way of multiplying two 2D arrays. I will be using 3x3 matrices because they are more complex, hence harder to decode by someone other than the intended recipient. Matrix multiplication works by multiplying the top row of the first matrix by the first column in the second matrix and then adding all the values. For simplicity I will treat each 3x3 2D matrix as a single 1D array. This is because it is significantly harder to work with 3x3 matrices rather than singular arrays. It also provides no major benefits over just using lists so I will opt for using them instead.



The key matrix will consist of nine randomly generated numbers that will change day to day in order to maximise its security. However you are left with numbers larger than 26 which means they cannot correspond to a position in the alphabet. To counter this problem we can use (mod 26) to get the number within the scope of the alphabet.
Here is an outline of how the

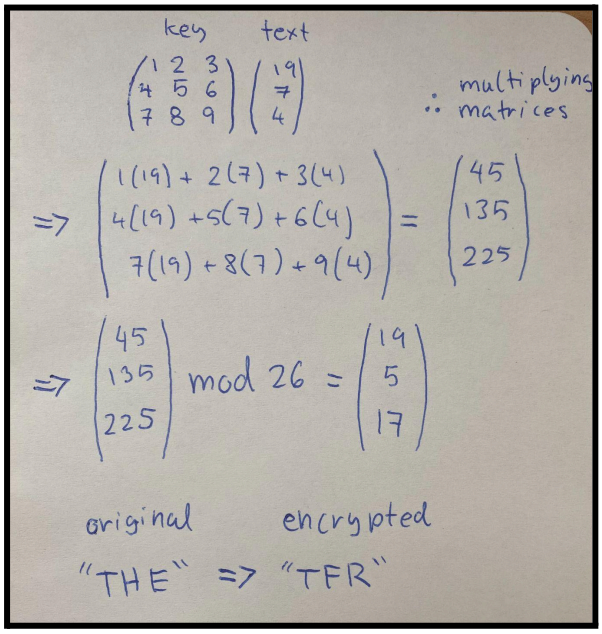
$$(\alpha\beta)\ mod\ 26 \qquad \Leftarrow Encryption$$

$$(\alpha\beta^{-1})\ mod\ 26 \qquad \Leftarrow Decryption$$

Example 1:

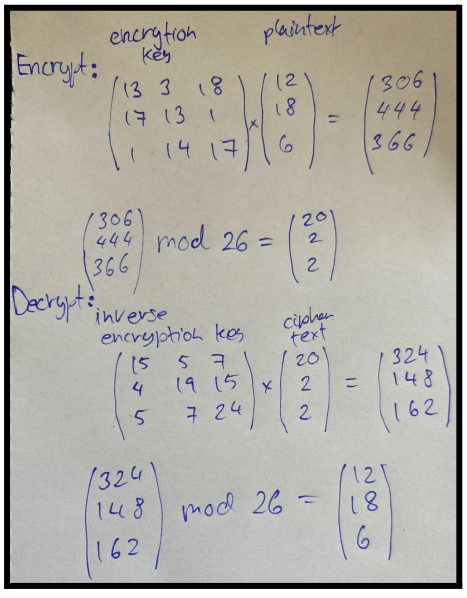
To better explain the logical process of how this will work I will use the phrase “the quick brown fox jumped over the lazy dog”. The encryption program should first strip the sentence of any spaces to create the single string “thequickbrownfoxjumpedoverthelazydog” ; this would then be split into groups of 3 letters in order to be able to encrypt it. After that, we would convert each letter into its position in the alphabet, resulting in an array of integers e.g (‘thequickb’ -> [19,7,4,16,18,20,8,2,10,1]). To actually encrypt the message we would then multiply this matrix by the key matrix and then mod 26 e.g :

Encryption:



Decryption:
For this particular case the decryption is impossible since the key matrix does not have an inverse. This is caused when its determinant is equal to 0, hence my encryption matrix must now satisfy the rule : $(determinant \neq 0)$ The key matrix can still be randomly generated, however there should be validation in place to make sure that the matrix satisfies the rule, if not, then a new key must be generated until it does.

Example 2:



As you can see in this case there exists an inverse matrix to the encryption matrix; making decryption possible. The original plaintext 12,18,6 is turned into ciphertext 20,2,2 and can then be transformed back into its plaintext version by being multiplied by the inverse matrix.
To ensure that the determinant is greater than zero I can have the random values be smaller at positions 2,4,6,7,9. I can do this by having the range be from 1-5 instead of 1-20 which is the range for the rest of the values.

Decryption Explanation:

The first step is getting the multiplicative inverse, which is a singular value which satisfies this rule:

$$Multiplicative\ inverse\ =\ x$$
$$(determinant\ *\ x)\ Mod\ 26\ \equiv\ 1$$

For example if my determinant is 1243 , 1243*5 = 6215, 6215 mod 26 = 1.
Therefore 5 is the multiplicative inverse. The program should iterate through a given amount of values until the multiplicative inverse is found.
The next step is to find the adjugate matrix. Which is just the inverse matrix without multiplying each value by 1/determinant. The next step is making any negative values positive; this is done by repeatedly adding 26 to the value until it is positive.

$$\begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{12} \\ a_{33} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{23} & a_{21} \\ a_{33} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{11} \\ a_{23} & a_{21} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{11} \\ a_{32} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

Step 1: getting the matrix of minors
Step 2: getting the matrix of cofactors
Step 3: transposing the matrix

Following this we would multiply the Adjugate matrix by the multiplicative inverse and mod 26 it to finally get the inverse matrix. From which we can decrypt the text file.

Encrypting Special Characters:

For now I have only considered encrypting lower case letters. It should be possible to encrypt upper case letters, numbers, and even special characters. There are a few steps I need to do to be able to achieve this. The way It works now, is that I have the regular alphabet of 26 letters and I will convert the letters in the message I want to decrypt into their position in the alphabet. Currently this alphabet only has lower case letters. To change this I will need to create a custom alphabet that has all the characters I want. Here is one I intend on using:

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890.,/\()+-=:;?!"

This has a total of 75 characters. This means that when I make the inverse matrix I need to MOD 75 at the end instead of the original MOD 26. And when I encrypt each file I will have to use this alphabet.

Encrypting other types of files:

Given that I can now encrypt numbers and special characters it now means that I can encrypt binary. This therefore means I would be able to encrypt almost any file. Since all files are made from machine code it now means I can even encrypt pictures and videos.
The binary code for images and videos is extremely large, as a result I need to make sure that my program can handle encryption and decryption efficiently. Therefore nothing should be a greater time complexity than O(n^2) otherwise It will take an unreasonable amount of time to encrypt.
I will need a python library to convert images to binary. One library that does this is called cv2. Prior to converting the file I will need to determine the file type. If it is a text file then no converting will have to be done.

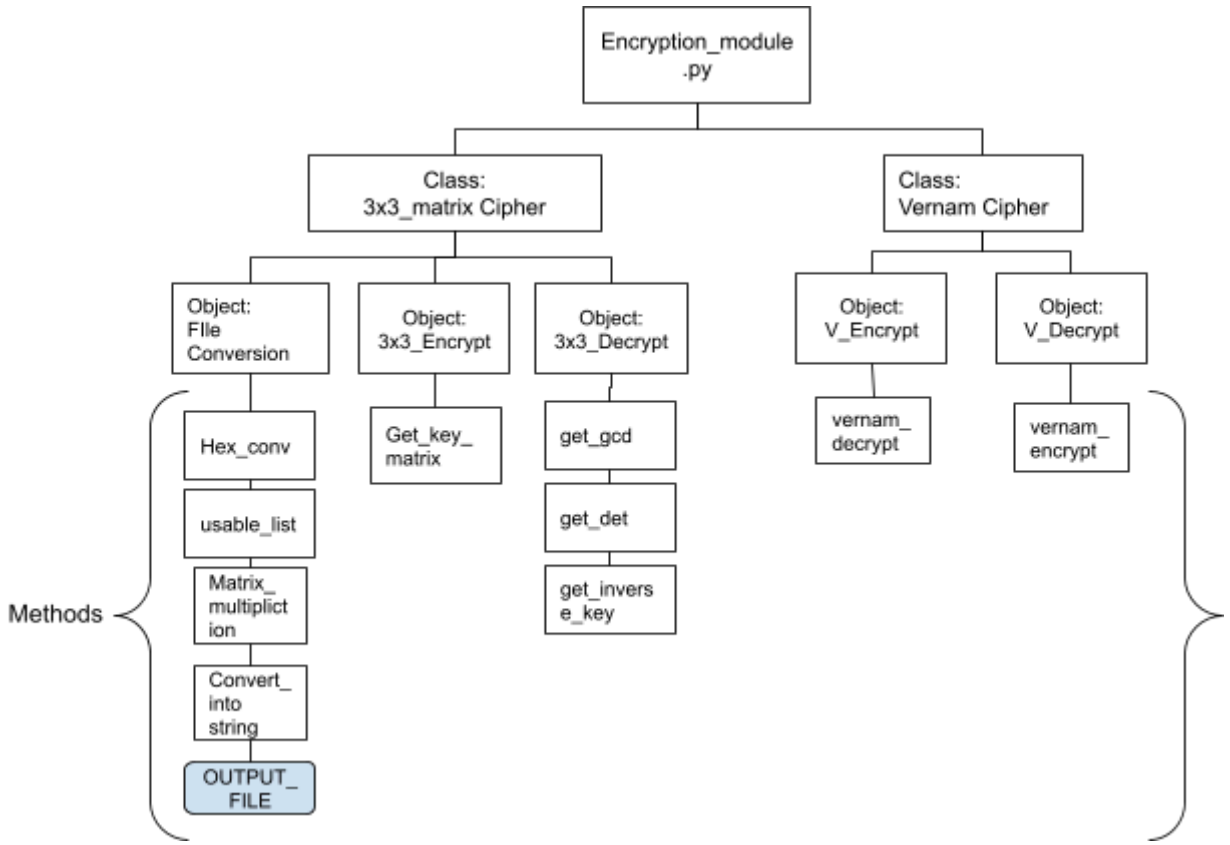
Vernam Cipher encryption:

The Vernam cipher is a symmetric encryption technique where each bit of the plaintext is combined with the corresponding bit of a secret key using the XOR (exclusive OR) operation. It is essentially a form of the one-time pad cipher, which is quite renowned for its theoretical perfect security when used correctly. The key must be as long as the plaintext and consists of truly random bits. This is one element of the Shannon theorem that I mentioned previously. The encryption process involves XORing each bit of the plaintext with the corresponding bit of the key, producing the ciphertext. XOR is a reversible operation, decryption is performed in the same manner: XORing the ciphertext with the same key restores the original plaintext. This makes the Vernam cipher an example of a symmetric key cipher, where the same key is used for both encryption and decryption. This could be a potential advantage since I only need to store one key in my database, which saves space.

One of the key strengths of the vernam cipher is that it is almost unbreakable as long as it meets the following conditions: if the key is random, used only once, and kept secret, it is mathematically impossible to decrypt the message without access to the key. This ensures that even if an attacker intercepts the ciphertext, they cannot deduce the plaintext without knowing the key. If the Vernam cipher I design is able to implement all the aspects of the Shannon theorem it is considered to be mathematically unbreakable. Considering this I want to give the user an option with which cipher they want to encrypt their files with. For example if they would like to encrypt a more important file such as a classified document the vernam cipher would be a better option than the matrix cipher. For my implementation I will use the random library that is integrated in python. This already makes its security questionable as the random library implements mathematical algorithms to convert a ‘seed’ into a number, this makes it predictable. However I meet the other requirements of the shannon theorem. I can make the key the same length as the ciphertext relatively easily although it would be extremely long since the hex data of the file tends to be quite extensive for images and larger files. I can also mitigate

someones access to the key since the database storing it will be on my local machine.

Class Diagram for Ciphers:

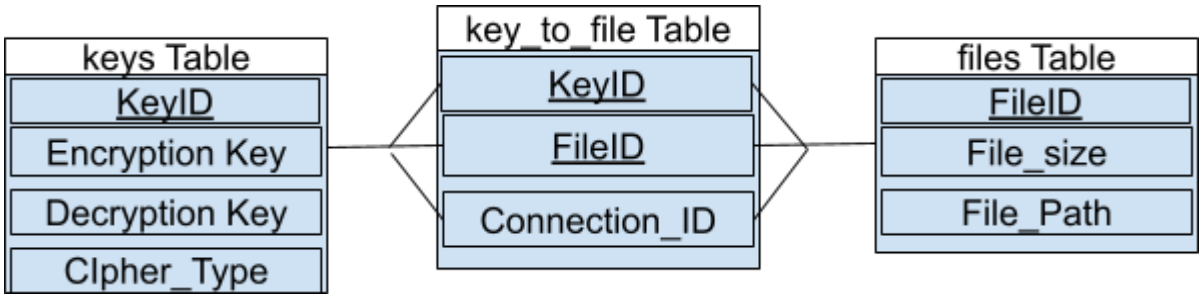


DB to store encryption keys and file:

One thing that should be considered, is that after a file is encrypted with one key it will no longer be able to be decoded in the future since the key changes. Therefore I will either need to store the keys that correspond to each file, or first decrypt every single file that is stored on the NAS and encrypt them again with the new key. The second option will require more processing power since there will potentially be hundreds of files on the NAS. On the other hand the first option would require a separate database that will map the key to its respective file. It would however lower the security of the encryption since someone could remotely access this database. Although, it does now mean that each file has a different encryption key, therefore it would be more challenging

for someone to decrypt more than one file. Therefore I will opt for the database option.

The entity relationship diagram should look something like this.



As visible in the diagram I will have 3 separate databases which are as follows:

Key_Table(KeyID, Encryption_key, Decryption_key,Cipher_Type)
Files_Table(File_Name, File_Path, File_Size)
Relation_Table(Key_ID, File_Name, File_path)

The databases should be normalised to Third normal form, meaning there should be no redundancy or repeats. The tables could be stored on 3rd party websites such as google cloud or Amazon RDS. Although it could provide some security benefits, it wouldn't be economically viable since they are paid services. So a more logical location would be directly on my own machine. .
The method I will be using to store the database is through a Virtual Private Server/ virtual environment allowing me to view the website on other devices. SQLite databases are usually stored as a file within the website's application directory. (e.g 'db.sqlite3').
The databases should be SQL inclusive and must have the python integration built in so that I can access it within my program.

Some viable options include:
3rd party:

1) Google Cloud SQL
2) Amazon Relational Database Service

Local:

1) SQLite
2) MySQL

Since my aim is to make my Cloud service as cheap as possible I will opt for hosting it on my local machine. MySQL is simpler to integrate so I will be using that. The database itself can be programmed and made in python directly. They are usually stored in this directory /var/lib/mysql. This can be changed later.
Another option would be to host the databases with the encryption keys on the raspberry pi itself. Setting up a database such as MySQL, MariaDB, or SQLite is relatively simple because the raspberry pi has support for all of them. However, there are notable drawbacks if I should take this approach. The Raspberry Pi has limited processing power and RAM, which can lead to performance bottlenecks, especially with large datasets or multiple concurrent users. This would not be an issue for me as I would be the only user. However , If I needed to retrieve the decryption keys for multiple files then I may run into issues. Storage can also be a limitation, as SD cards have lower read/write speeds and durability compared to SSDs.

SFTP server:

The SFTP or Secure file transfer protocol will be a set of rules by which packets will abide in order to be transmitted over the internet. It will be responsible for the backend function of the website. Its purpose should be to send files from the user's computer onto the Raspberry pi NAS.

Notes:
There are two states in which an FTP server can be , passive and active. Passive mod is generally safer since it operates in an environment with necessary firewalls in place. It is referred to as passive mode since the server

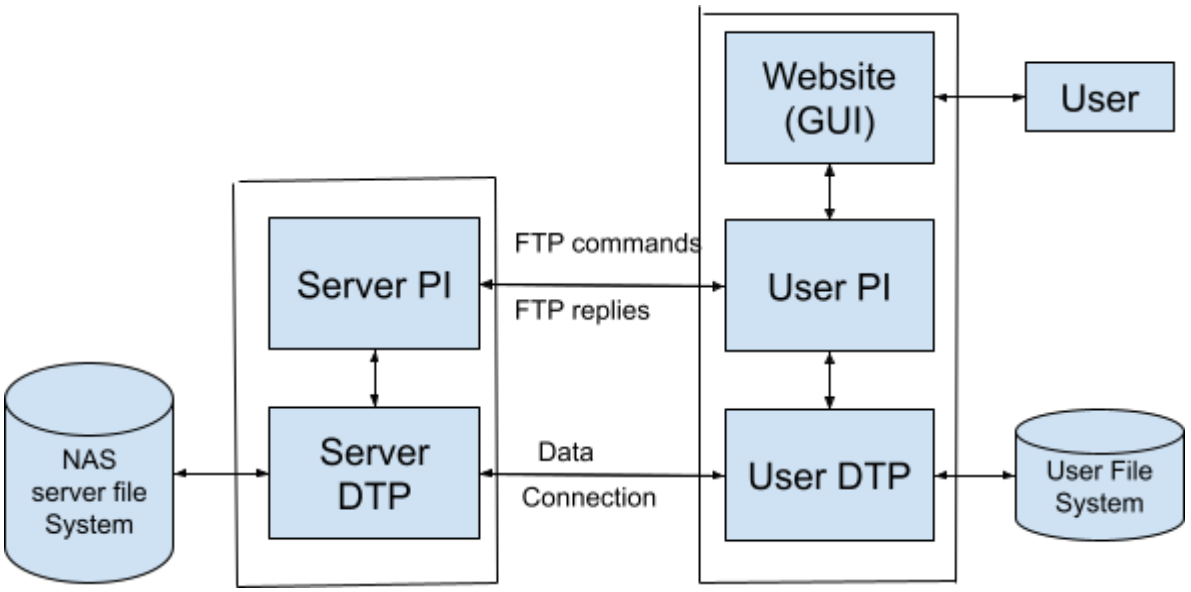
opens ports and listens passively. This allows clients to connect to it. However it does not allow for inbound connections to be initiated from the internet. This is referred to as the Data Transfer Process or DTP
As stated in my analysis there are two channels that are used when using FTP. The data channel and the control channel :

- The control connection is used for the exchange of commands and replies and follows the telnet protocol.
- The data connection is a full duplex connection by which the actual binary data is transferred. The data transferred could be a complete file, but it could also be a partition of one.

Terminology:

- ➔ There are two types of byte size. The logical byte size of the file which is the magnitude of the data located inside the file, and the transfer byte size which is used for the transmission of data. The transfer byte size will always be 8 bits.
- ➔ EOF (end of file) is a command that I will use to state the end of a file transfer.
- ➔ User PI initiates the control connection between its own port X and the server process.
- ➔ Server PI ‘listens’ on port X
- ➔ User DTP listens on the data port for a connection from the server FTP process
- ➔ The data transfer process, in its normal "active" state,
- ➔ establishes the data connection with the "listening" data port. It sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate a connection on the data port.

My FTP model:



Notes:

The FTP commands are responsible for specifying the parameters for the data connection which include things such as data ports, transfer mode, representation type and its structure. They can also determine the file system operations such as (store, retrieve, append, delete). These will be essential in providing certain functions to my website which will allow the user to manipulate the storage on the NAS.

There is also the concern of data representation and storage. This is when there are two storage mediums which use different ways of storing data. For example the users file system may use 7-bit ASCII while the Servers file system uses 8-bit EBCDIC codes. Since FTP is limited to the types of data translations it can do, the data translation should be done by the users themselves. However, in my case, the NAS uses a 3.5 inch Hard Drive which is identical to the ones found in laptops. Therefore the Storage method is also the same.

I also stated in the security section of my analysis that I will use non-default port numbers as a means of security through obscurity. This change can only be done through the User-PI. I will do this by using the PORT command, That will tell the Server-PI to listen on the alternate port.

Error Handling:

Error handling is done by TCP as there is no method for error detection within FTP. However there is an error recovery and restart procedure. This is done by implementing a marker within the file. If there is an error that occurs after transmission then the marker will also be affected. This should trigger a file resend.

Order of commands for verification:

1) The first command that is exchanged is the USER command. This acts as a user access control command.
2) The next command exchanged should be the PASS command. This requests the user's password.
3) ACCT command is then executed and acts as a secondary layer for the USER command and double checks the identity of the user. It stands for account.

Notes:
I will be using -paramiko- which is a library within python that allows you to create your own ftp server that is secured with SSH.

RSA key-based authentication:

RSA key-based authentication is a secure method for accessing remote systems, such as a Raspberry Pi, using cryptographic keys instead of passwords. It relies on a pair of keys: a private key, stored securely on the user's device, and a public key, uploaded to the server. When a connection is initiated, the server uses the public key to verify the private key's authenticity, granting access if they match. This approach eliminates the need for passwords, reducing the risk of brute-force attacks or phishing attempts. To set up RSA authentication, I need to generate a key pair using tools like ssh-keygen, then transfer the public key to the server, saving it in the ~/.ssh/authorized_keys file of my user account. Once configured, the user can connect without entering a password, streamlining access and enabling secure automation of tasks. RSA authentication is significantly stronger than password-based methods, as the cryptographic keys are much harder to compromise. For added security, the private key can be protected with a passphrase and strict file permissions to

prevent unauthorized access. I could use RSA key based authentication as opposed to the password authentication during SFTP initiation. This will make my data transfer more secure.

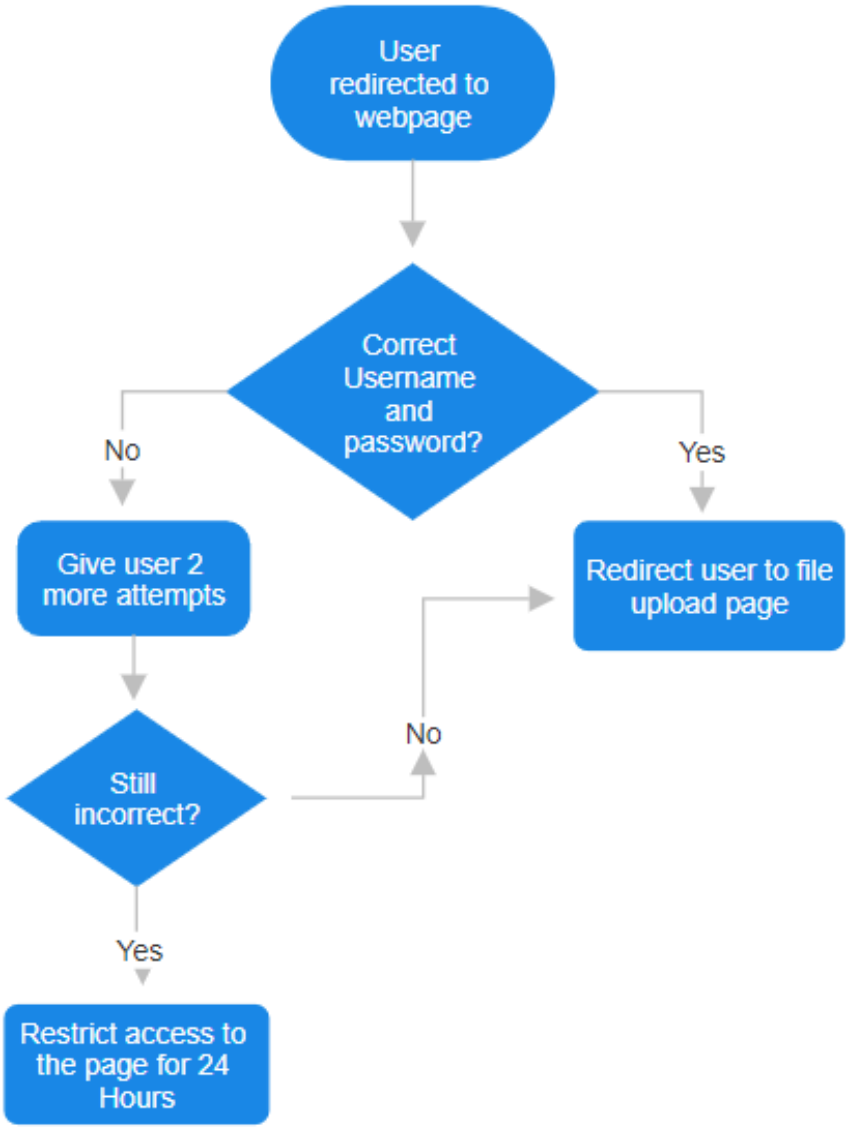
Web Server:

As stated in my analysis the web service should act as a Graphical User Interface (GUI) between the Secure File Transfer Protocol (SFTP) and the end user. In the future I will aim to have the website hosted publicly instead of just on my local machine. Hence there are some security concerns I must consider.

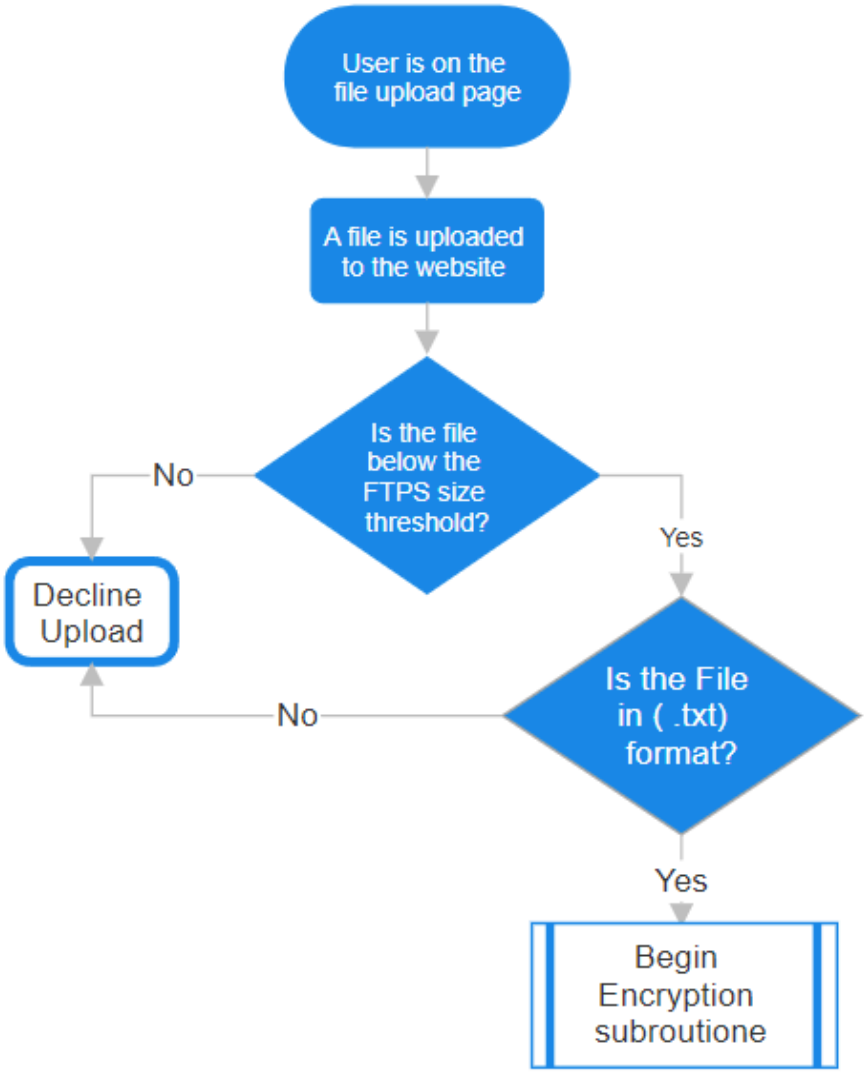
For my website framework I will be using Flask which has python integration built in. It will allow me to add functionality to the website. Some of the things that I will plan to add are listed here:

1. User login page (no need for an external database with users and login since there is only one user for now)
2. Menu page
3. Have a file upload inbox. This will validate that the file does not exceed the predetermined limit, as well as the file type. (should only be a text file)
4. File retrieval page
5. Use of HTTPS and environment variables
6. Be able to view, delete, and append new files

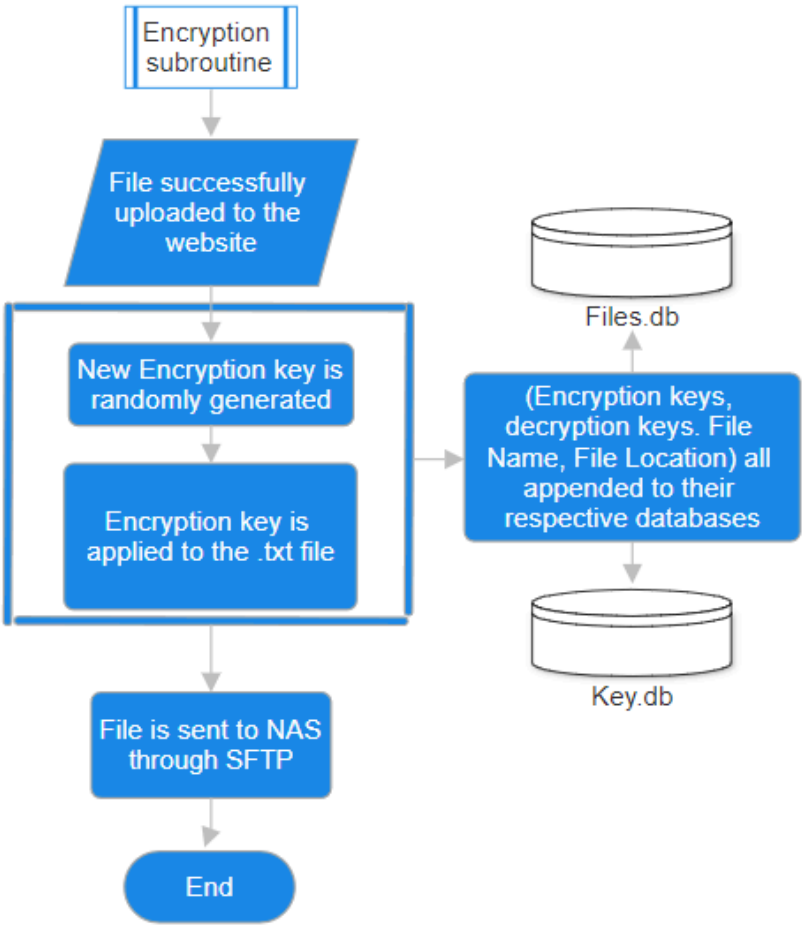
Simple Data Flow diagram demonstration:



Once the user's credentials are validated they will be redirected to another page with the option of uploading files. If the login process fails the first time then they will get two more attempts before it stops taking any more inputs . This will prevent unwanted users from brute forcing the login.

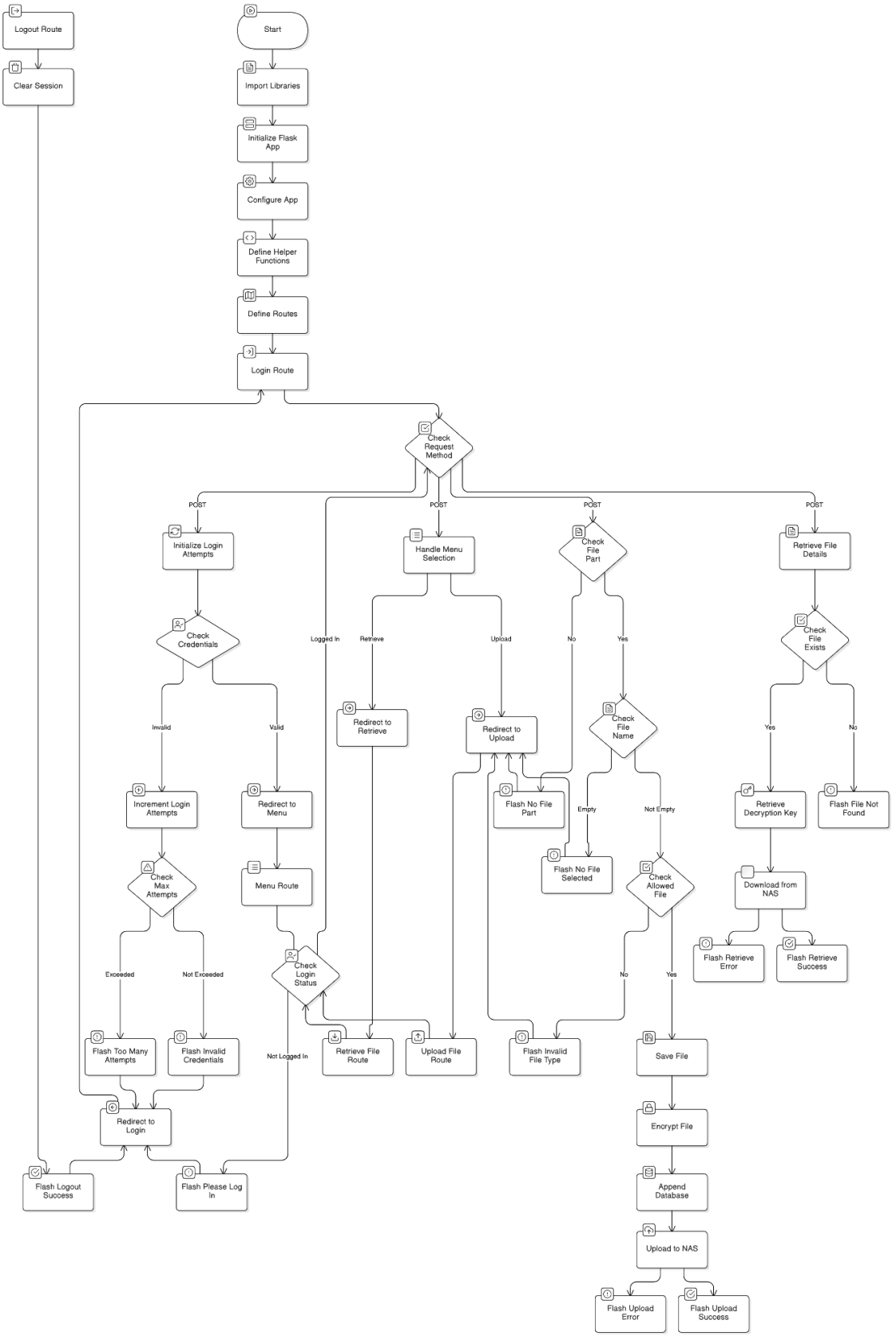


In the first Decision block, the FTPS size threshold is the maximum file size that a user can upload. This is there to prevent potential DDOS attacks that could flood the network of my NAS with unwanted traffic. Additionally it restricts the user uploading absurdly large files. The third security measure that is implemented in my data flow diagram is the file format based checking. This will inhibit the transmission of other file formats such as .exe , .img which could contain malicious and executable code. After the file is successfully validated it can then be encrypted and sent to the NAS.



Once an encryption key is generated it will first have the inverse matrix algorithm applied. Creating an inverse matrix. These two Matrices will be stored on the same database called keys.db . It will be located directly on the website itself (more specifically on the website hosting servers), but not accessible to the public of course.

Detailed Data Flow:



Data Flow Explanation:

It begins with a Start node, followed by the Import Libraries step, which makes sure all necessary dependencies are loaded before initializing the app and configuring it. I then define some helper functions, which allows for modular and reusable components to help the functionality of my system. The module then defines the main routes:e.g Login Route, which serves as the entry point for the user. The flowchart then expands into different branches depending on the user's interactions and request types. If a user attempts to log in, the system checks the Request method, validating whether it is a POST request before proceeding with initialise login attempts. The system verifies the user credentials, classifying them as either valid or Invalid. If credentials are invalid, login attempts are incremented, and a check max Attempts condition determines whether the user has exceeded login limits. If exceeded, the system triggers a 'too many attempts' warning, redirecting the user to the login page. Otherwise, an 'invalid credentials' message is flashed before reattempting login. If credentials are valid, the system redirects the user to the Menu Route, allowing further actions such as file retrieval and uploads.

When a user successfully logs in, the system continuously checks their Login Status before granting access to retrieval or upload functionalities. The Retrieve File Route and Upload File Route are separately managed. This guarantees authentication verification before file access. Within the file upload process, the system checks whether a file part exists in the request. If no file is selected, it flashes an error prompting the user to upload a file. If a file is provided, it checks whether the file name is empty or valid before proceeding. If invalid, an error message is displayed, but if the file passes validation, the system checks whether the file type is allowed. Unsupported file types trigger an error, while valid files are saved to the system. Once a file is successfully stored, it undergoes encryption, followed by Database Appending to maintain records, and then Upload to NAS, ensuring secure external storage. The File Retrieval process follows a similar validation flow. When a file retrieval request is received, the system checks for File Details to verify existence. If the file is not found, an error message is flashed; otherwise, the system retrieves the Decryption Key and proceeds with NAS Downloading. If retrieval fails, the

user is notified with an error message, but if successful, the file is provided. Additional system functionality includes a Logout Route, which clears the user session and confirms logout success. Throughout the flowchart, decision points are carefully structured to ensure authentication, proper file handling, and security compliance. The diagram systematically organizes the logic for handling requests, processing user authentication, managing file uploads and downloads, and securely encrypting and retrieving stored data while maintaining a structured approach to session management, error handling, and database integrity. The entire flow adheres to a modular architecture, allowing independent handling of authentication, menu navigation, file transactions, and system security while ensuring smooth user interactions and proper error messaging for unsuccessful operations.

Notes:
Uploading files within html requires a POST request method, which is a type of request which sends data to a server to create or update a resource. However this will still only send the file name rather than its contents. Therefore in order to send it as binary data we must change the content type (enc-type attribute) to multipart form data which will label the input as a file which can be split into smaller packets.

Website Security:

For testing purposes and considering I am the only user, the website will be private hence not accessible by the wider internet. However, in the future once multiple users will be using the website I must ensure that it has sufficient security. Besides input validation, the website should also have other forms of security. One of which being environment variables. They are used to store confidential information within the source code.
The website should also be encrypted with HTTPS.

I can disable remote root access by binding the databases with a fixed ip address of the local host. In the trial stages I will only use a single IP address from which I will upload data to the NAS. I will specifically bind this rule in

my router and in the FTP side of the code. This should prevent someone with a different IP from accessing the network.
The website also has its

Automatic encryption:

There must be a program within my website that will:

1, generate a random key
2, Apply the encryption algorithm to a copy of the file
3, send and store the encryption and decryption keys to the database

Generating a Random key can be done by importing the randInt function from the random library, and then appending 9 random numbers to an array. The encryption algorithm will use a function to split the key matrix and the contents of the file into blocks of three. Here is a diagram to better convey how it will work. The first row is the index values for the contents. The second row is the index values for the key matrix. The numbers in the same column should be multiplied together.
Diagram:

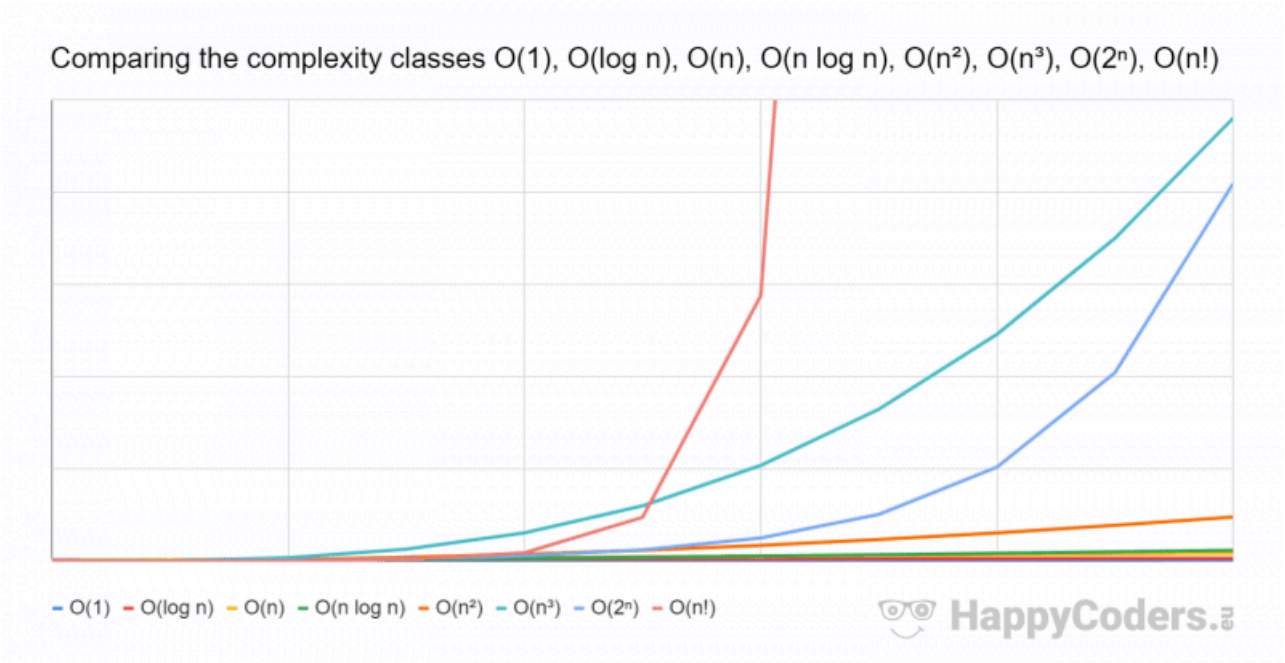
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8	0	...

Every time three values are multiplied they should be added together to get an encrypted value which will be then made smaller by using MOD (size of alphabet). It will be a nested loop where the outside loop is going through the whole text and the inside loop is going through the key_matrix.

Big O analysis:

To better understand the complexity of my functions I will be using Big O analysis to graphically analyse the space and time complexity. This will allow me to visualise how well my encryption algorithms will work for larger inputs.

This will be very useful for when I decide where to run the algorithms. For example If I run it on my raspberry pi it may only efficiently be able to encode files up to a certain size limit before overloading the CPU.



The complexity of my functions:

Encryption_module.py:

- Class MatrixEncrypt
- encrypt(self, plaintext, key_matrix):
  - Time Complexity: O(n^2), where n is the length of plaintext.
    - Mapping characters to indices is O(n).
    - The matrix_multiplication is O(n^2).

    - Class MatrixDecrypt
- decrypt(self, encrypted_text):
  - Time Complexity: O(n), where n is the length of encrypted_text.
    - The matrix_multiplication is O(n^2).

    - Class VernamCipher
- generate_key(length):
  - Time Complexity: O(length).

- The os.urandom(length) function generates a random sequence of length bytes.

- encrypt(plaintext, key):
  - Time Complexity: O(n), where n is the length of plaintext.
    - The list comprehension iterates over n elements.
- decrypt(ciphertext, key):
  - Time Complexity: O(n), where n is the length of ciphertext.
    - The loop iterates over n elements.

SFTP_module.py

- Function: download_file_with_progress

- Steps:
  - sftp.stat(remote_file_path).st_size: O(1) (fetching file metadata is a constant time operation on my SFTP server).
  - tqdm: Progress bar updates, which are O(1) per update.
  - File read and write in chunks of 32KB:
    - O(n/32768), where n is the size of the file. This simplifies to O(n), as the chunk size is constant.
- Overall Time Complexity:
  O(n), where n is the size of the file.
- ---
- Function: send_file_sftp
- Steps:
  - transport.connect: Authentication and connection setup depend on the server but are typically O(1) (constant).
  - sftp.put: Uploading a file typically involves reading the local file and transferring it in chunks, so it's O(n), where n is the file size.
- Overall Time Complexity:
  O(n), where n is the size of the file being uploaded.
- ---
- Function: download_files_from_nas
- Steps:
  - SSH connection setup (ssh_client.connect): This is O(1).
  - For each file in remote_files:
    - Extracting file_name with os.path.basename: O(k), where k is the length of the file path.

        - Calling download_file_with_progress: O(m), where mmm is the size of the file.
- Overall Time Complexity:
  O(f · m)), where:
    - f: Number of files in remote_files.
    - m: Average size of the files.
- Assuming f files are independent of their size, the complexity scales linearly with both f and m.

My last module: web_app.py does not have any particularly complex functions with subroutines inside.

Configuring my Router's web administration:

The Router handles everything to do with packet routing and Network address Translation etc. This being the case I will need to configure my router to be able to send and receive data from the NAS . This can be done by changing and creating certain rule sets within the web administration for my router. The rule set should only grant the authorised user access to the NAS. Not allowing them to bridge to other devices connected to that same network.

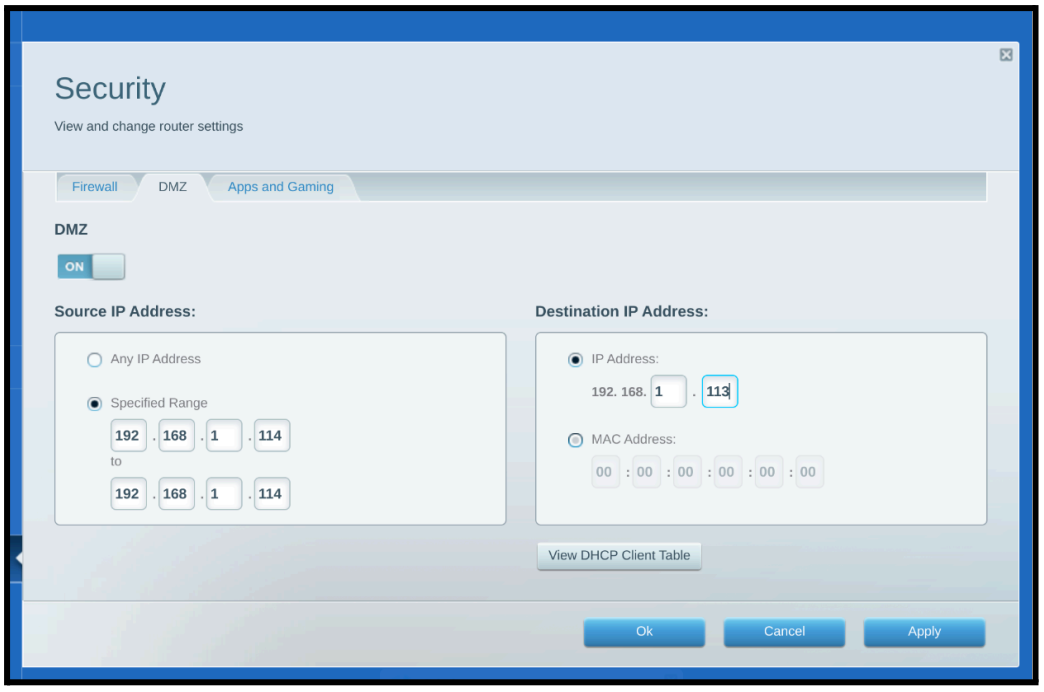
I will be using a separate router within my home network, on which no devices will be connected to. I am currently using a Virgin Media router and its web configuration is located on this ip address: 192.168.0.1 . It is password protected.

Configurations I must make:

- Make sure that the Router is in DMZ mode (demilitarized mode) makes the primary router publicly invisible when it comes to NAT (network address translation).
- Incoming SFTP transmission at the specified obscure port (to be determined), will always be redirected to the NAS. The data being transmitted should just be text files that have been sent by the website through SFTP.
- Configure the router to passive mode. So that it only 'listens' on one particular port.

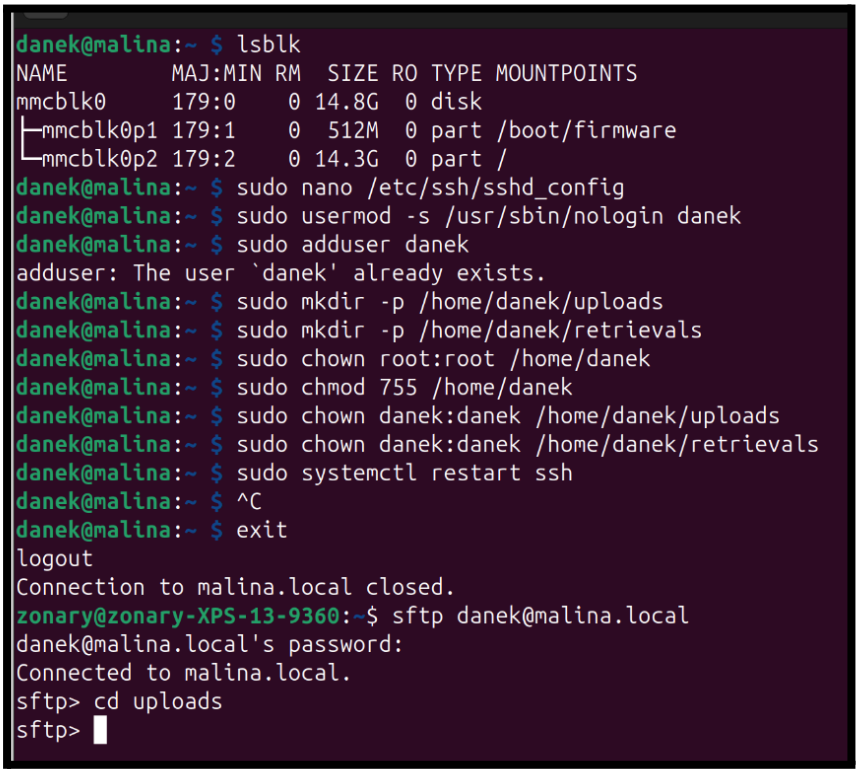
- Set up port forwarding for port 22 (SSH) to the website

The local IP address that I have assigned the raspberry pi is 10.0.0.1 and can be accessed through the address 10.0.0.2. I have named it 'malina' for ease of login. In my technical solution the IP of the NAS is static however the IP of the user will be dynamic. To determine the changing IP of the user I will be using a socket module. The socket.gethostname() function that is built into the library will have the hostname passed into it as the argument which will return the IP address of the user. The static private IP address of my raspberry pi is 192.168.1.113. In the DMZ settings you can change the source IP address to any IP from the internet, this should therefore let me connect from anywhere.



Technical Solution:

Bash Scripts:



Explanation:
Sudo nano /etc/ssh/ssh_config : this command allows me to change the ssh configuration file. Inside of the configuration I constricted the user privileges. First, I disabled TCP forwarding and changed the root directory to me. I also disabled X11 forwarding.

Pearl script:

```
Unset
Port (22 -> 2834)
Match User danek
       ChrootDirectory /home/danek
       ForceCommand internal-sftp
       AllowTcpForwarding no
       X11Forwarding no
```

Encryption_module.py

```
Python
import glob
```

```
import os
from os import urandom
from random import randint

ALPHABET =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890.,+-=:;?!
"


class FileConversion:
    def __init__(self, file_path):
        self.file_path = file_path  # Initialize with the file path

    def hex_conv(self):
        try:
            with open(self.file_path, 'rb') as file:
                bytes_array = file.read()  # Read file as bytes
                return [hex(byte) for byte in bytes_array]  # Convert
bytes to hex
        except FileNotFoundError:
            print("File not found.")
            return []

    def usable_list(self, hex_list):
        new_list = []
        for i in hex_list:
            for x in i:
                new_list.append(x)  # Flatten hex list into a single list
        return new_list

    @staticmethod
    def output_file(output_path, content):
        with open(output_path, "w") as f:
            f.write(content)  # Write content to the output file


class MatrixCipher:
    def __init__(self):
        self.key_matrix = None  # Initialize key matrix
        self.inverse_key = None  # Initialize inverse key matrix

    def generate_key_matrix(self):
        key_matrix = []
```

```
        for a in range(9):
            if a in [1, 5, 6]:
                key_matrix.append(randint(1, 3))  # Specific positions
get values 1-3
            else:
                key_matrix.append(randint(0, 20))  # Other positions get
values 0-20
        return key_matrix

    def calculate_determinant(self, matrix):
        # Calculate determinant of a 3x3 matrix
        first = matrix[0] * ((matrix[4] * matrix[8]) - (matrix[5] *
matrix[7]))
        second = -1 * (matrix[1] * ((matrix[3] * matrix[8]) - (matrix[5]
* matrix[6])))
        third = matrix[2] * ((matrix[3] * matrix[7]) - (matrix[4] *
matrix[6]))
        return first + second + third

    def calculate_gcd(self, det):
        # Find modular inverse of determinant modulo 71
        for b in range(1, det):
            if (det * b) % 71 == 1:
                return b
        return 0

    def adjugate_matrix(self, matrix):
        # Calculate adjugate of a 3x3 matrix
        adj_mat = [
            (matrix[4] * matrix[8]) - (matrix[5] * matrix[7]),
            -1 * ((matrix[1] * matrix[8]) - (matrix[2] * matrix[7])),
            (matrix[1] * matrix[5]) - (matrix[2] * matrix[4]),
            -1 * ((matrix[3] * matrix[8]) - (matrix[5] * matrix[6])),
            (matrix[0] * matrix[8]) - (matrix[2] * matrix[6]),
            -1 * ((matrix[0] * matrix[5]) - (matrix[2] * matrix[3])),
            (matrix[3] * matrix[7]) - (matrix[4] * matrix[6]),
            -1 * ((matrix[0] * matrix[7]) - (matrix[1] * matrix[6])),
            (matrix[0] * matrix[4]) - (matrix[1] * matrix[3])
        ]
        return adj_mat

    def finalize_matrix(self, adj_matrix, gcd):
        # Normalize and finalize the inverse matrix
```

```
        new = []
        for c in range(0, 9):
            if adj_matrix[c] < 0:
                temp = adj_matrix[c]
                while temp < 0:
                    temp += 71  # Ensure positive values
                new.append(temp)
            else:
                new.append(adj_matrix[c])

        final_matrix = []
        for d in new:
            final_matrix.append((d * gcd) % 71)  # Apply modular
arithmetic
        return final_matrix

    def matrix_multiplication(self, matrix, length, data):
        # Perform matrix multiplication for encryption/decryption
        encrypted = []
        for i in range(0, length, 3):
            for k in range(0, 8, 3):
                total = (
                    matrix[k] * data[i] +
                    matrix[k+1] * data[i+1] +
                    matrix[k+2] * data[i+2]
                )
                encrypted.append(total % len(ALPHABET))  # Apply modulo
operation
        return encrypted


class MatrixEncrypt(MatrixCipher):
    def encrypt(self, plaintext, key_matrix):
        # Encrypt plaintext using the key matrix
        length = len(plaintext)
        indexed = [ALPHABET.find(ch) for ch in plaintext]  # Map
characters to indices
        return self.matrix_multiplication(key_matrix, length, indexed)


class MatrixDecrypt(MatrixCipher):
    def decrypt(self, encrypted_text):
        # Decrypt encrypted text using the inverse key matrix
```

```
        length = len(encrypted_text)
        return self.matrix_multiplication(self.inverse_key, length,
encrypted_text)


class VernamCipher:
    @staticmethod
    def generate_key(length):
        return os.urandom(length)  #generate a random key of the given
length

    @staticmethod
    def encrypt(plaintext, key):
        # Encrypt plaintext using XOR with the key
        return bytes([int(plaintext[i]) ^ key[i] for i in
range(len(plaintext))])

    @staticmethod
    def decrypt(ciphertext, key):
        # decrypt ciphertext using XOR with the key
        if len(ciphertext) != len(key):
            raise ValueError("Key and text lengths do not match.")
        return bytes([ciphertext[i] ^ key[i] for i in
range(len(ciphertext))])
```

My encryption module supports two encryption types: the Matrix Cipher and the Vernam Cipher. For the Matrix Cipher, you first generate a 3x3 key matrix using the generate_key_matrix function, which creates a matrix with specific constraints on certain positions. This makes it more likely for the determinant to not equal zero.

The determinant of the matrix is calculated using the calculate_determinant function, and the modular inverse of the determinant is found using the calculate_gcd function to ensure the matrix is invertible .

The adjugate matrix is then calculated using the adjugate_matrix function, and the final inverse matrix is done using the finalize_matrix function.

The file content is padded to align with the matrix dimensions, encrypted using matrix multiplication in the matrix_multiplication function, and saved as a new file.

For the Vernam Cipher, a random key of the same length as the plaintext is generated using the generate_key function. The file content is encrypted using the XOR operation in the encrypt function, and the encrypted file is saved. Both ciphers store the encryption/decryption keys and file metadata in a MySQL database for later retrieval.

Webapp_module.py

Python

```python
from flask import Flask, render_template, request, redirect, url_for,
flash, session
import os
import paramiko
import mysql.connector
from encryption_module import *
from sftp_module import *

app = Flask(__name__)

app.secret_key = 'HgOr15Wpzx£2'  # Secret key for session management
app.config['max_filesize'] = 500 * 1024 * 1024  # Max file size for
uploads (500MB)
app.config['upload_fd'] = 'uploads'  # Directory for uploaded files
app.config['retrieve_fd'] = 'retrievals'  # Directory for retrieved
files
allowed_ext = {'txt', 'jpeg', 'pdf'}  # Allowed file extensions
max_attempts = 3  # Max login attempts before lockout

def allowed_file(filename):
   # Check if the file extension is allowed
   return '.' in filename and filename.rsplit('.', 1)[1].lower() in
allowed_ext

def hex_to_file(file):
   # convert hex data back to a binary file
   with open(file, 'r') as f:
       hex_data = f.read().strip()
   binary_data = bytes.fromhex(hex_data)
   with open('nameofthefilefromthedatabase.jpeg', 'w') as img:
       img.write(binary_data)
```

```
def append_database(cipher_type, enc_matrix, dec_matrix, file_name,
file_path, file_size):
   # Insert encryption keys and file details into the database
   mydb = mysql.connector.connect(
       host="localhost",
       user="myusername",
       password="mypassword",
       database="key_file_storage"
   )
   mycursor = mydb.cursor()
   sql1 = "INSERT INTO enc_keys(cipher_type, encryption_key,
decryption_key) VALUES (%s, %s, %s)"
   val1 = (cipher_type, str(enc_matrix), str(dec_matrix))

   sql2 = "INSERT INTO enc_files(file_name, unique_file_path, file_size)
VALUES (%s, %s, %s)"
   val2 = (file_name, file_path, file_size)

   mycursor.execute(sql1, val1)
   mycursor.execute(sql2, val2)

   mydb.commit()
   mycursor.close()
   mydb.close()


# Login Route
@app.route('/', methods=['GET', 'POST'])
def login():
   if request.method == 'POST':
       # Initialize login attempts if not already in session
       if 'login_attempts' not in session:
           session['login_attempts'] = 0

       username = request.form['username']
       password = request.form['password']
       correct_username = 'myusername'
       correct_password = 'mypassword'

       # Check credentials
       if username == correct_username and password == correct_password:
           session.pop('login_attempts', None)  # Reset login attempts
           session['logged_in'] = True  # Mark user as logged in
```

```
            return redirect(url_for('menu'))
        else:
            session['login_attempts'] += 1
            if session['login_attempts'] >= max_attempts:
                flash('Too many login attempts', 'danger')
                return redirect(url_for('login'))
            flash('Invalid credentials', 'danger')
            return redirect(url_for('login'))
    return render_template('login.html')


# Menu Route with Dropdown
@app.route('/menu', methods=['GET', 'POST'])
def menu():
    if not session.get('logged_in'):  # Check if user is logged in
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'POST':
        selected_page = request.form.get('page')
        if selected_page == 'upload':
            return redirect(url_for('upload_file'))
        elif selected_page == 'retrieve':
            return redirect(url_for('retrieve_file'))
    return render_template('menu.html')


# Upload File Route
@app.route('/upload_file', methods=['GET', 'POST'])
def upload_file():
    if not session.get('logged_in'):
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'POST':
        if 'file' not in request.files:
            flash('No file part', 'danger')
            return redirect(request.url)
        file = request.files['file']

        if file.filename == '':
            flash('No file selected', 'danger')
            return redirect(request.url)
```

```
        elif file and allowed_file(file.filename):
            if not os.path.exists(app.config['upload_fd']):
                os.makedirs(app.config['upload_fd'])

            filepath = os.path.join(app.config['upload_fd'],
file.filename)
            file.save(filepath)

            list_of_file_uploads =
glob.glob('/home/myusername/PycharmProjects/NEA/uploads/*')
            latest_file_upload = max(list_of_file_uploads,
key=os.path.getctime)
            file_converter = FileConversion(latest_file_upload)
            hex_list = file_converter.hex_conv()
            usable_text = file_converter.usable_list(hex_list)

            file_size_bytes = os.path.getsize(filepath)

            cipher = request.form.get('cipher')
            if cipher == 'M':
                # Matrix Cipher Encryption
                matrix_cipher = MatrixCipher()
                enc = MatrixEncrypt()

                # Key Matrix Generation
                inverse_key = []
                for _ in range(10):
                    key_matrix = matrix_cipher.generate_key_matrix()
                    det = matrix_cipher.calculate_determinant(key_matrix)
                    gcd = matrix_cipher.calculate_gcd(det)
                    if gcd != 0:
                        matrix_cipher.key_matrix = key_matrix
                        matrix_cipher.inverse_key =
matrix_cipher.finalize_matrix(matrix_cipher.adjugate_matrix(key_matrix),
gcd)
                        inverse_key = matrix_cipher.inverse_key
                        break

                # Text Padding
                while len(usable_text) % 3 != 0:
                    usable_text.append('!')

                encrypted = enc.encrypt(usable_text, key_matrix)
```

```
                converted_text = ''.join([ALPHABET[i] for i in
encrypted])

                real_file_name =
latest_file_upload.replace('/home/myusername/PycharmProjects/NEA/uploads
/', '')
                minimised_file_name = real_file_name.replace('.txt' or
'.jpeg' or '.pdf' or '.jpg', '')
                unique_file_path =
'/home/myusername/PycharmProjects/NEA/encrypted/' + minimised_file_name
+ '_encrypted' + '.txt'
                file_converter.output_file(unique_file_path,
converted_text)

                append_database('M', key_matrix, inverse_key,
real_file_name, latest_file_upload, file_size_bytes)

            elif cipher == 'V':
                # Vernam Cipher Encryption
                new = ''.join(usable_text)
                plaintext = ''.join(format(ord(i), '08b') for i in new)

                vernam_cipher = VernamCipher()
                key = vernam_cipher.generate_key(len(plaintext))
                ciphertext = vernam_cipher.encrypt(plaintext, key)
                decrypted_text = vernam_cipher.decrypt(ciphertext, key)

                real_file_name =
latest_file_upload.replace('/home/myusername/PycharmProjects/NEA/uploads
/', '')
                minimised_file_name = real_file_name.replace('.txt' or
'.jpeg' or '.pdf' or '.jpg', '')
                unique_file_path =
'/home/myusername/PycharmProjects/NEA/encrypted/' + minimised_file_name
+ '_encrypted' + '.txt'

                with open(unique_file_path, "wb") as file:
                    file.write(ciphertext)

                append_database('V', key, key, real_file_name,
latest_file_upload, file_size_bytes)

            # SFTP Upload to NAS
```

```
            remote_path = f"/pi/danek/pi/downloads/{file.filename}"
            try:
                send_file_sftp(
                    local_file_path=filepath,
                    remote_file_path=remote_path,
                    hostname='192.168.1.113',
                    port=2834,
                    username='danek',
                    password='mypassword'
                )
                flash(f"File '{file.filename}' uploaded to NAS
successfully!", 'success')
            except Exception as e:
                flash(f"Error uploading file to NAS: {e}", 'danger')
        else:
            flash('Invalid file type. Only .txt, .jpeg, and .pdf
allowed.', 'danger')
            return redirect(request.url)
    return render_template('upload_file.html')


# Retrieve File Route
@app.route('/retrieve_file', methods=['GET'])
def retrieve_file():
    mydb = mysql.connector.connect(
        host="localhost",
        user="myusername",
        password="mypassword",
        database="key_file_storage"
    )
    myc = mydb.cursor()

    if not session.get('logged_in'):
        flash('Please log in first.', 'warning')
        return redirect(url_for('login'))

    if request.method == 'POST':
        user_input = request.form.get('user_input')  # Retrieve file name
from form
        enc_type = request.form.get('cipher')  # Retrieve cipher type
from form

        query = "SELECT * FROM enc_files WHERE file_name = %s"
```

```
        myc.execute(query, (user_input,))
        value = ''
        results = myc.fetchall()
        if results:
            for row in results:
                value += (row[1])

        if value == user_input:
            retrieve = """
                    SELECT ek.decryption_key
                    FROM enc_keys ek
                    JOIN keys_to_files ktf ON ek.key_ID =  ktf.key_ID
                    JOIN enc_files ef ON ktf.file_ID = ef.file_ID
                    WHERE ef.file_name = %s;
                    """
            myc.execute(retrieve, (user_input,))
            fetch = myc.fetchone()
            if fetch:
                decrypt = ''
                for i in fetch:
                    decrypt += fetch[3]

                try:
                    # SFTP Download from NAS
                    the_file = download_files_from_nas(
                        HOSTNAME,  # NAS IP/domain
                        PORT,  # SSH port
                        USERNAME,  # SSH username
                        PASSWORD,  # SSH password
                        KEY_FILENAME,  # SSH private key path
                        REMOTE_FILES,
                        LOCAL_DIRECTORY
                    )
                    local_file_path =
os.path.join(app.config['retrieve_fd'], decrypt)
                    flash(f"File '{decrypt}' retrieved successfully!",
'success')
                except Exception as e:
                    flash(f"Error retrieving file: {e}", 'danger')
        else:
            flash('Cannot find that file.', 'warning')

        myc.close()
```

```
        mydb.close()
    return render_template('retrieve_file.html')


# Logout Route
@app.route('/logout')
def logout():
    session.pop('logged_in', None)  # Clear the session
    flash('You have been logged out.', 'success')
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)  # Run the Flask app
```

The webapp module handles user interactions, file management, and integration with encryption and SFTP modules. It starts by initializing Flask with a secret key (app.secret_key) to securely manage user sessions. Configuration settings like max_filesize (500MB limit), upload_fd (folder for uploaded files), and retrieve_fd (folder for decrypted files) make it so that file handling is organised. The allowed_file function restricts uploads to specific extensions (.txt, .jpeg, .pdf) to prevent unsafe file types such as .exe (executable files) which
User authentication is managed through the login.html route. The user must enter a predefined username and password. These can only be altered in the source code. The login system tracks attempts using Flask sessions: after three failed tries (max_attempts = 3), the user is locked out. Successful login sets session['logged_in'] = True, granting access to the main menu.

The menu.html route acts as a hub which redirects users to either the /upload_file or /retrieve_file pages. During file uploads, the app validates the file, converts it to hexadecimal using the FileConversion class, and encrypts it using either the Matrix Cipher or Vernam Cipher which is selected by the user (me) . Encrypted files are renamed (e.g., filename_encrypted.txt) and stored locally, while metadata (file name, path, size) and encryption keys are saved to a MySQL database via the append_database function. The app also transfers the encrypted file to a NAS (Network Attached Storage) using SFTP for secure remote storage.

For file retrieval (/retrieve_file), the app queries the database for the decryption key linked to the requested file. Retrieves the encrypted file from the NAS via SFTP, and prepares it for decryption. Error handling and user feedback are provided using Flask's flash messages, which display success or failure notifications (e.g., "File uploaded successfully" or "Invalid credentials").

In terms of security I added sessions to protect user activity. The file validation prevents malicious uploads, and SFTP ensures secure file transfers. The modular design separates tasks like encryption, database operations, and file handling, making my code maintainable and scalable. The app is run in debug mode (app.run(debug=True)) for testing, though I would disable this if I were to deploy it.

Sftp_module.py

Python

```
import paramiko
import os
from tqdm import tqdm

def download_file_with_progress(sftp, remote_file_path,
local_file_path):
    """
    Downloads a file with a progress bar.

    :param sftp: An active SFTP session.
    :param remote_file_path: Path to the remote file.
    :param local_file_path: Path where the file will be saved
locally.
    """
    try:
        file_size = sftp.stat(remote_file_path).st_size
        with sftp.open(remote_file_path, 'rb') as remote_file,
open(local_file_path, 'wb') as local_file:
            with tqdm(total=file_size, unit='B',
unit_scale=True, desc=os.path.basename(remote_file_path)) as
progress:
                while True:
```

```
                    data = remote_file.read(32768)  # 32KB
chunks
                    if not data:
                        break
                    local_file.write(data)
                    progress.update(len(data))
        print(f"dwnloaded {remote_file_path} to
{local_file_path}")
    except FileNotFoundError:
        print(f"remote {remote_file_path} does not exist.")
    except Exception as e:
        print(f"rrror downloading file: {e}")

def send_file_sftp(local_file_path, remote_file_path,
hostname, port, username, password):
    try:
        # create a SFTP client using Paramiko
        transport = paramiko.Transport((hostname, port))
        transport.connect(username=username,
password=password)

        sftp = paramiko.SFTPClient.from_transport(transport)

        # upload the file
        sftp.put(local_file_path, remote_file_path)
        print(f"file uploaded to {remote_file_path}")
        sftp.close()
        transport.close()
     except Exception as e:
        print(f"error in uploading yourr file: {e}")

def send_file_sftp(local_file_path, remote_file_path,
hostname, port, username, key_filename):
    """
    Uploads a file to the NAS using SSH key-based
authentication.
```

```
    :param local_file_path: Path to the file to upload.
    :param remote_file_path: Path on the NAS where the file
will be saved.
    :param hostname: NAS hostname or IP address.
    :param port: SSH port of the NAS.
    :param username: SSH username.
    :param key_filename: Path to the private key file for
authentication.
    """
    try:
        # Set up transport with key-based authentication
        private_key = paramiko.RSAKey(filename=key_filename)
        transport = paramiko.Transport((hostname, port))
        transport.connect(username=username, pkey=private_key)

        sftp = paramiko.SFTPClient.from_transport(transport)

        # Upload the file
        sftp.put(local_file_path, remote_file_path)
        print(f"File '{local_file_path}' uploaded to
'{remote_file_path}' successfully!")

        sftp.close()
        transport.close()
    except Exception as e:
        print(f"Error uploading file using SFTP: {e}")

def download_files_from_nas(
    hostname,
    port,
    username,
    password=None,
    key_filename=None,
    remote_files=None,
    local_directory=None
):
    """
```

```
    can downloads multiple files from the NAS using SFTP with
progress indication.

    :param hostname: IP address or domain of the NAS.
    :param port: my alternative SSH port other than the default
22
    :param username: SSH username.
    :param password: SSH password that I set in m
    :param key_filename: Path to the private key file
    :param remote_files: List of remote file paths to download.
    :param local_directory: Directory where files will be saved
locally.
    """
    if remote_files is None:
        remote_files = []
    if local_directory is None:
        local_directory =
'/home/myusername/PycharmProjects/NEA/retrievals'

    try:
        ssh_client = paramiko.SSHClient()

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy(
))

        if key_filename:
            ssh_client.connect(
                hostname=hostname,
                port=port,
                username=username,
                key_filename=key_filename
            )
        else:
            ssh_client.connect(
                hostname=hostname,
                port=port,
                username=username,
```

```
                password=password
            )

        print(f"Connected to {hostname} successfully.")

        sftp = ssh_client.open_sftp()
        print("SFTP session established.")

        for remote_file in remote_files:
            file_name = os.path.basename(remote_file)
            local_file = os.path.join(local_directory,
file_name)
            download_file_with_progress(sftp, remote_file,
local_file)

    except paramiko.AuthenticationException:
        print("authentication failed please check your
credentials.")
    except paramiko.SSHException as sshException:
        print(f"Unable to establish SSH connection:
{sshException}")
    except Exception as e:
        print(f"Error occurred: {e}")
    finally:
        if 'sftp' in locals():
            sftp.close()
            print("SFTP session closed.")
        if 'ssh_client' in locals():
            ssh_client.close()
            print("SSH connection closed.")


if __name__ == "__main__":
    HOSTNAME = '192.168.1.113'            # Public IP or domain
of my NAS
    PORT = 2834                           # SSH port
    USERNAME = 'myusername'             # SSH username
```

```
    PASSWORD = 'mypassword'             # SSH password (if using
password auth)
    KEY_FILENAME = '' # Path to SSH private sssh key
    REMOTE_FILES =  ''
    LOCAL_DIRECTORY =
'/home/myusername/PycharmProjects/NEA/retrievals/'
```

The download_file_with_progress function downloads a file from the remote server to the local system while displaying a progress bar. It reads the file in 32KB chunks to efficiently handle large files and updates the progress bar in real time. If the remote file does not exist or an error occurs during the download I have displayed appropriate error messages. This function ensures that users can monitor the download process and verify its completion. Although I do not integrate it within the webapp module, I added it as a means of guaranteeing that I have uploaded a file.

The send_file_sftp function uploads a file from the local system to the remote NAS. It supports two authentication methods: password-based and key-based. For password-based authentication, the function uses the provided username and password to establish an SFTP connection. For key-based authentication, it uses a private key file. Once the connection is established, the file is uploaded to the specified remote path. If the upload is successful, a confirmation message is displayed; otherwise, an error message is shown.

The download_files_from_nas function downloads multiple files from the remote NAS to a local directory. It establishes an SSH connection using either a password or a private key, depending on the provided user credentials. Once connected, it opens an SFTP session and iterates through the list of remote files, downloading each one using the download_file_with_progress function. The function handles authentication errors, SSH connection issues, and other exceptions. This provides informative error messages. It also ensures that the SFTP session and SSH connection are properly closed after the operation.

Upload_file.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bo
otstrap.min.css">
  <title>Upload File</title>
</head>
<body class="bg-light">
  <div class="container mt-5">
      <h2 class="text-center">Upload a File</h2>
      <form method="POST" enctype="multipart/form-data"
class="w-50 mx-auto mt-4">
          <div class="mb-3">
              <label for="file" class="form-label">Choose a
valid file (.txt, .jpeg, .pdf)</label>
              <input type="file" class="form-control"
id="file" name="file" required>
          </div>
          <div class="mb-3">
              <label for="cipher"
class="form-label">Encryption Type</label>
              <input type="text" id="cipher" name="cipher"
class="form-control" placeholder="Enter 'M' or 'V'" required>
          </div>
          <button type="submit" class="btn btn-primary
w-100">Submit</button>
      </form>
      {% with messages =
get_flashed_messages(with_categories=true) %}
          {% if messages %}
              <div class="alert alert-warning mt-3">
                  {% for category, message in messages %}
```

```
                    <p>{{ message }}</p>
                {% endfor %}
            </div>
        {% endif %}
    {% endwith %}
  </div>
</body>
</html>
```

Menu.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bo
otstrap.min.css">
   <title>Menu</title>
</head>
<body class="bg-light">
   <div class="container mt-5">
       <h1 class="text-center mb-4">Menu</h1>
       <form method="POST" class="text-center">
           <label for="page" class="form-label">Choose a
page:</label>
           <select name="page" id="page" class="form-select
w-50 mx-auto">
               <option value="upload">Upload File</option>
               <option value="retrieve">Retrieve File</option>
           </select>
```

```
            <button type="submit" class="btn btn-primary
mt-3">Go</button>
        </form>
        <div class="text-center mt-3">
            <a href="/logout" class="btn btn-danger">Logout</a>
        </div>
    </div>
</body>
</html>
```

Login.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bo
otstrap.min.css">
  <title>Login</title>
</head>
<body class="bg-light">
  <div class="container mt-5">
      <h2 class="text-center">Login</h2>
      <form method="POST" class="w-50 mx-auto mt-4">
          <div class="mb-3">
              <label for="username"
class="form-label">Username</label>
              <input type="text" class="form-control"
id="username" name="username" required>
          </div>
```

```
            <div class="mb-3">
                <label for="password"
class="form-label">Password</label>
                <input type="password" class="form-control"
id="password" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary
w-100">Login</button>
        </form>
        {% with messages =
get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div class="alert alert-warning mt-3">
                    {% for category, message in messages %}
                        <p>{{ message }}</p>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}
    </div>
</body>
</html>
```

Retrieve_file.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

```
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bo
otstrap.min.css">
    <title>Retrieve File</title>
</head>
<body class="bg-light">
    <div class="container mt-5">
        <h2 class="text-center">Retrieve File</h2>
        <form method="POST" class="w-50 mx-auto mt-4">
            <div class="mb-3">
                <label for="filename" class="form-label">Enter
File Name</label>
                <input type="text" id="filename"
name="filename" class="form-control" placeholder="Enter file
name" required>
            </div>
            <div class="mb-3">
               <label for="cipher"
class="form-label">Encryption Type</label>
                <input type="text" id="cipher" name="cipher"
class="form-control" placeholder="Enter 'M' or 'V'" required>
            </div>
            <button type="submit" class="btn btn-primary
w-100">Submit</button>
        </form>
        {% with messages =
get_flashed_messages(with_categories=true) %}
          {% if messages %}
              <div class="alert alert-warning mt-3">
                  {% for category, message in messages %}
                      <p>{{ message }}</p>
                  {% endfor %}
              </div>
          {% endif %}
      {% endwith %}
    </div>
</body>
```

```
</html>
```

DATABASE:

```
Unset
CREATE DATABASE key_file_storage;
USE key_file_storage;

CREATE TABLE enc_keys (
    key_ID BIGINT UNSIGNED AUTO_INCREMENT,
    cipher_type VARCHAR(1000),
    encryption_key BLOB,
    decryption_key BLOB,
    PRIMARY KEY (key_ID)
);

CREATE TABLE enc_files (
    file_ID BIGINT UNSIGNED AUTO_INCREMENT,
    file_name VARCHAR(1000),
    file_size_MB VARCHAR(10000),
    unique_file_path VARCHAR(2000),
    PRIMARY KEY (file_ID)
);

CREATE TABLE keys_to_files (
    connection_ID BIGINT UNSIGNED AUTO_INCREMENT,
    key_ID BIGINT UNSIGNED NOT NULL,
    file_ID BIGINT UNSIGNED NOT NULL,
    PRIMARY KEY (connection_ID),
    FOREIGN KEY (key_ID)
        REFERENCES enc_keys(key_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (file_ID)
        REFERENCES enc_files(file_ID)
        ON DELETE CASCADE
```

```
);
```

```
SQL issues:

-- ALTER TABLE enc_files ADD COLUMN file_name varchar(255) AFTER file_ID;
DESCRIBE enc_files;
-- ALTER TABLE enc_keys ADD COLUMN cipher_type varchar(255) AFTER key_ID;
-- DESCRIBE enc_keys;
INSERT INTO enc_files (file_name, file_size, unique_file_path) VALUES
('hello.txt', '1', 'path/');
SELECT * FROM enc_files;
SELECT * FROM enc_keys;
```

I had issues with seeing the second attribute under the primary keys of each table. As a result I had to add them on manually with the above sql.
RSA encryption key for later use:
Key =
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDQ3y9aaxJbNxti6V5JeVrregdBMbrP5zfV73IdnRFDnJKu
yv9PHEWRuNMlMTBdxTFhY71J2C6nt5FgkeLNqo4eisYayR/6v817IvJJ5zNxOf6deHXJjj93SHcXFKKy
YD+iocuupvp9fgWOjjT6SJKl4Hc7Mvxsn2GS5kX8hEi7ghO2llXv/23ilHC5Z+hB5BGjY3Ww8E8yclCIw
ZywImElcF658AxQ+gQt29c3Ja2v3ZXD1sEaAtmXezJMd//kDR/bS2iI5IOLvhHBRvsTrPZhvup5xo92Tw
flfJbBUeLv8ASr+81blk21Am5ZN7nFNtQ88w8Rx7ipddLZJdg7Vl9+jMhrmm7xDAybVbqRkqksUyMoHq
ga7f/Q+5179/R/8B7RDAo49Oha1Aeqgy+NoSX9b7TuYEiAmogrFtXLJU+eR2VGY2XZAffgPSAs6cJZoI6
bXfialjapeAjLcCYdXJSrmZ6EdsBnon+TbCo+FHejWzCPoaIW+8ahOK44yJmOQzM=
myusername@myusername-XPS-13-9360

Testing:

Video Proof:

https://youtube.com/shorts/ZTizHROWNpg?feature=share

Test_ID	Description	Data/Action	Expected result	Actual result	Objective met?
1.1	NAS contains a hard drive that is at least 50		n/a	n/a	✔

	GB in size and has a SATA port				
1.2	Has a capable microprocessor that has a wifi antenna and an internet port.		n/a	n/a	✔
1.4/1.5/1.6	Has a functional on/off switch, and a power bank to supply the HDD. should all be contained in a casing.		n/a	n/a	✔
2.1	Has a unique 3x3 matrix cipher.		n/a	n/a	✔
2.1.1	Can efficiently encrypt a file of varying magnitude. Does so in a timely fashion.	Matrix multiplication	Data from the file should be illegible	Example Output: 3zWc7adoW!jwryV5!io1GQ=Sm5bm5b3zW,CkRBNafs3zWNLqbTju3BgxU!FmS!mmVn-1X,CkNLqNLq:U1uGiKvSb4aC6xV4Y4qw	✔
2.2	Has appropriate	Deals with invalid	Should not allow	flash('invalid	✔

	exception handling	credentials	further access	credentials', 'danger')	
2.2.1	Adds text insulation for text lengths that are not divisible by the size of the key matrix	Adds a '!' at the end of the text	Hexadecimal example: Ox3fOx5!!!	Example output: ...0 x 7 d 0 x a 0 x 7 f 0 x c 0 0 x 1 8 0 x 3 e 0 x 4 9 0 x e 5 0 x b e 0 x e 0 0 x 3 c 0 x 8 0 0 x 0 0 x 0 ! ! The exclamation marks were appended to the end of the text	✔
2.2.2	Has a method for dealing with capital letters numbers and special characters	Use a larger alphabet which contains these letters and use MOD (size of alphabet) instead of mod 26	n/a	n/a	✔
2.3	The generated cipherkey is a truly randomly generated matrix	Uses the random library	n/a	n/a	✔
2.3.1	Frequency analysis of the ciphertext should not	Inputted a section of an example ciphertext	Shows the highest frequency in other	Shows a high frequency of the letter	✔

	show patterns of normal english		letters than e	‘q’.	
2.4	A database should hold the attributes of the files. As well as both the cipher keys	DESCRIBE enc_files; DESCRIBE enc_keys;	All relevant attributes	All relevant attributes 	✔
2.5.1	Cipher key must have a determinant value greater than zero for an inverse key to be possible	Using the equation for a determinant	A value greater than zero	Example output: 15845	✔
2.5.2	Multiplying the Ciphertext by the inverse key will give the original plaintext	Original text = `hellothisisasecretmessage01234` Inverse key = [5, 26, 10,	Converts the ciphertext back into its original hexadecimal so that it	The original hexadecimal of the file after conversion: 0 x 6 8 0 x 6	✔

		21, 4, 25, 40, 59, 47] Ciphertext = = n 7 ! J K - Z o n a 1 = n 7 6 k T P i k 6 + p = n 7 ! J K C ? 1 m , f = n 7 E P M = J k ? y I O o ; t y A - Z o k 1 c O o ; o s ? - Z o 6 + p = n 7 . q V - Z o m , f O o ; t y A 6 1 c p 5 ; = n 7 4 . E 9 . x , R ! J k c o s ? u 5 c x U d t I g	can be converted into its original file. Original hex: 0 x 6 8 0 x 6 5 0 x 6 c 0 x 6 c 0 x 6 f 0 x 7 4 0 x 6 8 0 x 6 9 0 x 7 3 0 x 6 9 0 x 7 3 0 x 6 1 0 x 7 3 0 x 6 5 0 x 6 3 0 x 7 2 0 x 6 5 0 x 7 4 0 x 6 d 0 x 6 5 0 x 7 3 0 x 7 3 0 x 6 1 0 x 6 7 0 x 6 5 0 x 3 0 0 x 3 1 0 x 3 2 0 x 3 3 0 x 3 4 0 x a	5 0 x 6 c 0 x 6 c 0 x 6 f 0 x 7 4 0 x 6 8 0 x 6 9 0 x 7 3 0 x 6 9 0 x 7 3 0 x 6 1 0 x 7 3 0 x 6 5 0 x 6 3 0 x 7 2 0 x 6 5 0 x 7 4 0 x 6 d 0 x 6 5 0 x 7 3 0 x 7 3 0 x 6 1 0 x 6 7 0 x 6 5 0 x 3 0 0 x 3 1 0 x 3 2 0 x 3 3 0 x 3 4 0 x a They are identical, proving the decryption algorithm works	
2.5.3	There should be an algorithm to convert the decrypted hexadecimal back into its original file format	Using a fromhex() function to convert hexadecimal back into binary and then writing that binary to a text file	The original file that the user uploaded	-Txt files generally work well and can be converted back into their original format. -Jpeg files can be converted back however I get an error	-

				when viewing them (Improper call to jpeg library in state 201)	
2.6	Implement a vernam cipher with the same key length as the message itself	Uses XOR gates. Single key. Symmetric	Should be able to be encrypted and decrypted	Files are able to be encrypted and decrypted	✔
3.1/3.2	Enable port forwarding in my routers web administration. This was done using a DMZ	Map 192.168.1.114 (IP of my machine) To 192.168.1.113 (IP of my NAS	Should connect server DTP with user DTP and have a stable connection between their file systems	Is able to initiate the connection if the source IP address is changed to all IP's from the internet. That way the user can send files from any computer	✔
3.3	Is able to upload an encrypted text file over SFTP	Uses paramiko library	File is uploaded to the uploads folder		✔
3.3.1.1	Is able to specify which HDD sector the file goes into	-	File is uploaded to a sector of the HardDrive that the user	Is able to specify which folder the file goes into.	-

			specifies		
3.3.2.2	Is able to retrieve a file using SFTP protocol that displays a download progress bar	Paramiko library	Decrypted File is retrieved into the 'retrievals' directory on the users computer	Method Error 304, EDIT: File was successfully retrieve	✔
3.5	Has a user login page to validate the user	Compares the correct username with the one inputted and prevents the user from trying again if they get the login wrong 3 times	Should not allow the user to try to login again if they get the credentials wrong 3 times. If the login is incorrect they should not be able to continue to the login page	Does not allow the user to continue to the login page	✔
3.6	Has an upload and retrieval page that allows the user to specify which cipher is used to encrypt/decrypt it	Matrix multiplicati on	Retrieves the decryption key from the database if they are retrieving a file and decrypts the file with that key. If the user is uploading a file it	Successful	✔

			encrypts it with that cipher type.		
3.7	Has a menu page displaying the retrieval and upload page options	-	Redirects user to their chosen page	Redirects user to their chosen page	✔
3.8					

Evaluation:

In my analysis I mentioned 4 key features that I wanted my project to have:

1. Has multiple encryption types that each provide unique benefits. The files should be stored in an encrypted state.
2. The user should be able to upload and retrieve files using SFTP. Be able to track the progress using a progress bar
3. A private web application serves as a graphical user interface between the user and the NAS. It should verify the users' identity through a login procedure and have separate pages for uploads and retrievals.
4. A database stores the keys and metadata of the file.

(1)) I had two encryption types that were unique and scrambled the hex content in different ways. Both had the ability to be decrypted with their respective decryption algorithm. The Vernam cipher had a significantly lower order big O notation than the 3x3 matrix cipher due to the fact that there are many more functions involved in calculating the inverse key. The vernam cipher has the advantage of having the same sized key as the text itself and being randomly generated. However this had its consequences as I had difficulties storing the large keys in the database and had to resort to altering the parameter data types. The 3x3 Matrix cipher has never been created before hence I had a few challenges to overcome. One of which was being able to

calculate the inverse key. This was due to the fact that it works differently than the usual mathematical formula because I had to implement the MOD function which mapped the indexes to the alphabet. I had to also consider the inclusion of numbers and special characters which I solved by changing the MOD value from 26 → 71 which was the size of the alphabet including the added numbers and special characters as well as capital letters.

(2)) My system had working SFTP functionality and I was able to upload files within the website. Unfortunately, I faced issues when retrieving files and a Method error would be displayed. I think this had to do with the relationship between my database tables and the queries I executed to retrieve the decryption key. It may have also been to do with how I named the files. Where I would upload each file with _encrypted attached to the end hence requesting a file with the original name would not work. I did however test only the retrieval function in a separate environment and it was successful and returned the encrypted hexadecimal. The progress bar was intended to be somewhat of a gimmick as all the files I uploaded were relatively small in size. And each iteration of the progress bar was in 32 kB chunks and all the files I attempted to upload were below that. As a result it was a little impractical.

(3)) The web application worked exactly as I had intended. I followed the data flows that I had designed. The web app module is initialised with the main function and calls the results of operations from the other two modules (SFTP and Encryption) to upload/retrieve a file. I did find a logic error within the web application where the user could bypass the login by entering the exact URL of the other pages. I wasn't sure how to solve this problem initially however after reviewing the code I realised that I wasn't checking that the user was logged in at each route. To solve this I should add a validation statement session.get('logged_in') at each redirect. I did not include this in the menu , upload and retrieve file pages. Otherwise I could also use a library that does this for you automatically. One library that does this is called wraps from functools. It's often referred to as a decorator. Otherwise the login procedure works as intended and the web application is able to upload and retrieve encrypted files from or to the user's computer.

(4)) The database is responsible for storing encryption keys and the files metadata. It played a pivotal role in my project as it stores which decryption key is associated with which file. This ensures that when the user requests to retrieve an encrypted file, the correct corresponding decryption key can be retrieved to successfully decrypt the file. The database was composed of three tables. The file table, encryption table and the relation table. The relation table was there to make the database normalised and so that there was a relation between the keys and the file name. One downfall I had was storing the cipher keys for the Vernam cipher within the keys table. This is due to the fact that the keys are the same length as the file itself. And because I convert the files to hexadecimal the keys tend to be quite long. As a consequence it was impossible to store the keys in the database because it exceeded the character limit. I tried to avoid this by changing the datatype to BLOB instead of JSON as there supposedly shouldn't be a character limit. However, it still produced the same error. Another difficulty of my project was storing the decryption keys on the database. This made it slightly less secure as access to the database would make it easier for someone to decrypt the files. Instead I should store the encryption keys and then on retrieval I will convert them to their inverse equivalent. That way it makes it harder for someone to get the inverse key if they do not have the conversion algorithm for it.

One improvement I would consider making to my project, if I were to do it again, would be to display all the available files on the NAS. That way they can still retrieve the file in case they have forgotten the name they have assigned it.

Overall my NEA was a success. I can upload and retrieve files from a remote NAS using SFTP which was the main objective of my project alongside having the . Additionally, the user has a choice in which cipher type they would like to use to encrypt their file. Both ciphers have a working method for decryption. I was successful in setting up a DMZ on my router. Which strengthened the security of my network.